isec
Engenharia

MESTRADO EM INFORMÁTICA E SISTEMAS

DEFINITIVO

Desenvolvimento de uma plataforma para classificação inteligente de famílias de proteínas

Autor

Hugo David Duarte Lucas

Orientador

Carlos Manuel Jorge da Silva Pereira

Coimbra, abril, 2021

INSTITUTO POLITÉCNICO
DE COIMBRA

INSTITUTO SUPERIOR
DE ENGENHARIA
DE COIMBRA

isec
Engenharia

DEPARTAMENTO DE INFORMÁTICA E SISTEMAS

Desenvolvimento de uma plataforma para classificação inteligente de famílias de proteínas

Relatório de Trabalho de Projeto para a obtenção do grau de Mestre em Informática e Sistemas

Especialização em Desenvolvimento de Software

Autor

Hugo David Duarte Lucas

Orientador

Carlos Manuel Jorge da Silva Pereira

Coimbra, abril, 2021

INSTITUTO POLITÉCNICO
DE COIMBRA

INSTITUTO SUPERIOR
DE ENGENHARIA
DE COIMBRA

Biology is the most powerful technology ever created. DNA is software, proteins are hardware, cells are factories.

Arvind Gupta

ABSTRACT

There is an urgent need to develop computational methods and large-scale bioinformatics protein annotation infrastructures that promote the discovery of biological knowledge, as decreasing costs and rapid sequencing have enabled increased availability of complete sequences of proteins genome for many organisms. To fully realize the value of the data, scientists need to identify proteins encoded by these genomes and understand how these proteins work. These data are sent to various public resources, which have become indispensable biological knowledge bases for biomedical research, as they provide data in easily accessible formats.

In this dissertation we present SmartGeno - a prototype that aims to assists the scientists with the task of classifying a protein family, which provides a set of important clues to its structure, activity and metabolic role, enabling the identification of proteins that are difficult to characterize, which is critical to understanding nature of the universe of proteins and the biology. SmartGeno allows the user to configure, train, and store machine learning models such as decision trees, Random Forest, and MLP neural networks that can be later used to classify protein families that must be submitted on the platform using a FASTA file. Using a dataset with 42086 protein sequences associated with 34 families and a vector representation of the sequences constructed from the primary structure of the protein, it was possible to obtain an F1Score = 88%, which proves that machine learning models such as Random Forest can be used successfully to predict the protein family. This study also demonstrates that deep learning techniques such as LSTM can be successfully used for this task, since we obtained a F1Score = 86% for the same dataset, using only raw protein sequences.

Keywords: Protein annotation, Machine learning, Protein classification, Protein family.

RESUMO

Existe uma necessidade urgente de desenvolver métodos computacionais e infraestruturas bioinformáticas para anotação de proteínas em larga escala que promovam a descoberta do conhecimento biológico pois, os custos decrescentes e o rápido sequenciamento permitiram um aumento da disponibilidade de sequências completas de genoma para muitos organismos. Para perceber completamente o valor dos dados, os cientistas precisam de identificar as proteínas codificadas por esses genomas e entender como essas proteínas funcionam. Esses dados são enviados para vários repositórios públicos que se tornaram em bases de conhecimento biológico indispensáveis para a pesquisa biomédica pois, fornecem dados em formatos de fácil acesso.

Neste projeto, apresentamos o SmartGeno - um protótipo que pretende auxiliar os cientistas na tarefa de classificar a família de proteínas que, fornece um conjunto de pistas importantes para sua estrutura, atividade e papel metabólico, permitindo a identificação de proteínas difíceis de caracterizar, o que é fundamental para entender a natureza do universo das proteínas e da biologia. O SmartGeno permite que o utilizador configure, treine e armazene modelos de *machine learning* como àrvores de decisão, Random Forest e redes neuronais MLP que podem ser usados posteriormente para classificar famílias de proteínas que devem ser submetidas na plataforma usando um ficheiro FASTA. Usando um *dataset* com 42086 sequências proteicas associadas a 34 famílias e uma representação vetorial das sequências construída a partir da estrutura primária da proteína foi possível obter um F1Score = 88%, o que comprova que modelos de *machine learning* como as Random Forest podem ser usadas com sucesso para prever a família das proteínas. Este estudo também demonstra que técnicas de aprendizagem profunda como as LSTM podem ser usadas com sucesso para essa tarefa, uma vez que obtivemos um F1Score = 86% para o mesmo *dataset*, usando apenas as sequências proteicas em bruto.

Palavras-Chave: Anotação de proteínas, *Machine learning*, Classificação de proteínas, Família de proteínas.

AGRADECIMENTOS

Agradeço ao professor Doutor. Carlos Pereira, orientador pelo ISEC, pelo apoio e ajuda prestada, sempre disponível para esclarecer qualquer dúvida em qualquer instante.

Agradeço ao Instituto Superior de Engenharia de Coimbra, e a todos os professores desta instituição que foram responsáveis pela minha formação académica, pela elevada qualidade de ensino praticado num espaço com excelentes instalações e condições de aprendizagem.

Agradeço à minha namorada pela ajuda, companhia e motivação prestada nesta etapa tão importante para mim.

Agradeço aos meus pais pela disponibilidade e pelo esforço que tiveram em ajudar-me a chegar até aqui.

Agradeço aos meus colegas e amigos de curso pela amizade, companhia, e disponibilidade para trabalharem em grupo.

Por último, um agradecimento à minha filha Olívia que nasceu durante a elaboração deste projeto, portanto a ela dedico em especial este trabalho.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

DEFINIÇÕES E ACRÓNIMOS

API – *"Application Programming Interface" é um protocolo de comunicação entre um cliente e um servidor que permite que o consumo de um conjunto de rotinas e funções de software sejam feitas sem a necessidade de se envolverem em processos demasiados complexos de implementação.*

AUC - *"Area Under the ROC Curve" é a área abaixo da curva de ROC e permite medir a capacidade de um classificador fazer a distinção entre classes.*

BPMN – *"Business Process Diagram" é um padrão para modulação de processos de negócios e fornece uma notação gráfica para a especificação de processos de negócios. O objetivo do BPMN é apoiar a gestão de processos de negócios tanto para utilizadores técnicos como para utilizadores de negócios, fornecendo uma notação intuitiva capaz de representar a semântica complexa do processo.*

CPU – *"Central Process Unit" é o principal componente de hardware de um computador. É responsável por calcular e executar as tarefas determinadas pelo utilizador e as suas características influenciam diretamente a velocidade de execução dos programas.*

CSS – *"Cascading Style Sheets" é uma linguagem de programação que serve para definir os estilos de uma página web (cores, fontes, tamanho de letra, etc.). Pode ser aplicado diretamente nas tags HTML ou pode ser definido num ficheiro próprio.*

CSV – *"Comma Separated Values" é um tipo de ficheiro de texto em que o seu conteúdo está normalmente separado por vírgulas. Tem como vantagem principal a capacidade de permitir importar e exportar informação que várias aplicações conseguem ler.*

GPU – *"Graphics Processing Unit" é um tipo de microprocessador responsável por processar os gráficos de um computador. As GPUs possuem milhares de núcleos o que é particularmente importante quando é necessário processar grandes conjuntos de dados.*

FTP – *"File Transfer Protocol"* é *um servidor que disponibiliza através de uma rede de computadores, um serviço que permite que os utilizadores acedam a um disco rígido ou servidor de ficheiros através do protocolo de transferência de ficheiros: File Transfer Protocal.*

HTML – *"HyperText Markup Language" é uma linguagem de marcação utilizada para o desenvolvimento de websites. Qualquer site na internet possui uma estrutura básica HTML e essa estrutura é constituída por tags que são responsáveis por mudar a formatação e a apresentação de uma página web.*

HTTP – *"Hypertext Transfer Protocol" é o protocolo usado pela World Wide Web para definir como as mensagens são formatadas e enviadas do browser para o utilizador de um website.*

IDE – *"Integrated Development Environment" é um programa de computador criado com o objetivo de apoiar os programadores no desenvolvimento de software promovendo um aumento na produtividade. As ferramentas mais comuns de um IDE incluem a edição do*

código-fonte numa linguagem de programação suportada pelo IDE e a compilação do mesmo em linguagem máquina.

JSON – *"JavaScript Object Notation" é um formato de dados compacto e simples de ler que permite a troca de dados entre diferentes sistemas de uma forma rápida e simples, tipicamente em webservices.*

ORM – "Object-Relational Mapping" *é uma técnica de conversão entre o modelo orientado a objetos usado na aplicação e o modelo relacional usado na base de dados relacional.*

POC – *"Proof of Concept"* é um *modelo prático que prova o conceito teórico estabelecido pela pesquisa realizada.*

RAM – *"Random Access Memory" é a memória principal de um computador usada para leitura e escrita de dados de forma rápida e temporária para que o computador consiga aceder-lhes agora ou nos próximos momentos.*

ROC – "Receiver Operating Characteristic Curve" *é um gráfico que mostra o desempenho de um modelo de classificação para todos os limites possíveis.*

SQL – *"Structured Query Language" é uma linguagem de programação padrão projetada para manipular dados em base de dados relacionais.*

SSD – *"Solid State Drive" é um componente de hardware com o propósito de armazenamento de dados de um computador. O SDD veio substituir o antigo disco rígido, conhecido como Hard Disk e é muito mais rápido uma vez que não possui discos físicos ou agulhas magnéticas, permitindo que o computador seja mais eficiente a abrir programas ou a executar tarefas.*

TPU – *"Tensor processing unit" é um circuito integrado de aplicação especifica desenvolvido pela Google com o objetivo de acelerar as cargas de trabalho das técnicas de machine learning.*

UML – *"Unified Modeling Language"* é uma *linguagem de notação utilizada em projetos de desenvolvimento de software. Esta linguagem é expressa através de diagramas que são usados para detalhar a estrutura ou o comportamento do sistema.*

UI – *"User Interface" é tudo aquilo que é percetível visualmente numa plataforma e leva a uma interação entre homem-máquina. Esta interação permite que o utilizador manipule um sistema que deverá produzir os efeitos das ações do utilizador.*

XML – *"Extensible Markup Language" é uma linguagem de marcação utilizada para criar documentos com dados organizados hierarquicamente.*

WSGI – *"Web Server Gateway Interface" é uma especificação universal para uma interface entre servidores web e aplicações web ou frameworks para a linguagem de programação Python.*

1 INTRODUÇÃO

Neste capítulo apresentam-se as principais motivações que levaram à realização do projeto e quais os objetivos a atingir.

1.1 Motivação

A classificação de proteínas em famílias é um dos problemas mais relevantes na área da bioinformática uma vez que há um esforço contínuo em organizar as proteínas em famílias. A identificação confiável de famílias de proteínas é fundamental para a análise filogenética, para a anotação funcional e para a exploração da diversidade da função das proteínas num determinado ramo filogenético, o que é fundamental para entender a natureza do universo das proteínas e em essência, a biologia [1].

As experiências de laboratório levam um tempo considerável para anotar sequências proteicas, no entanto, os custos decrescentes e a rapidez do sequenciamento permitiram um aumento na disponibilidade de sequências completas de genoma para um grande número de organismos. Dessas sequências apenas 1% estão anotadas manualmente [1], tornando-se cada vez mais necessário a utilização de técnicas de *machine learning* que permitam de forma automática e em tempo útil, uma correta anotação das restantes sequências proteicas.

1.2 Objetivos

Este trabalho divide-se em duas vertentes: uma vertente de investigação onde são explorados classificadores de aprendizagem supervisionada, nomeadamente árvores de decisão e redes neuronais simples e recorrentes para auxiliar os cientistas na tarefa de classificação das famílias das proteínas; e uma vertente de desenvolvimento que tem como objetivo a elaboração de um protótipo capaz de dar suporte a essa tarefa – o SmartGeno [2]. Pretende-se que a ferramenta computacional desenvolvida dê aos utilizadores a liberdade de configurar, parametrizar e armazenar modelos de *machine learning* que usam como entrada uma representação vetorial codificada das sequências proteicas, construída a partir de propriedades estruturais de proteínas e propriedades físico-químicas dos aminoácidos, que incluem conteúdo hidrofóbico, aromaticidade, peso molecular, ponto isoelétrico, tamanho da sequência e frequência absoluta de cada aminoácido.

1.3 Planeamento

O processo de desenvolvimento deste trabalho consistiu em 5 tarefas distintas:

1. Introdução ao tema;

2. Investigação e desenvolvimento de modelos de classificação;
3. Especificação da arquitetura do SmartGeno;
4. Desenvolvimento do SmartGeno;
5. *Deployment* do SmartGeno;
6. Documentação do trabalho realizado

Na primeira tarefa foi elaborada uma proposta do projeto a realizar. Para isso foi necessário realizar alguns estudos preliminares, um estado de arte e foi feita uma seleção das ferramentas e tecnologias a utilizar.

A segunda tarefa teve como objetivo a investigação e o desenvolvimento dos seguintes modelos de classificação:

- Árvores de decisão;
- Random Forest;
- Rede Neuronal Multilayer Perceptron (MLP);
- Rede Neuronal Long Short-Term Memory (LSTM)

A terceira tarefa iniciou-se após a aceitação da proposta e da demonstração de uma Proof of Concept (POC) funcional aos *stakeholders* do projeto. Esta tarefa subdividiu-se em 2 fases: A primeira fase foi o *design* da arquitetura do SmartGeno, onde se estabeleceu-se o *design* de alto nível da base de dados a ser utilizada, definiram-se os casos de uso, identificaram-se as principais áreas de *log* e auditoria e foram selecionadas as ferramentas de integração contínua. A segunda fase consistiu na listagem de requisitos do *software*, onde foi selecionada a abordagem para o levantamento de requisitos, trabalhou-se com os *stakeholders* para a consolidação de requisitos e identificaram-se as interfaces externas necessárias. A segunda tarefa terminou com a apresentação de um *workshop* intercalar em ambiente de sala de aula, onde foi possível recolher o *feedback* dos alunos e professores presentes.

Após a definição dos requisitos e da configuração de todos os ambientes necessários (teste, e produção) foi possível iniciar a tarefa de desenvolvimento, que incluiu as seguintes atividades:

- *Log* de auditoria;
- Projeção e criação da base de dados necessária (incluindo os *scripts* necessários);
- Desenvolvimento do protótipo SmartGeno;
- Criação e execução de testes unitários;
- Relatório de testes

Assim que a tarefa de desenvolvimento terminou, iniciou-se a configuração do ambiente de produção. De seguida, realizou-se o *deployment* do código-fonte desenvolvido, de modo a permitir que os *stakeholders* pudessem usar o SmartGeno em qualquer dispositivo com acesso à rede. Já em ambiente de produção, foi necessário monitorizar e resolver eventuais problemas no *software*.

Por fim, a última tarefa consistiu na documentação de todo o trabalho realizado.

O projeto teve início a 15 de outubro de 2018 e correspondeu a um total de 1170 horas de trabalho e o seu planeamento foi realizado segundo um gráfico de Gantt, tal como apresentado

na Figura 1 que contém a estimativa (em dias) para a execução das diferentes fases do projeto enquanto que, a Figura 2 contem o planeamento temporal real.

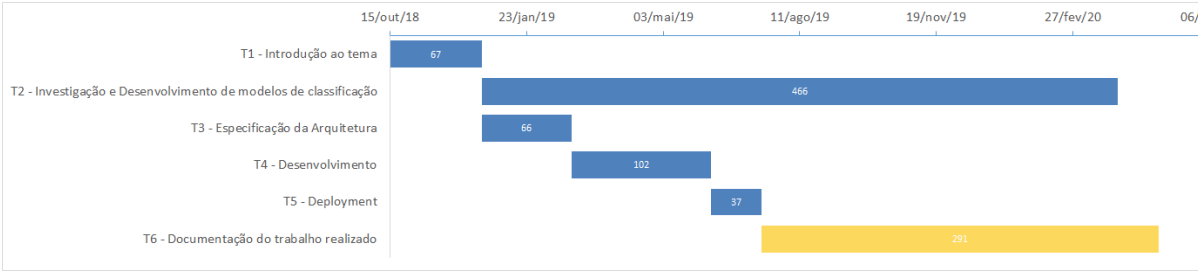

Figura 1 Diagrama de Gantt representativo do planeamento temporal previsto

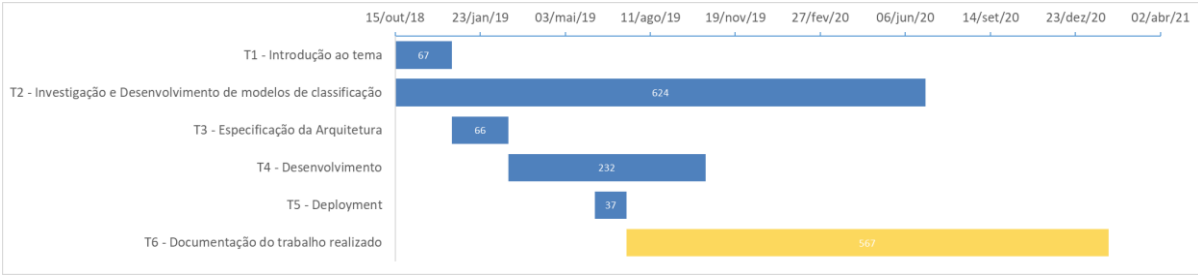

Figura 2 Diagrama de Gantt representativo do planeamento temporal real

Comparando o diagrama de Gantt planeado com o real, verifica-se que ocorreram ligeiros desvios relativamente às tarefas planeadas, mas nunca comprometendo a execução do projeto final. As principais diferenças verificam-se no tempo de desenvolvimento do protótipo, no *deployment* e no tempo da documentação do trabalho realizado.

Uma maior duração no tempo de desenvolvimento deveu-se a algumas alterações nas tecnologias planeadas, uma vez que inicialmente pensou-se em realizar o protótipo utilizando a linguagem de programação C# [3] juntamente com a biblioteca de *machine learning* scikit-learn [4]. Após uma pesquisa exaustiva e de várias tentativas falhadas de realizar uma POC funcional, conclui-se que não é possível carregar diretamente modelos scikit-learn em C#. Apesar do .NET permitir executar *scripts* Python usando bibliotecas como o IronPython [5], o scikit-learn depende fortemente de Numpy [6] e SciPy [7] que possuem muitas extensões em linguagem C e Fortran. Outra abordagem que foi tida em conta foi a substituição do scikit-learn pela Accord.NET [8] - uma *framework* de *machine learning* para .NET no entanto, a Accord.NET embora já tenha sido referenciado em várias publicações [9] e tenha uma boa documentação, ainda se encontra numa fase de maturação quando comparada com o scikit-learn. Por esses motivos optou-se por substituir a tecnologia .NET pela *framework* Django, exigindo algum tempo no *ramp-up* para a consolidação de conhecimentos numa nova linguagem de programação.

Uma maior duração no tempo da documentação do trabalho realizado deveu-se essencialmente ao aparecimento de uma pandemia causada por um surto de coronavírus (SARS-CoV-2) que inevitavelmente proporcionou constrangimentos e alterações nos métodos de trabalho.

O *deployment* foi realizado antes do protótipo final estar concluído de forma a mostrar aos *stakeholders* incrementos do *software* desenvolvido.

1.4 Estrutura do relatório

Este relatório de projeto encontra-se organizado em 7 capítulos. Após o primeiro capítulo introdutório, segue-se o segundo capítulo onde são apresentados alguns conceitos fundamentais e necessários ao longo deste trabalho.

O segundo capítulo contém também uma visão geral do problema de classificação de proteínas, descrevendo a sua importância e explica como é que se extrai conhecimento a partir de dados de sequências completas de genoma, enumerando algumas das principais fontes de origem desses dados.

No terceiro capítulo é apresentado o estado da arte (SoA) sobre plataformas e algoritmos de *machine learning* usados na tarefa de classificação da família de proteínas, o que serviu como um ponto de partida para o trabalho a ser desenvolvido.

O quarto capítulo descreve a metodologia usada para responder aos objetivos propostos neste trabalho. Contém a descrição do *dataset* utilizado, as técnicas de pré-processamento que foram aplicadas sobre a fonte de dados, os modelos de *machine learning* responsáveis pela construção de um classificador com base nesses dados e por fim, as métricas de avaliação usadas para medir a performance dos modelos.

O capítulo 5 está intimamente ligado ao SmartGeno. Este capítulo contém a visão da solução, enumera as principais funcionalidades, descreve as ferramentas e tecnologias que foram necessárias para o seu desenvolvimento e apresenta as várias etapas do seu ciclo de vida do desenvolvimento.

O capítulo 6 contém os resultados experimentais obtidos, incluindo uma análise e discussão dos mesmos.

Por último, no capítulo 7 é feita uma análise crítica ao trabalho desenvolvido e são propostos trabalhos futuros.

2 PROBLEMA DE CLASSIFICAÇÃO DE PROTEÍNAS

Este projeto enquadra-se na área da bioinformática, portanto, a revisão e a compreensão de alguns conceitos de biologia são fundamentais para estabelecer a ponte entre o problema na área da biologia e o objetivo a atingir na área da informática.

2.1 Conceitos Biológicos

2.1.1 ADN

O ADN é a sigla para ácido desoxirribonucleico, uma molécula que está presente no núcleo das células de todos os seres vivos e alguns vírus. Tem como função principal o armazenamento e transporte de informações genéticas do organismo [10]. O ADN apresenta uma estrutura dupla hélice onde as hélices são constituídas por fosfato e unidas por bases nitrogenadas que incluem a Adenina (A), Timina (T), Citosina (C) e a Guanina (G). A estrutura em dupla hélice é essencial para o sucesso da função de manutenção das características e informações básicas do organismo durante a divisão celular, possibilitando a replicação da estrutura. Na função de transporte de informações contidas nas sequências do ADN, também conhecidas como genes, as informações dos genes são copiadas para uma sequência de ácido ribonucleico (RNA) complementar, num processo conhecido como transcrição (Figura 3). Esta sequência pode usar usada posteriormente para construir outra sequência proteica num processo conhecido como tradução (Figura 4).

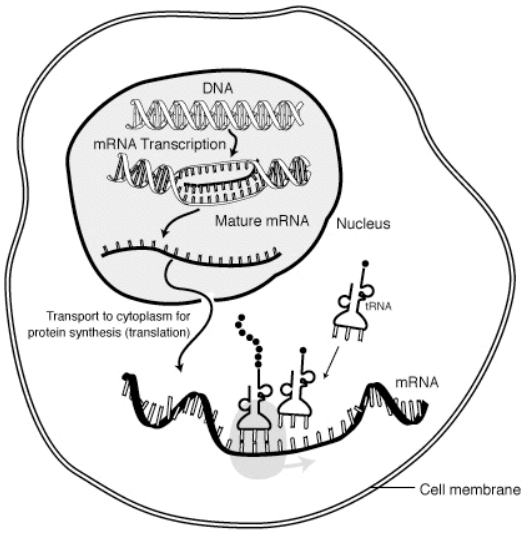

Figura 3 Transcrição [11]

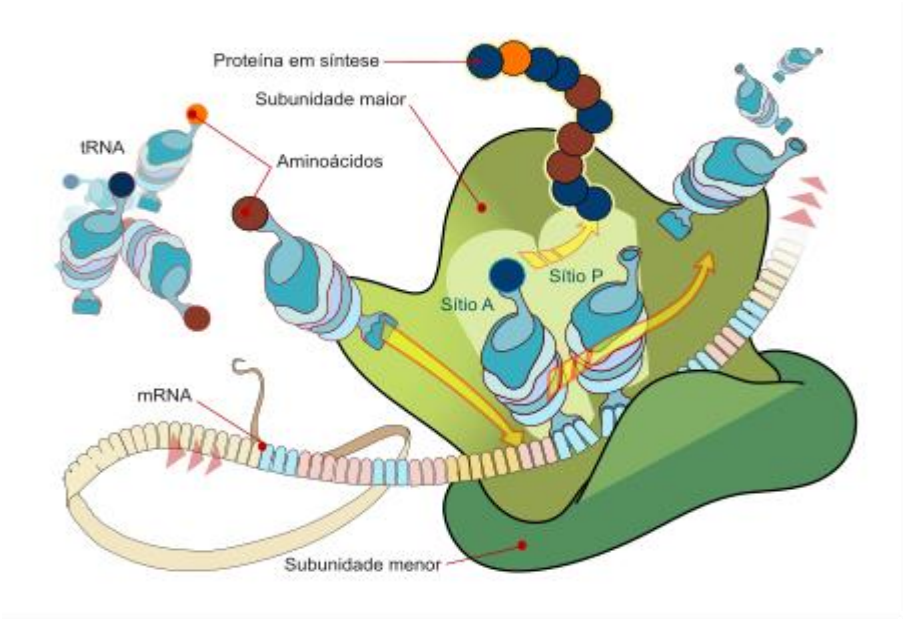

Figura 4 Tradução [12]

2.1.2 Sequenciamento do Genoma

O sequenciamento do genoma é uma técnica utilizada para obter o número e ordem das bases nitrogenadas adenina (A), guanina (G), citosina (C) e timina (T) que fazem parte do ADN de um ser vivo, o que permite determinar toda a sequência de ADN do organismo.

A primeira geração de sequenciamento do ADN surgiu em 1977 e ficou marcada pelo uso de técnicas de degradação química e processos pouco escaláveis e extremamente demorados. Das técnicas desenvolvidas na primeira geração, também conhecida como sequenciamento clássico, destacam-se o método de Maxam–Gilbert desenvolvido por Allan Maxam e Walter Gilbert e o método de Sanger, desenvolvido por Frederick Sanger [13].

Uns anos mais tarde, iniciou-se aquele viria a ser um marco na história da ciência – o Projeto Genoma Humano [14]. O projeto foi iniciado formalmente em 1990 e tinha uma estimativa de 15 anos para o seu desenvolvimento. Os principais objetivos desse projeto eram a identificação de todos os genes de um humano e a determinação da sequência dos cerca de 3,2 mil milhões de pares de bases que compõem o genoma do *homo sapiens*. O projeto foi concluído com sucesso em 2003, permitindo o aparecimento de uma magnífica fonte de informação biológica que serviu de base para a pesquisa e descoberta de várias aplicações práticas e possibilitou a comparação com outros projetos de genomas de outros organismos.

O avanço da tecnologia impulsionou o desenvolvimento de novas tecnologias para o sequenciamento do ADN com menor custo (Figura 5), maior qualidade e maior rapidez. Estes novos métodos surgiram em 2005 e marcaram o início da era do Sequenciamento da Nova Geração (SNG). Dentre desses métodos, destacam-se o 454 Roche, o Illumina, o Ion-torrent, o PacBio e o Nanopore [13]. Atualmente o Illumina (Figura 6) domina aproximadamente 75% do mercado de sequenciamento do ADN devido à sua alta precisão e alta taxa de transferência [15].

O sequenciamento do ADN promove:

- A compreensão da linha de evolução de uma espécie;
- A comparação do ADN entre espécies diferentes;
- O diagnóstico de doenças;
- O desenvolvimento de novos medicamentos;
- Ajuda a prevenir males que se possam vir a desenvolver

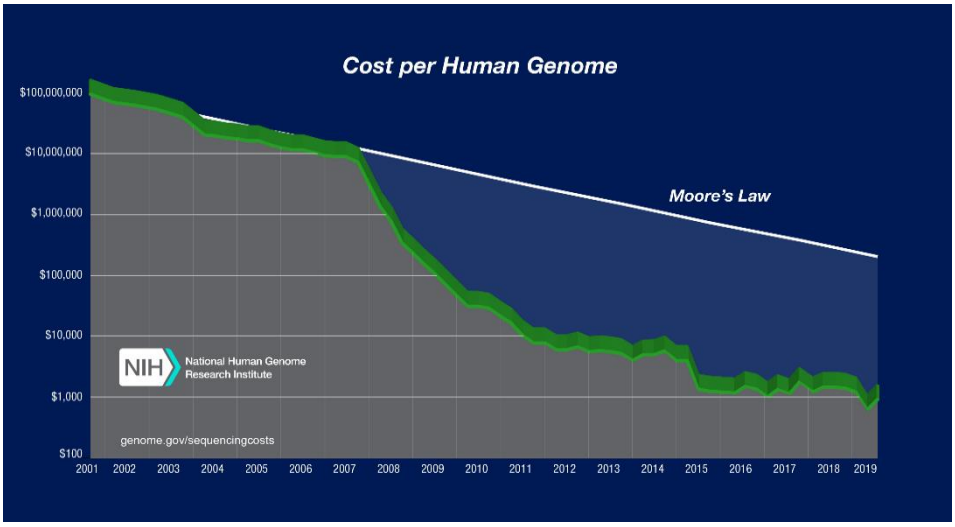

Figura 5 Evolução do custo do sequenciamento do genoma humano ao longo do tempo [16]

Figura 6 Illumina – um sequenciador da nova geração que domina 75% do mercado [17]

2.1.3 Aminoácido

Os aminoácidos são compostos orgânicos que se combinam para formar proteínas. São moléculas compostas por átomos de azoto, carbono, hidrogénio, oxigénio e por vezes podem conter enxofre. A Figura 7 demonstra a estrutura geral dos aminoácidos, que envolve um grupo

amina e um grupo carboxilo, ambos ligados ao carbono. O carbono também é ligado a um hidrogénio e a uma cadeira lateral, que é representada pela letra R. O grupo R determina a identidade de um aminoácido específico.

O corpo humano possui um total de 20 aminoácidos que podem ser representados por uma letra do alfabeto - uma abordagem simplista bastantemente utilizada para dar suporte a técnicas computacionais para armazenamento de centenas de resíduos de aminoácidos e para a sua avaliação. A Tabela 1 contém a notação mais recente e foi proposta pela Commission on Biochemical Nomenclature (CBN) [18] como resposta ao crescente número de diferentes notações e os problemas presentes, garantido assim uma única notação. Além da representação para os 20 aminoácidos, a Tabela 1 contém uma representação adicional para um aminoácido desconhecido - a letra do alfabeto 'X'.

Tabela 1 Nomenclatura dos aminoácidos [18]

Aminoácido	***Símbolo 1 Letra***	***Símbolo 3 Letras***
Alanine	A	Ala
Arginine	*R*	Arg
Asparagine	*N*	Asn
Aspartic acid	*D*	Asp
Cysteine	*C*	Cys
Glutamic acid	*E*	Glu
Glutamine	*Q*	Gln
Glycine	G	Gly
Histidine	*H*	His
Isoleucine	*I*	Ile
Leucine	L	Leu
Lysine	K	Lys
Methionine	M	Met
Phenylalanine	F	Phe
Proline	*P*	Pro
Serine	*S*	Ser
Threonine	*T*	Thr
Tryptophan	*W*	Trp
Tyrosine	*Y*	Tyr
Valine	*V*	Val
Desconhecido	*X*	-

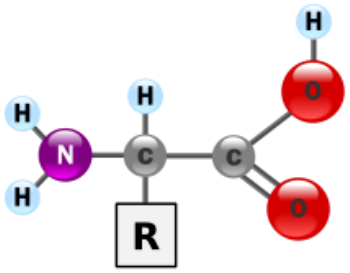

Figura 7 Estrutura geral de um aminoácido [19]

2.1.4 Proteína

As proteínas são uma classe de macromoléculas constituídas por uma ou mais cadeias de aminoácidos. Existem muitas proteínas diferentes num organismo, ou mesmo numa única célula e apresentam tamanho e formas distintas. Dependendo das interações com o seu ambiente e das sequências de aminoácidos as proteínas dobram-se numa estrutura tridimensional, o que lhes permite interagir com outras proteínas e moléculas e executar a sua função. As proteínas executam essas funções conforme a informação codificada nos genes e podem ser divididas em três classes principais: proteínas globulares, proteínas fibrilares e proteínas membranares. As proteínas são consideradas fundamentais para a vida uma vez que executam a maioria das funções biológicas, que incluem [20]:

- Replicação do ADN;
- Transporte de moléculas de uma célula para a outra;
- Aceleração das reações metabólicas;
- Proteção do corpo, uma vez que algumas podem funcionar como anticorpos ao ligarem-se a partículas estranhas específicas, como vírus e bactérias

2.1.5 Estrutura de Proteína

A estrutura de uma proteína é crítica para a função e pode ser classificada em quatro diferentes níveis de estrutura: primária, secundária, terciária e quaternária, tal como apresentado na Figura 8.

Na estrutura primária a cadeia principal da proteína é formada por ligações dos aminoácidos apresentados na Tabela 1. Qualquer mudança na sequência do ADN pode levar à mudança da sequência dos aminoácidos, o que conduz à mudança da estrutura geral da proteína e a sua função. Um exemplo disso é a anemia falciforme, uma doença que resulta da troca do ácido glutamato (E) com a valina (V) na sexta posição da hemoglobina β, um dos tipos de cadeias proteicas que formam a hemoglobina [21].

A estrutura secundária resulta da dobra (*fold*) da cadeia de aminoácidos em estruturas secundárias locais e os tipos mais comuns de estruturas são a α-hélice e a folhas β. As formas

de ambas as estruturas são mantidas por ligações que ocorrem entre o hidrogénio do grupo N-H com o oxigénio do grupo C=O. A estrutura secundária é determinada pela sequência, classificando cada aminoácido no elemento da estrutura, por exemplo α-hélice.

A estrutura terciária é descrita no espaço (x, y, z) pelas coordenadas de todos os átomos de uma proteína e resulta da dobra da estrutura primária sobre si, mediada por pontes de enxofre.

Por último, as várias estruturas terciárias podem interagir ou unir-se para formar proteínas mais complexas. Essas proteínas mais complexas correspondem à estrutura quaternária que, é representada através das coordenadas de todos os átomos e de todas as cadeias principais.

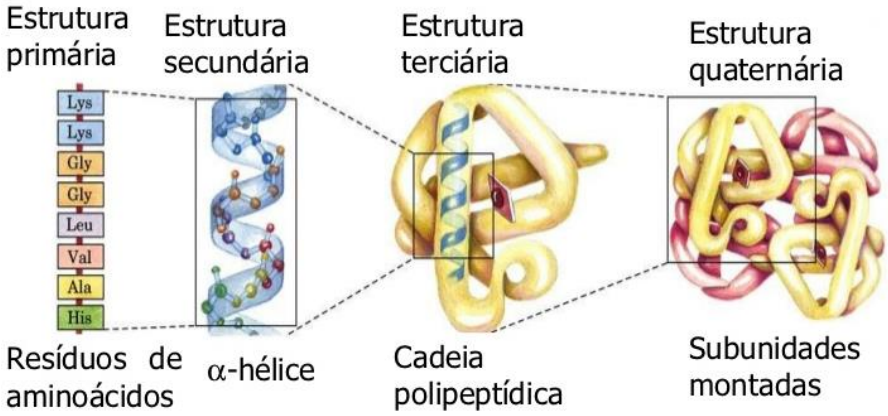

Figura 8 Tipos de estruturas das proteínas [22]

2.1.6 Família de Proteínas

Uma família de proteínas é um grupo de proteínas que descendem de um ancestral comum e geralmente possuem estruturas tridimensionais, funções e sequências com semelhança significativa. A disposição dos aminoácidos na sequência é o indicador mais estrito da homologia e, portanto, o indicador mais claro de ancestralidade comum [23]. As famílias de proteínas são frequentemente organizadas em hierarquias, com proteínas que partilham um ancestral comum subdivididas em grupos menores e mais relacionados, tal como apresentado na Figura 9. O termo superfamília é o maior agrupamento de proteínas para o qual a ancestralidade comum pode ser inferida. Essa ancestralidade comum é definida pelo alinhamento estrutural (Figura 10), mesmo que não exista homologia a nível da sequência. O termo subfamília é o nível mais abaixo e agrupa proteínas com uma estreita relação evolutiva. As subfamílias são identificadas por métodos que incluem a semelhança da sequência [24], função [25] ou ramo filogenético [26].

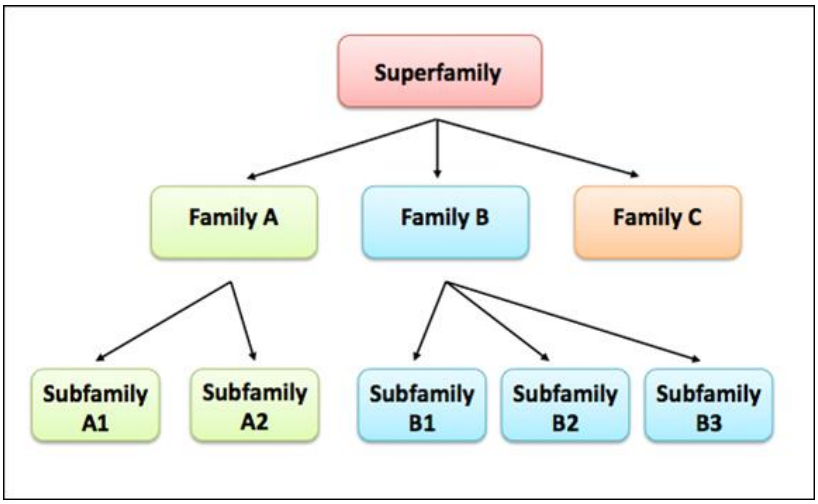

Figura 9 Hierarquias de famílias de proteínas [27]

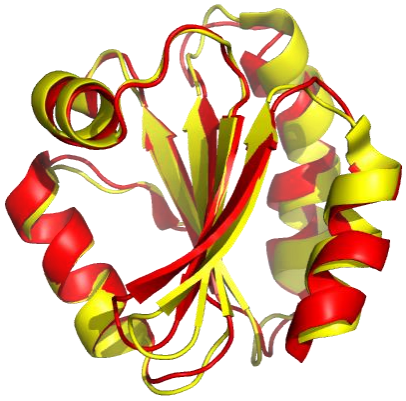

Figura 10 Alinhamento estrutural de uma proteína de humanos (tierredoxina), representada a vermelho, com uma proteína da mosca da fruta, representada a amarelo [28]

2.2 Classificação de Família de Proteínas

Não existe uma maneira única de classificar famílias de proteínas e a fronteira entre famílias diferentes pode ser subjetiva. As técnicas mais comuns de classificação de famílias de proteína incluem a semelhança da sequência, estrutura ou função e a sua escolha depende em parte do problema.

O objetivo da classificação da família de proteínas é agrupar proteínas em famílias, uma vez que uma proteína no contexto de família é muito mais informativa do que a própria proteína. Por exemplo, duas proteínas classificadas numa mesma família podem sugerir que partilham entre si estruturas semelhantes, mesmo quando as suas sequências não apresentam semelhanças. A classificação de famílias de proteínas fornece pistas valiosas para a sua estrutura, atividade e papel metabólico.

2.2.1 Repositórios Públicos

Um dos primeiros passos na tarefa de classificação de proteínas passa pela pesquisa de uma base de dados uma vez que efetuar a comparação entre proteínas fornece informação sobre a relação entre proteínas dentro de um genoma ou entre espécies diferentes, portanto, oferece muito mais informação do que estudar uma proteína de forma isolada. As bases de dados de proteínas atualmente estão disponíveis maioritariamente na *Internet* permitindo que os dados sejam facilmente mantidos e atualizados com o mínimo de custo. A maiorias das bases de dados fornecem aos seus utilizadores mecanismos de pesquisa interativa para que os mesmos possam especificar as suas necessidades e obter as informações relacionadas de forma interativa. Algumas bases de dados permitem também o *upload* de dados que são imediatamente verificados e validados pelo servidor de base de dados.

De seguida, são apresentadas algumas das bases de dados públicas de famílias de proteínas mais conhecidas:

2.2.1.1 Pfam

O Pfam [29] é uma base de dados de famílias de proteínas curadas (validadas experimentalmente), cada uma das quais definida por dois alinhamentos e um perfil do modelo oculto de Markov (HMM) - um modelo probabilístico usado para a inferência estatística de homologia construída a partir de um conjunto de sequências representativas de famílias alinhadas [30]. Na Pfam, o perfil HMM é pesquisado numa grande coleção de sequências obtidas do UniProt Knowledgebase [31]. Os dados da Pfam estão disponíveis em *flatfiles* (uma base de dados armazenada num ficheiro) MySQL e podem ser descarregados através de um File Transfer Protocal (FTP). Os dados podem também ser obtidos de uma forma interativa a partir do acesso ao portal *web*, que fornece diferentes maneiras de aceder aos dados da base de dados, com recurso a várias representações gráficas.

A *release* 32.0 da Pfam foi publicada em Setembro de 2018 e contém um total de 17929 famílias de proteínas [29].

2.2.1.2 PANTHER

PANTHER [32], sediada na University of Southern California, CA, EUA, é uma base de dados que subdividiu famílias de proteínas em subfamílias com funções relacionadas, usando a experiência humana. Essa divisão permitiu uma associação mais precisa da função, bem como a determinação dos aminoácidos importantes para o desempenho da sua função [33]. No PANTHER para cada família e subfamília é construído um modelo HMM que é usado para classificar sequências proteínas adicionais. Os dados do PHANTER podem ser utilizados para obter informações sobre um gene específico, descobrir famílias e subfamílias, processos

biológicos, funções moleculares, componentes da célula e para associar uma lista de genes relacionados a uma família / subfamília em particular.

A *release* 14.1 do PANTER foi publicada em Dezembro de 2019 e contém um total de 107627 subfamílias e 15524 famílias [34].

2.2.1.3 CATH - Gene3D

CATH – Gene3D (Class, Architecture, Topology and Homologous superfamily) [35] foi criada nos anos 90 e continua a ser desenvolvido pelo grupo Orengo na Universidade Pública de Londres. É uma base de dados que fornece uma classificação hierárquica de estruturas e domínios de proteínas. O CATH obtém as estruturas tridimensionais do Protein Data Bank [36] e dividi-as em cadeias polipeptídicas consecutivas. A combinação de ferramentas computacionais e a validação experimental permite a identificação dos domínios dentro dessas cadeias e classifica-os de acordo com a hierarquia estrutural apresentada na Tabela 2.

Tabela 2 Os quatro níveis principais da hierarquia CATH

Nível	***Descrição***
C*lass*	Os domínios são atribuídos de acordo com o conteúdo da estrutura secundária.
A*rchitecture*	Alta semelhança estrutural, mas nenhuma evidência de homologia.
T*opology*	Um agrupamento em larga escala de topologias que partilham características estruturais particulares.
H*omologous superfamily*	Indicativo de uma relação evolutiva demonstrável.

A *release* v.4.2 foi publicada em Julho de 2017 e contém um total de 6119 de superfamílias [37].

2.2.1.4 SCOP

O Structural Classification of Proteins (SCOP) é uma base de dados que classifica manualmente os domínios de proteínas com base na semelhança estrutural e na sequência de aminoácidos, permitindo determinar uma relação evolutiva entre as proteínas. Proteínas com as mesmas formas (Figura 10) mas com poucas semelhanças a nível de sequência ou semelhança funcional são colocadas em superfamílias diferentes e supõe-se que tenham um ancestral comum distante. As proteínas que têm a mesma forma e alguma semelhança a nível de sequência ou função são colocadas em famílias e presume-se que tenham um ancestral comum mais próximo [38].

O SCOP obtém as estruturas das proteínas do Protein Data Bank [36] e classifica-as com base na seguinte hierarquia [39]:

Tabela 3 Níveis de Hierarquia do SCOP

Nível	***Descrição***
Class	Tipo de dobra (Ex: folhas β)
Fold	Diferentes formas de domínios numa classe.
SuperFamily	Os domínios numa dobra são agrupados em superfamílias que têm pelo menos um ancestral comum distante.
Family	Os domínios de uma superfamília são agrupados em famílias que têm um ancestral comum. Para isso têm de ter semelhança sequencial >30% ou apresentar alguma semelhança sequencial e executarem a mesma função.
Protein Domain	Os domínios nas famílias são agrupados em domínios proteicos.
Species	Os domínios proteicos são agrupados de acordo com as espécies.
Domain	Parte de uma proteína ou para proteínas simples pode ser a proteína inteira.

A *release* v2 foi publicada em janeiro de 2020 e contém um total de 5134 famílias e 2485 superfamílias. Os níveis de classificação organizam mais de 41000 domínios não redundantes que representam mais de 504000 estruturas de proteínas [39].

3 ESTADO DA ARTE

Este capítulo descreve o SoA de algumas das plataformas para classificação de proteínas, bem como o SoA do uso de algoritmos de *machine learning* com o objetivo de predizer as famílias de proteínas.

3.1 Plataformas

3.1.1 EBI – InterPro

O InterPro [40] é uma ferramenta de análise funcional de proteínas utilizada para classificar famílias e prever domínios. O InterPro usa modelos preditivos, também conhecidos como assinaturas, fornecidos for várias bases de dados que no seu conjunto compõe o seu consórcio.

Quando assinaturas diferentes correspondem ao mesmo conjunto de proteínas na mesma região da sequência, presume-se que pertencem à mesma família ou domínio e são colocadas numa entrada única no InterPro por um curador [41]. Agrupar assinaturas equivalentes traz benefícios óbvios uma vez que fornece assinaturas, nomes e anotações consistentes numa única base de dados. Atualmente o InterPro é composto por assinaturas do CATH-Gene3D [35], CDD [42], HAMAP [43], PANTHER [34], Pfam [30], PIRSF [44], PRINTS [45], PROSITE [46], SFLD [47], SMART [48], SUPERFAMILY [49] e TIGRFAMS [50].

Os métodos computacionais usados para produzir assinaturas de proteínas são os seguintes:

- Expressões regulares (PROSITE patterns);
- Impressões digitais que usam matrizes de sequência de posição específica (PRINTS);
- Agrupamento de sequências usando PSI-BLAST (PRODOM);
- Matrizes de sequências (PROSITE profiles, HAMAP);
- Modelos ocultos de Markov (Pfam, TIGRFAMs, PIRSF, SUPERFAMILY, CATH-Gene3D, PANTHER, SMART)

No InterPro a análise funcional de proteínas pode ser feita através da plataforma *web* (https://www.ebi.ac.uk/interpro/search/sequence/), através de um serviço de *email* interativo usando instruções específicas, através de *webservices* baseados em SOAP ou REST e por fim, para utilizadores que exigem um alto rendimento das suas aplicação podem fazer o *download* de uma versão independente em ftp://ftp.ebi.ac.uk/pub/databases/interpro/iprscan. A Figura 11 mostra o menu da plataforma *web* onde o utilizador poderá carregar uma sequência no formato FASTA com um comprimento máximo de 40000 aminoácidos ou carregar múltiplas sequências usando um ficheiro. Outra das funcionalidades oferecidas neste menu é a possibilidade de o utilizador poder filtrar as bases de dados de onde o UniProt recolhe as assinaturas. Após concluir o treino será aberta uma janela com o aspeto apresentado na Figura 12, onde o utilizador poderá observar a(s) família(s) prevista(s).

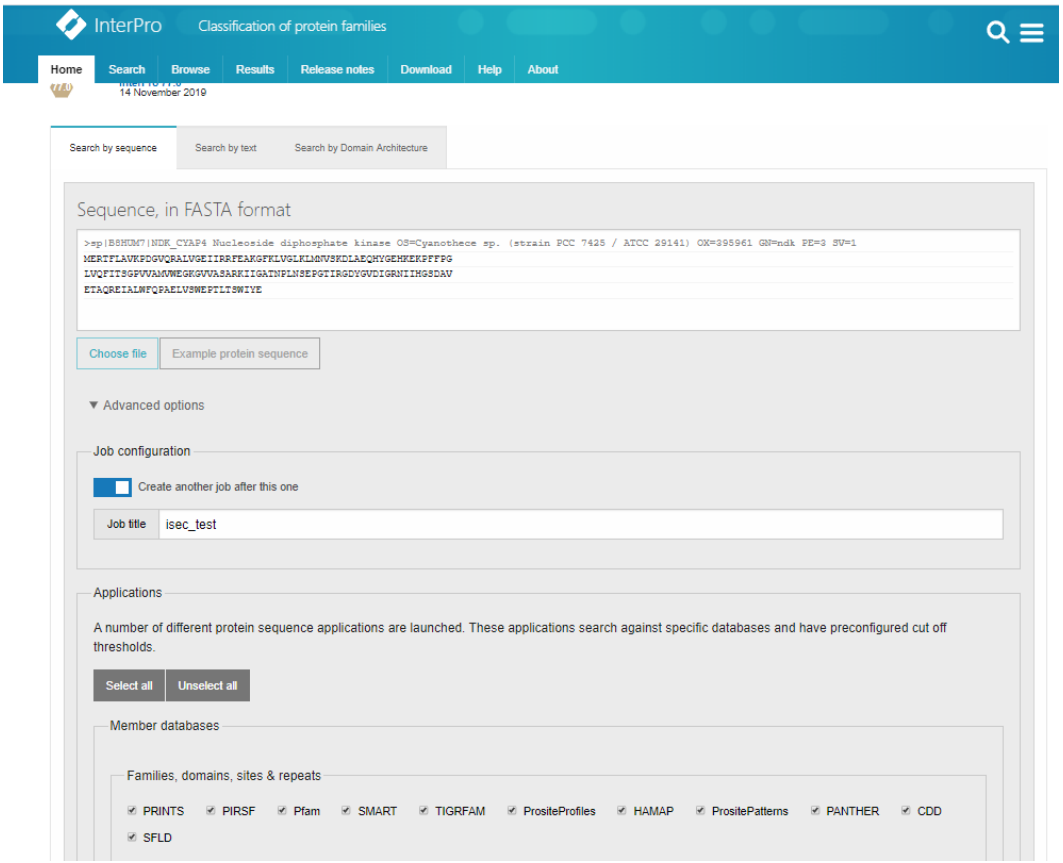

Figura 11 Aplicação *web* EBI – InterPro [51]

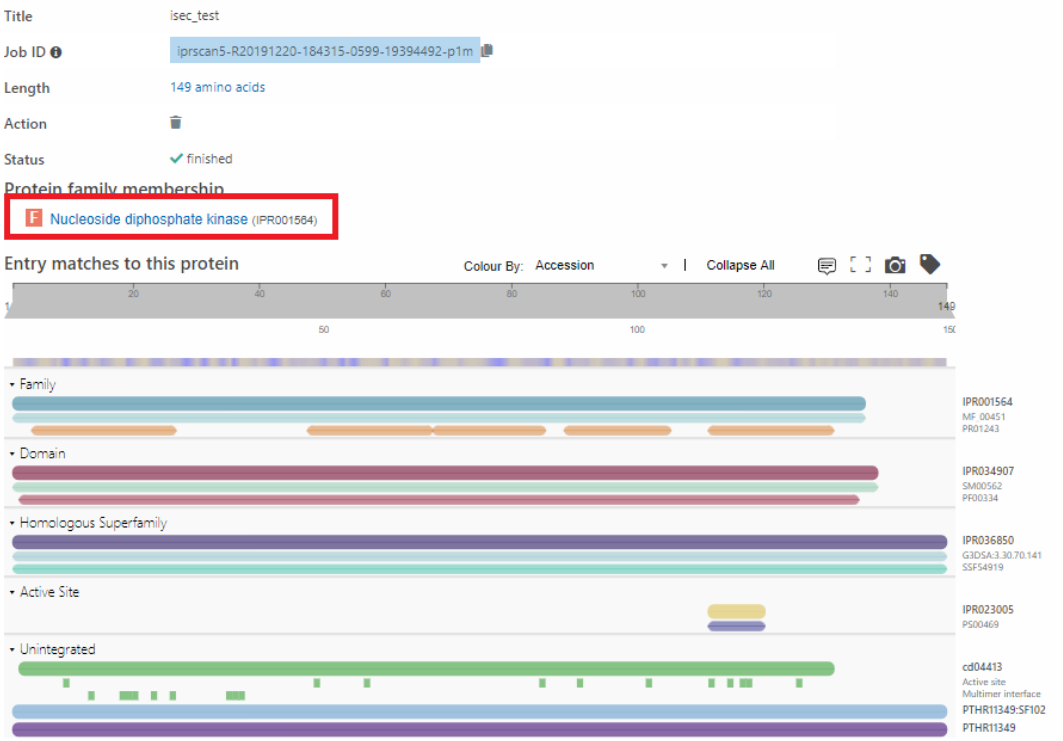

Figura 12 Classificação de família de proteína usando a aplicação *web* EBI – InterPro [51]

3.1.2 HAMAP-Scan

O HAMAP-Scan [52] é um sistema usado para classificação e anotação automática de sequências proteicas. Consiste numa coleção de perfis de famílias de proteínas que determinam a associação à família da proteína e regras especializadas para anotação funcional de membros da família [53]. Os perfis de família e as regras de anotação são criados e mantidos por curadores experientes e usam dados do UniProtKB / Swiss-Prot [54]. Tal como referido anteriormente, o HAMAP é também uma base de dados do consórcio do InterPro e também é usado para anotar proteínas no UniProtKB através de uma *pipeline* de anotação automática do UniProt que faz parte do UniRule [55].

O *webservice* do HAMAP-Scan (https://hamap.expasy.org/hamap_scan.html) pode ser visto como uma boa alternativa para pequenos projetos de pesquisa no entanto, grandes projetos de sequenciamento de genoma não devem depender de *webservices* para processar um grande volume de dados.

3.2 Algoritmos de *Machine Learning*

Existem diversos trabalhos para identificar as funções das proteínas com base nas sequências proteicas, no entanto as técnicas usadas tendem a acompanhar a evolução tecnológica. As técnicas antigas geralmente faziam uso do alinhamento das sequências. Prova disso foi um trabalho realizado em 1970 por Needleman e Wunsch [56], onde desenvolveram um algoritmo para encontrar semelhanças nas sequências de aminoácidos de duas proteínas. A partir desses resultados era possível determinar se existia homologia significativa entre as proteínas e essa informação era utilizada para rastrear o seu possível desenvolvimento evolucionário. Com a evolução do conhecimento biológico essas técnicas tornaram-se limitadas, uma vez que propunham um método que apenas faria sentido utilizar quando as sequências não partilham padrões semelhantes. Atualmente as pessoas confiam principalmente em técnicas computacionais que usam *machine-learning* para o reconhecimento de padrões, em vez de técnicas antigas [20]. Os modelos de redes neuronais (NNs) alcançaram um desempenho de última geração e são cada vez mais usados como suporte a problemas na área da biologia.

Lee e Tuan Nguyen [57] utilizaram um *dataset* com 550960 sequências de proteínas obtidas da base de dados UniProt [58] com mais de 10345 famílias com o objetivo de treinar um classificador para a tarefa de identificação da família das proteínas. Dessas famílias de proteínas, utilizaram apenas as que continham mais de 200 exemplos, o que resultou num total de 589 famílias e 317460 sequências usadas para treino e teste. Para treinar os modelos de classificação de proteínas dividiram os dados de treino, validação e teste numa escala de 70/15/15 e codificaram as sequências das proteínas representando cada sequência como uma série de trigramas (um bloco de aminoácidos) e criaram uma representação de cada trigrama usando o GloVe [59]. Os autores experimentaram várias arquiteturas de redes neuronais e

usaram como método de avaliação o F1-Score, justificando tal escolha pelo facto de se tratar de um problema de classificação multiclasse e concluíram que as famílias de proteínas podem ser previstas com precisão a partir de sequências de aminoácidos mas que a utilização de modelos mais complexos como o LSTM bidirecional [60] não traz grandes melhorias nos resultados, ou seja não são necessários para alta precisão. Dos modelos utilizados obtiveram um melhor desempenho geral com o modelo Gated Recurrent Neural Networks (GRU), atingindo um F1-Score = 0.953141.

Anu Vazhayl, Vinayakumar e Soman [20], reuniram informações de 40433 sequências proteicas revistas e anotadas manualmente. A informação foi obtida da base de dados de famílias de proteínas (Pfam) [29] e continha um total 30 famílias distintas. Dividiram as sequências de proteínas em 12000 sequências para dados de teste e as restantes 28433 foram usadas para treinar os modelos. Avaliaram o desempenho de diferentes modelos como o Recurrent Neural Network (RNN), a LSTM, o GRU, árvores de decisão, Naive Bayes, k-nearest neighbors (KNN) e Support Vector Machine (SVM), conseguindo atingir um máximo de 78.4% de precisão com a LSTM.

Um dos aspetos que difere este trabalho dos demais é a forma como as sequências proteicas são representadas. A seleção da representação das sequências proteicas é uma tarefa desafiadora uma vez que afeta a precisão dos modelos de aprendizagem [20]. Um trabalho de Ehsaned-din e Mohammad [61] foca-se a 100% nesta tarefa, e propõe um novo método de representação e extração de características das sequências de proteínas com o nome de ProtVec. Os autores defendem que essa representação pode ser aplicada a um conjunto diverso de problemas da bioinformática como a visualização de proteínas, classificação de famílias de proteínas, predição de estrutura e a extração de domínios – unidades funcionais e /ou estruturais distintas numa proteína, geralmente responsáveis por uma função ou interação específica. Nessa abordagem cada sequência biológica é incorporada num vetor de n-dimensões que caracteriza propriedades biofísicas e bioquímicas das sequências usando redes neuronais. No caso da classificação da família das proteínas, usaram como classificador o SVM [62] e além das estruturas primárias (sequências) usaram como entrada da rede informações adicionais, como a hidrofobicidade, a equação de Van der Waals, a polaridade, a polarizabilidade, a estrutura secundária e a tensão superficial. Para avaliar a eficiência do ProtVec os autores classificaram 7027 famílias de proteínas e obtiveram uma precisão média de 93.06 % e concluíram que a utilização apenas da sequência primária era suficiente para atingirem uma alta precisão.

4 METODOLOGIA

4.1 Pipeline

Uma *pipeline* de *machine learning* consiste em várias etapas para treinar um modelo. Os *pipelines* são iterativos uma vez que cada etapa é repetida para melhorar continuamente a precisão do modelo até obter um algoritmo de sucesso. Tal como apresentado na Figura 13, este trabalho seguiu o *workflow* típico de uma *pipeline* de *machine learning* que é composto pelas seguintes etapas:

- Recolha de dados - É uma etapa crucial para o resultado final pois, a quantidade e a qualidade dos dados do *dataset* determinam o sucesso do modelo de *machine learning*;
- Preparação dos dados - Verifica se os dados do *dataset* estão bem distribuídos e se são ou não tendenciosos. Nesta etapa deve ser garantido que os dados são formatados da maneira que melhor se adequa ao modelo de aprendizagem escolhido;
- Configuração e treino do modelo - Há vários modelos de *machine learning* disponíveis e a escolha do modelo mais adequado não é uma tarefa trivial. A sua escolha dependerá da natureza do problema, do tipo de dados e do tempo necessário para resolver o problema. A etapa do treino consiste no ajuste do hiperparâmetros, que vai sendo feita até aprimorar as capacidades de previsão do modelo;
- Validação do modelo - Permite testar o modelo com informações que não foram usadas no treino, permitindo concluir se a configuração escolhida para o modelo é ou não eficaz.

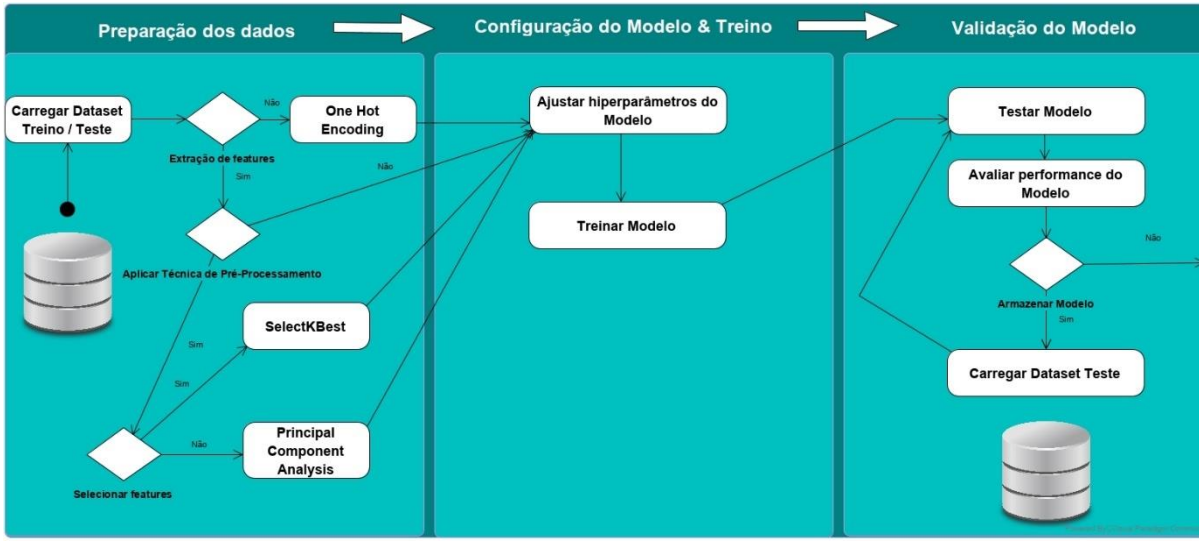

Figura 13 Diagrama de atividade com o *workflow* da *pipeline* utilizada

4.2 Dataset

O UniProt Knowledgebase (UniProtKB) [58] atua como um repositório central de conhecimento sobre proteínas. Contém informação de sequências proteicas e as suas funções, sendo que muitas das entradas provêm de projetos de sequenciação de genomas. Além da

vantagem de ser gratuito, a informação que o UniProtKB armazena, é considerada de alta qualidade e completa.

O *dataset* utilizado neste trabalho (disponível em https://github.com/hddlucas/thesis_data/blob/main/Datasets/all_original.csv) consiste num conjunto de sequências proteicas na sua estrutura primária, obtidas a partir do Swiss-Prot [31], uma seção do UniProtKB que contém sequências anotadas e revistas manualmente que foram obtidas a partir de resultados de experiências, ferramentas computacionais e conclusões científicas. Swiss-Prot fornece um alto nível de anotação, um nível mínimo de redundância e um alto nível de integração com outras bases de dados.

Os dados foram descarregados no formato *tab-separated*, um dos formatos disponíveis no UniProtKB onde as colunas são separadas por tabulações e são configuráveis (à exceção do identificador da entrada no UniProtKB), tal como demonstrado na Figura 14. Tendo em conta os objetivos a atingir, foi necessário selecionar a sequência e a família a que a proteína pertence, resultando num *dataset* com o aspeto exemplificado na Figura 15. É de notar que além da família, algumas das sequências proteicas descarregadas também contêm a informação da superfamília a que pertencem. A associação da superfamília a uma sequência proteica desconhecida também é importante para um biólogo por diversas razões práticas, como a descoberta de medicamentos, previsão da função molecular e diagnostico médico [63].

Figura 14 UnitProt permite a configuração das colunas do *dataset* que o utilizador pretende visualizar / descarregar [65]

A coluna sequência representa a estrutura primária da proteína, composta por um conjunto de aminoácidos representados por um código de uma única letra, tal como apresentado na Tabela 1. No UniProtKB o papel de atribuir uma família a uma sequência é assegurado pelo InterPro (consultar seção 3.1) [64].

As diferentes sequências utilizadas pertencem a um conjunto de 4 organismos diferentes: 20415 proteínas de humanos, 17013 de ratos, 15834 de Arabidopsis thaliana e 6721 de Sac-charomyces cerevisiae. Deste volumoso conjunto de dados, removeram-se as sequências que continham aminoácidos com o caracter 'X', uma vez que representa um aminoácido desconhecido (consultar Tabela 1) e as sequências que não continham a informação da família a que pertenciam, uma vez que algumas sequências apenas continham a informação da superfamília a que pertenciam. Depois de removidas as sequências não desejadas, o *dataset* a ser utilizado ficou com 14.147 proteínas de humanos, 12166 de ratos, 12115 de Arabidopsis thaliana e 4132 de Sac-charomyces cerevisiae, associadas a um total de 5017 famílias distintas.

4.3 Representação vetorial da sequência

Uma questão importante nos modelos de aprendizagem supervisionada, é como codificar as entradas. Boas representações de entrada são cruciais para o sucesso da aprendizagem do modelo, uma vez que tornam mais fácil o reconhecimento de padrões [66].

4.3.1 Representação baseada em propriedades estruturais e físico-químicas

Esta seção descreve uma representação vetorial (exemplificada no Anexo E – Extração de *features* da sequência proteica – exemplo prático), em que cada sequência proteica é representada por um vetor específico construído a partir de representações codificadas das seguintes propriedades:

- Propriedades físico-químicas;
  - Índice hidropático;
  - Aromaticidade;
  - Ponto isoelétrico;
  - Peso molecular
- Propriedades estruturais
  - Tamanho da sequência;
  - Frequência absoluta de cada aminoácido

```
Entry   Protein families       Sequence
Q8YW84  Complex I NdhM subunit family
MENATLLKSTTRHIRIFAAEIDRDGELVPSNQVLTLDIDPDNEFNWNEDALQKIYRKFDELVEASSGADLTDYNLRRIGSDLEHYLRSLLQKGEISYNLSARVTNYSLGLPQVAVEDK
B8HUM7  NDK family
MERTFLAVKPDGVQRALVGEIIRRFEAKGFKLVGLKLMNVSKDLAEQHYGEHKEKPFFPGLVQFITSGPVVAMVWEGKGVVASARKIIGATNPLNSEPGTIRGDYGVDIGRNIIHGSDAVETAQR
FQPAELVSWEPTLTSWIYE
```

Figura 15 Exemplo de sequência proteica no formato tab-separated obtida do UniProtKb/Swiss-Prot

4.3.1.1 Índice hidropático

O índice hidropático ou, do termo inglês Grand average of hydropathicity (GRAVY), dos aminoácidos reflete a tendência que as suas cadeias laterais têm para interagir com o meio aquoso. Um índice de hidropaticidade elevado (+) indica uma região hidrofóbica (apolar) do polipeptídeo e um índice baixo (-) indica o contrário, ou seja, região hidrofílica (polar) do polipeptídeo. O aumento da pontuação positiva indica uma maior hidrofobicidade.

$$\text{GRAVY} = \sum_{i=1}^{20} f_i * \alpha_i,$$

onde f_i é a frequência relativa do aminoácido do tipo i na proteína e α_i o índice hidropático desse aminoácido (Tabela 4) [67].

Tabela 4 Índice hidropático dos aminoácidos [19]

Aminoácido	***Índice hidropático***
Alanine	1.8
Arginine	-4.5
Asparagine	-3.5
Aspartic acid	-3.5
Cysteine	2.5
Glutamic acid	-3.5
Glutamine	-3.5
Glycine	-0.4
Histidine	-3.2
Isoleucine	4.5
Leucine	3.8
Lysine	-3.9
Methionine	1.9
Phenylalanine	2.8
Proline	-1.6
Serine	-0.8
Threonine	-0.7
Tryptophan	-0.9
Tyrosine	-1.3
Valine	4.2

4.3.1.2 Aromaticidade

A aromaticidade é a frequência relativa dos aminoácidos aromáticos: Phenylalanine, Tryptophan e Tyrosine [68]:

$$\text{AROMACIDADE} = \sum_{i=1}^{20} f_i * \delta_i,$$

onde f_i é a frequência relativa do aminoácido do tipo i na proteína e δ_i = 1 quando o aminoácido é aromático e δ_i = 0 caso contrário.

4.3.1.3 Ponto Isoelétrico

Os aminoácidos são moléculas que podem apresentar carga total positiva, negativa ou neutra, influenciada pela sua composição química e pelo pH da solução onde o aminoácido está inserido. O ponto isoelétrico (pI) é o valor de pH para o qual a carga total do aminoácido é nula, ou seja, quando sujeito a esse valor de pH, o total das cargas positivas iguala o total de cargas negativas.

O pI foi obtido utilizando o método de Bjellqvist [67], uma abordagem que simula um gel de focalização isoelétrica. Primeiro o algoritmo calcula a carga de uma proteína com o pH = 7 e de seguida o próximo pH proposto move 3.5 (metade de 7) na direção apropriada, dependendo da carga. Essas iterações continuam em *loop* até que a carga esteja próxima de zero. O espaço de pesquisa é dividido pela metade em cada iteração, resultando numa pesquisa eficiente pelo ponto isoelétrico.

O primeiro elemento do programa consiste numa função de chamada que executa duas tarefas: conta o número de cópias dos aminoácidos que desempenham um papel na determinação do pI e propõe valores de pH nos quais a carga deve ser avaliada. Os aminoácidos que desempenham um papel na determinação do pI são os que nas várias áreas da proteína podem assumir uma carga positiva (*Lysine*, *Arginine* e *Histidine*), ou uma carga negativa (*Aspartic* e *Glutamic acids*, *Cysteine* e *Tyrosine*). O segundo elemento do programa é uma função que calcula a carga da proteína quando sujeita a um determinado pH. A carga da proteína é equivalente à soma das frações dos grupos com carga positiva e negativa na proteína:

$$Z = Nterm + Cterm + \alpha * K + \beta * R + \gamma * H + \delta * D + \epsilon * E + \zeta * C + \eta * Y,$$

onde $Nterm + Cterm, K, R, H, D, E, C\ e\ Y$ são as cargas que esses grupos assumem num determinado pH e as letras gregas antes deles são a contagem de cada resíduo de aminoácido na sequência proteica. A soma desses termos é a carga sobre toda a proteína.

A estrutura e a composição de uma molécula determinam a capacidade de assumir uma carga. Devido aos milhares de cópias de um determinado grupo químico ionizável, é possível

que alguns não assumam uma carga quando sujeitos a um determinado pH. A proporção de cópias moleculares que recebem uma carga em resposta a alterações de pH é dada pelos valores de pK (Tabela 5) para esse grupo químico.

Tabela 5 Valores de pK [69]

Aminoácido	***pK***
Lysine	10.0
Arginine	12.0
Histidine	5.98
Aspartic acids	4.05
Glutamic acids	4.45
Cysteine	9.0
Tyrosine	10.0
N-terminal	7.5
C-terminal	3.55

A percentagem de um grupo de moléculas que recebe uma carga é calculada por duas funções. Uma determina a percentagem de iões positivos para o N-terminal e a outra determina a percentagem de iões negativos para o C-terminal. Cada função gera uma taxa de concentração (CR).

Para grupos positivos:

$$CR = 10^{pK-pH}$$

Para grupos negativos a ordem inverte:

$$CR = 10^{pH-pK}$$

Depois de obtido o valor de CR, a carga parcial é obtida usando a seguinte fórmula:

$$CR = \frac{CR}{CR+1}$$

4.3.1.4 Peso Molecular

O peso molecular da proteína é frequentemente estimado a partir da mobilidade eletroforética na presença de dodecil sulfato de sódio, ou do volume de eluição em cromatografia de filtração em gel [68].

$$\text{PESO MOLECULAR} = \sum_{i=1}^{20} f_i * \alpha i - (len-1) * k,$$

onde f_i é a frequência absoluta do aminoácido do tipo i na proteína, αi o peso molecular (daltons) desse aminoácido (Tabela 6), len o número de carateres da sequência e k uma constante que representa a massa média de uma molécula de água com o valor 18.0153.

Tabela 6 Peso molecular dos aminoácidos [19]

Aminoácido	***Peso Molecular (daltons)***
Alanine	89.094
Arginine	174.203
Asparagine	132.119
Aspartic acid	133.104
Cysteine	121.154
Glutamic acid	147.131
Glutamine	146.146
Glycine	75.067
Histidine	155.156
Isoleucine	131.175
Leucine	131.175
Lysine	146.189
Methionine	149.208
Phenylalanine	165.192
Proline	115.132
Serine	105.093
Threonine	119.119
Tryptophan	204.228
Tyrosine	181.191
Valine	117.148

4.3.1.5 Tamanho de Sequência

Corresponde ao número total de caracteres da sequência.

4.3.1.6 Frequência absoluta de cada aminoácido

$$aa_i = a_i ,$$

onde a_i é a frequência absoluta dos 20 diferentes tipos de aminoácidos i (Tabela 1) na proteína.

4.3.2 Representação baseada na sequência em bruto

A sequência em bruto foi utilizada com o objetivo de perceber se existe uma relação entre a disposição das sequências de aminoácidos e as famílias das proteínas.

As sequências de aminoácidos são representadas com um código de uma letra correspondente (Tabela 1), tal como exemplificado:

MVMGLGVLLLVFVLGLGLTPPTLAQDNSRYTHFLTQHYDAKPQGRDDRYCESIM
RRRGLTSPCKDINTFIHGNKRSIKAICENKNGNPHRENLRISKSSFQVTTCKLHGGSPW
PPCQYRATAGFRNVVVACENGLPVHLDQSIFRRP

Para a construção de modelos de aprendizagem é necessário transformar esses dados textuais na forma numérica de modo a que o modelo de *machine learning* os possa processar. Para realizar essa tarefa foram executados os seguintes passos:

1. Criou-se um dicionário de 20 aminoácidos convertidos em valores inteiros:

{'A': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6, 'H': 7, 'I': 8, 'K': 9, 'L': 10, 'M': 11, 'N': 12, 'P': 13, 'Q': 14, 'R': 15, 'S': 16, 'T': 17, 'V': 18, 'W': 19, 'Y': 20}

2. Para cada sequência de aminoácidos, substitui-se o código de 1 letra por um valor inteiro, usando o dicionário criado no passo 1. Se o código não estiver presente no dicionário, o valor é substituído por 0, considerando apenas os 20 aminoácidos comuns:

[11, 18, 11, 6, 10, 6, 18, 10, 10, 10, 18, 5, 18, 10, 6, 10, 6, 10, 17, 13, 13, 17, 10, 1, 14, 3, 12, 16, 15, 20, 17, 7, 5, 10, 17, 14, 7, 20, 3, 1, 9, 13, 14, 6, 15, 3, 3, 15, 20, 2, 4, 16, 8, 11, 15, 15, 15, 6, 10, 17, 16, 13, 2, 9, 3, 8, 12, 17, 5, 8, 7, 6, 12, 9, 15, 16, 8, 9, 1, 8, 2, 4, 12, 9, 12, 6, 12, 13, 7, 15, 4, 12, 10, 15, 8, 16, 9, 16, 16, 5, 14, 18, 17, 17, 2, 9, 10, 7, 6, 6, 16, 13, 19, 13, 13, 2, 14, 20, 15, 1, 17, 1, 6, 5, 15, 12, 18, 18, 18, 1, 2, 4, 12, 6, 10, 13, 18, 7, 10, 3, 14, 16, 8, 5, 15, 15, 13]

3. Uma vez que as sequências apresentam tamanhos diferentes foi necessário definir um comprimento máximo para a sequência de modo a que o vetor tivesse uma dimensão $M * N$, onde M representa o total de sequências e N o comprimento máximo das sequências. Se o comprimento máximo definido for maior do que o tamanho da sequência então a restante sequência é preenchida com zeros ou, em caso contrário, a sequência é truncada até ao comprimento definido. A biblioteca Keras [70] dispõe do método pad_sequences [71] que pode ser utilizado para realizar essa tarefa, tal como demonstrado na Figura 16. A abordagem utilizada para definir o comprimento máximo da sequência foi usar a média do comprimento de todas as sequências.

4. Por fim cada sequência é convertida num vetor codificado usando a técnica One Hot Encoding descrita na seção 4.4.3.

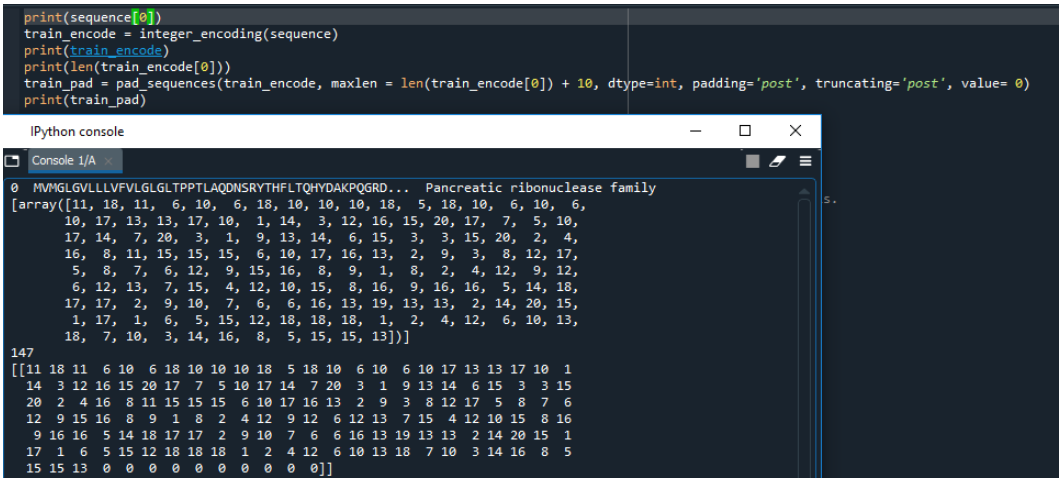

Figura 16 Exemplo da aplicação do tf.keras.preprocessing.sequence.pad_sequences

4.4 Técnicas de Pré-Processamento

4.4.1 Redução de Dimensionalidade

Para avaliar o impacto da redução de dimensionalidade, foi utilizado o método de Análise de Componentes Principais (ACP). O ACP, ou do termo inglês Principal Component Analysis (PCA) é provavelmente o método da estatística multivariada mais utilizado para transformar o espaço multidimensional num subespaço menor [72]. A ideia é que a redução da dimensão de um conjunto de dados composto por um número considerável de variáveis, perca o mínimo de informação possível, através de técnicas que exploram correlações entre as variáveis originais e transformam-nas num menor conjunto de variáveis, designadas por componentes principais. Cada componente principal é uma combinação linear de todas as variáveis originais que são linearmente independentes entre si e estimadas com o propósito de reter o máximo de informação, em termos da variância total contida nos dados [73]. A ACP, através da procura de propriedades que diferem fortemente entre as classes, permite remover redundâncias e descrever cada sequência de proteína utilizando menos propriedades.

4.4.2 Seleção de *features*

A seleção de *features* é um processo no qual se selecionam as *features* que mais contribuem para a variável de saída [74]. Este processo deve ser uma das primeiras etapas de uma *pipeline* de *machine learning* porque utilizar *features* irrelevantes pode diminuir a precisão dos modelos uma vez que as *features* que são usadas para treinar um modelo de *machine learning* têm uma influência enorme no desempenho que pode ser alcançado. Além de remover redundância dos dados, a seleção de *features* reduz a dimensão do conjunto de dados o que permite melhorar o desempenho em conjuntos de dados de dimensões muito altas [75]. Apesar de ambos os métodos reduzirem o número de atributos no conjunto de dados, a seleção de *features* é diferente da redução de dimensionalidade porque o método de redução de dimensionalidade

cria novas combinações, enquanto que os métodos de seleção incluem e excluem atributos presentes nos dados sem alterá-los.

4.4.3 One Hot Encoding

O One Hot Encoding é uma representação de variáveis categóricas em vetores binários que tem como pré-requisito o mapeamento dos valores categóricos em valores inteiros. Em seguida, cada valor inteiro é representado como um vetor binário com todos os valores zero, exceto o índice do número inteiro, marcado com 1, tal como exemplificado na Figura 17.

Muitos dos algoritmos de *machine learning* não podem usar dados categóricos diretamente, portanto, as categorias devem ser convertidas em números de modo a que possam ser usados como variáveis de entrada e saída. Poderia ser utilizada a codificação inteira diretamente redimensionando-a onde necessário, mas isso apenas é eficaz para problemas em que existe um relacionamento ordinal natural entre as categorias, no entanto o consumo de recursos aumenta devido à dimensionalidade.

Figura 17 Exemplo ilustrativo da técnica One Hot Encoding

4.5 Modelos de Aprendizagem

O objetivo de um modelo de aprendizagem supervisionada é construir um classificador com base num conjunto de dados de treino que pertencem a uma determinada classe ou *target*, usando um conjunto de propriedades comuns. As famílias de proteínas anotadas obtidas a partir do UniProtKB/Swiss-Prot correspondem às classes a serem previstas.

4.5.1 Árvores de Decisão

As árvores de decisão são um tipo de aprendizagem supervisionada, conhecidas pela sua boa performance na criação de modelos com bom desempenho para classificação e regressão a partir de um conjunto variado de *features* [76]. O seu objetivo principal é criar um modelo que prevê o valor de uma classe através da aprendizagem de regras de decisão simples, explicadas facilmente pela lógica booleana, tal como exemplificado na Figura 18. As árvores decisão destacam-se pela sua simplicidade de interpretação e compreensão, uma vez que podem ser

visualizadas e ao contrário de outras técnicas de aprendizagem, não necessitam de uma grande normalização dos dados, sendo capazes de lidar com dados numéricos e categóricos. São também capazes de lidar com a classificação de problemas multiclasse, permitindo assim a classificação de proteínas de diferentes famílias. Apesar das vantagens enumeradas, as árvores de decisão apresentam algumas desvantagens, uma vez que quando a sua profundidade aumenta, a sua complexidade aumenta, afetando as regras de decisão. As árvores de decisão podem tornar-se instáveis porque pequenas variações nos dados podem resultar na geração de uma árvore completamente diferente. Para conjuntos de dados onde as *features* são substituídas aleatoriamente, as árvores decisão não são uma boa escolha, uma vez que em grande parte, os algoritmos de aprendizagem das árvores de decisão são baseados em algoritmos heurísticos, como o caso do algoritmo guloso, que inicialmente coloca todos os exemplos num único nó chamado de raiz e de seguida procura sobre um conjunto de atributos, aqueles que melhor dividem o conjunto de exemplos em subconjuntos, repetindo esse processo de forma recursiva, até que todos os exemplos já estejam classificados ou até que todas os atributos preditivos já tenham sido utilizados.

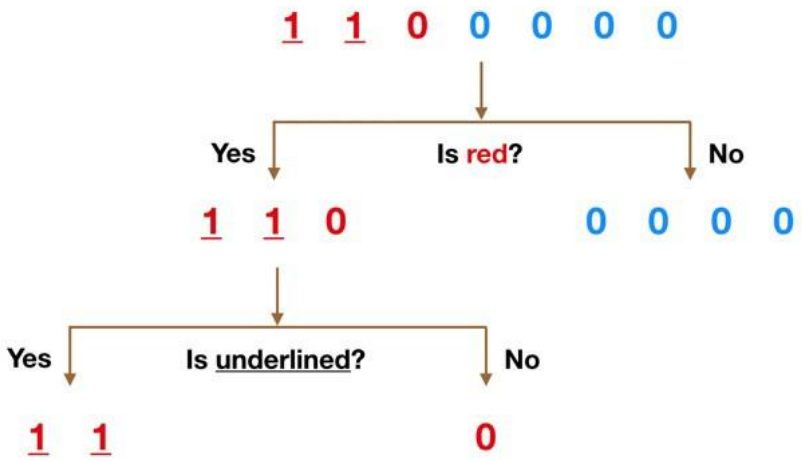

Figura 18 Exemplo de árvore de decisão simples [77]

4.5.2 Classificador Random Forest

O classificador Random Forest consiste num grande número de árvores de decisão individuais que funcionam como um conjunto. Cada árvore de decisão apresenta uma previsão de classe individual e a classe com mais votos é usada como a previsão do modelo (Figura 19), permitindo uma melhoria na precisão.

Enquanto que as árvores de decisão geram regras e nodos, normalmente utilizando o cálculo de ganho de informação e o gini *index*, o Random Forest faz isso de modo aleatório. Outra diferença é o facto de as árvores de decisão profundas poderem sofrer de sobreajuste (*overfitting*), algo que as Random Forest evitam na maioria dos casos, uma vez que trabalham com subconjuntos aleatórios das *features* e constroem árvores menores a partir desses

subconjuntos e no final combinam todas. Dependendo do número de árvores geradas, esta abordagem poderá tornar a computação mais lenta.

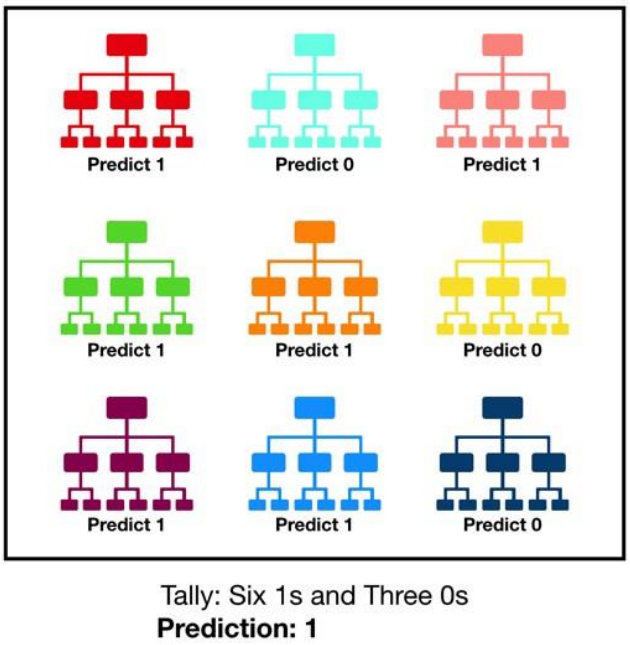

Figura 19 Exemplo do classificador Random Forest [77]

4.5.3 Rede Neuronal Multilayer Perceptron

As redes neuronais são algoritmos computacionais inspirados no paradigma de realizar tarefas cognitivas à semelhança do sistema biológico que é o cérebro humano. Este paradigma baseia-se num largo número de interconexões entre os diversos elementos (neurónios), que trabalham como um todo para a resolução de um dado problema.

As redes MLP têm vindo a ser usadas com sucesso em diversas áreas, incluindo na bioinformática, por exemplo para a tarefa de prever a estrutura secundária de uma proteína [78]. Uma Rede Neuronal Artificial (RNA) do tipo MLP é constituída por um conjunto de nós, formados por uma camada de entrada da rede, uma ou mais camadas escondidas onde os dados recebidos da camada de entrada são processados e uma camada de saída onde os valores de saída são obtidos de acordo com as informações recebidas da camada intermédia. À exceção da camada de entrada, todas as outras camadas contêm neurónios, portanto, apresentam capacidade computacional. É possível resolver problemas de complexidade crescente através de RNA, aumentando o número de camadas, no entanto esse aumento embora se refira a um mecanismo de decisão mais profundo, exige um maior poder de processamento.

O MLP é um modelo RNA *feedforward* que é treinado com *stochastic gradient descent* (sgd), *stochastic gradient-based* (adam) ou por um optimizador com base nos métodos quasi-Newton (lbfgs) usando *backpropagation*. A saída de cada neurónio é definida como [78]:

$$f(\alpha) = f\left(\sum_{i=1} w_i \; x_i + \; b\right)$$

onde w_i e x_i representam os valores de entrada do neurónio e w_i os respetivos pesos. A função f representa a função de ativação não linear usada em toda a rede e b o *treshold* de ativação do neurónio.

4.5.4 Rede Neuronal Long Short-Term Memory

Uma rede neuronal recorrente (RNN) tem a capacidade de aprender dependências a longo prazo, uma vez que recebe sinais tanto da camada de entrada como da camada oculta na iteração de tempo anterior. Essa é uma das características que a distingue das redes neuronais *feedforward*, que apenas recebem sinais da camada de entrada (os dados brutos) e processam esses dados para a camada de saída.

A camada oculta das RNN funciona como uma "memória" no sentido em que a cada período no tempo é capaz de guardar no seu estado oculto a informação dos dados analisados naquele período de tempo, bem como recuperar informações do estado oculto anterior, que num cenário ideal, contem toda a informação relevante que aconteceu no passado. Nesse mesmo período, o estado oculto também é capaz de fornecer informações para a camada de saída, no momento de realizar uma predição, passando essa informação para o estado oculto do período de tempo seguinte.

A rede neuronal LSTM é um tipo de rede neuronal RNN que pode aprender dependências de longo prazo entre as etapas do tempo dos dados da sequência, ou seja, é capaz de aprender quando deve manter ou descartar informações. Em bioinformática têm sido usadas para resolver problemas como a previsão da estrutura secundária de uma proteína [79], predição do número de resíduos de contacto [80] e também para classificação das famílias de proteínas [20]. A LSTM usa células de memória que são uma camada oculta e podem armazenar informações por um longo período de tempo. Tal como apresentado na Figura 20, a célula de memória consiste em três tipos de portais que controlam um fluxo de informações para dentro dessas células:

- Portal de entrada - Controla o quanto a nova memória influencia a memória anterior, permitindo adicionar informação à memória da célula;
- Portal de esquecimento - Remove as informações que não são mais úteis no estado da célula;
- Portal de saída - Controla a quantidade de memória nova que deve ser enviada para a próxima unidade LSTM

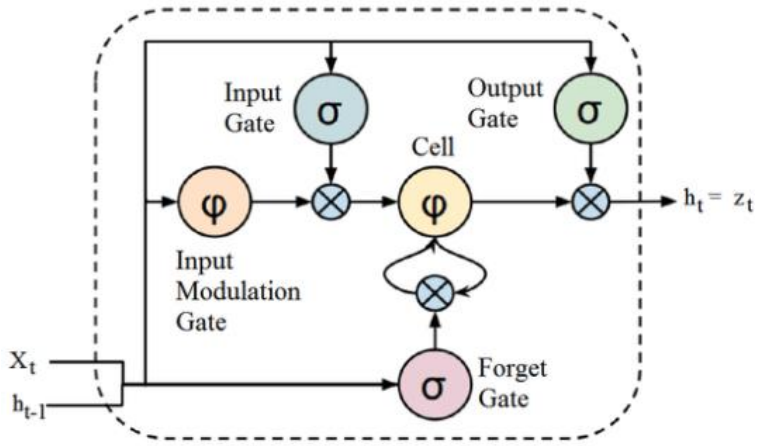

Figura 20 Portais numa célula de memória de uma Long Short-Term Memory [81]

4.6 Métricas de Avaliação dos Modelos

As métricas de avaliação de um modelo de *machine learning* permitem medir a performance do modelo com base num conjunto de dados de teste. Um modelo pode fornecer resultados satisfatórios quando avaliado com uma determinada métrica mas pode apresentar resultados menos bons quando avaliado em a relação outra métrica. A escolha das métricas influencia a forma como o desempenho do algoritmo de *machine learning* é medido e comparado. Esta seção descreve algumas das principais métricas usadas em problemas de classificação.

4.6.1 Matriz de Confusão

A matriz de confusão é usada com o objetivo de avaliar a qualidade da saída de um modelo de classificação aplicado a um conjunto de dados. Tal como pode ser observado na Figura 21, a matriz confusão corresponde a uma tabela de $(N * N)$ entradas onde as colunas representam as observações de uma classe prevista e cada linha representa as observações de uma classe real, ou vice-versa. A partir da matriz de confusão de um classificador binário (ex.: classificar se uma classe é positiva ou negativa) é possível obter as seguintes frequências:

- Verdadeiros Positivos (TP) - Ocorrem quando classificador obteve o valor positivo e a classe é positiva;
- Verdadeiros Negativos (TN) - Ocorrem quando o classificador obteve o valor negativo e a classe é negativa;
- Falsos Positivos (FP) - Ocorrem quando o classificador obteve o valor positivo e a classe é negativa;
- Falsos Negativos (FN) - Ocorrem quando o classificador obteve o valor negativo e a classe é negativa

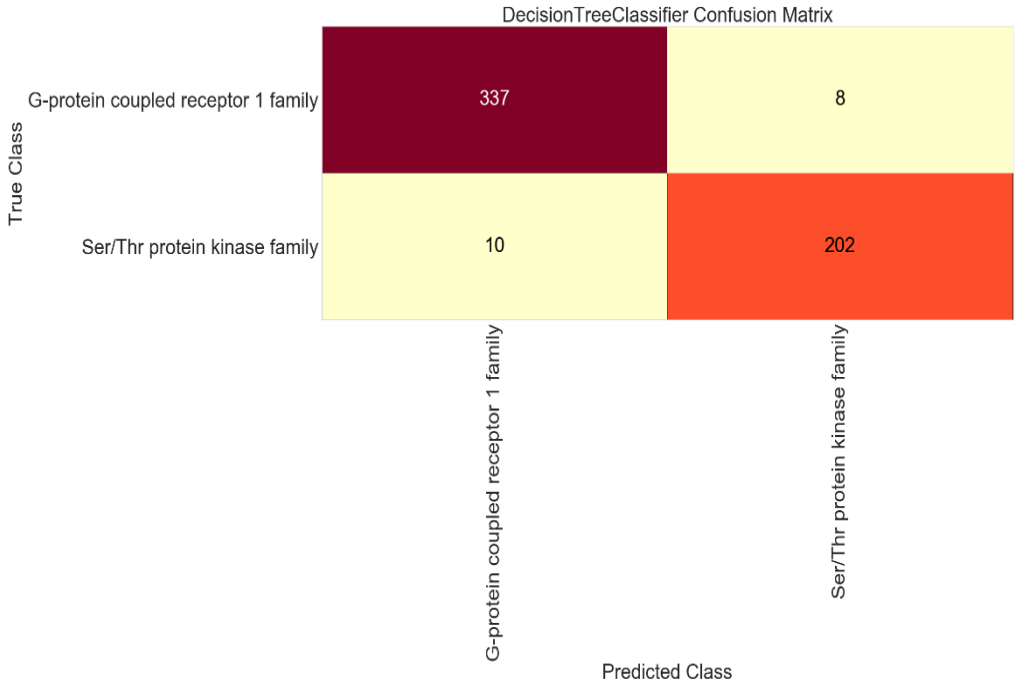

Figura 21 Exemplo de matriz de confusão gerada a partir do SmartGeno

No caso do problema de classificação de famílias em que existem múltiplas classes a serem previstas, as métricas são calculadas da seguinte forma:

- Verdadeiros Positivos (TP) - Correspondem aos elementos na diagonal que representam o número de pontos para os quais a classe prevista é igual à classe verdadeira (pintados a vermelho na Tabela 7). Os Verdadeiro Negativos (TN) correspondem á soma de todos os valores da matriz de confusão, excluindo a linha e a coluna dessa classe;
- Falsos Positivos (FP) e Falsos Negativos (FP) - São os elementos fora da diagonal e são aqueles que são classificados incorretamente pelo classificador. FP ficam na última linha da tabela enquanto que os FN ficam na última coluna à direita e tal como apresentado na Tabela 7 podemos ver que a soma destes conjuntos é igual.

Tabela 7 Exemplo de matriz de confusão de um problema multiclasse

		PREVISTO			FN
		Ser/Thr protein kinase	Sugar transporter	Helicase	
CLASSE REAL	Ser/Thr protein kinase	15	0	4	4
	Sugar transporter	0	11	0	0
	Helicase	1	1	8	2
FP		1	1	4	6

4.6.2 Accuracy

A *accuracy*, também conhecida por taxa de acerto, é uma fração entre o total de classes que foram classificadas corretamente e o total de amostras de um problema. Pode ser utilizada para perceber se um modelo está a ser treinado corretamente, no entanto, não deve ser usada como métrica principal uma vez que não fornece informações detalhadas sobre a sua aplicação ao problema no caso de existir um desequilíbrio entre as classes. A *accuracy* é obtida a partir da seguinte fórmula:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

4.6.3 Precisão

A precisão corresponde ao balanço entre aqueles que foram classificados como corretos, quando efetivamente eram e traduz-se na seguinte fórmula:

$$Precisão = \frac{TP}{TP + FP}$$

4.6.4 Taxa de verdadeiros positivos

A taxa de verdadeiros positivos, também conhecida por *recall*, corresponde à frequência que o classificador encontra exemplos de uma classe, ou seja, quando realmente é da classe X, o quão frequente o modelo o classifica como X? O *recall* é obtido a partir da seguinte fórmula:

$$Taxa\ de\ verdadeiros\ positivos = \frac{TP}{TP + FN}$$

4.6.5 Taxa de falsos positivos

A taxa de falsos positivos corresponde à frequência que o classificador encontra um "falso alarme", isto é quantas vezes uma classe negativa é classificada como positiva. A taxa de falsos positivos é obtida a partir da seguinte fórmula:

$$Taxa\ de\ falsos\ positivos = \frac{FP}{FP + TN}$$

4.6.6 F1-Score

O F1-Score é uma média ponderada da precisão e do *recall*. Pode variar entre 0 a 1 (quanto maior melhor o modelo) e permite obter uma qualidade geral do modelo. O F1-Score é calculado a partir da seguinte fórmula:

$$F1Score = 2 * \frac{precisão * recall}{precisão + recall}$$

4.6.7 ROC

Uma das opções mais óbvias e utilizadas para medir a performance de um classificador é a taxa de acerto. Uma taxa de acerto de 100% é o objetivo ideal, no entanto ela pode ser vista como uma grande incerteza. Considere-se por exemplo um *dataset* em que 90% dos exemplos pertençam a uma classe. Só de classificar corretamente todos os exemplos naquela classe já se atinge uma taxa de acerto de 90%, mesmo que todos os exemplos da outra classe sejam classificados incorretamente. Uma alternativa eficaz consiste na comparação do desempenho geral de um classificador a partir do gráfico de ROC (Receiver Operating Characteristic).

O gráfico ROC é uma representação gráfica bidimensional que permite ver o quão bem um classificador separa os exemplos positivos dos negativos e qual o melhor limite para separá-los. Tal como pode ser observado na Figura 22, o gráfico é produzido através do cálculo da taxa de verdadeiros positivos (representada no eixo YY) em relação à taxa de falsos positivos (representada no eixo do XX) para os vários limites possíveis.

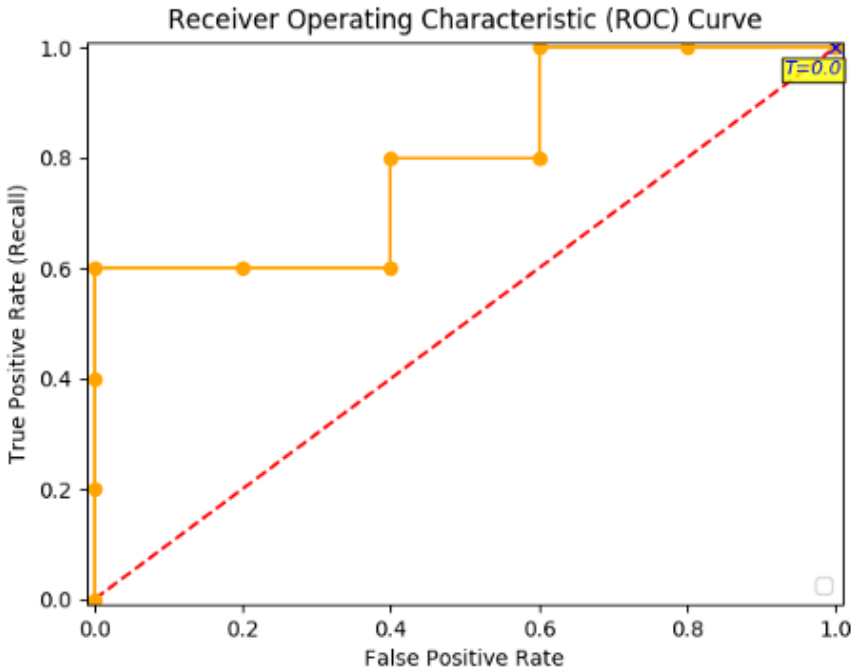

Figura 22 Exemplo de um gráfico ROC [82]

4.6.8 AUC

Para comparar classificadores é necessário reduzir a curva ROC a um escalar. Um método comum para esta redução consiste em calcular a AUC (Area Under Curve). A AUC é composta pela área abaixo da curva de ROC e uma vez que faz parte da área do quadrado unitário (espaço de ROC), o seu resultado varia entre 0.0 e 1.0. Quanto maior a AUC melhor será o modelo a prever "positivos como positivos e negativos como negativos". A sua principal vantagem em relação ao F1-Score é que ela mede o desempenho em vários pontos de corte. Observando a Figura 23, podemos verificar que a AUC corresponde à combinação da área dos retângulos azul, verde e roxo, portanto, $AUC = 0.4 * 0.6 + 0.2 * 0.8 + 0.4 * 1.0 = 0.80$.

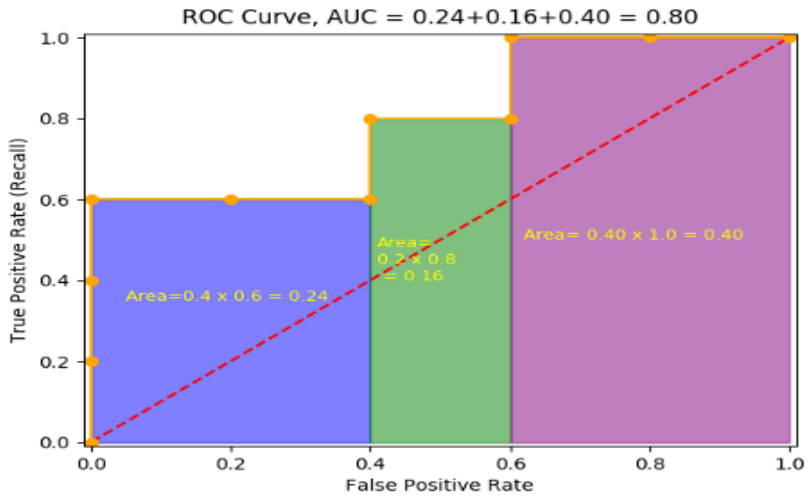

Figura 23 Cálculo da área abaixo da curva de ROC [82]

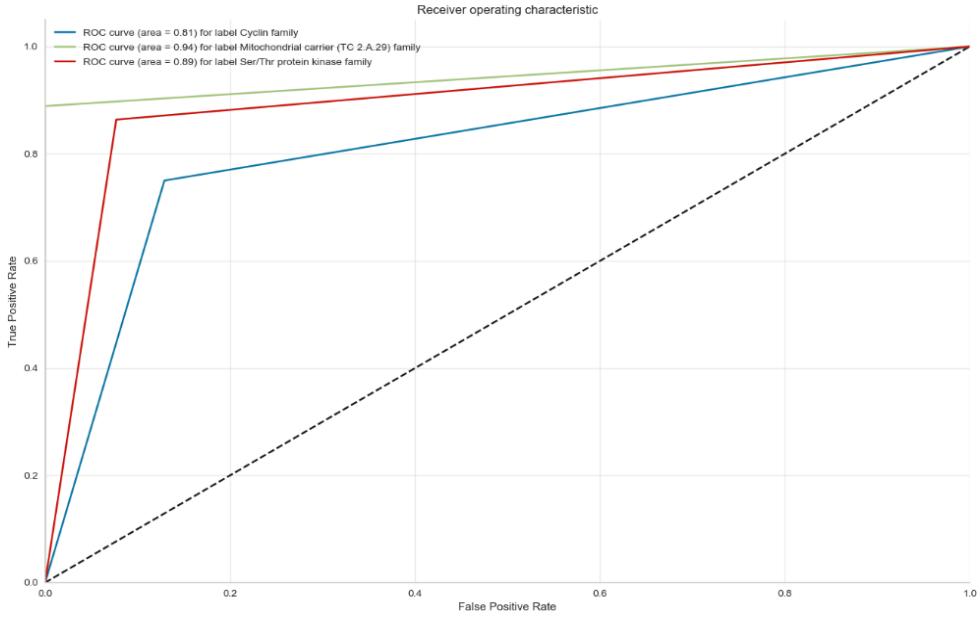

Figura 24 Exemplo de gráfico de ROC obtido a partir do protótipo SmartGeno

Quando se trata de um problema em que existem múltiplas classes, o gráfico de ROC contém um número N de curvas de ROC para cada uma das N classes a serem previstas, tal como exemplificado na Figura 24.

4.6.9 Micro-Average vs Macro Average

Num problema de classificação multiclasse as métricas precisão, *recall*, F1-Score e *support* podem ser calculadas usando uma *macro-average* ou uma *micro-average*, ambas disponíveis como configuração do parâmetro "*average"* do método precision_recall_fscore_support[83] do scikit-learn. O mesmo se aplica ao cálculo da métrica AUC, através da configuração dos parâmetros "average" e "multi_class" do método roc_auc_score [84]. Como elas calculam coisas ligeiramente diferentes, a sua escolha deve ser baseada no *dataset* do problema em questão.

A *macro-average* calcula a métrica para cada uma das classes e em seguida obtém a média, portanto, trata todas as classes de forma igual. A *micro-average* agrega as contribuições de todas as classes para calcular a métrica média, fazendo com que todas as métricas se tornem iguais (precisão $=$ *recall* $=$ F1-Score $=$ *support*). A *micro-average* é recomendável para problemas onde há um desequilíbrio entre as classes.

Para ilustrar a diferença, a matriz confusão apresentada na Figura 21 será usada como exemplo para o cálculo da métrica precisão usando a *macro-average* e a *micro-average*.

- Classe "Ser/Thr protein kinase family": 15 TP e 1 FP;
- Classe "Sugar transporter (TC 2.A.1.1) family": 11 TP e 1 FP;
- Classe "Helicase family": 8 TP e 4 FP

Cálculo da precisão usando a *macro-average*:

$$\text{Precisão da classe "Ser/Thr protein kinase family"} = \frac{\left(\frac{15}{16}\right)+\left(\frac{11}{12}\right)+\left(\frac{8}{12}\right)}{3} \cong 0.8403$$

Cálculo da precisão usando a *micro-average*:

$$\text{Precisão da classe "Ser/Thr protein kinase family"} = \frac{15+11+8}{16+12+12} = 0.85$$

5 PLATAFORMA COMPUTACIONAL

5.1 Visão da Solução

O SmartGeno consiste num protótipo que utiliza métodos computacionais com o objetivo de dar suporte aos cientistas na tarefa de anotação de sequências proteicas, uma vez que funciona como uma alternativa às experiências de laboratório, que levam um tempo considerável para anotação de sequências proteicas. O SmartGeno permite que o utilizador realize os seus próprios estudos ao escolher e configurar diferentes modelos de *machine learning* que podem ser armazenados para usar posteriormente com conjuntos de teste. Os resultados obtidos podem ser partilhados por *email* e consultados a qualquer instante, uma vez que a plataforma mantém histórico dos dados dos utilizadores.

5.1.1 Principais Funcionalidades

Esta seção apresenta uma visão de alto nível das principais funcionalidades do SmartGeno, que incluem:

- Autenticação e registo de utilizadores;
- Treino de modelos de *machine learning*;
- Teste de modelos de *machine learning* previamente armazenados;
- Visualização e partilha de resultados

5.1.2 Dependências

Existem funcionalidades que dependem umas das outras para poderem ser executadas com sucesso. Um exemplo disso é a funcionalidade "Testar modelo de *machine learning* previamente armazenado" que, tal como o nome indica, tem como pré-requisito a existência de pelo menos um modelo de *machine learning* na base de dados, armazenado pelo próprio utilizador ou por outro utilizador que o tenha armazenado como público.

5.2 Ferramentas e Tecnologias

Esta seção descreve as ferramentas e as tecnologias que foram utilizadas para o desenvolvimento do SmartGeno. As ferramentas e as tecnologias selecionadas foram escolhidas com base em critérios como a performance e a qualidade que apresentam.

5.2.1 Ferramentas

5.2.1.1 Visual Paradigm Community

O Visual Paradigm [85] é uma ferramenta de gestão de projetos de *software*, que fornece todas as ferramentas necessárias para arquitetura, gestão de projetos, desenvolvimento de *software* e colaboração em equipa. O Visual Paradigm foi projetado para uma ampla gama de utilizadores, incluindo arquitetos, gestores de projeto, programadores de *software*, analistas de sistemas, analistas de negócios, de sistemas e qualquer outro que precise de criar sistemas de *software* em grande escala através da metodologia Scrum e de uma abordagem orientada a objetos. O Visual Paradigm suporta os padrões Unified Modeling Language (UML) e Business Process Diagram (BPMN) mais recentes e também é um complemento perfeito para processos baseados na metodologia Scrum. Projetos diferentes podem exigir combinações específicas de ferramentas ágeis devido a várias situações como, por exemplo, a natureza do problema. Um ambiente de desenvolvimento ágil também deve incluir um conjunto rico de ferramentas que os programadores possam adotar de maneira flexível nas suas próprias escolhas. As principais ferramentas ágeis no Paradigma Visual são [86]:

- Modelação UX e Wireframe;
- Modelação visual;
- *Report designer*;
- Modelação de base de dados;
- Engenharia de código e integração em Integrated Development Environment (IDE);
- Gestão de mudanças e trabalho em equipa

Visual Paradigm foi utilizado neste projeto essencialmente durante a fase de desenho do sistema com o objetivo de produzir os diagramas de casos de uso, modelação da base de dados e para a especificação da arquitetura.

5.2.1.2 DBeaver Community

O DBeaver Community [87] é uma ferramenta *open-source* grátis para gestão de base de dados. Uma das principais vantagens da sua utilização é o facto de permitir a sua utilização com diferentes Sistemas de Gestão de Base de Dados (SGBDs), como Orable, Postgres, MySQL, MariaDB, Microsoft SQL Server, entre outros. Algumas das suas principais funcionalidades incluem [88]:

- Diagramas Entidade-Relacionamento (ER);
    - Capaz de gerar diagramas ER automaticamente para uma base de dados / esquema para uma ou mais tabelas;
    - Exportação dos diagramas para os formatos: GIF, PNG, BMP e GraphML.
- Transferência de dados;

  - Exportação de dados para um ficheiro ou para outra tabela da base de dados, com a opção de criar a tabela de destino se ela não existir;
  - Suporta formatos Comma Separated Values (CSV), HyperText Markup Language (HTML), Extensible Markup Language (XML), JavaScript Object Notation (JSON), etc.

- Manipulação de *queries* Structured Query Language (SQL) visualmente;
  - É possível construir *queries* SQL simples ou complexas sem necessidade de ter um conhecimento real em SQL, a partir de junções / filtros realizados visualmente, através de uma interface com o utilizador amigável.

- Editor de SQL
  - Organizar scripts em pastas;
  - Destaque da sintaxe SQL determinado pela base de dados associada a um *script*;
  - Suporte para diferentes conjuntos de palavras-chave reservadas e funções do sistema para diferentes bases de dados;
  - Importação e exportação de *scripts*.

Neste projeto o DBeaver Community foi utilizado para gerir a base de dados Postgres utilizada para manter os dados da aplicação SmartGeno.

5.2.1.3 Scikit-learn

O scikit-learn [4] é um projeto Python de *machine learning* que dispõe de um conjunto de algoritmos para *data-mining* e para tarefas de análise de dados como classificação, regressão, *clustering*, redução de dimensionalidade e seleção de modelos. Os algoritmos são construídos usando algumas bibliotecas científicas como o NumPy [6], SciPy [7] e Matplotlib [89]:

- NumPy;
  - Fornece o tipo de dados *ndarray*, utilizado para representar dados com n-dimensões e considerados como eficientes para computação numérica baseada em *array*.

- SciPy;
  - Possui um conjunto de funções matemáticas de alto nível que operam em *ndarrays*. Os principais domínios incluem álgebra linear, otimização e processamento de sinais. O SciPy está ligado a bibliotecas como o BLAS [90] e o MKL [91], que garantem um alto desempenho. Juntos, o NumPy e o SciPy fornecem um ambiente científico robusto para computação numérica e são a base dos algoritmos do scikit-learn.

- Matplotlib
  - É uma biblioteca de plotagem 2D que permite gerar gráficos, histogramas, espectros de potência, gráficos de barras, gráficos de erros, dispersão, etc. Neste projeto,

conjugando as bibliotecas Matplotlib e Yellowbrick [92], foi possível plotar gráficos como o relatório de classificação dos modelos de aprendizagem e a matriz de confusão.

No scikit-learn, todos os objetos e algoritmos aceitam dados de entrada na forma de matrizes bidimensionais com o formato "amostras × características". Os objetos scikit-learn partilham entre si um conjunto uniforme de métodos utilizados para diferentes propósitos:

- Modelo;
  - É uma interface que expõe o método *fit*, utilizado para ajustar modelos a partir dos dados de treino. Todos os algoritmos supervisionados e não supervisionados estão disponíveis como objetos que implementam essa interface. Além dessa tarefa o estimador é responsável por tarefas de *machine learning* como a seleção de características ou a redução de dimensionalidade.
- Preditor;
  - É um estimador com um método *predict*, que recebe como entrada um *array* de entrada X_test e retorna os valores previstos para cada amostra, calculados a partir do modelo estimado. A partir dos valores retornados por este método, é possível obter avaliar a performance dos modelos de aprendizagem.
- Transformador
  - O transformador é um estimador que implementa um método *transform*, que pode ser usado com algoritmos de pré-processamento para redução de dimensionalidade (consultar seção 4.4.1) e seleção de recursos (consultar seção 4.4.2).

Neste projeto o scikit-learn foi utilizado para reduzir a dimensão ou selecionar *features* do conjunto de entrada, para transformar características de valores não numéricos (nome da família da proteína) em valores numéricos, para treinar e testar *datasets* com dados de sequências proteicas aplicando diferentes modelos de *machine learning* e para avaliar a performance desses modelos.

5.2.1.4 Microsoft Visual Studio

O Microsoft Visual Studio [93] é um ambiente de desenvolvimento integrado (IDE) da Microsoft utilizado para o desenvolvimento de *software*, especialmente dedicado ao .NET Framework e ás linguagens de programação Visual Basic (VB), C, C++, C# e F#. À semelhança de outros IDEs o visual Studio tem como principal finalidade a edição de código-fonte numa linguagem de programação suportada pelo IDE e a compilação do mesmo em linguagem máquina.

Neste projeto utilizou-se o Microsoft Visual Studio para facilitar o processo de *deployment* da aplicação num ambiente de produção, que pode ser realizado no próprio IDE, uma vez que o Visual Studio e o serviço aplicacional do Azure fornecem um mecanismo poderoso para criar,

publicar e manter aplicações *web* no Azure. Publicar no serviço de aplicações do Azure significa copiar os ficheiros necessários para o servidor e configurar um ficheiro "web.config" que contém as configurações necessárias para instruir o servidor da *web* como iniciar a aplicação. Outro motivo para a escolha do Microsoft Visual Studio como IDE foi o facto do mesmo fornecer um suporte *open-source* para a linguagem Python na versão 2.7 ou na versão 3.5 e superiores. Embora seja possível usar o Visual Studio para editar código noutras versões, algumas das funcionalidades ou recursos poderão não funcionar. O SmartGeno foi desenvolvido utilizando a versão 3.6.4 para 64 bits. Por fim e não menos importante, escolheu-se o Microsoft Visual Studio também pelo facto de o mesmo oferecer ao utilizador um local único para a gestão de todos os ambientes globais do Python (virtuais ou conda). Para cada ambiente, o utilizador pode gerir *packages*, abrir uma janela interativa para esse ambiente e aceder às pastas desse ambiente.

5.2.1.5 GitHub

GitHub [94] é uma plataforma de alojamento de código-fonte com controlo de versões que usa o Git. Permite que programadores, utilizadores ou qualquer utilizador registado na plataforma contribua em projetos privados e/ou *open-source* de qualquer lugar do mundo. GitHub é muito utilizado por programadores para divulgação dos seus trabalhos ou para que outros programadores contribuam com o projeto, além de promover fácil comunicação através de recursos que relatam problemas (*issues)* ou mesclam repositórios remotos (*pull request*). O GitHub tem mais de 3 milhões de utilizadores ativos em todo o mundo e é usado por empresas como a Google, Microsoft e WordPress.

As suas principais funcionalidades são:

- *Code review*;
- Gestão de projeto;
- Integração de código;
- Gestão de equipas;
- Gestão de *issues*;
- Documentação;
- Alojamento do código-fonte

O GitHub foi utilizado neste projeto para alojar o código-fonte da aplicação desenvolvida e a documentação do projeto.

5.2.2 Tecnologias

5.2.2.1 Django Framework

Qualquer pessoa com a experiência mínima em desenvolvimento de *websites* conhece a dor de reinventar certas *features* vezes sem conta. É necessário criar *schemas* de bases de dados, é

necessário colocar e extrair dados da base de dados, fazer *parse* de URLs, ter em conta a segurança e a usabilidade, etc. As *frameworks* são a chave para a resolução desse problema, uma vez que facilitam a reutilização e a geração de código, o que permite um aumento na velocidade e produtividade no desenvolvimento de aplicações *web*. "*São compostas por um conjunto de interfaces, classes e padrões dedicados a resolver um grupo de problemas através de uma arquitetura de programação flexível e extensível* [95]*"*.

O Django fornece uma *framework* de alto nível que permite que os utilizadores construam aplicações *web* com poucas linhas de código, usando a linguagem Python (consultar exemplo prático no Anexo D – Django Framework – exemplo prático). É simples, robusto e flexível, permitindo desenhar soluções sem demasiado *overhead*. A *framework* Django utiliza o padrão de *design* de *software* Model View Template (MVT). O MVT é um padrão de arquitetura de *software* que separa a apresentação dos dados da lógica de interação com o utilizador, tal como acontece como o padrão Model View Controller (MVC), no entanto o padrão utilizado pelo Django difere-se pelo facto de ser a própria *framework* a gerir a parte do "*Controller*". O MVT é composto por 3 componentes: o componente "Model" - usado para representar os dados armazenados na base de dados, o componente "View" - usado para manipular o fluxo de apresentação na visualização, lidando essencialmente com a lógica de negócio e interação com os modelos para transportar dados e o componente "Template" - a camada de apresentação que lida com a parte da interface do utilizador.

A estrutura de um projeto Django [96] tem um aspeto semelhante ao apresentado na Figura 25. Desta estrutura destacam-se:

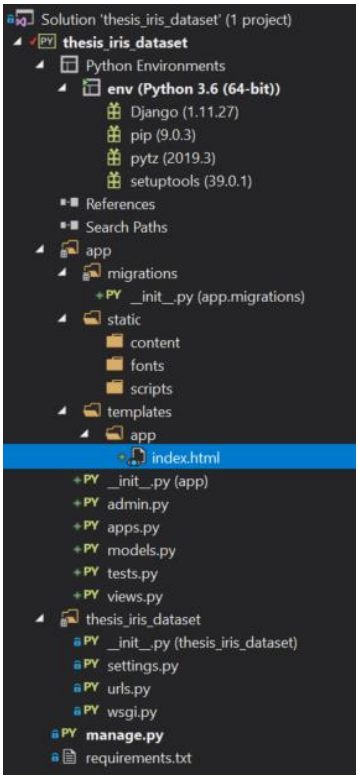

Figura 25 Estrutura de um projeto Django

- A raiz do projeto que é composta por:
  - [nomedoprojeto]/settings.py;

    Contém configurações que são partilhadas entre configurações de desenvolvimento e produção.
  - [nomedoprojeto]/urls.py;

    Contém as configurações dos URLs e permitem realizar um mapeamento entre expressões de caminho de URL e funções Python.
  - [nomedoprojeto]/wsgi.py.

    Contém as configurações de *deployment* que usam o padrão *web* Server Gateway Interface (WSGI).
- A diretoria [nomedoprojeto]/[nomedaaplicação]/static/;

  Contém todos os ativos estáticos (Cascading Style Sheets (CSS), ficheiros *javascript*, imagens, etc.);
- A diretoria [nomedoprojeto]/[nomedaaplicação]/templates/;

Contém as páginas HTML que são renderizadas e apresentadas ao utilizador. Em HTML não podemos incluir código Python porque os *browsers* apenas entendem HTML, no entanto, usando os *templates* Django é possível transformar *tags* de *template* [97] (sintaxe própria do Django) em HTML, permitindo a construção de sites dinâmicos.

- A diretoria [nomedoprojeto]/[nomedaaplicação]/migrations/;

  Contém os *scripts* necessários para atualizar a base de dados que se encontra alinhada com as alterações efetuadas nos modelos, permitindo assim um versionamento da mesma.

- O ficheiro [nomedoprojeto]/manage.py;

  É um *script* que ajuda a gestão da aplicação *web*. Pode ser usado para iniciar o servidor *web*, criar e executar migrações, criar um superutilizador, entre outras coisas.

- O ficheiro [nomedoprojeto]/[nomedaaplicação]/models.py;

  É o local onde os modelos Object-Relational Mapping (ORM) são definidos.

- O ficheiro [nomedoprojeto]/[nomedaaplicação]/tests.py;

  É um ficheiro Python que contém a estrutura básica dos testes unitários.

- O ficheiro [nomedoprojeto]/[nomedaaplicação]/views.py;

  É o local onde são definidas as vistas Python cuja responsabilidade é tipicamente renderizar o HTML para enviar ao *browser*, no entanto com exceções, uma vez que uma vista não necessita necessariamente de ser visível (ex.: a validação de um formulário). Por norma recebem um pedido Hypertext Transfer Protocol (HTTP) e devolvem uma resposta.

- O ficheiro requeriments.txt

  Permite gerir todas as dependências dos ambientes de desenvolvimento Python. O ficheiro contém uma lista dos pacotes com as respetivas versões a serem instaladas.

5.2.2.2 PostgreSQL

PostgreSQL [98] é sistema de gestão de base de dados objeto-relacional (SGBD) que começou a ser desenvolvido em 1977 na Universidade Berkeley na Califórnia, num projeto chamado Ingres. Mais tarde passou de um projeto académico para um projeto comercial desenvolvido pela Relational Technologies/Ingres Corporation. Em 1986 outra equipa liderada por Michael Stonebraker continuou o desenvolvimento do código da Ingres para criar uma base de dados objeto-relacional, chamada de Postgres, mais tarde renomeada para PostgreSQL. PostgreSQL é de longe considerada a base de dados *open-source* mais avançada do mundo e é considerada uma excelente escolha para aqueles que procuram proteger a integridade dos dados das suas aplicações e criar ambientes mais tolerante a falhas. Além da vantagem de ser gratuito, o PostgreSQL é altamente extensível uma vez que permite que o utilizador defina os seus

próprios tipos de dados, crie funções personalizadas ou até escreva código de diferentes linguagens de programação sem a necessidade de recompilar a base de dados [99].

As principais funcionalidades do PostgreSQL são as seguintes:

- Tipos de dados;
    - Suporta dados primitivos (inteiro, numérico, *string*, booleano), estruturados (data /hora, matrizes, intervalos, UUID), documentais (JSON, XML, Key-Value), geométricos (ponto, linha, círculo, polígono) ou customizados.
- Integridade dos dados;
    - Chaves Primárias;
    - Chaves Estrangeiras;
    - Dados não nulos (NOT NULL) ou únicos (UNIQUE).
- Concorrência, Performance;
    - Indexação;
    - Indexação avançada;
    - Transações;
    - Controlo de simultaneidade de várias versões (MVCC);
    - Paralelização de consultas de leitura e construção de índices B-tree
    - Particionamento de tabela.
- Segurança;
    - Sistema de controlo de acesso robusto;
    - Segurança a nível de coluna e linha.
- Extensibilidade
    - Funções e Stored Procedures;
    - Linguagem Procedimental (PL/PGSQL, Perl, Python);
    - Muitas extensões que fornecem funcionalidades adicionais, incluindo o PostGIS – permite o uso de objetos para sistemas de informação geográfica.

O PostgreSQL foi utilizado neste projeto principalmente pelo facto de ser um projeto *open-source* gratuito e de elevada qualidade.

5.2.2.3 Microsoft Azure

O Microsoft Azure [100] é uma plataforma profissional em nuvem disponibilizada pela empresa Microsoft. Utilizando as suas funcionalidades avançadas, é possível lançar máquinas virtuais, bases de dados SQL, criar cópias de segurança dos recursos e muito mais, sem a preocupação de realização de manutenção a falhas, equipamentos e *software* obsoleto. Uma das características mais importantes é o facto de suportar todos os sistemas operativos e uma vasta gama de linguagens de programação. O Azure oferece acesso aos serviços integrados que

abrangem a computação, armazenamento, aplicações, dados e redes, o que facilita a gestão da infraestrutura, realização de mais tarefas e acima de tudo, permite alcançar maiores poupanças financeiras. A partir de 1 de agosto de 2014, o Microsoft Azure passou a estar também disponível no licenciamento *open-source* o que aumentou a sua disponibilidade para pequenas e médias empresas. Alguns dos produtos e serviços mais populares do Microsoft Azure incluem [100]:

- Máquinas virtuais;
  - A *cloud* Microsoft Azure permite aprovisionar máquinas virtuais do Windows e do Linux de forma rápida.
- Serviços de aplicações;
  - Permite criar rapidamente poderosas aplicações *web* e móveis disponíveis na *cloud*.
- Base de dados SQL do Azure;
  - É uma base de dados SQL relacional gerida como serviço.
- Armazenamento;
  - Permite um armazenamento na *cloud* durável, de elevada disponibilidade e extremamente dimensionável.
- Funções;
  - Permite processar eventos com código sem servidor.
- Azure Cosmos DB;
  - É uma base de dados com múltiplos modelos distribuída globalmente para qualquer dimensão.
- Azure Active Directory;
  - Permite a sincronização de diretorias locais e ativar o início de sessão única.
- *Backup*
  - Permite um *backup* do servidor de forma simples e fiável para a *cloud*.

O Microsoft Azure foi utilizado neste projeto para alojar a aplicação SmartGeno, disponível em https://smartgeno.azurewebsites.net/.

5.2.2.4 Python

O Python é uma linguagem de programação confiável, flexível, fácil de aprender, gratuita para uso em qualquer sistema operativo e é suportado por uma forte comunidade de programadores e por muitas bibliotecas gratuitas. Utilizando o Python é possível desenvolver aplicações *web*, aplicações *desktop*, *scripts* e ferramentas de computação científica, usadas por muitas universidades, cientistas e programadores casuais ou profissionais.

O Python foi utilizado neste projeto como linguagem de programação uma vez que permite a combinação da biblioteca scikit-learn escrita em linguagem Python, com uma aplicação *web* também escrita em Python, utilizando a *framework* Django.

5.3 Ciclo de vida do desenvolvimento

5.3.1 Análise de Requisitos

Os requisitos funcionais e não funcionais do protótipo SmartGeno podem ser consultados no Anexo C – Análise de Requisitos.

5.3.2 Desenho

5.3.2.1 Diagrama de Casos de Uso

A Figura 26 contém o diagrama de casos de uso do SmartGeno.

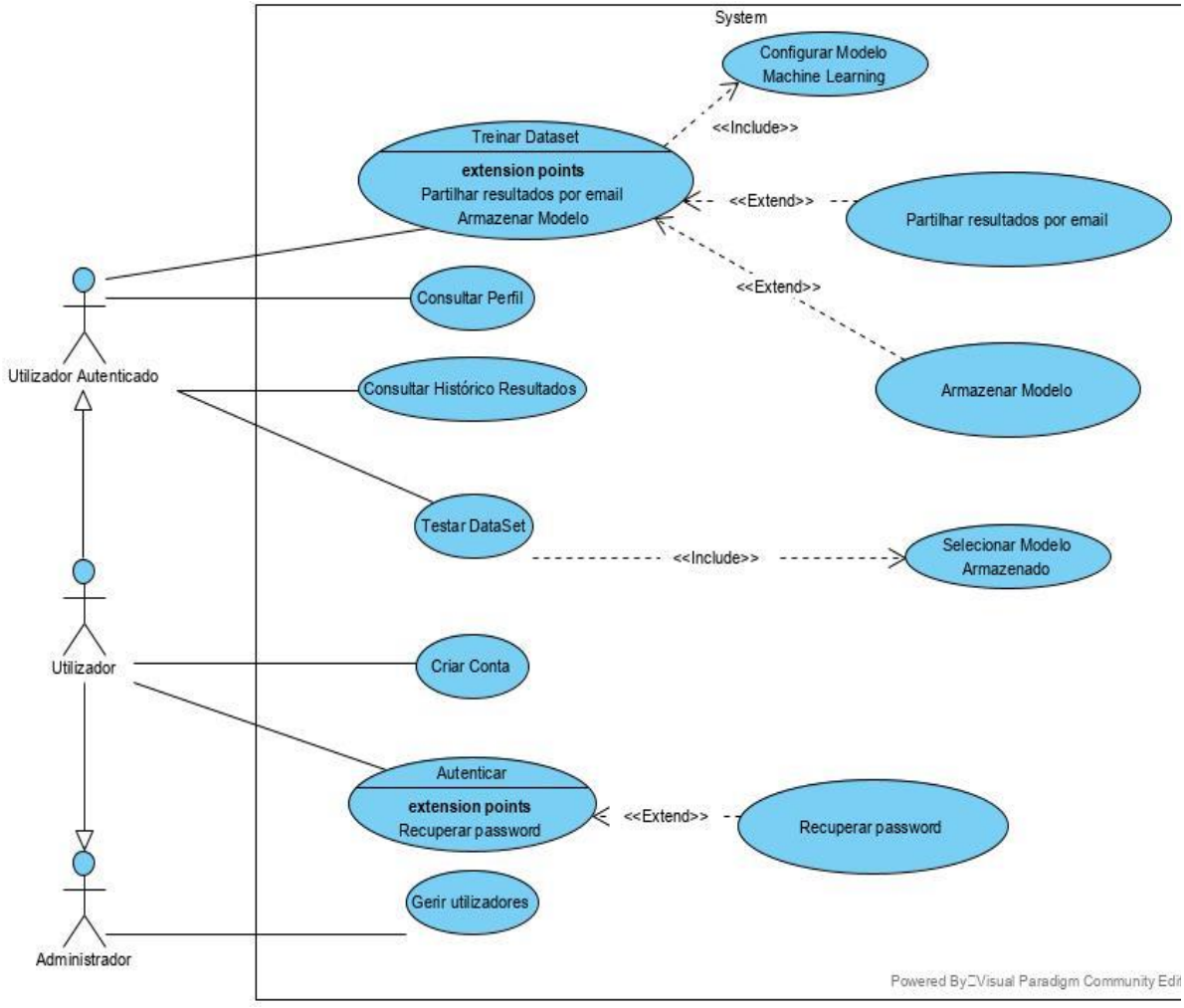

Figura 26 Diagrama de Casos de Uso do SmartGeno

5.3.2.2 Modelo de Dados

O modelo entidade relacionamento (ER) que contém as tabelas utilizadas (consultar Anexo B – Modelo de Dados) é apresentado na Figura 27.

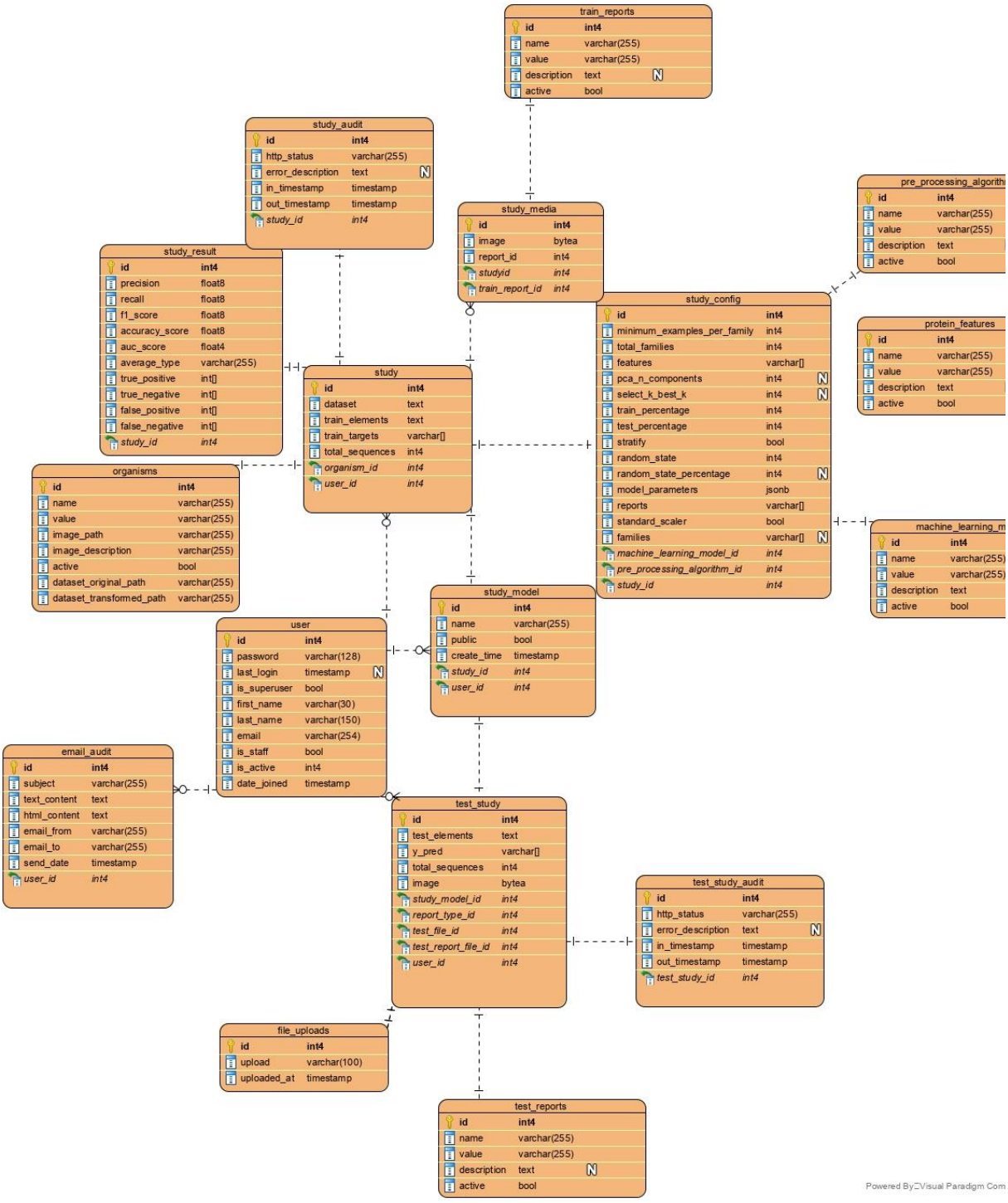

Figura 27 Modelo ER do SmartGeno

5.3.3 Manual de Utilizador

O manual de utilizador pode ser consultado no Anexo A – Manual de Utilizador.

6 RESULTADOS

Os modelos de aprendizagem descritos na seção 4.5 foram treinados com base no conjunto de dados descrito na seção 4.2. As fases de treino e teste envolveram diferentes etapas como a codificação das sequências de proteínas, seleção dos subconjuntos das características e implementação do algoritmo de classificação. Para treinar os modelos de classificação de proteínas dividiu-se aleatoriamente os dados de treino e teste numa escala de 70/30 utilizando o método train_test_split [101]. O conjunto de teste serviu para avaliar o modelo usando algumas das métricas descritas na seção 4.6 - o F1-Score e a AUC.

As Tabelas 8,9 e 10 descrevem *os* hiperparâmetros dos modelos de aprendizagem do scikit-learn utilizados neste trabalho.

Tabela 8 Hiperparâmetros da rede neuronal MLP, disponível na classe MLPClassifier [102] do scikit-learn

Modelo	***Hiperparâmetro***	***Configuração por defeito***	***Descrição***
Rede Neuronal Multilayer Perceptron	max_iter	200	Define o número máximo de iterações.
	solver	'adam'	Especifica o algoritmo para otimização de peso entre os nós. Os algoritmos suportados são: 'lbfgs' – um optimizador da família de métodos Quasi-Newton [103]; 'sgd' – refere-se à descida do gradiente estocástico; 'adam' – um optimizador estocástico baseado em gradiente que foi proposto por Kingma, Diederik e Jimmy Ba [104].
	hidden_layer_sizes	(100,),	Permite definir o número de camadas e o número de neurónios em cada camada da rede neuronal. Cada elemento da tupla representa o número de neurónios em cada camada e o comprimento da tupla indica o total de camadas ocultas na rede.
	activation	'relu'	Define a função de ativação para as camadas ocultas. As funções suportadas são: 'logistic' – função sigmoide que retorna: $f(x) = 1 / (1 + \exp(-x))$; 'tanh' – função tangente hiperbólica que retorna $f(x) = \tanh(x)$; 'relu' – função linear retificada que retorna $f(x) = max(0, x)$; 'identity' – faz com que o modelo seja capaz de aprender apenas combinações lineares, retornando $f(x) = x$

Tabela 9 Hiperparâmetros da árvore de decisão, disponível na classe DecisionTreeClassifier [105] do *scikit-learn*

Modelo	***Hiperparâmetro***	***Configuração por defeito***	***Descrição***
Árvore de Decisão	criterion	'gini'	Define a função usada para medir a qualidade da divisão de um nó. As funções suportadas são 'gini' para a impureza de Gini e 'entropy' para o ganho de informações.
	max_depth	None	Define a profundidade máxima da árvore. O valor por defeito é 'None', que significa ilimitado.
	min_samples_split	2	Define o número mínimo de amostras necessárias para dividir um nó interno.
	min_samples_leaf	1	Define o número mínimo de amostras que um nó da folha deve ter.

Tabela 10 Hiperparâmetros da Random Forest, disponível na classe RandomForestClassifier [106] do scikit-learn

Modelo	***Hiperparâmetro***	***Configuração por defeito***	***Descrição***
Random Forest	n_estimators	10	Define o número de árvores construídas pelo algoritmo antes de fazer uma média das predições.
	criterion	'gini'	Define a função usada para medir a qualidade da divisão de um nó. As funções suportadas são 'gini' para a impureza de Gini e 'entropy' para o ganho de informações.
	max_depth	None	Define a profundidade máxima da árvore. O valor por defeito é None, que significa ilimitado.
	min_samples_split	2	Define o número mínimo de amostras necessárias para dividir um nó interno.
	min_samples_leaf	1	Define o número mínimo de amostras que um nó da folha deve ter.
	n_jobs	None	Define o número de tarefas a serem executadas em paralelo. Se ele tiver o valor 'None' (valor por defeito) utiliza apenas um processador enquanto que quando é definido com o valor -1 significa que não há limite na quantidade de processadores a utilizar.

Para efeitos de avaliação dos resultados experimentais do *dataset* descrito em 4.2 removeram-se 475 entradas duplicadas e selecionaram-se apenas as famílias que contêm mais de **100** exemplos, resultando num total de **7919** sequências e **34** famílias. A Tabela 11 contém o detalhe das famílias usadas e demonstra que existe um desequilíbrio entre as mesmas, portanto, a micro-average (consultar seção 4.6.6) foi a escolha adequada para calcular as métricas apresentadas nos resultados da seção 6.1.

Tabela 11 Descrição do *dataset*

Nome da família	***Instâncias de treino***	***Instâncias de teste***
AAA ATPase	97	37
AGC Ser/Thr protein kinase	90	52
AP2/ERF transcription factor	101	46
ATP-dependent AMP-binding enzyme	82	23
BZIP	102	64
CAMK Ser/Thr protein kinase	139	60
Cation transport ATPase (P-type) (TC 3.A.3)	90	43
CMGC Ser/Thr protein kinase	131	52
Cyclin	87	44
Cytochrome P450	185	82
DEAD box helicase	147	69
DEFL	207	88
'GDSL' lipolytic enzyme	87	33
G-protein coupled receptor 1	787	339
Helicase	75	34
Intermediate filament	106	39
Kinesin	120	33
Krueppel C2H2-type zinc-finger protein	512	204
MHC class I	64	42
Mitochondrial carrier (TC 2.A.29)	118	57
Peptidase C19	129	44
Peptidase S1	171	79
PP2C	81	34
PPR	289	166
Protein-tyrosine phosphatase	144	60
Rab	147	61
Ser/Thr protein kinase	520	209
Short-chain dehydrogenases/reductases (SDR)	129	46
STE Ser/Thr protein kinase	90	51
Sugar transporter (TC 2.A.1.1)	84	29
TRIM/RBCC	81	29
Tyr protein kinase	132	59
Ubiquitin-conjugating enzyme	113	34
UDP-glycosyltransferase	106	34

6.1 Resultados experimentais

De forma a avaliar o desempenho dos modelos de aprendizagem foram testados diferentes *workflows* da *pipeline* apresentada na Figura 13.

Os resultados apresentados neste capítulo correspondem à média de um total de 10 amostras realizados por cada modelo / configuração e podem ser visualizados com maior detalhe em https://github.com/hddlucas/thesis_data/tree/main/Resultados.

6.1.1 Conjunto de *features* na sua totalidade

Nesta análise usou-se a representação vetorial descrita em 4.3.1 com o conjunto de *features* na sua totalidade sem aplicar qualquer algoritmo de pré-processamento à exceção do StandardScaler [107] – um algoritmo do sckit-learn que permite um reescalonamento das *features* de modo que elas tenham as propriedades de uma distribuição normal padrão. O StandardScaler foi usado durante a *pipeline* que usa o modelo MLPClassifier uma vez que para treinar uma rede neuronal o conjunto de dados deve ser normalizado visto que os dados do *dataset* estão em diferentes escalas (ex.: *kilodaltons*, pH, etc.) e com intervalos de valores distintos. Para normalizar as *features* foi aplicada a seguinte fórmula:

$$Standardization = \frac{x-u}{\sigma}, \text{ onde } u = \frac{1}{N}\sum_{i=1}^{n}(x_i) \text{ e } \sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i-u)^2}$$

Tal como observado na Figura 28 a média (u) dos valores das colunas do *dataset* original corresponde a [7.8954; 61316.6] e desvio padrão (σ) corresponde a [1.52453;12501.3]. Por exemplo para calcular a *standardization* para a posição (0,0) é feito o seguinte cálculo:

$$Standardization\ (0,0) = \frac{(5.79608 - 7.8954)}{1.52453} = -1.37703$$

Ao usar o StandardScaler assegura-se que as diferentes *features* são tidas em conta de igual forma uma vez que os dados são transformados de forma a que a sua distribuição tenha um valor médio 0 e um desvio padrão de 1, tal como demonstrado na Figura 28.

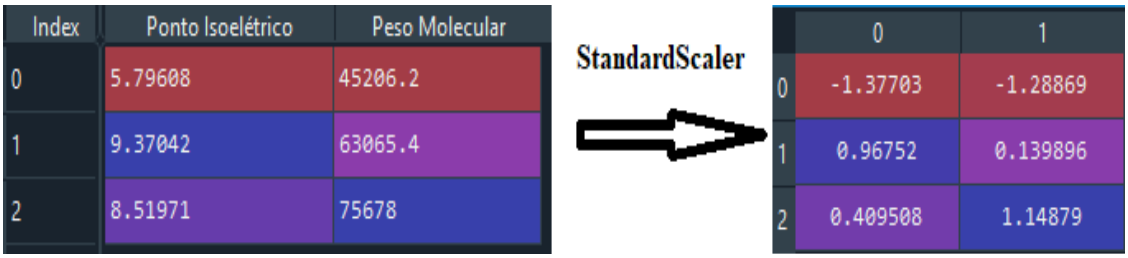

Figura 28 Exemplo ilustrativo da aplicação do StandardScaler

Tabela 12 Resultados dos testes da árvore de decisão com o conjunto de *features* na sua totalidade sem aplicar qualquer algoritmo de pré-processamento à exceção do StandardScaler

Modelo do sckit-learn	***Configuração***	***F1-Score***	***AUC***	***Tempo de execução (segundos)***
DecisionTreeClassifier	*default*	0.70	0.81	0.12
DecisionTreeClassifier	criterion="entropy"	0.71	0.82	0.32
DecisionTreeClassifier	max_depth=8	0.54	0.89	0.07
DecisionTreeClassifier	max_depth=32	0.70	0.82	0.13
DecisionTreeClassifier	min_samples_split = 0.1	0.46	0.90	0.06
DecisionTreeClassifier	min_samples_split = 0.5	0.34	0.81	0.03
DecisionTreeClassifier	min_samples_split =1.0	0.22	0.60	0.01
DecisionTreeClassifier	min_samples_split =5	0.68	0.83	0.12
DecisionTreeClassifier	min_samples_split= 10	0.67	0.85	0.12
DecisionTreeClassifier	min_samples_leaf = 20	0.61	0.91	0.09

Por observação da Tabela 12 concluímos que a os resultados da aplicação de uma árvore de decisão ficaram um pouco aquém do desejado, tendo sido atingido um F1-Score máximo de 71% ainda que, de um modo geral tenha sido o classificador mais rápido. Das funções usadas para medir a qualidade da divisão de um nó - "gini" e "entropy", a que proporcionou melhores resultados foi a "entropy". A função "gini" tem como objetivo medir com que frequência um elemento escolhido aleatoriamente do conjunto seria identificado incorretamente, enquanto que a função "entropy" em vez de utilizar probabilidades usa a base de $\log 2$ das probabilidades:

$$Gini = 1 - \sum_j p_j^2$$

$$Entropy = -\sum_j p_j \log 2\ p_j$$

Os resultados da Tabela 12 sugerem também que um aumento da profundidade máxima da árvore – hiperparâmetro "max_depth", proporcionou melhores resultados ainda que a decisão de expandir os nós até que todas as folhas estejam puras ou até que todas as folhas contenham menos amostras do que o número mínimo de amostras necessárias para dividir um nó interno ("min_samples_split") tenha proporcionado melhores resultados.

Tabela 13 Resultados dos testes da Random Forest com o conjunto de *features* na sua totalidade sem aplicar qualquer algoritmo de pré-processamento à exceção do StandardScaler

Modelo do sckit-learn	***Configuração***	***F1-Score***	***AUC***	***Tempo de execução (segundos)***
RandomForestClassifier	*default*	0.81	0.95	0.20
RandomForestClassifier	n_estimators=100	0.87	0.99	1.84
RandomForestClassifier	n_estimators=200	0.87	0.99	4.00
RandomForestClassifier	n_estimators=300	0.87	0.99	5.92
RandomForestClassifier	n_estimators=400	0.88	0.99	8.11
RandomForestClassifier	n_estimators=500	0.88	0.99	9.70
RandomForestClassifier	n_estimators=1000	0.87	0.99	19.79
RandomForestClassifier	n_estimators=200, n_jobs = -1	0.87	0.99	1.18
RandomForestClassifier	criterion="entropy"	0.80	0.95	0.38
RandomForestClassifier	max_depth=8	0.66	0.96	0.11
RandomForestClassifier	max_depth=32	0.80	0.95	0.18
RandomForestClassifier	min_samples_split = 0.1	0.45	0.93	0.08
RandomForestClassifier	min_samples_split = 0.5	0.31	0.81	0.05
RandomForestClassifier	min_samples_split =1.0	0.14	0.50	0.04
RandomForestClassifier	min_samples_split =5	0.80	0.96	0.19
RandomForestClassifier	min_samples_split= 10	0.78	0.97	0.16
RandomForestClassifier	min_samples_leaf = 20	0.69	0.97	0.13

Por observação da Tabela 13 e comparando com os resultados das Tabelas 12 e 13, verificamos que o Random Forest foi o algoritmo que proporcionou melhores resultados tendo sido atingido um F1-Score de 88%. Analisando os resultados podemos concluir que o ajuste do hiperparâmetro "n_estimador" permitiu um aumento no poder preditivo, mas também tornou a computação mais lenta. A combinação de 400 árvores de decisão foi o valor ótimo uma vez que o incremento desse valor não trouxe melhorias nos resultados. Tal como os resultados sugerem, outro hiperparâmetro bastante importante de ajustar foi o n_jobs uma vez que quando definido com o valor -1, permite que tarefas como a decisão dos caminhos, o ajuste e a previsão possam ser executadas em paralelo nas diferentes árvores da floresta o que proporciona um aumento significativo na velocidade do modelo.

Tabela 14 Resultados dos testes da rede neuronal MLP com o conjunto de *features* na sua totalidade sem aplicar qualquer algoritmo de pré-processamento à exceção do StandardScaler

Modelo do sckit-learn	***Configuração***	***F1-Score***	***AUC***	***Tempo de execução (segundos)***
MLPClassifier	*default*	0.83	0.99	7.51
MLPClassifier	max_iter=500	0.85	0.99	19.55
MLPClassifier	solver='lbfgs'	0.84	0.99	6.08
MLPClassifier	solver='sgd'	0.65	0.96	7.69
MLPClassifier	hidden_layer_sizes= (500,100), max_iter=500	0.85	0.99	19.20
MLPClassifier	activation=identity	0.77	0.98	4.67
MLPClassifier	activation=logistic	0.79	0.99	6.42
MLPClassifier	activation=tanh	0.83	0.99	8.06
MLPClassifier	hidden_layer_sizes= (1000,500), max_iter=500	0.85	0.99	22.42
MLPClassifier	hidden_layer_sizes= (1000,1000), max_iter=500	0.85	0.99	22.10
MLPClassifier	hidden_layer_sizes= (1000,500,500), max_iter=500	0.85	0.99	22.03
MLPClassifier	hidden_layer_sizes= (1000,500), max_iter=2500	0.85	0.99	45.8
MLPClassifier	hidden_layer_sizes= (1000,1000), max_iter=2500	0.85	0.99	45.11
MLPClassifier	hidden_layer_sizes= (1000,500,500), max_iter=2500	0.85	0.99	46.15
MLPClassifier	hidden_layer_sizes= (1000,500), max_iter=5000	0.86	0.99	40.37
MLPClassifier	hidden_layer_sizes= (1000,1000), max_iter=5000	0.85	0.99	41.34
MLPClassifier	hidden_layer_sizes= (1000,500,500), max_iter=5000	0.85	0.99	42.73
MLPClassifier	hidden_layer_sizes= (1000,500,500), max_iter=5000, solver=lbfgs	0.84	0.99	8.08
MLPClassifier	hidden_layer_sizes= (1000,500,500), max_iter=5000, solver=sgd	0.83	0.99	130.14

O sucesso de uma rede neuronal artificial depende não só dos dados que alimentam o modelo, mas também da definição da arquitetura da rede que incluem a definição do tipo de

neurónios (i.e., funções de ativação usadas), do número de camadas, da quantidade de neurónios em cada camada e da forma que se ligam as camadas. Os resultados apresentados na Tabela 14 sugerem que a utilização da função de ativação "relu" com 2 camadas cada uma com 1000 e 500 neurónios respetivamente e com um total de 5000 iterações foi a melhor escolha uma vez que permitiram obter um F1-Score de 86%.

6.1.2 Conjunto de dados dimensionado usando a ACP

Nesta análise usou-se a representação vetorial descrita em 4.3.1 aplicando o método de ACP (consultar seção 4.4.1) que decompõe um conjunto de dados multivariados num conjunto de componentes ortogonais sucessivos que explicam uma quantidade máxima de variância. No sckit-learn, a ACP utiliza o Linear Algebra Package (LAPACK) da decomposição de valor singular proposta por Halko [108], implementado como um objeto transformador que aprende componentes no método *fit* e pode ser usado em novos dados para projetá-los nesses componentes. Antes de aplicar a ACP, foi feito o dimensionamento das *features* por meios de normalização utilizando o StandardScaler [107]. A ACP é um excelente exemplo do quão importante é a normalização, uma vez que estamos interessados nos componentes que maximizam a variação [109]. Se um componente (ex.: peso molecular) varia menos do que outro (ex.: ponto isoelétrico) por causa das suas respetivas escalas (*kilodaltons* vs. pH), a ACP pode determinar que a direção da variância máxima corresponde mais de perto ao eixo *'kilodaltons'*, se essas *features* não forem dimensionadas. Tal como apresentado nas tabelas seguintes, foram testados diferentes números de componentes para avaliar qual o impacto da projeção do conjunto de *features* na sua totalidade nesse número de dimensões.

Tabela 15 Resultados dos testes da arvore decisão com a ACP e o StandardScaler

Modelo do sckit-learn	***Configuração***	***Número de Componentes do ACP***	***F1-Score***	***AUC***	***Tempo de execução (segundos)***
DecisionTreeClassifier	*default*	2	0.40	0.66	0.02
DecisionTreeClassifier	*default*	3	0.51	0.71	0.03
DecisionTreeClassifier	*default*	4	0.54	0.73	0.04
DecisionTreeClassifier	*default*	5	0.58	0.74	0.05
DecisionTreeClassifier	*default*	6	0.60	0.76	0.06
DecisionTreeClassifier	*default*	10	0.63	0.77	0.13
DecisionTreeClassifier	*default*	15	0.65	0.78	0.15
DecisionTreeClassifier	*default*	20	0.64	0.78	0.19

Tabela 16 Resultados dos testes da Random Forest com a ACP e o StandardScaler

Modelo do sckit-learn	***Configuração***	***Número de Componentes do ACP***	***F1-Score***	***AUC***	***Tempo de execução (segundos)***
RandomForestClassifier	*default*	2	0.43	0.79	0.14
RandomForestClassifier	*default*	3	0.54	0.85	0.10
RandomForestClassifier	*default*	4	0.61	0.89	0.19
RandomForestClassifier	*default*	5	0.65	0.90	0.15
RandomForestClassifier	*default*	6	0.69	0.92	0.15
RandomForestClassifier	*default*	10	0.73	0.93	0.21
RandomForestClassifier	*default*	15	0.76	0.94	0.20
RandomForestClassifier	*default*	20	0.76	0.94	0.25

Tabela 17 Resultados dos testes da rede neuronal MLP com a ACP e o StandardScaler

Modelo do sckit-learn	***Configuração***	***Número de Componentes do ACP***	***F1-Score***	***AUC***	***Tempo de execução (segundos)***
MLPClassifier	*default*	2	0.44	0.91	6.42
MLPClassifier	*default*	3	0.51	0.93	6.50
MLPClassifier	*default*	4	0.58	0.95	6.63
MLPClassifier	*default*	5	0.63	0.96	6.72
MLPClassifier	*default*	6	0.67	0.97	6.78
MLPClassifier	*default*	10	0.76	0.98	6.84
MLPClassifier	*default*	15	0.82	0.99	6.90
MLPClassifier	*default*	20	0.84	0.99	6.93
MLPClassifier	hidden_layer_sizes=(1000,500,500), max_iter=5000, solver=sgd	2	0.42	0.90	46.97
MLPClassifier	hidden_layer_sizes=(1000,500,500), max_iter=5000, solver=sgd	3	0.50	0.93	57.97
MLPClassifier	hidden_layer_sizes=(1000,500,500), max_iter=5000, solver=sgd	4	0.58	0.95	82.34

MLPClassifier	hidden_layer_sizes=(1000,500,500), max_iter=5000, solver=sgd	5	0.64	0.96	107.54
MLPClassifier	hidden_layer_sizes=(1000,500,500), max_iter=5000, solver=sgd	6	0.68	0.97	114.46
MLPClassifier	hidden_layer_sizes=(1000,500,500), max_iter=5000, solver=sgd	10	0.75	0.98	135.44
MLPClassifier	hidden_layer_sizes=(1000,500,500), max_iter=5000, solver=sgd	15	0.81	0.99	115.89
MLPClassifier	hidden_layer_sizes=(1000,500,500), max_iter=5000, solver=sgd	20	0.83	0.99	117.4

Os resultados apresentados nas Tabela 15,16 e 17 sugerem que o aumento do número de componentes da ACP proporciona melhores resultados. Se compararmos os resultados da Tabela 14 com os resultados da Tabela 17 para uma projeção do conjunto de *features* em 20 dimensões, verificamos que um classificador MLP configurado com três camadas com respetivamente 1000, 500 e 500 neurónios e um máximo de 5000 iterações proporcionou em ambos um F1Score médio de 83%. A principal diferença reside no tempo de treino uma vez que um demorou 130.14 segundos e o outro 117.4 segundos o que nos permite concluir que a ACP melhora o desempenho dos algoritmos ao livrar-se de variáveis correlacionadas que não contribuem em nenhuma tomada de decisão. O tempo de treino dos algoritmos reduz significativamente com um menor número de recursos, portanto usar a ACP para acelerar os algoritmos é uma escolha razoável.

6.1.3 Seleção de k-*features* do conjunto de dados

Nesta análise usou-se a representação vetorial descrita em 4.3.1 e foi uma seleção de *features* (consultar seção 4.4.2) com base na avaliação da importância de cada uma para o modelo de aprendizagem.

Tabela 18 Resultados dos testes da árvore de decisão com a seleção das melhores *features*

Modelo do sckit-learn	***Configuração***	***Número de features***	***F1-Score***	***AUC***	***Tempo de execução (segundos)***
DecisionTreeClassifier	*default*	2	0.29	0.58	0.03
DecisionTreeClassifier	*default*	3	0.39	0.64	0.03
DecisionTreeClassifier	*default*	4	0.45	0.68	0.07
DecisionTreeClassifier	*default*	5	0.50	0.70	0.05
DecisionTreeClassifier	*default*	6	0.57	0.74	0.04
DecisionTreeClassifier	*default*	10	0.64	0.78	0.07
DecisionTreeClassifier	*default*	15	0.67	0.80	0.08
DecisionTreeClassifier	*default*	20	0.68	0.81	0.11

Tabela 19 Resultados dos testes da Random Forest com a seleção das melhores *features*

Modelo do sckit-learn	***Configuração***	***Número de features***	***F1-Score***	***AUC***	***Tempo de execução (segundos)***
RandomForestClassifier	*default*	2	0.33	0.70	0.14
RandomForestClassifier	*default*	3	0.44	0.79	0.12
RandomForestClassifier	*default*	4	0.51	0.84	0.16
RandomForestClassifier	*default*	5	0.58	0.87	0.15
RandomForestClassifier	*default*	6	0.65	0.90	0.14
RandomForestClassifier	*default*	10	0.74	0.93	0.19
RandomForestClassifier	*default*	15	0.77	0.95	0.15
RandomForestClassifier	*default*	20	0.78	0.95	0.22

Tabela 20 Resultados dos testes da rede neuronal MLP com a seleção das melhores *features*

Modelo do sckit-learn	***Configuração***	***Número de features***	***F1-Score***	***AUC***	***Tempo de execução (segundos)***
MLPClassifier	*default*	2	0.36	0.87	10.97
MLPClassifier	*default*	3	0.45	0.92	10.33
MLPClassifier	*default*	4	0.49	0.93	10.67
MLPClassifier	*default*	5	0.57	0.95	9.27
MLPClassifier	*default*	6	0.62	0.96	9.38

MLPClassifier	*default*	10	0.74	0.97	9.31
MLPClassifier	*default*	15	0.80	0.98	9.35
MLPClassifier	*default*	20	0.81	0.99	9.33

Para obter as *features* que mais contribuem para o modelo de aprendizagem foi utilizado o algoritmo SelectKBest [110] com a função de pontuação f_classif [111], que executa um teste ANOVA (um tipo de teste de hipótese) em cada *feature* e atribui a essa *feature* um valor p. O SelectKBest classifica as *features* por esse valor p (quanto menor, melhor) e seleciona apenas as melhores *features* k.

Analisando os resultados apresentados nas Tabelas 18,19 e 20 verifica-se que um aumento do número *features* proporcionou um maior tempo de execução, mas um menor erro de classificação, ou seja, assume-se que a maior parte das *features* propostas desempenham um papel importante na correta classificação da família da proteína.

A Figura 29 contém um exemplo da pontuação de cada uma das *features* segundo o algoritmo SelectKBest, revelando a importância de cada uma para a resolução do problema de classificação de famílias de proteínas.

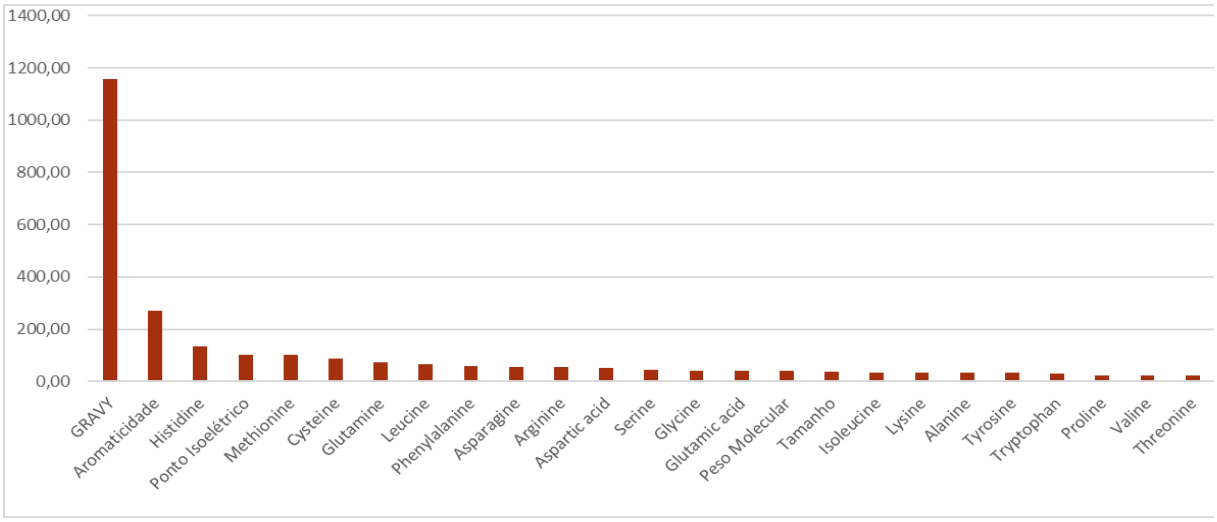

Figura 29 Importância das features segundo o algoritmo SelectKBest

Tal como observado na Figura 29, segundo o algoritmo SelectKBest as features que mais contribuem para o sucesso de aprendizagem dos modelos são essencialmente as propriedades físico-químicas GRAVY, aromaticidade, ponto isoelétrico e também a contagem dos aminoácidos Histidine e Methionine.

Os resultados apresentados na Tabela 21 foram usados para avaliar uma *feature* ou um grupo de *features* usando um classificador MLP.

Tabela 21 Avaliação das *features* ou grupo de *features* usando um classificador MLP

Modelo do sckit-learn	***Configuração***	***Feature ou grupo(s) de features***	***F1-Score***	***AUC***	***Tempo de execução (segundos)***
MLPClassifier	*default*	GRAVY	0.32	0.82	8.11
MLPClassifier	*default*	Tamanho da sequência	0.30	0.81	8.20
MLPClassifier	*default*	Propriedades físico-químicas	0.57	0.94	8.04
MLPClassifier	*default*	Propriedades físico-químicas + Tamanho da sequência	0.57	0.94	8.79
MLPClassifier	*default*	Propriedades físico-químicas excluindo o GRAVY	0.43	0.90	7.90
MLPClassifier	*default*	Contagem dos Aminoácidos	0.82	0.99	8.43
MLPClassifier	*default*	Contagem dos Aminoácidos + Tamanho da sequência	0.83	0.99	8.63
MLPClassifier	*default*	Contagem dos Aminoácidos + GRAVY	0.83	0.99	8.39
MLPClassifier	*default*	Contagem dos Aminoácidos + Propriedades físico-químicas	0.84	0.99	8.45

Analisando os resultados apresentados na Tabela 21 conclui-se que apesar de existirem *features* que contribuem mais do que outras, o sucesso da aprendizagem do modelo depende de um ou mais grupos de *features* e não de uma única *feature* em específico. O grupo de *features* que proporcionou melhores resultados foi a contagem dos aminoácidos, atingindo um F1-Score de 82%, portanto pode-se afirmar que existe uma relação entre a frequência absoluta de cada aminoácido da sequência primária da proteína e a família a que pertence. Combinando este grupo com o grupo propriedades físico-químicas permitiu subir o F1-Score para 84%.

6.1.4 Modelo treinado com sequências proteicas de um organismo para predizer sequências proteicas de outro organismo

- Modelo treinado com sequências proteicas de humanos para predizer sequências proteicas da Arabidopsis thaliana

Tal como descrito na introdução desta seção, do *dataset* utilizado apenas foram selecionadas as famílias que contêm mais de 100 exemplos, no entanto, para esta análise foi necessário diminuir esse valor para **90** de modo a existirem famílias em comum em ambos os *datasets*. Assumindo que o *dataset* de treino corresponde aos humanos e o *dataset* a testar à Arabidopsis thaliana, a Tabela 22 contém o número de instâncias de treino e teste para cada uma das famílias comuns em ambos os *datasets*.

Tabela 22 Descrição do *dataset* de treino (humanos) e do *dataset* de teste (arabidopsisthaliana)

Nome da família	***Instâncias de treino (Humano)***	***Instâncias de teste (Arabidopsis thaliana)***
Protein-tyrosine phosphatase	93	11
Tyr protein kinase	90	9

Tabela 23 Resultados da aplicação de um modelo treinado com sequências proteicas de humanos para predizer sequências proteicas de Arabidopsis thaliana

Modelo do sckit-learn	***Configuração***	***F1-Score***	***AUC***	***Tempo de execução (segundos)***
DecisionTreeClassifier	*default*	0.67	0.64	0.10
RandomForestClassifier	*default*	0.51	0.74	0.02
MLPClassifier	*default*	0.80	0.86	0.22
MLPClassifier	*hidden_sizes: (1000,1000), max_iter: 5000*	0.85	0.90	0.59

- Modelo treinado com sequências proteicas de humanos para predizer sequências proteicas de ratos;

Nesta análise usou-se a representação vetorial descrita em 4.3.1 e começou-se por analisar quais das famílias descritas na Tabela 11 existiam tanto no *dataset* do humano como no *dataset* do rato. Assumindo que o *dataset* de treino corresponde aos humanos e o *dataset* a testar aos ratos, a Tabela 24 contém o número de instâncias de treino e teste para cada uma das famílias comuns em ambos os *datasets*.

Tabela 24 Descrição do *dataset* de treino (humanos) e do *dataset* de teste (ratos)

Nome da família	***Instâncias de treino (humano)***	***Instâncias de teste (rato)***
G-protein coupled receptor 1	724	402
Krueppel C2H2-type zinc-finger protein	545	171
Peptidase S1	121	129

Tabela 25 Resultados da aplicação de um modelo treinado com sequências proteicas de humanos para predizer sequências proteicas de ratos

Modelo do sckit-learn	***Configuração***	***F1-Score***	***AUC***	***Tempo de execução (segundos)***
DecisionTreeClassifier	*default*	0.97	0.97	0.01
RandomForestClassifier	*default*	0.99	1.0	0.02
MLPClassifier	*default*	0.97	1.0	0.99

Analisando os resultados das Tabela 23 e 25 podemos concluir que um classificador treinado com sequências proteicas de humanos pode ser usado com sucesso para predizer as famílias de sequências proteicas de ratos ou mesmo organismos mais distantes como a Arabidopsis thaliana. Estes resultados sugerem que estes seres vivos, ainda que apresentem diferenças marcantes na sua forma, tamanho e ciclo de vida, compartilham alguns aspetos metabólicos, fisiológicos e anatómicos [112]. A explicação para as principais diferenças entre os humanos e os outros organismos justifica-se essencialmente pela diferença na expressão génica, no entanto, muitas das próprias funções das proteínas são semelhantes.

6.1.5 Rede Neuronal Recorrente - Long Short Term-Memory

Nesta análise usou-se a representação vetorial descrita em 4.3.2 para testar o desempenho da LSTM (consultar seção 4.5.4). As representações de aminoácidos a longo prazo na sequência são cruciais na previsão da família da proteína uma vez que a LSTM é capaz de encontrar dependências de longo prazo na sequência e memorizá-las por um longo período de tempo.

Tal como descrito na seção 7.3, uma das limitações deste trabalho foi o facto da biblioteca scikit-learn não oferecer suporte para técnicas de aprendizagem profunda. Para mitigar esse problema, o treino das LSTM foi realizado com recurso às bibliotecas TensorFlow [113] e Keras [70]. O TensorFlow foi criado pela Google Brain e é uma biblioteca *open-source* de *machine learning* e computação numérica que permite criar e treinar vários modelos e algoritmos de *machine learning* e aprendizagem profunda para detetar e decifrar padrões e correlações. Ele é compatível com o Python e é essa linguagem que usa para fornecer uma Application Programming Interface (API) de *front-end* no entanto, esta serve apenas como uma abstração uma vez que as aplicações são executadas em código C++ de alto desempenho. O TensorFlow opera em larga escala e em ambientes heterogéneos [113]: pode ser executado numa máquina local, em dispositivos Android ou IOS, na Central Process Unit (CPU), na Graphics Processing Unit (GPU) e quando usado na própria nuvem da Google pode ser executado num silício personalizado da unidade de processamento TensorFlow (TPU) desenvolvido pela Google (Figura 30), que permite acelerar ainda mais os algoritmos, tal como a Figura 33 comprova.

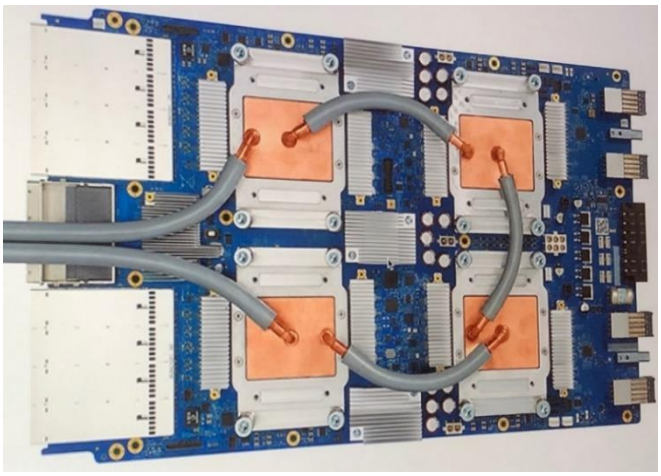

Figura 30 Google Tensor Processing Unit 3.0 [114]

O Keras é uma biblioteca *open-source* que fornece uma interface Python para as redes neuronais artificiais. Tal como a Figura 31 demonstra o Keras pode ser executado "em cima" do TensorFlow e foi desenvolvido para permitir que os utilizadores interajam com redes neuronais de aprendizagem profunda de forma rápida e intuitiva, através de APIs simples e consistentes. Estas APIs [115] permitem minimizar o número de ações do utilizador para casos de uso comuns numa pipeline de *machine learning*.

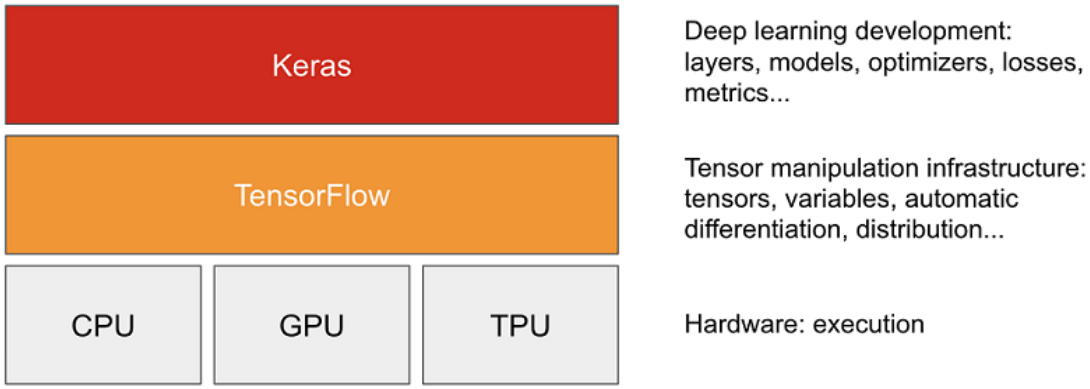

Figura 31 Arquitetura usada para treinar a LSTM [116]

Antes de obter os resultados da LSTM (apresentados na Tabela 27) realizou-se um *benchmark* de forma a avaliar o desempenho da mesma quando executada em diferentes tipos de *hardware* usando 100 iterações e a arquitetura apresentada na Figura 32 (descrita na Tabela 26).

```
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 519)]             0
_________________________________________________________________
embedding_2 (Embedding)      (None, 519, 128)          2688
_________________________________________________________________
bidirectional_2 (Bidirection (None, 128)               98816
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 34)                4386
=================================================================
Total params: 105,890
Trainable params: 105,890
Non-trainable params: 0
```

Figura 32 Arquitetura da LSTM usada para realizar o *benchmark*

Para executar o código nos servidores da nuvem da Google foi necessário recorrer ao Google Colab [117] - uma ferramenta que permite executar código Python usando apenas um *browser*, o que permite que os utilizadores tirem proveito do poder do *hardware* do servidor Google, incluindo GPUs e TPUs, independentemente da potência da sua máquina local. A máquina local que foi usada tem um processador Intel Core I7 – 7700HQ, 16 gigabytes de Random Access Memory (RAM) e um disco Solid State Drive (SSD) de 256 gigabytes. Para executar a LSTM numa máquina local usando a GPU seria necessária uma placa GPU NVIDIA [118] com arquiteturas CUDA [119] 3.5, 3.7, 5.2, 6.0, 6.1, 7.0 ou mais recentes [120]. Uma vez que a máquina local usada não possui esses requisitos, a LSTM foi treinada apenas com recurso á CPU. Por observação da Figura 33 conclui-se que apesar da GPU ter acelerado a execução do treino da LSTM, a TPU é o *hardware* ideal uma vez que cada iteração demorou em média apenas 1.03 segundos, o que corresponde a um tempo aproximadamente 67 vezes inferior ao tempo de execução na CPU da nuvem da Google, que demorou em média 68.5 segundos por iteração.

Figura 33 *Benchmark* usado para avaliar o desempenho de uma LSTM em diferentes tipos de *hardware*

Para obter os resultados apresentados na Tabela 27 usou-se uma arquitetura semelhante à apresentada na Figura 32 e foram testados vários hiperparâmetros até obter um modelo com uma boa capacidade de previsão. A Tabela 26 descreve as camadas usadas nesta arquitetura.

Tabela 26 Descrição da arquitetura usada para treinar a LSTM

Camada	***Descrição***
Embedding [121]	Esta camada só pode ser usada como a primeira camada de um modelo de aprendizagem profunda do Keras e foi usada para mapear cada caracter (aminoácido) num vetor de 128 elementos. Este valor pode ser considerado um hiperparâmetro ajustável, no entanto escolhemos o valor 128 para fornecer um maior nível de liberdade para as arquiteturas de aprendizagem profunda. Esta camada funciona de forma colaborativa com as outras camadas da rede durante a retro propagação e facilita o agrupamento de aminoácidos semelhantes na sequência. Este *cluster* de aminoácidos permite detetar a semântica e a semelhança estrutural de uma sequência proteica. Para usar esta camada foi necessário especificar 3 argumentos: • input_dim - define o tamanho do vocabulário nos dados do texto. Por exemplo, se os dados forem codificados por inteiro para valores entre 0 e 20 (total de aminoácidos), o tamanho do vocabulário será de 21 palavras; • output_dim - define o tamanho do espaço vetorial no qual as palavras serão inseridas; • input_lenght - define o comprimento das sequências de entrada que no nosso caso foi definido com o valor 519 (valor correspondente à média do comprimento de todas as sequências, tal como descrito na seção 4.3.2).
LSTM [122]	As entradas para cada camada de uma LSTM devem ser compostas por um *array* de 3 dimensões: • batch_size – define o número de amostras e um *batch* pode ser composto por uma ou mais amostras; • timesteps – define o número de passos no tempo a considerar. Um intervalo de tempo corresponde a um ponto de observação na amostra; • input_dim – corresponde à dimensão das *features* / embedding. A camada LSTM permite especificar múltiplos argumentos, no entanto neste trabalho avaliámos apenas a variação do seguinte parâmetro: • units - Define o número de neurónios da LSTM.
Droupout [123]	É uma camada que atua como um parâmetro de regularização para evitar *overfitting* (ocorre quando um modelo se ajusta muito bem ao conjunto de dados anteriormente observado, mas se mostra ineficaz para prever novos resultados). Esta camada permite especificar 3 argumentos, no entanto neste trabalho avaliámos apenas a variação do seguinte parâmetro: • rate - valor percentual (varia entre 0 (sem *dropout*) e 1) que define a probabilidade dos neurónios (e as suas conexões) serem removidos aleatoriamente durante o treino do modelo de aprendizagem profunda.
Dense [124]	Corresponde à camada onde cada neurónio está conectado a todos os neurónios da camada seguinte. A saída de cada operação é dada pela função $y = X * W + b$, em que X representa a camada da entrada e W e b são os pesos e o *bias* da camada. Para usar esta camada foi necessário especificar os seguintes parâmetros: • units - define a dimensão do espaço de saída, que no problema em questão corresponde ao número de classes a serem previstas (34 famílias). • activation - especifica a função de ativação a ser usada. A sua escolha é muito importante para a camada de saída uma vez que define o formato que as previsões terão. Uma vez que o problema em questão é multiclasse, foi utilizada a função softmax [125].

Tabela 27 Resultados dos testes da LSTM com uma representação vetorial baseada na sequência em bruto

Modelo do keras	***Nº de neurónios***	***Iterações***	***Droupout***	***F1-Score***	***AUC***	***Tempo de execução (segundos)***
LSTM	64	100	0	0.46	0.89	76.45
LSTM	64	100	0.2	0.44	0.87	75.87
LSTM	64	100	0.5	0.37	0.83	76.31
LSTM	64	100	0.8	0.37	0.84	76.22
LSTM	128	100	0	0.51	0.91	83.60
LSTM	128	100	0.2	0.45	0.88	83.70
LSTM	128	100	0.5	0.45	0.88	83.33
LSTM	128	100	0.8	0.40	0.86	83.33
LSTM	64	200	0	0.59	0.94	146.58
LSTM	64	200	0.2	0.53	0.92	145.33
LSTM	64	200	0.5	0.50	0.91	153.43
LSTM	64	200	0.8	0.41	0.86	152.66
LSTM	128	200	0	0.71	0.97	159.29
LSTM	128	200	0.2	0.67	0.96	161.26
LSTM	128	200	0.5	0.61	0.95	162.39
LSTM	128	200	0.8	0.48	0.90	162.45
LSTM	256	200	0	0.83	0.98	316.78
LSTM	256	200	0.2	0.84	0.99	315.61
LSTM	256	200	0.5	0.81	0.98	321.34
LSTM	256	200	0.8	0.73	0.96	321.82
Bidirectional LSTM	64	100	0	0.69	0.96	107.26
Bidirectional LSTM	64	100	0.2	0.65	0.95	105.39
Bidirectional LSTM	64	100	0.5	0.62	0.95	107.85
Bidirectional LSTM	64	100	0.8	0.57	0.93	106.56
Bidirectional LSTM	128	100	0	0.74	0.97	121.61
Bidirectional LSTM	128	100	0.2	0.76	0.98	123.14
Bidirectional LSTM	128	100	0.5	0.70	0.96	122.18
Bidirectional LSTM	128	100	0.8	0.61	0.95	122.16
Bidirectional LSTM	64	200	0	0.74	0.97	199.85
Bidirectional LSTM	64	200	0.2	0.74	0.97	202.72

Bidirectional LSTM	64	200	0.5	0.72	0.97	208.57
Bidirectional LSTM	64	200	0.8	0.55	0.93	208.37
Bidirectional LSTM	128	200	0	0.82	0.98	249.56
Bidirectional LSTM	128	200	0.2	0.82	0.98	245.43
Bidirectional LSTM	128	200	0.5	0.81	0.98	235.42
Bidirectional LSTM	128	200	0.8	0.72	0.97	243.19
Bidirectional LSTM	256	300	0	0.84	0.99	476.94
Bidirectional LSTM	256	300	0.2	0.86	0.99	474.60
Bidirectional LSTM	256	300	0.5	0.86	0.99	478.58
Bidirectional LSTM	256	300	0.8	0.86	0.99	496.83

Por observação dos resultados da Tabela 27 verificamos que quanto maior é o número de neurónios da rede LSTM, mais poderosa a rede fica, no entanto também aumenta o tempo de treino. Observando os resultados desta tabela verificamos também que a arquitetura bidirecional da LSTM proporcionou melhores resultados, mas exigiu mais recursos computacionais. Estes resultados justificam-se pelo facto da LSTM simples usar apenas a subsequência passada e ignorar as subsequências futuras enquanto que a arquitetura bidirecional considera a sequência passada e futura nas proteínas. As arquiteturas bidirecionais originais calculam duas camadas ocultas, uma que processa a sequência de entrada em uma direção direta e a outra que processa a sequência de trás para frente e concatena as suas saídas. Uma vez que usando uma técnica de aprendizagem profunda conseguimos obter um F1-Score = 86%, podemos concluir que existe uma relação entre a disposição das sequências de aminoácidos e as famílias da proteína.

Comparando os resultados da Tabela 14 com os resultados da Tabela 27 verificamos que tanto as redes neuronais tradicionais como as redes de aprendizagem profunda atingiram o mesmo F1-Score (86%). Apesar da rede MLP ter sido bastante mais rápida a realizar o treino e o teste do modelo, não nos podemos esquecer que a técnica de pré-processamento dos dados usada em cada uma é diferente. A técnica de *encoding* usada na MLP (consultar seção 4.3.1) demorou em média 3.47 segundos enquanto que a técnica de *encoding* usada na LSTM (consultar seção 4.3.2) demorou em média apenas 1.05 segundos. Por esse motivo e considerando o tempo total necessário para executar cada *pipeline* (descrita na seção 4.1) podemos concluir que para o *dataset* em questão o uso das redes neuronais tracionais foi mais eficiente do que as técnicas de aprendizagem profunda.

7 CONCLUSÕES

7.1 Visão geral do problema

Os custos decrescentes e a rapidez do sequenciamento permitiram um aumento na disponibilidade de sequências completas de genoma para um grande número de organismos. Apenas 1% dessas sequências são anotadas manualmente, portanto cada vez é mais necessário investigar técnicas para classificar de forma automática e em tempo útil as restantes sequências.

Neste projeto comprovámos que as técnicas de *machine learning* podem ser usadas para classificar corretamente e de forma automática as famílias de um conjunto de sequências proteicas obtidas a partir do UniPro-tKB/Swiss-Prot. Para isso extraiu-se um conjunto de *features* dessas sequências, que incluem propriedades estruturais primárias e propriedades físico-químicas dos aminoácidos e foram usadas como entrada de diferentes modelos de aprendizagem supervisionada do sckit-learn, como árvores de decisão, Random Forest e a rede neuronal MLP. Nalgumas das experiências realizadas aplicaram-se técnicas de pré-processamento como a análise de componentes principais e a seleção das melhores *features*. Os resultados obtidos sugerem que a aplicação dessas técnicas, combinadas com a boa performance dos modelos RandomForest e da rede neuronal MLP, permitem classificar famílias de proteínas com elevado desempenho, tal como um F1-Score = 0.88 e uma AUC = 0.99 comprova. Outra abordagem para comprovar o sucesso das técnicas de *machine learning* foi utilizar as bibliotecas Keras e o Tensorflow para treinar um modelo de aprendizagem profunda - uma rede neuronal recorrente LSTM, na nuvem da Google. Como entrada deste modelo foram usadas as próprias sequências e atingiu-se um F1-Score = 0.86.

7.2 Contribuições do projeto

7.2.1 Contribuições Teóricas

A metodologia proposta para classificar as famílias de proteínas baseada em técnicas de *machine learning* é a principal contribuição teórica deste projeto. Os resultados obtidos sugerem que estas técnicas podem substituir a determinação experimental realizada em laboratórios, que é um processo trabalhoso e bastante demorado. Os biólogos e cientistas podem aplicar esta metodologia a um conjunto diverso de problemas da bioinformática como a visualização de proteínas, classificação de famílias de proteínas, predição de estrutura e a extração de domínio.

7.2.2 Contribuições Práticas

O desenvolvimento de ferramentas computacionais para anotação de famílias de proteínas tem um papel essencial no conhecimento das funções de todas as proteínas. Atualmente existe uma necessidade urgente de desenvolver métodos computacionais e infraestruturas

bioinformáticas para anotação de proteínas em larga escala. Nesse sentido o SmartGeno pode ser visto como uma mais valia uma vez que é um protótipo capaz de agrupar proteínas em famílias com base na sua sequência, contribuindo para a resolução da tarefa desafiadora de atribuir uma função a uma proteína. A identificação da função de uma proteína é tão importante que pode ajudar a entender os mecanismos de uma doença ou a produzir medicamentos para a mesma.

Outra das contribuições práticas deste projeto é a descrição das ferramentas e tecnologias utilizadas bem como os tutoriais com explicações passo a passo que, podem ser usados como guia para quem tenha o interesse de criar uma ferramenta *web* com um ou mais classificadores para resolver problemas numa determinada área.

7.3 Limitações

Parece-nos pertinente referir algumas das dificuldades que foram sentidas durante a elaboração deste projeto. Uma das principais limitações reside no facto da biblioteca de *machine learning* scikit-learn não oferecer suporte para técnicas de aprendizagem profunda (ex.: LSTM) ao contrário do TensorFlow ou do Keras. A investigação da aplicação dessas técnicas para classificação de famílias de proteínas foi iniciada após o desenvolvimento do protótipo SmartGeno e, portanto, o acréscimo desse requisito no SmartGeno obrigaria a um *refactor* grande na lógica e no código-fonte para suportar todas as bibliotecas necessárias.

O SmartGeno apenas suporta o treino de modelos de *machine learning* para 4 organismos: humanos, ratos, Arabidopsis thaliana e de Sac-charomyces cerevisiae. Os *datasets* desses organismos foram obtidos do UniProtKB e colocados manualmente no servidor o que representa uma limitação, uma vez que se o UniProtKB lançar uma nova versão, o SmartGeno fica com a informação desatualizada.

Por último, mas não menos importante, gostaríamos de mencionar que não foi possível encontrar um *dataset* com as mesmas características do usado neste projeto, portanto, não foi possível confrontar os nossos resultados com os resultados analisados no estado da arte.

7.4 Análise crítica e trabalhos futuros

Tal como identificado na seção anterior, no SmartGeno as sequências proteicas dos organismos são atualizadas manualmente. Uma futura melhoria seria utilizar a Application Programming Interface (API) do UnitProt [126] para permitir a atualização dos dados a cada versão do UnitProt.

Há também várias direções em que esta investigação poderia ser expandida. Uma delas seria perceber se é possível predizer a estrutura de uma proteína usando como entrada dos modelos de *machine learning* as representações vetoriais propostas na seção 4.3.1 e 4.3.2. As estruturas fornecem informações valiosas para ajudar a entender melhor como uma cadeia de proteína se

dobra na sua estrutura 3D original, como as cadeias interagem na forma quaternária, e como prever estruturas a partir da sequência (estrutura primária) [127]. A determinação experimental de uma estrutura de proteínas ainda é um processo muito dispendioso, demorado, e nem sempre possível. A comparação de proteínas com informação de bases de dados anotadas é uma maneira de entender a sua estrutura e combate a elevada discrepância entre o número de sequências proteicas e o número de estruturas conhecidas.

Outra direção a seguir seria disponibilizar um *webservice* que permitisse a classificação de famílias de um conjunto de dados no formato FASTA.

Por último, sugere-se como trabalho futuro a integração das bibliotecas TensorFlow e Keras na plataforma SmartGeno, de modo a que a mesma permita o treino, teste e configuração de modelos de aprendizagem profunda.

REFERÊNCIAS

[1] S. Das e C. A. Orengo, «Protein function annotation using protein domain family resources», *Methods*, vol. 93, pp. 24–34, Jan. 2016, doi: 10.1016/j.ymeth.2015.09.029.

[2] «Home». https://smartgeno.azurewebsites.net/ (acedido Set. 03, 2019).

[3] «Guia de introdução – Guia do C# | Microsoft Docs». https://docs.microsoft.com/pt-br/dotnet/csharp/getting-started/ (acedido Dez. 12, 2019).

[4] «scikit-learn: machine learning in Python — scikit-learn 0.20.2 documentation». https://scikit-learn.org/stable/ (acedido Dez. 30, 2018).

[5] «IronPython.net /». https://ironpython.net/ (acedido Dez. 12, 2019).

[6] «NumPy — NumPy». http://www.numpy.org/ (acedido Jan. 14, 2019).

[7] «SciPy.org — SciPy.org». https://www.scipy.org/ (acedido Jan. 14, 2019).

[8] «Accord.NET Machine Learning Framework». http://accord-framework.net/ (acedido Set. 04, 2019).

[9] J. Kyte e R. F. Doolittle, «A simple method for displaying the hydropathic character of a protein», *Journal of Molecular Biology*, vol. 157, n. 1, pp. 105–132, Mai. 1982, doi: 10.1016/0022-2836(82)90515-0.

[10] «Ácido desoxirribonucleico», *Wikipédia, a enciclopédia livre*. Mar. 03, 2020, Acedido: Mar. 14, 2020. [Em linha]. Disponível em: https://pt.wikipedia.org/w/index.php?title=%C3%81cido_desoxirribonucleico&oldid=57647193.

[11] «Transcrição (genética)», *Wikipédia, a enciclopédia livre*. Dez. 08, 2019, Acedido: Dez. 16, 2019. [Em linha]. Disponível em: https://pt.wikipedia.org/w/index.php?title=Transcri%C3%A7%C3%A3o_(gen%C3%A9tica)&oldid=56908803.

[12] «Tradução (genética)», *Wikipédia, a enciclopédia livre*. Out. 25, 2019, Acedido: Dez. 16, 2019. [Em linha]. Disponível em: https://pt.wikipedia.org/w/index.php?title=Tradu%C3%A7%C3%A3o_(gen%C3%A9tica)&oldid=56561208.

[13] «Sequenciamento de DNA», *Wikipédia, a enciclopédia livre*. Out. 31, 2019, Acedido: Dez. 18, 2019. [Em linha]. Disponível em: https://pt.wikipedia.org/w/index.php?title=Sequenciamento_de_DNA&oldid=56609148.

[14] «Human Genome Project Information». https://web.ornl.gov/sci/techresources/Human_Genome/index.shtml (acedido Dez. 18, 2019).

[15] S. A. Jeon *et al.*, «Comparison of the MGISEQ-2000 and Illumina HiSeq 4000 sequencing platforms for RNA sequencing», *Genomics Inform*, vol. 17, n. 3, Set. 2019, doi: 10.5808/GI.2019.17.3.e32.

[16] «The Cost of Sequencing a Human Genome», *Genome.gov*. http://www.genome.gov/about-genomics/fact-sheets/Sequencing-Human-Genome-cost (acedido Mar. 29, 2020).

[17] «MiSeq Sequencer - Global Bios». https://globalbios.com/index.php?id_product=95&controller=product (acedido Mar. 29, 2020).

[18] «IUPAC-IUB Commission on Biochemical Nomenclature A One-Letter Notation for Amino Acid Secruences1-3 Tentative Rules», The Journal of Biological Chemistry.

[19] «Aminoácido», *Wikipédia, a enciclopédia livre*. Mai. 21, 2019, Acedido: Ago. 06, 2019. [Em linha]. Disponível em: https://pt.wikipedia.org/w/index.php?title=Amino%C3%A1cido&oldid=55226990.

[20] A. Vazhayil, V. R, e S. KP, «DeepProteomics: Protein family classification using Shallow and Deep Networks», ArXiv, 2018.

[21] R. Fitzsimmons, N. Amin, e V. N. Uversky, «Understanding the roles of intrinsic disorder in subunits of hemoglobin and the disease process of sickle cell anemia», *Intrinsically Disord Proteins*, vol. 4, n. 1, Dez. 2016, doi: 10.1080/21690707.2016.1248273.
[22] R. Beduka, «▷ O que são as PROTEÍNAS? - Estrutura, Funções, Classificações e Síntese!», *Beduka - Tudo sobre ENEM, SISU, FIES, ProUni e vestibulares*, Mai. 24, 2019. https://beduka.com/blog/materias/biologia/o-que-sao-as-proteinas/ (acedido Mar. 29, 2020).
[23] «Protein family», *Wikipedia*. Fev. 04, 2020, Acedido: Mar. 29, 2020. [Em linha]. Disponível em: https://en.wikipedia.org/w/index.php?title=Protein_family&oldid=939070179.
[24] D. P. Brown, N. Krishnamurthy, e K. Sjölander, «Automated Protein Subfamily Identification and Classification», *PLoS Comput Biol*, vol. 3, n. 8, Ago. 2007, doi: 10.1371/journal.pcbi.0030160.
[25] J. A. Eisen, K. S. Sweder, e P. C. Hanawalt, «Evolution of the SNF2 family of proteins: subfamilies with distinct sequences and functions.», *Nucleic Acids Res*, vol. 23, n. 14, pp. 2715–2723, Jul. 1995.
[26] H. Mi, S. Poudel, A. Muruganujan, J. T. Casagrande, e P. D. Thomas, «PANTHER version 10: expanded protein families and functions, and analysis tools», *Nucleic Acids Res*, vol. 44, n. Database issue, pp. D336–D342, Jan. 2016, doi: 10.1093/nar/gkv1194.
[27] «What are protein families? | EMBL-EBI Train online». https://www.ebi.ac.uk/training/online/course/introduction-protein-classification-ebi/protein-classification/what-are-protein-families (acedido Abr. 02, 2019).
[28] «Structural alignment», *Wikipedia*. Set. 12, 2019, Acedido: Mar. 29, 2020. [Em linha]. Disponível em: https://en.wikipedia.org/w/index.php?title=Structural_alignment&oldid=915276711.
[29] S. El-Gebali *et al.*, «The Pfam protein families database in 2019», *Nucleic Acids Res*, vol. 47, n. D1, pp. D427–D432, Jan. 2019, doi: 10.1093/nar/gky995.
[30] R. D. Finn *et al.*, «Pfam: the protein families database», *Nucleic Acids Res*, vol. 42, n. Database issue, pp. D222–D230, Jan. 2014, doi: 10.1093/nar/gkt1223.
[31] «reviewed:yes in UniProtKB». https://www.uniprot.org/uniprot/?query=reviewed:yes (acedido Mar. 26, 2019).
[32] «PANTHER - Gene List Analysis». http://pantherdb.org/ (acedido Dez. 17, 2019).
[33] H. Mi *et al.*, «The PANTHER database of protein families, subfamilies, functions and pathways», *Nucleic Acids Res*, vol. 33, n. suppl_1, pp. D284–D288, Jan. 2005, doi: 10.1093/nar/gki078.
[34] «PANTHER - PANTHER data». http://www.pantherdb.org/data/ (acedido Dez. 17, 2019).
[35] «CATH: Protein Structure Classification Database at UCL». http://www.cathdb.info/ (acedido Ago. 24, 2019).
[36] R. P. D. Bank, «RCSB PDB: Homepage». https://www.rcsb.org/ (acedido Dez. 17, 2019).
[37] «CATH». https://www.cathdb.info/wiki/doku/?id=release_notes (acedido Dez. 17, 2019).
[38] J.-M. Chandonia, N. K. Fox, e S. E. Brenner, «SCOPe: classification of large macromolecular structures in the structural classification of proteins—extended database», *Nucleic Acids Res*, vol. 47, n. D1, pp. D475–D481, Jan. 2019, doi: 10.1093/nar/gky1134.
[39] «Structural Classification of Proteins database», *Wikipedia*. Mar. 20, 2020, Acedido: Mar. 29, 2020. [Em linha]. Disponível em: https://en.wikipedia.org/w/index.php?title=Structural_Classification_of_Proteins_database&oldid=946562555.
[40] «InterPro». https://www.ebi.ac.uk/interpro/ (acedido Dez. 20, 2019).

[41] «InterPro: the integrative protein signature database | Nucleic Acids Research | Oxford Academic». https://academic.oup.com/nar/article/37/suppl_1/D211/1010485 (acedido Dez. 20, 2019).
[42] «Conserved Domains Database (CDD) and Resources». https://www.ncbi.nlm.nih.gov/Structure/cdd/cdd.shtml (acedido Dez. 20, 2019).
[43] «HAMAP home». https://hamap.expasy.org/ (acedido Dez. 20, 2019).
[44] «Welcome to PIR [Protein Information Resource]». https://proteininformationresource.org/pirwww/index.shtml (acedido Dez. 20, 2019).
[45] «PRINTS». http://130.88.97.239/PRINTS/index.php (acedido Dez. 20, 2019).
[46] «PROSITE user manual». https://prosite.expasy.org/prosuser.html (acedido Dez. 20, 2019).
[47] «Structure Function Linkage Database». http://sfld.rbvi.ucsf.edu/archive/django/index.html (acedido Dez. 20, 2019).
[48] I. Letunic e P. Bork, «20 years of the SMART protein domain annotation resource», *Nucleic Acids Research*, vol. 46, n. D1, pp. D493–D496, Jan. 2018, doi: 10.1093/nar/gkx922.
[49] A. P. Pandurangan, J. Stahlhacke, M. E. Oates, B. Smithers, e J. Gough, «The SUPERFAMILY 2.0 database: a significant proteome update and a new webserver», *Nucleic Acids Research*, vol. 47, n. D1, pp. D490–D494, Jan. 2019, doi: 10.1093/nar/gky1130.
[50] «TIGRFAMS | J. Craig Venter Institute». https://www.jcvi.org/tigrfams (acedido Dez. 20, 2019).
[51] «InterProScan - InterPro». https://www.ebi.ac.uk/interpro/search/sequence/ (acedido Mar. 29, 2020).
[52] «HAMAP-Scan». https://hamap.expasy.org/hamap_scan.html (acedido Jan. 02, 2020).
[53] I. Pedruzzi *et al.*, «HAMAP in 2015: updates to the protein family classification and annotation system», *Nucleic Acids Res*, vol. 43, n. D1, pp. D1064–D1070, Jan. 2015, doi: 10.1093/nar/gku1002.
[54] «About HAMAP». https://hamap.expasy.org/hamap_about.html (acedido Jan. 02, 2020).
[55] «UniRule». https://www.uniprot.org/help/unirule (acedido Jan. 02, 2020).
[56] S. B. Needleman e C. D. Wunsch, «A general method applicable to the search for similarities in the amino acid sequence of two proteins», *Journal of Molecular Biology*, vol. 48, n. 3, pp. 443–453, Mar. 1970, doi: 10.1016/0022-2836(70)90057-4.
[57] T. K. Lee e T. Nguyen, «Protein Family Classification with Neural Networks», p. 9.
[58] «UniProt». https://www.uniprot.org/ (acedido Dez. 16, 2018).
[59] «GloVe: Global Vectors for Word Representation». https://nlp.stanford.edu/projects/glove/ (acedido Dez. 17, 2018).
[60] S. Hochreiter, «Long Short-Term Memory», Neural Computation, pp. 1735–1780, 1997.
[61] E. Asgari e M. Mofrad, «Continuous Distributed Representation of Biological Sequences for Deep Proteomics and Genomics», Plos One, 2015.
[62] C. Cai, L. Y. Han, Z.-L. Ji, X. Chen, e Y. Chen, «SVM-Prot: Web-Based Support Vector Machine Software for Functional Classification of a Protein from Its Primary Sequence», *Nucleic acids research*, vol. 31, pp. 3692–7, Ago. 2003, doi: 10.1093/nar/gkg600.
[63] P. N. Rao, G. Edu, T. U. Devi, D. Kaladhar, G. Sridhar, e A. A. Rao, «A PROBABILISTIC NEURAL NETWORK APPROACH FOR PROTEIN SUPERFAMILY CLASSIFICATION», p. 7, 2005.
[64] «How is protein family membership assigned in UniProtKB?» https://www.uniprot.org/help/family_membership (acedido Jan. 06, 2020).
[65] «reviewed:yes in UniProtKB». https://www.uniprot.org/uniprot/?query=reviewed:yes#customize-columns (acedido Mar. 29, 2020).

[66] J. Wang, S. Dennis, Q. Ma, e C. Wu, «Application of Neural Networks to Biological Data Mining: A Case Study in Protein Sequence Classification», KDD, 2000.
[67] B. Bjellqvist *et al.*, «The focusing positions of polypeptides in immobilized pH gradients can be predicted from their amino acid sequences», *Electrophoresis*, vol. 14, n. 10, pp. 1023–1031, Out. 1993.
[68] R. G. Duggleby, «Calculation of the molecular weight of proteins from electrophoretic and gel exclusion chromatographic experiments», *Bioinformatics*, vol. 10, n. 2, pp. 133–135, Abr. 1994, doi: 10.1093/bioinformatics/10.2.133.
[69] «Bio.SeqUtils.IsoelectricPoint». https://biopython.org/DIST/docs/api/Bio.SeqUtils.IsoelectricPoint-module.html (acedido Dez. 13, 2019).
[70] «Home - Keras Documentation». https://keras.io/ (acedido Jan. 02, 2020).
[71] «tf.keras.preprocessing.sequence.pad_sequences | TensorFlow Core v2.1.0», *TensorFlow*. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/sequence/pad_sequences?hl=pt (acedido Mar. 28, 2020).
[72] I. Jolliffe, «Principal Component Analysis», em *International Encyclopedia of Statistical Science*, M. Lovric, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1094–1096.
[73] R. Alves, «Aplicação de Modelos de Redes Neuronais para Previsão de Consumos de Energia», Instituto Superior Técnico Lisboa, Dissertação para obtenção do Grau de Mestre em Engenharia Mecânica.
[74] J. Brownlee, *Machine Learning Mastery With Python: Understand Your Data, Create Accurate Models, and Work Projects End-to-End*. Machine Learning Mastery, 2016.
[75] «1.13. Feature selection — scikit-learn 0.20.2 documentation». https://scikit-learn.org/stable/modules/feature_selection.html (acedido Jan. 21, 2019).
[76] A. Sikandar *et al.*, «Decision Tree Based Approaches for Detecting Protein Complex in Protein Protein Interaction Network (PPI) via Link and Sequence Analysis», *IEEE Access*, vol. 6, pp. 22108–22120, 2018, doi: 10.1109/ACCESS.2018.2807811.
[77] T. Yiu, «Understanding Random Forest», *Medium*, Ago. 14, 2019. https://towardsdatascience.com/understanding-random-forest-58381e0602d2 (acedido Mar. 29, 2020).
[78] B. Ç. Yavuz, N. Yurtay, e O. Ozkan, «Prediction of Protein Secondary Structure With Clonal Selection Algorithm and Multilayer Perceptron», *IEEE Access*, vol. 6, pp. 45256–45261, 2018, doi: 10.1109/ACCESS.2018.2864665.
[79] M. Tsubaki, M. Shimbo, e Y. Matsumoto, «Protein Fold Recognition with Representation Learning and Long Short-Term Memory», *IPSJ Transactions on Bioinformatics*, vol. 10, n. 0, pp. 2–8, 2017, doi: 10.2197/ipsjtbio.10.2.
[80] A. Jacobsson e C. Gustavsson, «Prediction of the Number of Residue Contacts in Proteins using LSTM Neural Networks», 2003.
[81] E. Kang, «Long Short-Term Memory (LSTM): Concept», *Medium*, Set. 01, 2017. https://medium.com/@kangeugine/long-short-term-memory-lstm-concept-cb3283934359 (acedido Mar. 29, 2020).
[82] O. Shalev (@ofirdi), «Recall, Precision, F1, ROC, AUC, and everything», *Medium*, Mai. 31, 2019. https://medium.com/swlh/recall-precision-f1-roc-auc-and-everything-542aedf322b9 (acedido Nov. 07, 2020).
[83] «sklearn.metrics.precision_recall_fscore_support — scikit-learn 0.22.1 documentation». https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html (acedido Jan. 14, 2020).

[84] «Receiver Operating Characteristic (ROC) — scikit-learn 0.23.2 documentation». https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html#sphx-glr-auto-examples-model-selection-plot-roc-py (acedido Nov. 07, 2020).
[85] «Ideal Modeling & Diagramming Tool for Agile Team Collaboration». https://www.visual-paradigm.com/ (acedido Set. 04, 2019).
[86] «Visual Paradigm - UML, Agile, PMBOK, TOGAF, BPMN and More!» https://www.visual-paradigm.com/features/ (acedido Set. 04, 2019).
[87] «DBeaver Community | Free Universal Database Tool». https://dbeaver.io/ (acedido Set. 04, 2019).
[88] «Main Features – DBeaver». https://dbeaver.com/features/ (acedido Set. 04, 2019).
[89] «Matplotlib: Python plotting — Matplotlib 3.0.2 documentation». https://matplotlib.org/ (acedido Jan. 14, 2019).
[90] «BLAS (Basic Linear Algebra Subprograms)». http://www.netlib.org/blas/ (acedido Abr. 04, 2020).
[91] admin, «Intel® Math Kernel Library (Intel® MKL)», 23:50:13 UTC. https://software.intel.com/en-us/mkl (acedido Abr. 04, 2020).
[92] «Yellowbrick: Machine Learning Visualization — yellowbrick 0.9 documentation». http://www.scikit-yb.org/en/latest/ (acedido Jan. 23, 2019).
[93] «IDE do Visual Studio, Editor de Código, Azure DevOps e App Center», *Visual Studio*. https://visualstudio.microsoft.com/pt-br/ (acedido Set. 05, 2019).
[94] «Build software better, together», *GitHub*. https://github.com (acedido Set. 04, 2019).
[95] A. Pereira, V. V. Cogo, e A. S. Charao, «Frameworks para Desenvolvimento Ra´pido de Aplicac¸o˜es Web: um Estudo de Caso com CakePHP e Django», p. 6.
[96] «Project Structure — django-project-skeleton 1.4 documentation». https://django-project-skeleton.readthedocs.io/en/latest/structure.html (acedido Jan. 07, 2020).
[97] «Built-in template tags and filters | Django documentation | Django». https://docs.djangoproject.com/en/3.0/ref/templates/builtins/ (acedido Jan. 07, 2020).
[98] «PostgreSQL: The world's most advanced open source database». https://www.postgresql.org/ (acedido Set. 05, 2019).
[99] «PostgreSQL: About». https://www.postgresql.org/about/ (acedido Set. 05, 2019).
[100] «Plataforma de Computação na Cloud e Serviços do Microsoft Azure». https://azure.microsoft.com/pt-pt/ (acedido Set. 04, 2019).
[101] «sklearn.model_selection.train_test_split — scikit-learn 0.20.2 documentation». https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html (acedido Jan. 21, 2019).
[102] «sklearn.neural_network.MLPClassifier — scikit-learn 0.20.2 documentation». https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html (acedido Jan. 27, 2019).
[103] «Quasi-Newton method», *Wikipedia*. Mai. 27, 2020, Acedido: Jun. 07, 2020. [Em linha]. Disponível em: https://en.wikipedia.org/w/index.php?title=Quasi-Newton_method&oldid=959247803.
[104] D. P. Kingma e J. Ba, «Adam: A Method for Stochastic Optimization», *arXiv:1412.6980 [cs]*, Jan. 2017, Acedido: Jun. 07, 2020. [Em linha]. Disponível em: http://arxiv.org/abs/1412.6980.
[105] «sklearn.tree.DecisionTreeClassifier — scikit-learn 0.20.3 documentation». https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html (acedido Mar. 27, 2019).

[106] «3.2.4.3.1. sklearn.ensemble.RandomForestClassifier — scikit-learn 0.22.1 documentation». https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (acedido Jan. 02, 2020).
[107] «sklearn.preprocessing.StandardScaler — scikit-learn 0.20.3 documentation». https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html#sklearn.preprocessing.StandardScaler (acedido Mar. 26, 2019).
[108] N. Halko, P.-G. Martinsson, e J. A. Tropp, «Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions», *arXiv:0909.4061 [math]*, Set. 2009, Acedido: Jan. 21, 2019. [Em linha]. Disponível em: http://arxiv.org/abs/0909.4061.
[109] «sklearn.decomposition.PCA — scikit-learn 0.20.2 documentation». https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html (acedido Jan. 21, 2019).
[110] «sklearn.feature_selection.SelectKBest — scikit-learn 0.20.2 documentation». https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html#sklearn.feature_selection.SelectKBest (acedido Jan. 21, 2019).
[111] «sklearn.feature_selection.f_classif — scikit-learn 0.20.2 documentation». https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif.html#sklearn.feature_selection.f_classif (acedido Fev. 01, 2019).
[112] Y. Suzuki *et al.*, «Sequence Comparison of Human and Mouse Genes Reveals a Homologous Block Structure in the Promoter Regions», *Genome Res.*, vol. 14, n. 9, pp. 1711–1718, Jan. 2004, doi: 10.1101/gr.2435604.
[113] «TensorFlow». https://www.tensorflow.org/?hl=pt (acedido Jan. 02, 2020).
[114] «Tensor Processing Unit», *Wikipedia*. Set. 15, 2020, Acedido: Nov. 10, 2020. [Em linha]. Disponível em: https://en.wikipedia.org/w/index.php?title=Tensor_Processing_Unit&oldid=978594015.
[115] K. Team, «Keras documentation: Keras API reference». https://keras.io/api/ (acedido Nov. 10, 2020).
[116] «3 Introduction to Keras and TensorFlow · Deep Learning with Python, Second Edition MEAP V04». https://livebook.manning.com/book/deep-learning-with-python-second-edition/chapter-3/v-4/ (acedido Nov. 10, 2020).
[117] «Google Colaboratory». https://colab.research.google.com/notebooks/intro.ipynb (acedido Nov. 11, 2020).
[118] «Buy NVIDIA Graphics Cards | NVIDIA Store». https://www.nvidia.com/en-gb/graphics-cards/ (acedido Abr. 02, 2020).
[119] «CUDA Zone», *NVIDIA Developer*, Jul. 18, 2017. https://developer.nvidia.com/cuda-zone (acedido Nov. 11, 2020).
[120] «Suporte a GPUs», *TensorFlow*. https://www.tensorflow.org/install/gpu?hl=pt-br (acedido Nov. 11, 2020).
[121] K. Team, «Keras documentation: Embedding layer». https://keras.io/api/layers/core_layers/embedding/ (acedido Nov. 30, 2020).
[122] K. Team, «Keras documentation: LSTM layer». https://keras.io/api/layers/recurrent_layers/lstm/ (acedido Nov. 30, 2020).
[123] K. Team, «Keras documentation: Dropout layer». https://keras.io/api/layers/regularization_layers/dropout/ (acedido Nov. 30, 2020).
[124] K. Team, «Keras documentation: Dense layer». https://keras.io/api/layers/core_layers/dense/ (acedido Nov. 30, 2020).

[125] «Softmax function - Wikipedia». https://en.wikipedia.org/wiki/Softmax_function (acedido Nov. 30, 2020).
[126] «Programmatic access - Downloading data at every UniProt release». https://www.uniprot.org/help/api_downloading (acedido Jan. 02, 2020).
[127] J.-M. Chandonia e S. E. Brenner, «The Impact of Structural Genomics: Expectations and Outcomes», *Science*, vol. 311, n. 5759, pp. 347–351, Jan. 2006, doi: 10.1126/science.1121018.
[128] «OpenStreetMap», *OpenStreetMap*. https://www.openstreetmap.org/ (acedido Nov. 18, 2019).
[129] «Authentication - Django REST framework». https://www.django-rest-framework.org/api-guide/authentication/ (acedido Nov. 18, 2019).
[130] «sklearn.datasets.load_iris — scikit-learn 0.22.1 documentation». https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html (acedido Jan. 07, 2020).
[131] F. D. Gregorio, *psycopg2: psycopg2 - Python-PostgreSQL Database Adapter*. .
[132] «sklearn.preprocessing.LabelEncoder — scikit-learn 0.22 documentation». https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html (acedido Dez. 13, 2019).

ANEXO A – MANUAL DE UTILIZADOR

Este manual pretende auxiliar o utilizador a interagir com a plataforma SmartGeno, descrevendo os passos necessários para elaborar as suas principais funcionalidades:

- Treinar modelo de *machine learning*;
- Testar modelo de *machine learning* previamente armazenado

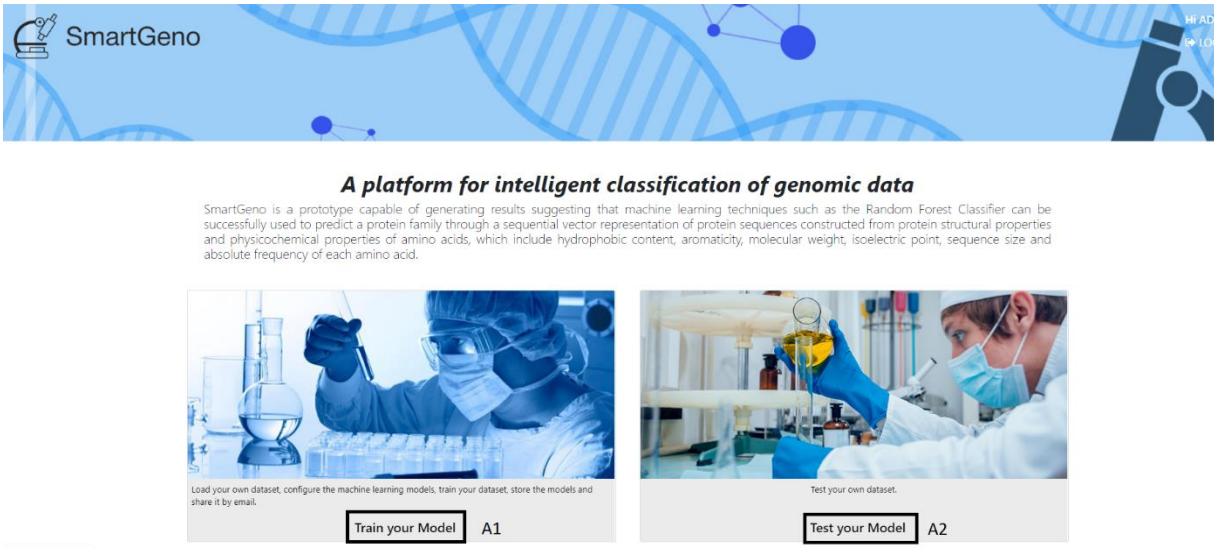

Figura 34 Vista da página principal do SmartGeno

Legenda:

- A1 – Treinar Modelo de *machine learning*;
- A2 – Testar Modelo de *machine learning* previamente armazenado

Funcionalidade "Treinar modelo de *machine Learning*"

Para treinar um modelo de *machine learning* o utilizador deve deslocar-se à página principal e clicar em A1 - "Train your Model". Se o utilizador ainda não estiver autenticado o sistema irá redirecioná-lo para a página de *Login* ou, em caso contrário, apresenta uma página onde o utilizador deverá selecionar qual o *dataset* (pertencente a um organismo) que pretende treinar, tal como apresentado na Figura 35.

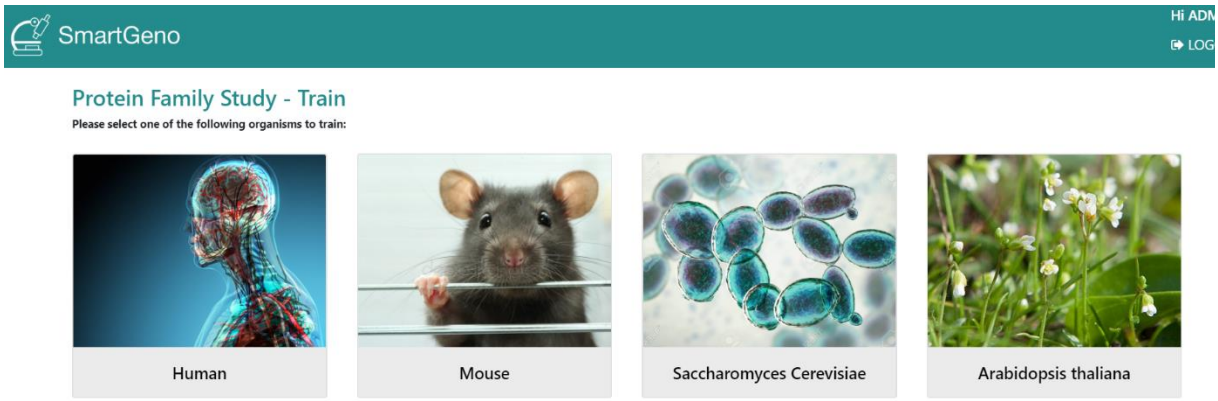

Figura 35 Treinar modelo de *machine learning* - Selecionar organismo / *dataset*

Tal como demonstrado na Figura 36, de seguida o sistema irá abrir uma janela onde o utilizador necessita de passar por vários passos até configurar por completo o treino do modelo de *machine learning*. Em cada passo o sistema validará se o utilizador preencheu corretamente os campos do formulário e é possível voltar ao passo anterior ou vice-versa.

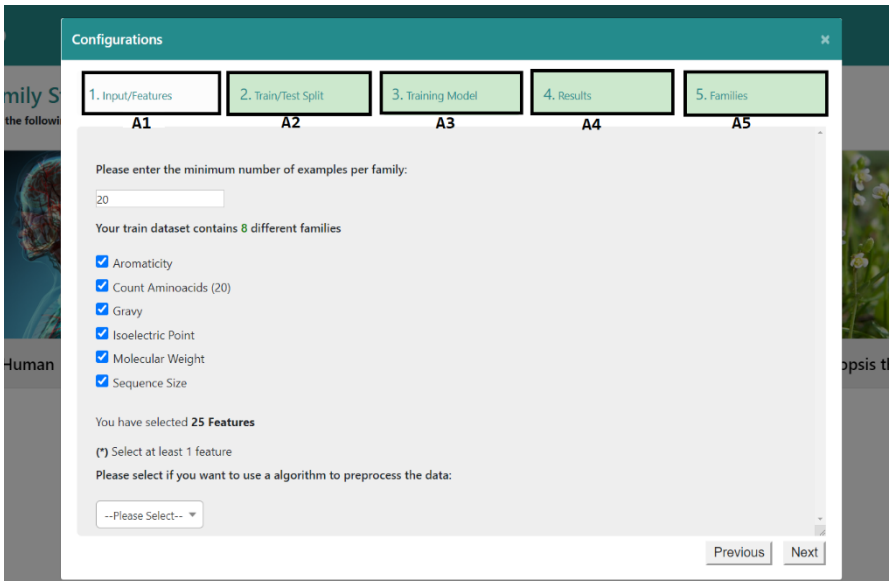

Figura 36 Treinar modelo de *machine learning* - Configurações

Legenda:

- A1 – Configuração das *features* e das técnicas de processamento a utilizar (se aplicável);
- A2 – Configuração da divisão do *dataset* num conjunto de treino e teste;
- A3 – Configuração dos hiperparâmetros do modelo de *machine learning*;
- A4 – Configuração dos relatórios (imagens ilustrativas) com os resultados do treino do modelo de *machine learning* (matriz confusão, *report* de classificação, balanceamento das classes e curva de ROC);
- A5 – Configuração das famílias a serem usadas para treinar o modelo de *machine learning*.

Após o utilizador realizar as configurações necessárias o sistema irá iniciar o treino do modelo. Quando terminar, uma página com os relatórios configurados pelo utilizador será apresentada. Por defeito todos os relatórios são apresentados, no entanto, o utilizador durante a configuração pode selecionar que deseja apenas visualizar a matriz de confusão por exemplo.

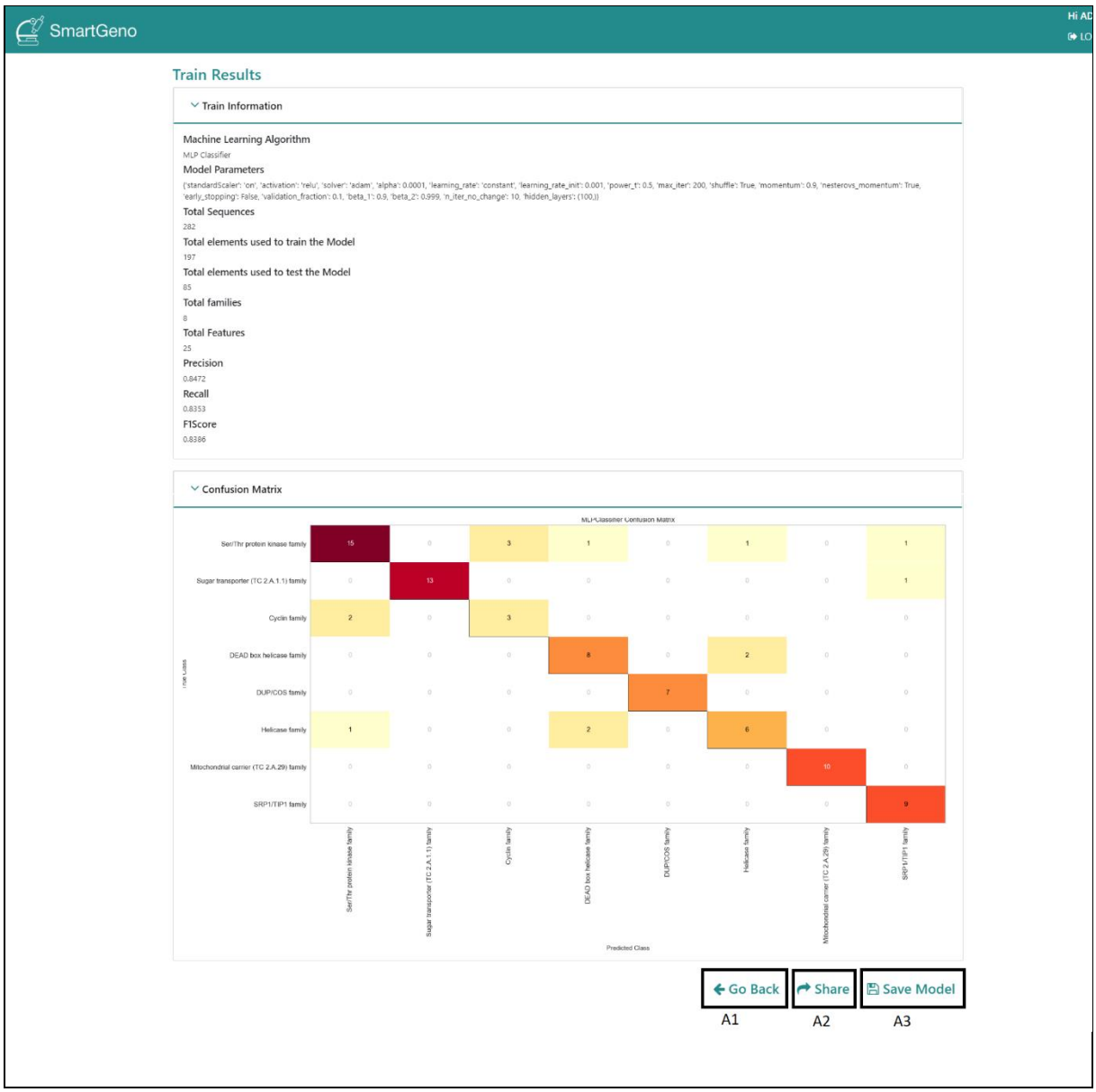

Figura 37 Treinar modelo de *machine learning* - Resultados

Legenda:

- A1 - Voltar atrás;
- A2 – Partilhar resultados por *email*;
- A3 – Armazenar modelo de *machine learning*

Tal como apresentado na Figura 37, o utilizador tem 3 opções ao seu dispor. Caso opte por A2 – “Share”, será apresentada uma janela com o aspeto da Figura 38.

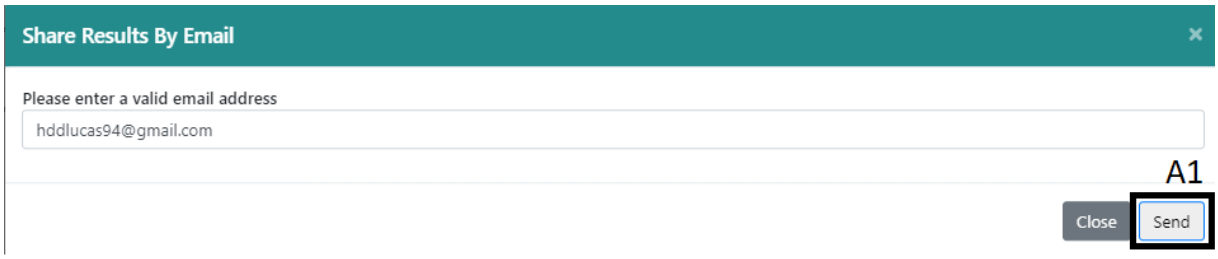

Figura 38 Treinar modelo de *machine learning* – Partilha dos resultados por *email*

Após preencher o *email* do destinatário e clicar em A1 – "Send", o utilizador irá receber na caixa do correio do *email* indicado os resultados do treino do modelo de *machine learning*, tal como demonstrado na Figura 39.

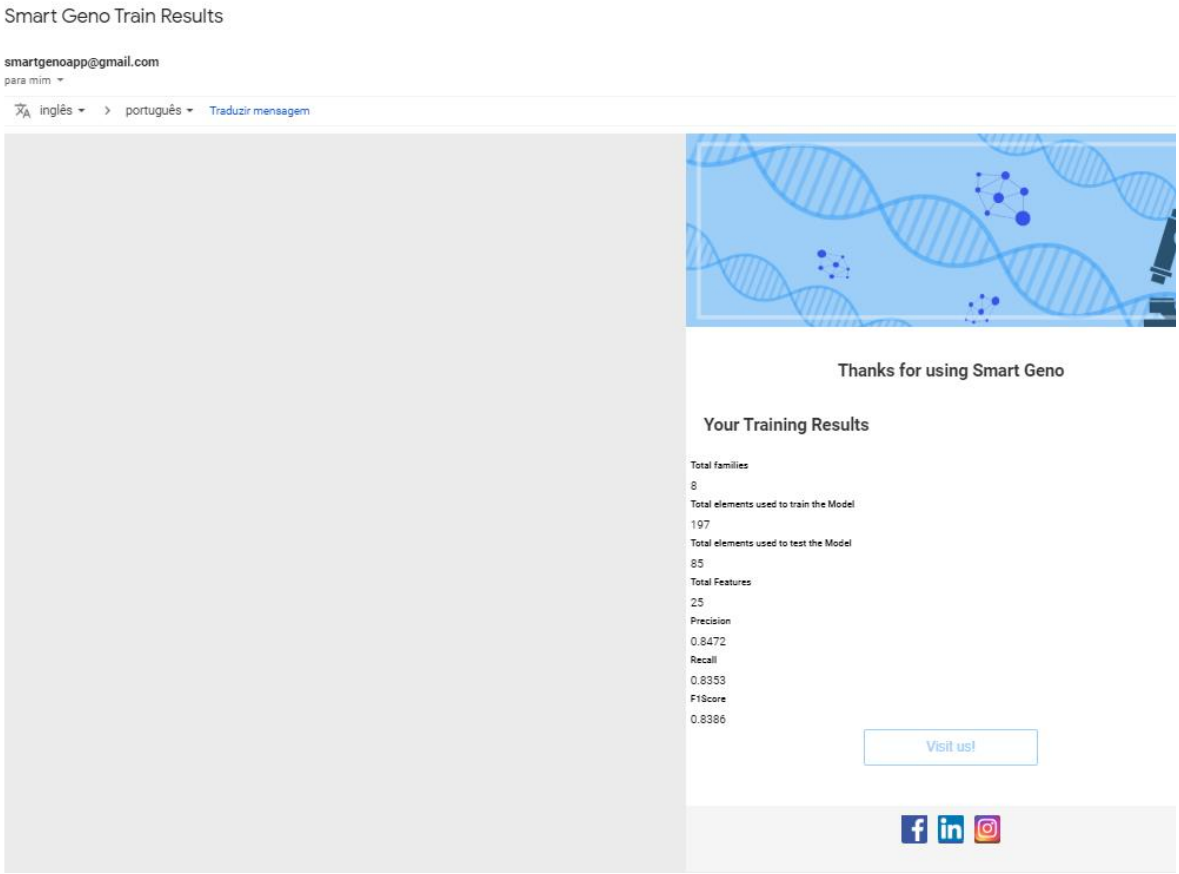

Figura 39 Treinar modelo de *machine learning* - Exemplo de *email* com os resultados

Se após receber os resultados do treino do modelo de *machine learning* o utilizador optar pela opção "Save Model", o sistema apresentará uma nova janela onde o utilizador deverá dar um nome ao modelo e defini-lo como público (disponível aos outros utilizadores) ou privado, tal como demonstrado na Figura 40.

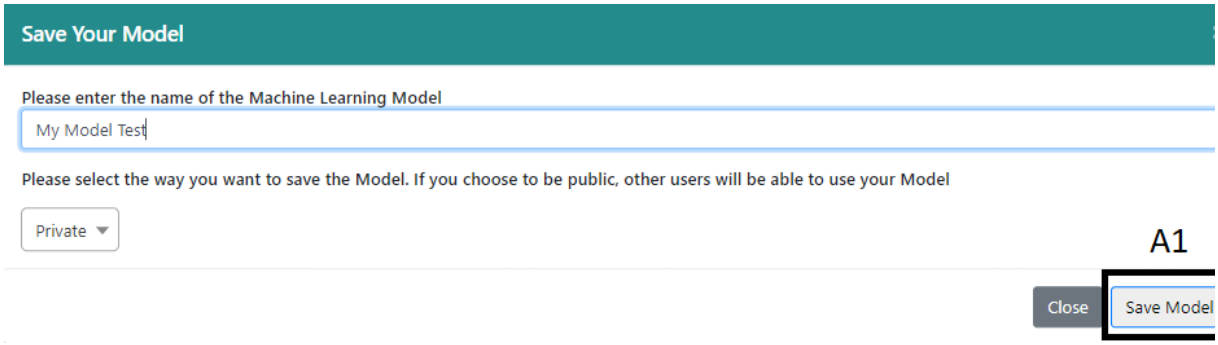

Figura 40 Treinar modelo de *machine learning* - Armazenar modelo

Após clicar em "Save Model" o sistema apresentará uma mensagem ao utilizador informando-o se modelo foi armazenado ou não com sucesso.

Funcionalidade "Testar modelo de *machine learning* previamente armazenado"

Para testar um modelo de *machine learning* previamente armazenado o utilizador deve deslocar-se à página principal e clicar em "Test your Model". Se o utilizador ainda não estiver autenticado o sistema irá redirecioná-lo para a página de *login* ou, em caso contrário, será apresentada uma página onde o utilizador poderá iniciar (apenas se e só se existir um modelo público armazenado por outro utilizador ou se o utilizador tiver armazenado pelo menos armazenado um modelo) o seu teste carregando um *dataset* (ficheiro .FASTA) com um conjunto de sequências proteicas à sua escolha. Após carregamento do *dataset* no servidor o sistema irá validar o mesmo, informando o utilizador se é ou não possível prosseguir com o estudo, tal como demonstrado na Figura 41.

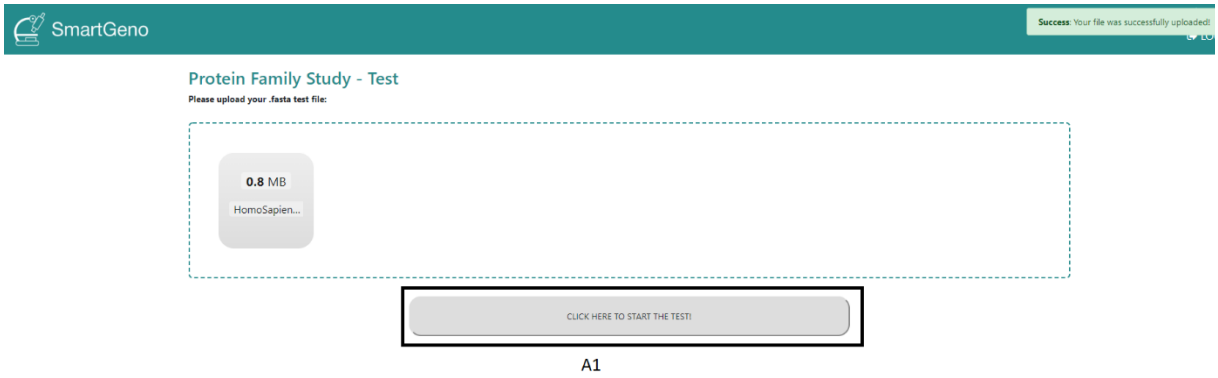

Figura 41 Testar modelo de *machine learning* - *Upload* ficheiro .FASTA

Clicando em A1 – "Click here to start the test!" o sistema irá abrir uma janela onde o utilizador necessitará de passar por alguns passos até configurar por completo o teste do modelo de *machine learning* previamente armazenado, tal como demonstrado na Figura 42.

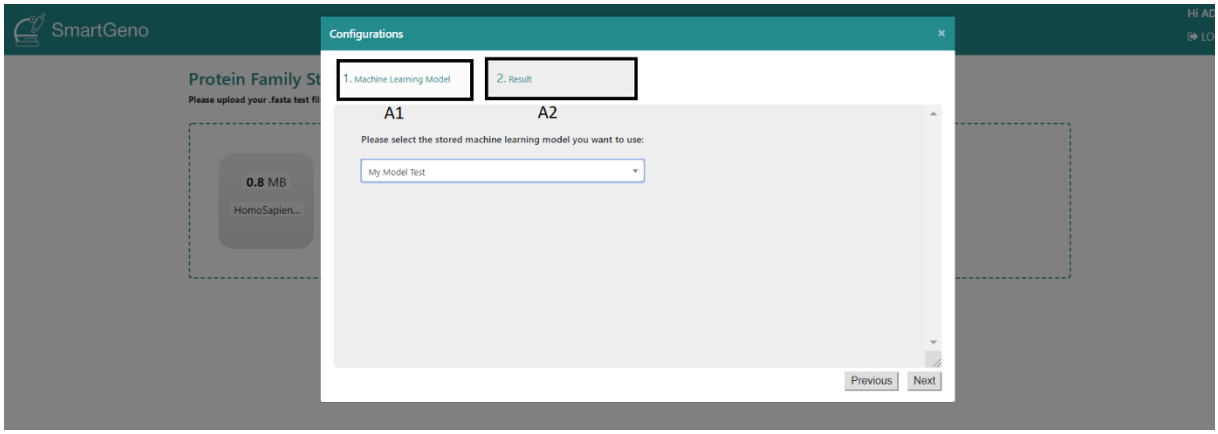

Figura 42 Testar modelo de *machine learning* - Configurações

Legenda:

- A1 - Seleção do modelo de *machine learning* previamente armazenado;
- A2 - Configuração do tipo de ficheiro que utilizador pretende observar resultados do teste do modelo de *machine learning* (ex.: .TXT,.JSON, .XML etc.)

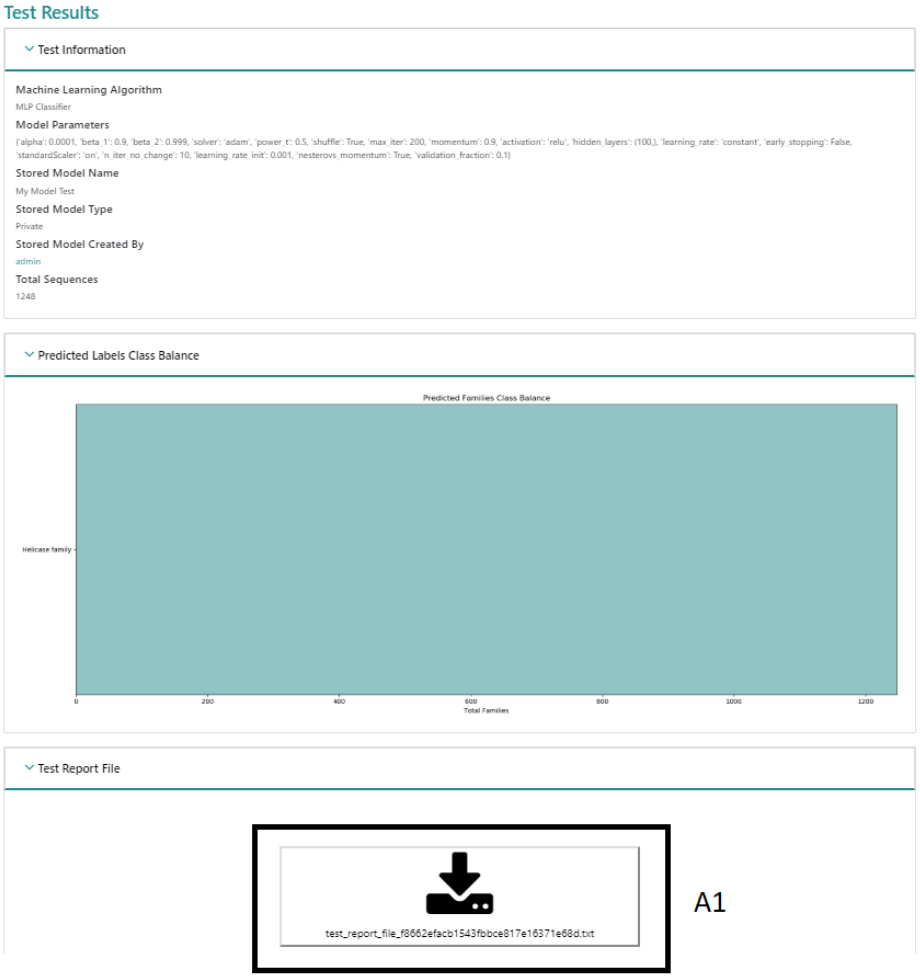

Figura 43 Testar modelo de *machine learning* – Resultados

Após o utilizador realizar as configurações necessárias o sistema irá iniciar o teste do modelo e quando terminar uma página com os resultados do teste é apresentada.

Para descarregar o ficheiro com os resultados o utilizador apenas necessita de carregar no botão A1 (Figura 43) para iniciar o *download* do ficheiro do servidor.

ANEXO B – MODELO DE DADOS

O SmartGeno contém as seguintes tabelas de base de dados:

- django_migrations;
- study;
- study_config;
- study_model;
- study_result;
- study_media;
- study_audit;
- test_study;
- test_study_audit;
- email_audit;
- file_uploads;
- organisms;
- user;
- protein_features;
- train_reports;
- test_reports;
- pre_processing_algorithms;
- machine_learning_models

De seguida, para cada uma das tabelas de base de dados acima listadas é apresentada uma descrição e são listados todos os seus atributos. Para cada um dos seus atributos é identificado o tipo de dados a que pertence, se é ou não nulo, algumas observações relevantes e também uma descrição.

Tabela "Django_migrations"

O Django cria automaticamente esta tabela na base de dados na primeira vez em que uma migração é aplicada. Para cada migração aplicada uma nova linha é inserida nesta tabela.

Tabela 28 Descrição detalhada da tabela migrations

Nome do atributo	*Tipo de dados*	*Nulo*	*Observações*	*Descrição*
id	int4	Não	Chave primária	Identificador único da tabela
app	varchar(255)	Não	-	Nome da aplicação onde a migração é utilizada
name	varchar(255)	Não	-	Nome da migração
applied	timestamp	Não	-	Data em que a migração foi aplicada

Tabela "study"

Cada estudo que um utilizador realiza na plataforma quando treina um modelo de *machine learning* é representado por uma linha nesta tabela.

Tabela 29 Descrição detalhada da tabela study

Nome do atributo	*Tipo de dados*	*Nulo*	*Observações*	*Descrição*
id	int4	Não	Chave Primária	Identificador único da tabela
dataset	text	Não	-	*Dataset* utilizado
train_elements	text	Não	-	*Features* utilizadas no conjunto de treino
train_targets	varchar[]	Não	-	*Targets* do conjunto de treino
total_sequences	int4	Não	-	Total de sequências no conjunto de treino
organism_id	int4	Não	Chave Estrangeira (organisms)	Identificador do organismo utilizado
user_id	int4	Não	Chave Estrangeira (user)	Identificador do utilizador autenticado

Tabela "study_config"

Contém as configurações do treino de um modelo de *machine learning.*

Tabela 30 Descrição detalhada da tabela study_config

Nome do atributo	*Tipo de dados*	*Nulo*	*Observações*	*Descrição*
id	int4	Não	Chave Primária	Identificador único da tabela
minimum_examples_per_family	int4	Não	-	Número mínimo de exemplos de sequências proteicas por família
total_families	int4	Não	-	Total de famílias no conjunto de treino
features	varchar[]	Não	-	Lista de *features* utilizadas para treinar o modelo de *machine learning (*ex.: aromaticidade, ponto isoelétrico)
pca_n_components	int4	Sim	-	Número de Componentes do PCA
select_k_best_k	int4	Sim	-	Número de melhores *features* a serem utilizadas
train_percentage	int4	Não	-	Percentagem de elementos do conjunto de treino
test_percentage	int4	Não	-	Percentagem de elementos do conjunto de teste
stratify	bool	Não	-	*Flag* que indica se os dados são para serem divididos de forma estratificada, usando-os como rótulos de classe
random_state	bool	Não	-	*Flag* que indica se os dados do conjunto de treino/teste são para dividir de forma aleatória
random_state_percentage	int4	Sim	-	Gerador de número aleatórios
model_parameters	jsonb	Não	-	Parâmetros do modelo de *machine learning* utilizado
reports	varchar[]	Não	-	Configuração com a lista dos relatórios utilizados para apresentar os resultados
standard_scaler	bool	Não	-	*Flag* que indica se o algoritmo de pré-processamento StandardScaler [107] deve ser aplicado
families	varchar[]	Não	-	Configuração com a lista de famílias a usadas para treinar o modelo

machine_learning_model_id	int4	Não	Chave Estrangeira (machine_learning_models)	Identificador do modelo de *machine learning* utilizado
pre_processing_algorithm_id	int4	Sim	Chave Estrangeira (pre_processing_algorithms)	Identificador do algoritmo de pré-processamento utilizado
study_id	int4	Não	Chave Estrangeira (study)	Identificador do estudo

Tabela "study_model"

Contém os modelos de *machine learning* armazenados pelos utilizadores. Estes modelos podem ser posteriormente utilizados para predizer as famílias de um conjunto de sequências proteicas armazenadas num ficheiro. Os modelos armazenados podem ser públicos (disponíveis a qualquer utilizador) ou privados.

Tabela 31 Descrição detalhada da tabela study_model

Nome do atributo	***Tipo de dados***	***Nulo***	***Observações***	***Descrição***
id	int4	Não	Chave Primária	Identificador único da tabela
name	varchar(255)	Não	-	Nome do modelo
public	bool	Não	-	*Flag* que indica se o modelo armazenado é público ou privado
create_time	timestamp	Não	-	Data de criação
study_id	int4	Não	Chave Estrangeira (study)	Identificador do estudo
user_id	int4	Não	Chave Estrangeira (user)	Identificador do utilizador autenticado

Tabela "study_result"

Contém os resultados utilizados para medir a performance do modelo de *machine learning* utilizado.

Tabela 32 Descrição detalhada da tabela study_result

Nome do atributo	*Tipo de dados*	*Nulo*	*Observações*	*Descrição*
id	int4	Não	Chave Primária	Identificador único da tabela
precision	float8	Não	-	Precisão do modelo de *machine learning* utilizado
recall	float8	Não	-	*Recall* do modelo de *machine learning* utilizado
f1_score	float8	Não	-	F1-Score do modelo de *machine learning* utilizado
accuracy_score	Float8	Não	-	Accuracy do modelo de *machine learning* utilizado
auc_score	Float8	Não	-	AUC do modelo de *machine learning* utilizado
average_type	varchar(255)	Não	-	Determina o tipo de média a usar como métrica de avaliação do modelo (*macro-average* ou *micro-average*)
true_positive	int[]	Não	-	Verdadeiros positivos para cada classe
true_negative	int[]	Não	-	Verdadeiros negativos para cada classe
false_positive	int[]	Não	-	Falsos positivos para cada classe
false_negative	int[]	Não	-	Falsos negativos para cada classe
study_id	int4	Não	Chave Estrangeira (study)	Identificador do estudo

Tabela "study_media"

Contém os relatórios (imagens ilustrativas) com os resultados do treino do modelo de *machine learning* (ex.: matriz confusão, relatório de classificação, etc.).

Tabela 33 Descrição detalhada da tabela study_media

Nome do atributo	*Tipo de dados*	*Nulo*	*Observações*	*Descrição*
id	int4	Não	Chave Primária	Identificador único da tabela
image	bytea	Não	-	Imagem no formato base 64
study_id	int4	Não	Chave Estrangeira (study)	Identificador do estudo
train_report_id	int4	Não	Chave Estrangeira (train_reports)	Identificador do relatório

Tabela "study_audit"

Regista a auditoria relativamente ao treino de um modelo de *machine learning*, indicando se o mesmo foi ou não efetuado com sucesso.

Tabela 34 Descrição detalhada da tabela study_audit

Nome do atributo	*Tipo de dados*	*Nulo*	*Observações*	*Descrição*
id	int4	Não	Chave Primária	Identificador único da tabela
http_status	varchar(255)	Não	-	Indica se um pedido HTTP específico foi efetuado com sucesso.
error_description	text	Sim	-	Contém uma descrição do erro caso o pedido HTTP não tenha sido efetuado com sucesso
in_timestamp	timestamp	Não	-	Regista o tempo em que o estudo foi iniciado
out_timestamp	timestamp	Não	-	Regista o tempo em que o estudo foi concluído
study_id	int4	Não	Chave Estrangeira (study)	Identificador do estudo

Tabela “test_study”

Cada estudo que um utilizador realiza na plataforma quando testa um modelo de *machine learning* previamente armazenado é representado por uma linha nesta tabela.

Tabela 35 Descrição detalhada da tabela test_study

Nome do atributo	***Tipo de dados***	***Nulo***	***Observações***	***Descrição***
id	int4	Não	Chave primária	Identificador único da tabela
test_elements	text	Não	-	*Dataset* utilizado (após codificado através de uma representação vetorial)
y_pred	varchar[]	Não	-	*Labels* do conjunto de teste
total_sequences	int4	Não	-	Total de famílias no conjunto de teste
image	bytea	Não	-	Imagem com o total de sequências por família
study_model_id	int4	Não	Chave Estrangeira (study_models)	Identificador do modelo de *machine learning* armazenado
report_type_id	int4	Não	Chave Estrangeira (report_type)	Identificador do relatório utilizado para apresentar os resultados
test_file_id	int4	Não	Chave Estrangeira (file_uploads)	Identificador do ficheiro que contém o *dataset* utilizado para treino
test_report_file_id	int4	Não	Chave Estrangeira (file_uploads)	Identificador do ficheiro que contém os resultados
user_id	int4	Não	Chave Estrangeira (user)	Identificador do utilizador autenticado

Tabela “test_study_audit”

Regista a auditoria relativamente ao teste de um modelo de machine learning previamente armazenado, indicando se o mesmo foi ou não efetuado com sucesso.

Tabela 36 Descrição detalhada da tabela test_study_audit

Nome do atributo	*Tipo de dados*	*Nulo*	*Observações*	*Descrição*
id	int4	Não	Chave primária	Identificador único da tabela
http_status	varchar(255)	Não	-	Indica se um pedido HTTP específico foi efetuado com sucesso.
error_description	text	Sim	-	Contém uma descrição do erro caso o pedido HTTP não tenha sido efetuado com sucesso
in_timestamp	timestamp	Não	-	Regista o tempo em que o estudo é iniciado
out_timestamp	timestamp	Não	-	Regista o tempo em que o estudo é concluído
test_study_id	int4	Não	Chave Estrangeira (test_study)	Identificador do estudo

Tabela "file_uploads"

Contém a lista de ficheiro armazenados no servidor, servindo essencialmente para controlar o *upload* e *download* de ficheiros.

Tabela 37 Descrição detalhada da tabela file_uploads

Nome do atributo	*Tipo de dados*	*Nulo*	*Observações*	*Descrição*
id	int4	Não	Chave primária	Identificador único da tabela
upload	varchar(100)	Não	-	Caminho para a pasta no servidor onde o ficheiro é armazenado
uploaded_at	timestamp	Não	-	Data em que o ficheiro é armazenado no servidor

Tabela "email_audit"

Regista a informação relativa a qualquer *email* enviado através da plataforma para um servidor de *email* externo.

Tabela 38 Descrição detalhada da tabela email_audit

Nome do atributo	*Tipo de dados*	*Nulo*	*Observações*	*Descrição*
id	int4	Não	Chave primária	Identificador único da tabela
subject	varchar(255)	Não	-	Assunto
text_content	text	Não	-	Conteúdo do *email* (formato texto)
html_content	text	Não	-	Conteúdo do *email* (formato HTML)
email_from	varchar(255)	Não	-	Remetente
email_to	varchar(255)	Não	-	Destinatário
send_date	timestamp	Não	-	Data de envio
user_id	int4	Sim	Chave Estrangeira (user)	Identificador do utilizador autenticado

Tabela "organisms"

Contém a lista de organismos no sistema que têm um *dataset* de sequências proteicas disponíveis para serem usadas como conjunto de treino.

Tabela 39 Descrição detalhada da tabela organisms

Nome do atributo	*Tipo de dados*	*Nulo*	*Observações*	*Descrição*
id	int4	Não	Chave primária	Identificador único da tabela
name	varchar(255)	Não	-	Nome do organismo
value	varchar(255)	Não	-	Identificador do organismo
image_path	varchar(255)	Não	-	Caminho para a pasta no servidor onde a imagem do organismo está armazenada
image_description	varchar(255)	Não	-	Descrição da imagem
active	bool	Não	-	*Flag* que indica se o organismo está ou não disponível

dataset_original_path	varchar(255)	Não	-	Caminho para a pasta no servidor onde o *dataset* com as sequências proteicas do organismo está armazenado
dataset_transformed_path	varchar(255)	Não	-	Caminho para a pasta no servidor onde o *dataset* codificado (através de uma representação vetorial) com as sequências proteicas do organismo está armazenado

Tabela "user"

Contém a lista de utilizadores registados no sistema.

Tabela 40 Descrição detalhada da tabela user

Nome do atributo	*Tipo de dados*	*Nulo*	*Observações*	*Descrição*
id	int4	Não	Chave primária	Identificador único da tabela
password	varchar(128)	Não	-	Password calculada com o algoritmo PBKDF2, que utiliza um *hash* SHA256
last_login	timestamp	Sim	-	Data do último login
is_superuser	bool	Não	-	*Flag* que indica se o utilizador é administrador (beneficia do privilégio máximo)
username	varchar(150)	Não	-	Nome de utilizador
first_name	varchar(30)	Não	-	Primeiro nome
last_name	varchar(150)	Não	-	Último nome
email	varchar(254)	Não	-	*Email*
is_active	bool	Não	-	*Flag* que indica se o utilizador está ativo
date_joined	timestamp	Não	-	Data em que o utilizador se registou no sistema

Tabela "protein_features"

Contém a lista de *features* que podem ser usadas para treino de um modelo de *machine learning* (ex.: aromaticidade, ponto isoelétrico, etc.).

Tabela 41 Descrição detalhada da tabela protein_features

Nome do atributo	*Tipo de dados*	*Nulo*	*Observações*	*Descrição*
id	int4	Não	Chave primária	Identificador único da tabela
name	varchar(255)	Não	-	Nome da *feature*
value	varchar(255)	Não	-	Identificador da *feature* em formato texto
description	text	Sim	-	Descrição da *feature*
active	bool	Não		*Flag* que indica se a *feature* está ativa

Tabela "train_reports"

Contém a lista de relatórios (imagens ilustrativas) disponíveis para o utilizador poder observar resultados do treino do modelo de *machine learning* (ex.: matriz confusão, relatório de classificação, curva de ROC, etc.).

Tabela 42 Descrição detalhada da tabela train_reports

Nome do atributo	*Tipo de dados*	*Nulo*	*Observações*	*Descrição*
id	int4	Não	Chave primária	Identificador único da tabela
name	varchar(255)	Não	-	Nome do relatório
value	varchar(255)	Não	-	Identificador do relatório em formato texto
description	text	Sim	-	Descrição do relatório
active	bool	Não	-	*Flag* que indica se o relatório está ativo

Tabela "test_reports"

Contém a lista de tipos de ficheiro disponíveis para o utilizador poder observar resultados do teste do modelo de *machine learning* (ex.: .TXT, .JSON, .XML etc.).

Tabela 43 Descrição detalhada da tabela test_reports

Nome do atributo	*Tipo de dados*	*Nulo*	*Observações*	*Descrição*
id	int4	Não	Chave primária	Identificador único da tabela
name	varchar(255)	Não	-	Nome do relatório
value	varchar(255)	Não	-	Identificador do relatório em formato texto
description	text	Sim	-	Descrição do relatório
active	bool	Não	-	*Flag* que indica se o relatório está ativo

Tabela "pre_processing_algorithms"

Contém a lista de algoritmos de pré-processamento (ex.: análise de componentes principais, etc.).

Tabela 44 Descrição detalhada da tabela pre_processing_algorithms

Nome do atributo	*Tipo de dados*	*Nulo*	*Observações*	*Descrição*
id	int4	Não	Chave primária	Identificador único da tabela
name	varchar(255)	Não	-	Nome do algoritmo de pré-processamento
value	varchar(255)	Não	-	Identificador do algoritmo de pré-processamento em formato texto
description	text	Sim	-	Descrição do algoritmo de pré-processamento
active	bool	Não	-	*Flag* que indica se o algoritmo de pré-processamento está ativo

Tabela "machine_learning_models"

Contém a lista de algoritmos de modelos de *machine learning* que podem ser utilizados para treinar as sequências proteicas (ex.: rede neuronal MLP, árvore de decisão, etc.).

Tabela 45 Descrição detalhada da tabela machine_learning_models

Nome do atributo	***Tipo de dados***	***Nulo***	***Observações***	***Descrição***
id	int4	Não	Chave primária	Identificador único da tabela
name	varchar(255)	Não	-	Nome do modelo de *machine learning*
value	varchar(255)	Não	-	Identificador do modelo de *machine learning* em formato texto
description	text	Sim	-	Descrição do modelo de *machine learning*
active	bool	Não	-	*Flag* que indica se o modelo de *machine learning* está ativo

ANEXO C – ANÁLISE DE REQUISITOS

O SmartGeno contém as seguintes funcionalidades:

- Fazer autenticação;
- Registar utilizador;
- Restabelecer *password*;
- Treinar modelo de *machine learning*;
- Partilhar resultados do treino do modelo de *machine learning*;
- Armazenar modelo de *machine learning*;
- Consultar histórico de resultados do treino de modelo de *machine learning*;
- Testar modelo de *machine learning* previamente armazenado

De seguida, para cada uma das funcionalidades listadas acima é apresentada a sua descrição e prioridade, estímulos e sequência de resposta e os requisitos funcionais. Posteriormente são apresentados os requisitos de interfaces externas e os requisitos não funcionais.

Funcionalidade "Fazer autenticação"

Descrição e prioridade

O Sistema apresenta uma página de autenticação onde o utilizador deve preencher as suas credenciais (*username* e *password*).

Prioridade = Alta

Estímulos e Sequência de Resposta

Estímulo: O utilizador não autenticado clica em "Login" ou tenta aceder a uma página onde a autenticação é obrigatória.
Resposta: O sistema apresenta o formulário de autenticação.

Estímulo: O utilizador introduz as credenciais e submete o pedido de autenticação.
Resposta: O sistema invoca a base de dados e confirma os dados.

Estímulo: O utilizador faz autenticação com sucesso.
Resposta: O sistema redireciona o utilizador para a página principal.

Estímulo: O utilizador faz autenticação sem sucesso.
Resposta: O sistema mantém-se na mesma página e apresenta uma mensagem de erro.

Requisitos Funcionais

REQ.1: fazLogin – O sistema apresenta o ecrã de *login* do sistema mostrando um formulário com os campos *username* e *password*.

REQ.2: fazLogin.ValidaAutenticação – O utilizador submete o formulário de autenticação e o sistema verifica se as credenciais introduzidas estão corretas. Desencadeia de seguida uma autenticação com sucesso ou não.

REQ.3: fazLogin.SemSucesso – O sistema apresenta uma mensagem de erro ao utilizador em caso de falha de autenticação.

REQ.4: fazLogin.ComSucesso – O sistema faz *login* com sucesso, envia um pedido à base de dados para registar a data do último login do utilizador e de seguida redireciona o utilizador para a página principal.

Funcionalidade "Registar utilizador"

Descrição e prioridade

Permite registar um novo utilizador na plataforma SmartGeno.

Prioridade = Alta

Estímulos e Sequência de Resposta

Estímulo: O utilizador clica em "Register".
Resposta: O sistema apresenta o formulário de registo.

Estímulo: O utilizador submete o formulário de registo.
Resposta: O sistema envia um pedido à base de dados e confirma as informações do utilizador.

Estímulo: O utilizador faz registo com sucesso.
Resposta: O sistema redireciona o utilizador para a página de autenticação.

Estímulo: O utilizador faz registo sem sucesso.
Resposta: O sistema mantém-se na mesma página e apresenta uma mensagem de erro.

Requisitos Funcionais

REQ.1: registaUtilizador – O sistema apresenta um formulário de registo contendo um conjunto de campos a serem preenchidos pelo utilizador.

REQ.2: registaUtilizador.ValidaRegisto – O utilizador submete o formulário de registo e o sistema valida as informações do novo utilizador. Desencadeia de seguida um registo com sucesso ou não.

REQ.3: registaUtilizador.ComSucesso – O sistema valida o formulário de registo com sucesso, envia um pedido à base de dados e regista a informação do novo utilizador e de seguida redireciona o utilizador para a página de autenticação.

REQ.4: registaUtilizador.SemSucesso – O sistema apresenta uma mensagem de erro ao utilizador quando ocorre um erro ao validar o formulário de registo.

Funcionalidade "Restabelecer *password*"

Descrição e prioridade

Permite que o utilizador recupere o acesso ao sistema em caso de perda das suas credenciais.

Prioridade = Média

Estímulos e Sequência de Resposta

Estímulo: O utilizador clica em "*Reset Password*".
Resposta: O sistema apresenta o formulário para restabelecer *password*.

Estímulo: O utilizador submete o formulário de restabelecimento da *password*.
Resposta: O sistema envia um *email* ao utilizador com as instruções necessárias para restabelecer *password*.

Estímulo: O utilizador segue as instruções do *email* clicando no *link* para definir a nova password.
Resposta: O sistema identifica o utilizador e apresenta um formulário onde o utilizador deve definir a sua nova *password*.

Estímulo: O utilizador submete a nova *password*.
Resposta: O sistema envia um pedido à base de dados e confirma as informações do utilizador.

Estímulo: O utilizador edita a *password* com sucesso.
Resposta: O sistema redireciona o utilizador para uma nova página.

Estímulo: O utilizador edita a *password* sem sucesso.
Resposta: O sistema mantém-se na mesma página e apresenta uma mensagem de erro.

Requisitos Funcionais

REQ.1: restabelecerPassword – O sistema apresenta um formulário com um campo *email* - o destinatário para onde é enviado o *email* com as instruções necessárias para restabelecer a *password*.

REQ.2: restabelecerPassword.EmailRestabelecimento – O sistema envia um *email* ao utilizador com as instruções a seguir para restabelecer a sua *password*.

REQ.3: restabelecerPassword.LinkRestabelecimento – O utilizador clica no *link* do *email* de restabelecimento de *password*, o sistema identifica o utilizador e redireciona-o para o formulário onde poderá definir a sua nova *password*.

REQ.4: restabelecerPassword.ComSucesso – O sistema valida o formulário de edição de password com sucesso e redireciona o utilizador para uma nova página com uma mensagem de sucesso "Restabelecimento da *password* feito com sucesso"

REQ.4: registaUtilizador.SemSucesso – O sistema apresenta uma mensagem de erro ao utilizador quando ocorre um erro ao validar o formulário de edição da *password*.

Funcionalidade "Treinar modelo de *machine learning*"

Descrição e prioridade

Permite que o utilizador treine um modelo de *machine learning* usando um dos *datasets* disponíveis no sistema, cada um deles associado a um diferente organismo.

Prioridade = Alta

Estímulos e Sequência de Resposta

Estímulo: O utilizador clica em "Train your Model".
Resposta: O sistema apresenta a lista de *datasets* disponíveis.

Estímulo: O utilizador seleciona o *dataset*.
Resposta: O sistema apresenta uma janela com um conjunto de passos a seguir para configurar o treino do *dataset*.

Estímulo: O utilizador configura o treino do *dataset* com sucesso.
Resposta: O sistema armazena na base de dados as configurações e os resultados do treino e apresenta uma nova página com os resultados do mesmo.

Estímulo: O utilizador configura o treino *dataset* sem sucesso.
Resposta: O sistema mantém-se na mesma página e não deixa o utilizador avançar para o passo seguinte da configuração ou, em caso de o utilizador estar no último passo, não o deixa submeter as configurações.

Requisitos Funcionais

REQ.1: treinarModelo – O sistema apresenta uma janela com a lista de *datasets* disponíveis.

REQ.2: treinarModelo.ConfigurarTreino – O sistema apresenta uma janela de configurações com 4 passos. O utilizador só poderá submeter as configurações depois de todos os passos serem concluídos com sucesso e pode voltar atrás ou avançar a qualquer momento, desde que não existam erros no preenchimento das configurações em cada passo.

REQ.3: treinarModelo.ConfigurarTreino.ConfigurarFeatures – O sistema apresenta as configurações das *features* e das técnicas de processamento que podem ser aplicadas ao *dataset* original.

REQ.4: treinarModelo.ConfigurarTreino.ConfigurarDivisãoDataset – O sistema apresenta as configurações da divisão do *dataset* num conjunto de treino/teste.

REQ.5: treinarModelo.ConfigurarTreino.ConfigurarModeloMachineLearning – O sistema apresenta as configurações dos hiperparâmetros de cada modelo de *machine learning* disponível no sistema e possibilita que o utilizador use ou reponha as configurações por defeito. Cada configuração contém uma mensagem explicativa da função do hiperparâmetro.

REQ.6: treinarModelo.ConfigurarTreino.ConfigurarRelatórios – O sistema apresenta a lista de relatórios (matriz confusão, *report* de classificação, balanceamento das classes e curva de ROC) que o utilizador pode escolher para visualizar os resultados do treino, possibilitando a escolha de um ou mais relatórios. A curva de ROC apenas estará disponível se o número de classes for <=10.

REQ.7: treinarModelo.ConfigurarTreino.ConfiguraFamílias – O sistema apresenta a lista de famílias a serem usadas para treinar o modelo de *machine learning*. O utilizador deverá selecionar um número mínimo de 2 famílias, caso contrário o sistema deverá apresentar uma mensagem erro.

REQ.8: treinarModelo.SubmeteConfigurações – O sistema inicia o treino do modelo de *machine learning* e assim que termina apresenta os resultados numa nova página. Os resultados incluem os relatórios configurados pelo utilizador e as métricas F1-Score, *recall* e a precisão.

Funcionalidade "Partilhar resultados do treino do modelo de *machine learning*"

Descrição e prioridade

Permite que o utilizador partilhe por *email* os resultados to treino do modelo de *machine learning*.

Prioridade = Baixa

Estímulos e Sequência de Resposta

Estímulo: O utilizador realiza a funcionalidade "Treinar modelo de *machine learning*" e clica em "Share".
Resposta: O sistema apresenta o formulário de partilha de resultados.

Estímulo: O utilizador submete o formulário de partilha de resultados com sucesso.
Resposta: O sistema mantém-se na mesma página, armazena na base de dados a informação do *email* enviado e apresenta uma mensagem de sucesso.

Estímulo: O utilizador submete o formulário de partilha de resultados sem sucesso.
Resposta: O sistema mantém-se na mesma página e apresenta uma mensagem de erro.

Requisitos Funcionais

REQ.1: partilharResultados – O sistema apresenta uma janela com um formulário que contém um campo *email* - o destinatário para onde é enviado o *email* com os resultados do treino.

REQ.2: partilharResultados.SubmeteResultadosComSucesso – O sistema envia um *email* ao utilizador com os resultados do treino. O anexo do *email* contém os relatórios configurados no treino e inclui as métricas F1-Score, *recall*, AUC e a precisão.

REQ.3: partilharResultados.SubmeteResultadosSomSucesso – O sistema apresenta uma mensagem de erro informando o utilizador que ocorreu um erro ao enviar o *email* com os resultados do treino para o destinatário.

Funcionalidade "Armazenar modelo de *machine learning*"

Descrição e prioridade

Permite que o utilizador armazene um modelo de *machine learning* depois de treinado.

Prioridade = Alta

Estímulos e Sequência de Resposta

Estímulo: O utilizador realiza a funcionalidade "Treinar modelo de *machine learning*" ou "Consultar histórico de resultados do treino de modelo de *machine learning*" e clica em "Save Model".

Resposta: O sistema apresenta o formulário de armazenamento do modelo.

Estímulo: O utilizador submete o formulário de armazenamento do modelo com sucesso.

Resposta: O sistema mantém-se na mesma página, armazena na base de dados a informação do modelo e apresenta uma mensagem de sucesso.

Estímulo: O utilizador submete o formulário de armazenamento do modelo sem sucesso.

Resposta: O sistema mantém-se na mesma página e apresenta uma mensagem de erro.

Requisitos Funcionais

REQ.1: armazenaModelo – O sistema apresenta uma janela com um formulário que contém o nome do modelo a armazenar e uma configuração para indicar se o utilizador pretende guardá-lo como público ou privado.

REQ.2: armazenaModelo.ArmazenaModeloComSucesso – O sistema apresenta uma mensagem de sucesso e armazena a informação do modelo de modo a que possa ser usada posteriormente para predizer as famílias de um conjunto de sequências proteicas submetidas na plataforma através do *upload* de um ficheiro .FASTA.

REQ.3: armazenaModelo. ArmazenaModeloSemSucesso – O sistema apresenta uma mensagem de erro informando o utilizador que ocorreu um erro ao armazenar o modelo.

Funcionalidade "Consultar histórico de resultados do treino de modelo de *machine learning*"

Descrição e prioridade

Permite que o utilizador consulte o histórico de resultados do treino de um modelo de *machine learning*.

Prioridade = Baixa

Estímulos e Sequência de Resposta

Estímulo: O utilizador visita a sua página de perfil e clica em "Train Results"
Resposta: O sistema obtém da base de dados a lista de treinos efetuados e apresenta-a numa nova página usando uma tabela.

Estímulo: O utilizador clica no ícone que permite observar os detalhes dos resultados do treino.
Resposta: O sistema abre uma nova página e apresenta ao utilizador os resultados do treino do modelo de *machine learning*.

Requisitos Funcionais

REQ.1: consultarHistóricoResultadosTreino – O sistema apresenta uma tabela com a lista de treinos efetuados e disponibiliza alguma informação como o organismo que foi utilizado, o número de sequências utilizadas, o total de elementos do conjunto de treino, o total de elementos do conjunto de teste, o total de famílias, o total de *features*, o F1-Score e a data em que foi criado.

REQ.2: consultarHistóricoResultadosTreino.ConsultarDetalhes – O sistema apresenta os detalhes do treino do modelo de *machine learning* que devem incluir os relatórios configurados pelo utilizador e as métricas F1-Score, recall e a precisão.

Funcionalidade "Testar modelo de *machine learning* previamente armazenado"

Descrição e prioridade

Permite que o utilizador teste um modelo de *machine learning* previamente armazenado. O utilizar deverá escolher um modelo seu ou um modelo público de outro utilizador. O modelo será usado para predizer a família de um conjunto de sequências proteicas num ficheiro .FASTA que deve ser carregado no servidor.

Prioridade = Alta

Estímulos e Sequência de Resposta

Estímulo: O utilizador clica em "Test your Model".
Resposta: O sistema apresenta uma página com um formulário para *upload* de um ficheiro.

Estímulo: O utilizador faz upload de um ficheiro com sucesso.
Resposta: O sistema armazena na base de dados o ficheiro e valida o seu conteúdo.

Estímulo: O utilizador faz upload de um ficheiro sem sucesso.

Resposta: O sistema mantém-se na mesma página e apresenta uma mensagem de erro.

Estímulo: O utilizador faz *upload* de um ficheiro válido.
Resposta: O sistema adiciona um botão na página que permite que o utilizador inicie o teste do modelo de *machine learning*.

Estímulo: O utilizador faz *upload* de um ficheiro inválido.
Resposta: O sistema mantém-se na mesma página e apresenta uma mensagem de erro.

Estímulo: O utilizador clica em "Click here to start the test".
Resposta: O sistema apresenta uma janela com um conjunto de passos a seguir para configurar o teste do modelo.

Estímulo: O utilizador configura o teste do modelo com sucesso.
Resposta: O sistema armazena na base de dados um ficheiro com o resultado do teste e apresenta uma nova página com os resultados do mesmo.

Estímulo: O utilizador configura o teste do modelo sem sucesso.
Resposta: O sistema mantém-se na mesma página e não deixa o utilizador avançar para o passo seguinte da configuração ou, em caso de o utilizador estar no último passo, não o deixa submeter as configurações.

Requisitos Funcionais

REQ.1: testarModelo – O sistema apresenta uma janela com um formulário para *upload* de ficheiro. O formulário deve permitir *drag* & *drop* e deve suportar um único ficheiro até 50MB.

REQ.2: testarModelo.ConfigurarTeste – O sistema apresenta uma janela de configurações com 2 passos. O utilizador só poderá submeter as configurações depois de todos os passos serem concluídos com sucesso e pode voltar atrás ou avançar a qualquer momento, desde que não existam erros no preenchimento das configurações em cada passo.

REQ.3: testarModelo.ConfigurarTeste.SelecionarModelo – O sistema apresenta a lista dos modelos armazenados pelo utilizador ou os modelos públicos armazenados por outros utilizadores.

REQ.4: testarModelo.ConfigurarTeste.ConfiguraResultados– O sistema apresenta a lista de tipos de ficheiro que o utilizador pode escolher para observar os resultados do teste. O sistema deve permitir a geração de um único tipo de ficheiro com a extensão. JSON, .XML ou .TXT.

REQ.5: testarModelo.SubmeteConfigurações – O sistema inicia o teste do modelo de *machine learning* e assim que termina apresenta os resultados numa nova página. Os resultados incluem um gráfico com o total de sequências previstas por família e um ficheiro com os resultados que pode ser descarregado do servidor.

Funcionalidade "Consultar histórico de resultados do teste de modelo de *machine learning*"

Descrição e prioridade

Permite que o utilizador consulte o histórico de resultados do teste de um modelo de *machine learning* previamente armazenado.

Prioridade = Baixa

Estímulos e Sequência de Resposta

Estímulo: O utilizador visita a sua página de perfil e clica em "Test Results"
Resposta: O sistema obtém da base de dados a lista de testes efetuados e apresenta-a numa nova página usando uma tabela.

Estímulo: O utilizador clica no ícone que permite observar os detalhes dos resultados do teste.
Resposta: O sistema abre uma nova página e apresenta ao utilizador os resultados do teste do modelo de *machine learning*.

Requisitos Funcionais

REQ.1: consultarHistóricoResultadosTeste – O sistema apresenta uma tabela com a lista de teste efetuados e disponibiliza alguma informação como o nome do modelo armazenado, o tipo de modelo (público ou privado), o nome do utilizador que o criou, o número total de sequências no ficheiro carregado pelo utilizador e a data em que o teste foi feito.

REQ.2: consultarHistóricoResultadosTeste.ConsultarDetalhes – O sistema apresenta os detalhes do teste do modelo de *machine learning* que deve incluir o gráfico com o total de sequências previstas por família e o ficheiro com os resultados que pode ser descarregado do servidor.

Requisitos de Interfaces Externas

Interfaces de *hardware*

REQ-IH.1: Wireless ou rede móvel para a comunicação entre o cliente-servidor.

Interfaces de *software*

REQ-IS.1: A API usada para apresentação dos mapas fica a cargo da API Open Street Maps [58].

REQ-IS.2: A API usada para gestão de utilizadores fica a cargo da Django Rest Framework [129].

Requisitos Não Funcionais

Requisitos de Desempenho

REQ-D.1: As notificações de sucesso ou de erro na submissão de um formulário deverão ser enviadas imediatamente.

Requisitos de Segurança

REQ-S.1: Os dados dos utilizadores não serão fornecidos a entidades exteriores.

REQ-S.2: A comunicação entre o cliente-servidor deverá ser protegida.

REQ-S.3: Utilizador deverá estar autenticado para aceder às principais funcionalidades do sistema.

REQ-S.4: A *password* do utilizador deverá estar armazenada na base de dados na forma encriptada usando o algoritmo PBKDF2 com uma *hash* SHA256. Qualquer acesso à mesma nunca poderá resultar numa *string plain-text*.

REQ-S.5: O sistema apenas aceitará a *password* do utilizador se a mesma não for semelhante a uma lista de 20000 *passwords* comuns, se tiver um tamanho mínimo de 8 caracteres e se todos caracteres não forem numéricos.

Atributos de Qualidade de Software

REQ-AQS.1: A aplicação SmartGeno deverá estar sempre disponível a 100%.

REQ-AQS.2: A aplicação SmartGeno deverá ser robusta, portanto quando existe uma falha na comunicação entre o cliente e o servidor deverá ser mostrada a devida mensagem.

REQ-AQS.3: A portabilidade deve ser tida em conta, ou seja, a aplicação SmartGeno poderá ser usada por dispositivos com telas de ecrã de diferentes tamanhos ou com sistemas operativos diferentes.

ANEXO D – DJANGO *FRAMEWORK* – EXEMPLO PRÁTICO

Este anexo visa demonstrar os passos necessários para criar uma aplicação *web* que usa uma árvore de decisão para classificar o *dataset* iris [130] usando a Django *framework*, a biblioteca de *machine learning* scikit-learn, a biblioteca Yellowbrick e a base de dados PostgreSQL. Para treinar o classificador o utilizador deve indicar a percentagem de treino e teste e o sistema deve apresentar numa nova página os resultados que incluem as métricas precisão, *recall*, F1-Score, *accuracy*, AUC e a matriz confusão (descritas em 4.6). Cada estudo (classificação do *dataset*) e as respetivas configurações devem ser persistidas na base de dados bem como os resultados, usando o modelo de dados apresentado na Figura 44.

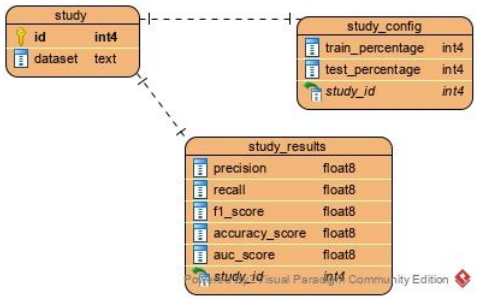

Figura 44 Modelo ER da aplicação para classificar o *dataset* iris

1- Criar um projeto Django no Visual Studio

No Visual Studio, selecionar **File** > **New** > **Project**, e pesquisar por "Django". Selecionar "**Blank Django Web Project**", tal como demonstrado na Figura 45.

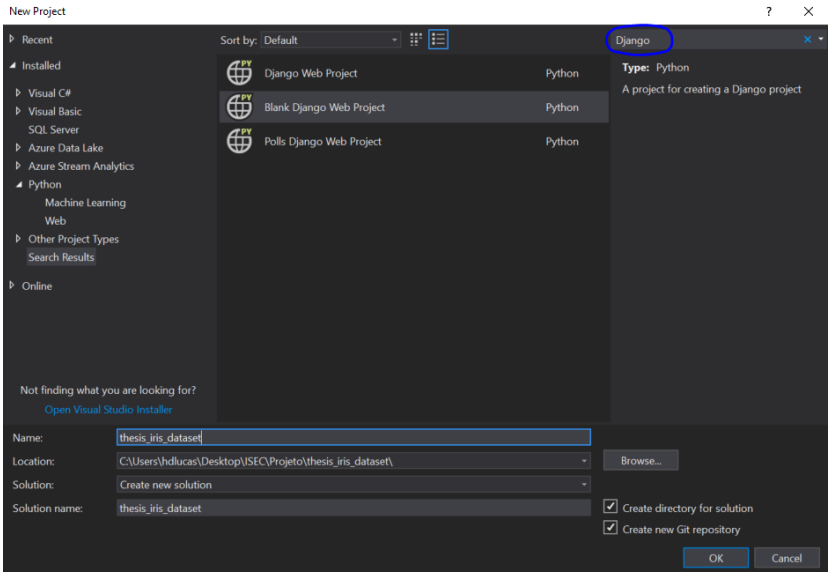

Figura 45 Como criar um projeto Django no Visual Studio

2- Criar uma aplicação dentro do projeto Django

Em "**Solution Explorer**" carregar no botão do lado direto e selecionar **Add** > **New item** e na janela **Add New Item** selecionar **Django 1.9 App**, especificar o nome da aplicação no campo **Name** e selecionar **OK**, tal como demonstrado na Figura 46.

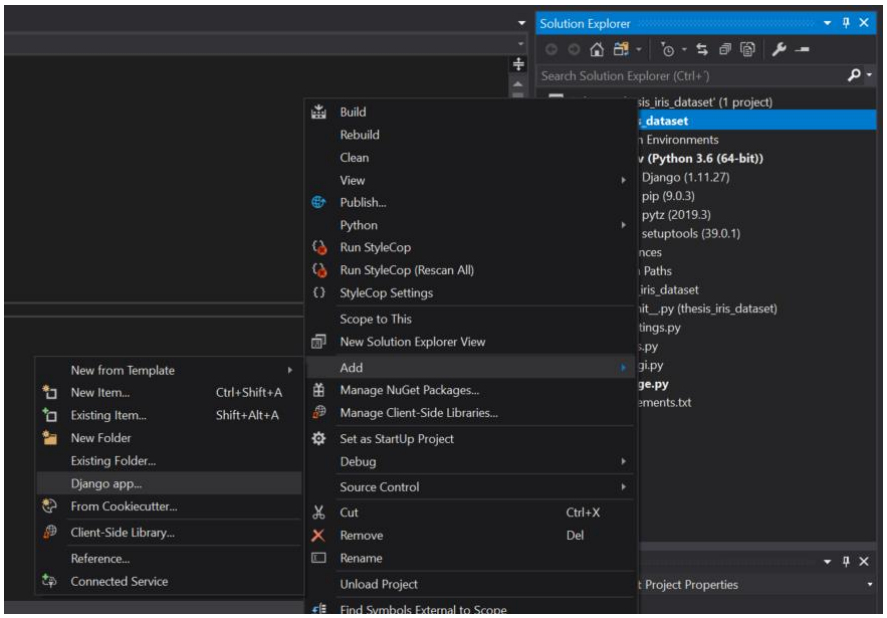

Figura 46 Como criar uma aplicação dentro de um projeto Django usando o Visual Studio

3- <u>Executar a aplicação a partir do projeto Django</u>

Para a aplicação ser executada restam apenas 2 passos:

a) Primeiro é necessário criar uma vista no ficheiro **views.py** para renderizar a página templates/index.html, tal como demonstrado na Figura 47.

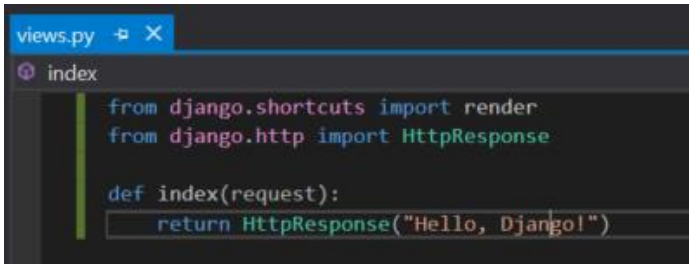

Figura 47 Criando uma nova vista no Django

b) De seguida, é necessário registar o(s) URL(s) para a vista criada no passo anterior, usando o ficheiro [nomedoprojeto]/**urls.py**. Analisando a Figura 48 conclui-se que é possível definir vários URLs para a mesma vista e verifica-se também que as *urlpatterns* começam com a expressão regular ^$ que é o caminho para a raiz do *site* "/".

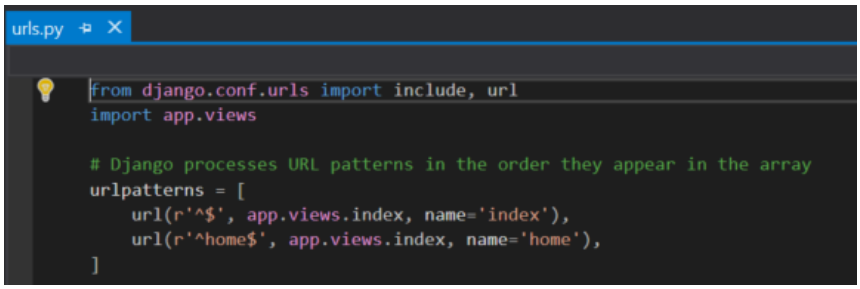

Figura 48 Definindo os URLs para a aplicação

Executando o projeto é possível observar que o browser apresenta uma página com a mensagem **Hello, Django**!

4- Configurar a ligação à base de dados PostgreSQL

a) Instalar o PostgreSQL e criar uma base de dados e um utilizador, garantindo que o utilizador tem privilégios suficientes para manipular a mesma, executando os seguintes comandos:

```
sudo su – postgres
psql
CREATE DATABASE iris_database;
CREATE USER iris_user WITH PASSWORD 'password';
GRANT ALL PRIVILEGES ON DATABASE iris_database TO iris_user;
```

b) Instalar o pacote psycopg2 [131] – é o adaptador de base de dados mais usado para a linguagem de programação Python. Uma das suas principais vantagens é o facto de permitir que várias *threads* partilhem a mesma conexão à base de dados. Para instalar o pacote é necessário adicionar uma nova entrada no ficheiro **requeriments.txt** com o nome do pacote e a respetiva versão "psycopg2==2.8.2", selecionar o ambiente Python e clicar na opção "**Install from requeriments.txt**", tal como demonstrado na Figura 49. Na janela de *output* o Visual Studio informará se a instalação foi ou não bem-sucedida.

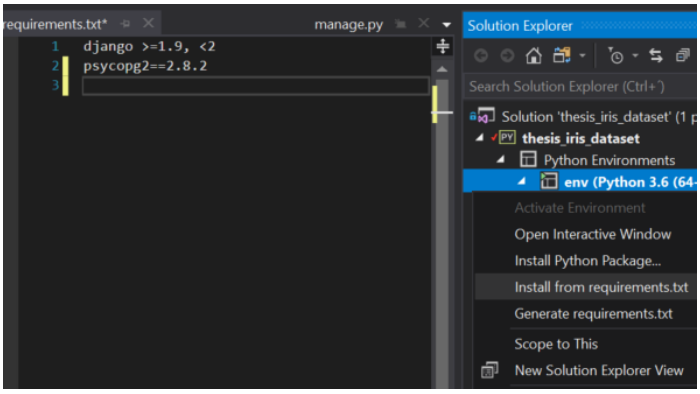

Figura 49 Instalação do pacote psycopg2 no ambiente Python

c) Configurar a ligação á base de dados usando o ficheiro **settings.py**, tal como exemplificado na Figura 50.

```
settings.py*

# https://docs.djangoproject.com/en/1.9/howto/static-files/

STATIC_URL = '/static/'

STATIC_ROOT = posixpath.join(*(BASE_DIR.split(os.path.sep) + ['static']))

DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.postgresql_psycopg2',
            'NAME': 'iris_database',
            'USER': 'iris_user',
            'PASSWORD': 'password',
            'HOST': 'localhost',
            'PORT': '5432',
        }
    }
```

Figura 50 Configuração da ligação á base de dados PostgreSQL usando o Django

5- Criar as tabelas na base de dados

Os passos a seguir para criar as tabelas na base de dados são os seguintes:

a) O Django usa o mapeamento de todas as tabelas da base de dados relacional no ficheiro **models.py** e implementa os relacionamentos definindo que um campo é chave estrangeira e indicando o seu tipo de dados. A Figura 51 contém o código-fonte com a definição dos modelos usados para produzir as tabelas apresentadas no modelo ER da Figura 27.

```
models.py | __init__.py (app.migrations) | views.py | results.html
StudyResult

from django.db import models
from django.conf import settings

# Create your models here.
class Study(models.Model):
    dataset = models.TextField()

    class Meta:
        db_table = "study"

class StudyConfig(models.Model):
    study = models.ForeignKey(Study, on_delete=models.CASCADE)
    train_percentage = models.IntegerField(default=0)
    test_percentage = models.IntegerField(default=0)

    class Meta:
        db_table = "study_config"

class StudyResult(models.Model):
    study = models.ForeignKey(Study, on_delete=models.CASCADE)
    precision = models.FloatField(default=0.0)
    recall = models.FloatField(default=0.0)
    f1_score = models.FloatField(default=0.0)
    support = models.FloatField(default=0.0, null=True)
    accuracy_score = models.FloatField(default=0.0)
    auc_score = models.FloatField(default=0.0)

    class Meta:
        db_table = "study_result"
```

Figura 51 Código fonte com os modelos usados para criar a base de dados pretendida

b) Criar as migrações executando o comando "**python .\manage.py makemigrations**" ou através da User Interface (UI) do Visual Studio selecionando a opção **Python** e de seguida clicando na opção **"Django Make Migrations"**, tal como demonstrado na Figura 52.

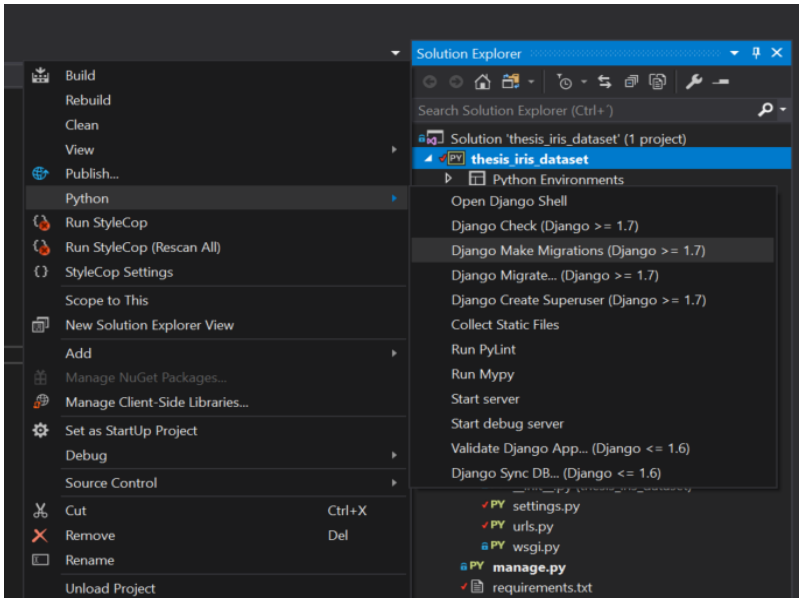

Figura 52 Como criar uma migração num projeto Django usando a UI do Visual Studio

c) Após executar as instruções do passo anterior, o projeto deverá agora conter uma nova migração na diretoria [nomedoprojeto]/ [nomedaaplicação]/migrations/. Para executar os scripts dentro dessa migração na base de dados o utilizar deve executar o comando "**python .\manage.py migrate**" ou usar a UI do Visual Studio selecionando a opção **Python** e de seguida clicando na opção "**Django Migrate**", tal como demonstrado na Figura 53.

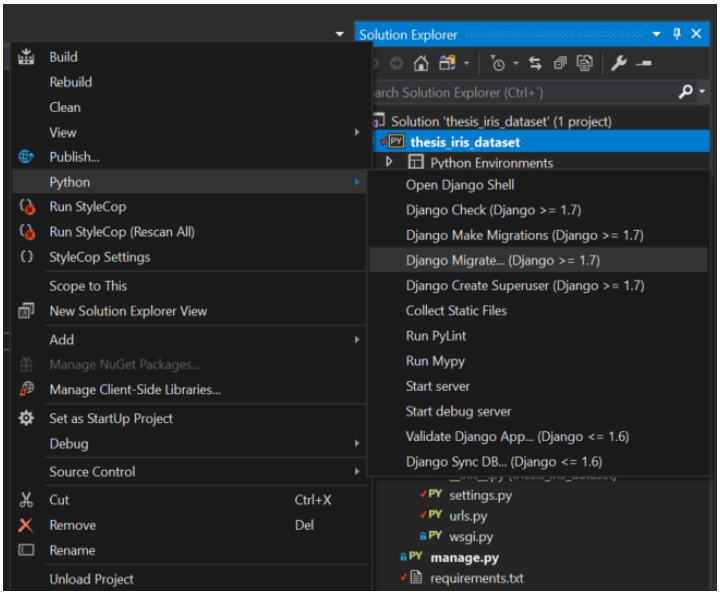

Figura 53 Como migrar *scripts* de base de dados num projeto Django usando a UI do Visual Studio

Após executar o passo anterior as tabelas passam a estar disponíveis na base de dados, tal como a Figura 54 comprova.

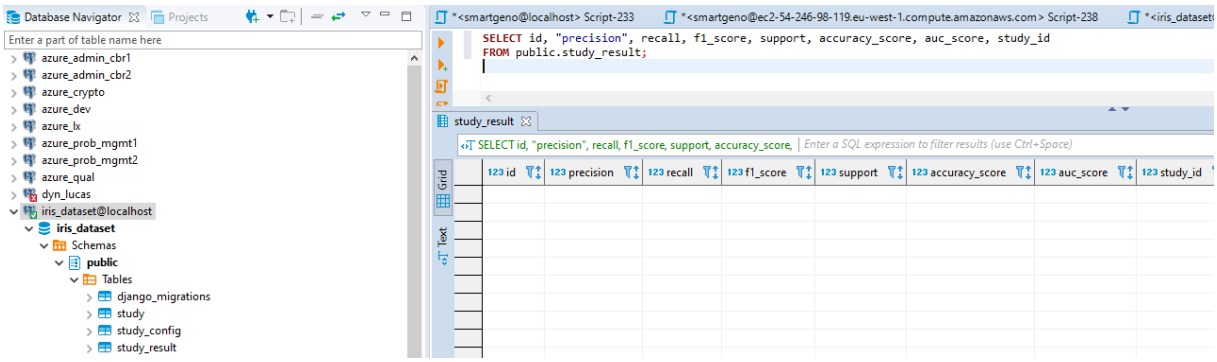

Figura 54 Tabelas criadas na base de dados usando modelos de um projeto Django

6- Criar um formulário para o utilizador iniciar a classificação do *dataset* iris

Para criar o formulário foi redefinido o HTML da página **index.html**, tal como demonstrado na Figura 55. Adicionou-se também um *script* para garantir que o formulário apenas é submetido se a soma das percentagens de treino e teste for igual a 100. Se o formulário for válido é submetido e a vista "**classify_iris_dataset**" é invocada. O código-fonte dessa vista é apresentado no passo seguinte.

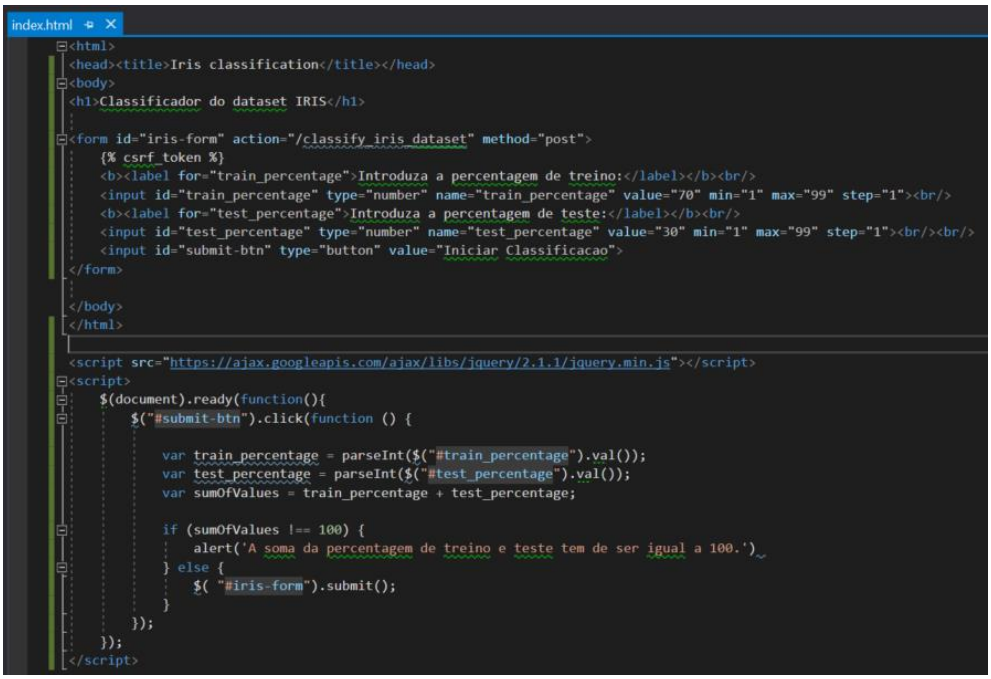

Figura 55 Código fonte da página index.html

Para renderizar a página a página **index.html** apresentada na Figura 57 foi necessário redefinir a vista index, definida no ficheiro **views.py**, tal como demonstrado na Figura 56.

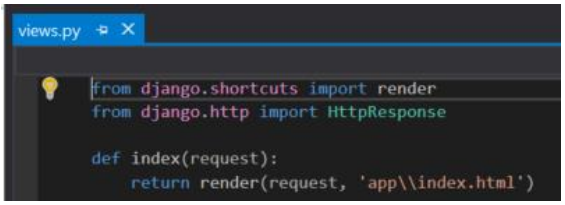

Figura 56 Vista index modificada para renderizar a página index.html

Figura 57 Aspeto da página index.html

7- Classificar o *dataset* iris usando o scikit-learn e persistir os resultados na base de dados

a) Em primeiro lugar é necessário instalar o pacote do scikit-learn, incluindo todas as suas dependências (Numpy, Scipy e Matplotlib) e o pacote Yellowbrick para construir a matriz de confusão. Os novos pacotes foram adicionados ao ficheiro "**requirements.txt**", tal como demonstrado na Figura 58.

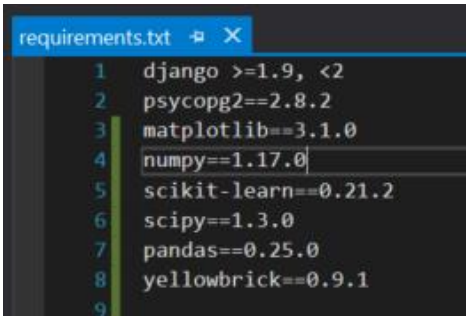

Figura 58 Instalação do pacote scikit-learn e as dependências

b) Implementar o código da vista "**classify_iris_dataset**" no ficheiro **views.py**.

```
def classify_iris_dataset(request):
    from sklearn.datasets import load_iris
    from sklearn import tree
    from http import HTTPStatus
    from app.models import Study, StudyConfig, StudyResult
```

```
    from django.http import JsonResponse
    from django.db import transaction
    from sklearn.metrics import precision_recall_fscore_support
    from sklearn.metrics import accuracy_score
    from sklearn.metrics import roc_auc_score
    from sklearn.model_selection import train_test_split
    from io import BytesIO
    from yellowbrick.classifier import ConfusionMatrix
    from sklearn.utils.multiclass import type_of_target

    import pandas as pd
    import matplotlib.pyplot as plt
    import base64

    httpStatus = HTTPStatus.OK

    try:
        with transaction.atomic():

            #Obtém dados do formulário criado no template index.html
            train_percentage = int(request.POST.get('train_percentage'))
            test_percentage = int(request.POST.get('test_percentage'))

            #Carrega o dataset iris
            iris = load_iris()

            dataset = pd.DataFrame(iris.data, columns=iris.feature_names)
            dataset['target'] = iris.target

            #Armazena uma nova entrada na tabela study (Apenas no fim do bloco 'with' uma vez que
estamos dentro de uma transação)
            study = Study(dataset = dataset.to_string())
            study.save()

            #Armazena as configurações do estudo na tabela study_config (Apenas no fim do bloco
'with' uma vez que estamos dentro de uma transação)
            study_config = StudyConfig(study = study, train_percentage = train_percentage,
test_percentage = test_percentage)
            study_config.save()

            X = iris.data
            y = iris.target

            # Divide o dataset num conjunto de treino e de teste conforme os valores configurados
pelo utilizador
            X_train, X_test, y_train, y_test = train_test_split(X, y, train_size= (train_percentage
/ 100), test_size= (test_percentage / 100))
```

```
            #Cria uma instância do classificador – Árvore de decisão
            clf = tree.DecisionTreeClassifier()

            #Estimador
            clf.fit(X_train, y_train)

            #Preditor
            y_pred = clf.predict(X_test)

            #Métricas usadas para medir a performance do classificador
            metrics = precision_recall_fscore_support(y_test, y_pred, average='macro')
            precision = metrics[0]
            recall = metrics[1]
            f1_score = metrics[2]
            support = metrics[3]
            accuracy_score = accuracy_score(y_test, y_pred)

            y_score=clf.predict_proba(X_test)

            if type_of_target(y) == 'binary':
                probs = y_score[:, 1]
                fpr, tpr, thresholds = roc_curve(y_test, probs)
                auc_score = auc(fpr, tpr)
            else:
                auc_score = roc_auc_score(y_test, y_score, multi_class='ovr')

            #Armazena as métricas na base de dados (Apenas no fim do bloco 'with' uma vez que
estamos dentro de uma transação)
            study_result = StudyResult(study = study,
                precision = precision,
                recall = recall,
                f1_score = f1_score,
                support = support,
                accuracy_score = accuracy_score,
                auc_score = score)
            study_result.save()

            visualizer = ConfusionMatrix(clf, classes=iris.target_names, label_encoder={0: 'setosa',
1: 'versicolor', 2: 'virginica'})
            visualizer.score(X_test, y_test)
            plt.autoscale(tight=True)
            ax = plt.gca()
            #ajusta a imagem à largura do ecrã
            ax.axis('auto')
            plt.tight_layout()

            outpath = BytesIO()
```

```
            #Armazena a matriz de confusão em memória
            plt.savefig(outpath)
            outpath.seek(0)
            confusion_matrix_image = base64.b64encode(outpath.getvalue())

    except Exception as ex:
        httpStatus = HTTPStatus.INTERNAL_SERVER_ERROR
        error_description = getattr(ex, 'message', repr(ex))

    if httpStatus == HTTPStatus.OK:
        return render(request, 'app\\results.html', {
            'precision': round(precision, 4),
            'recall' : round(recall, 4),
            'f1_score' : round(f1_score, 4),
            'accuracy_score' : round(accuracy_score, 4),
            'auc_score' : round(auc_score, 4),
            'confusion_matrix_image' : confusion_matrix_image
            })
    else:
        return JsonResponse({'errorDescription' : error_description}, status = httpStatus.value)
```

8- Apresentar os resultados ao utilizador.

Por fim, para apresentar os resultados ao utilizador foi implementada uma nova página HTML "**results.html**" que contém o código-fonte apresentado na Figura 59 que depois de renderizado terá um aspeto semelhante ao da Figura 60. Observando a Figura 59 verifica-se que os dados retornados pela vista "**classify_iris_dataset**" foram injetados no HTML usando *tags* de *template* (ex.: {{*precision*}}).

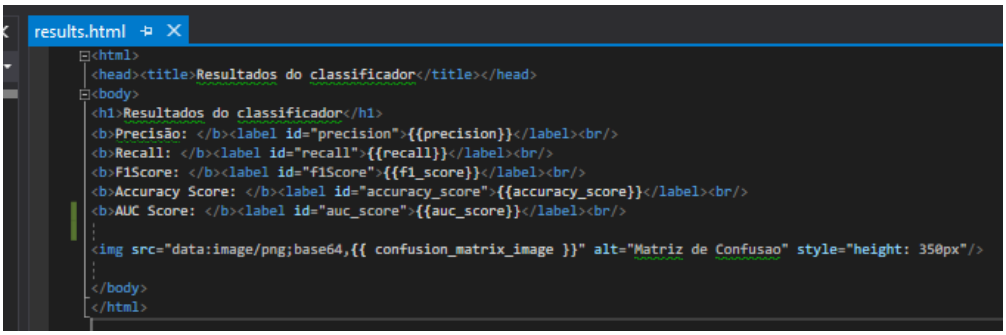

Figura 59 Código fonte da página results.html

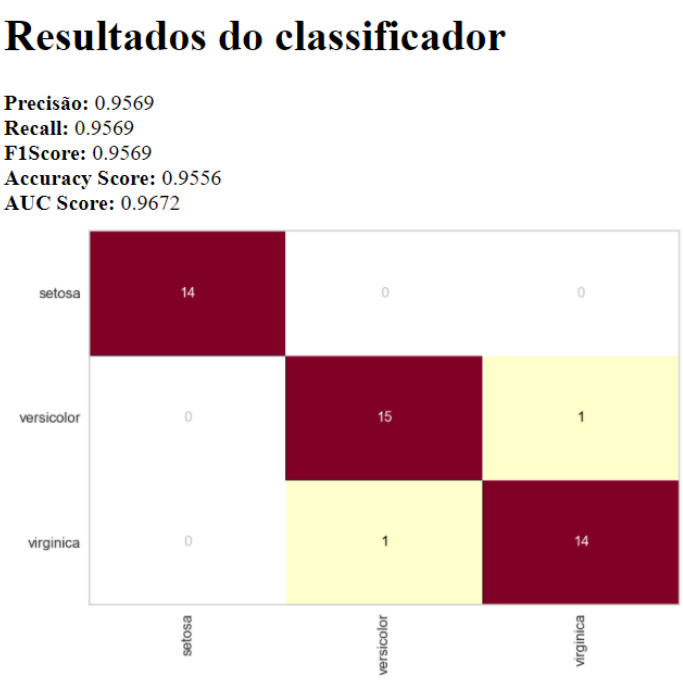

Figura 60 Aspeto da página results.html

ANEXO E – EXTRAÇÃO DE *FEATURES DA SEQUÊNCIA PROTEICA*– EXEMPLO PRÁTICO

Este anexo visa demonstrar exemplificar como uma sequência proteica foi codificada num vetor de 26 elementos (25 *features* + *target*) a ser usado como entrada de um modelo de *machine learning*.

Exemplo de sequência proteica da família GroES chaperonin

MTSTNFRPLHDRVYVRRVESEAKTKGGIIIPDTAKEKPQEGEIVAVGSGARDEAGK

Cálculo do Índice hidropático

$$0.09 * 1.8 + 0.0 * 2.5 + 0.05 * -3.5 + 0.11 * -3.5 + 0.02 * 2.8 + 0.11 * -0.4 + 0.02 * -3.2 + 0.07 * 4.5 + 0.09 * -3.9 + 0.02 * 3.8 + 0.02 * 1.9 + 0.02 * -3.5 + 0.05 * -1.6 + 0.02 * -3.5 + 0.09 * -4.5 + 0.05 * -0.8 + 0.07 * -0.7 + 0.09 * 4.2 + 0.0 * -0.9 + 0.04 * -1.3 = \mathbf{-0,76}$$

Cálculo da Aromaticidade

$$0.09 * 0 + 0.0 * 0 + 0.05 * 0 + 0.11 * 0 + 0.02 * 1 + 0.11 * 0 + 0.02 * 0 + 0.07 * 0 + 0.09 * 0 + 0.02 * 0 + 0.02 * 0 + 0.02 * 0 + 0.05 * 0 + 0.02 * 0 + 0.09 * 0 + 0.05 * 0 + 0.07 * 0 + 0.09 * 0 + 0.0 * 1 + 0.04 * 1 = \mathbf{0.06}$$

Cálculo do ponto isoelétrico

Contagem do número de cópias dos aminoácidos que desempenham um papel na determinação do pI (o resultado é convertido para *float* e o Nterm e Cterm são constantes com valor = 1.0):

{'Nterm': 1.0, 'K': 5.0, 'R': 5.0, 'H': 1.0, 'Cterm': 1.0 'D': 3.0, 'E': 6.0, 'C': 0.0, 'Y': 2.0}

Cálculo da carga da proteína quando sujeita a um pH = **7.0**

- Carga Positiva (Lysine, Arginine e Histidine):

$$1.0 * 10\ \hat{}\ (7.0 - 7.0) / (10\ \hat{}\ (7.0 - 7.0) + 1.0) + 5.0 * 10\ \hat{}\ (10.0 - 7.0) / (10\ \hat{}\ (10.0 - 7.0) + 1.0) + 5.0 * 10\ \hat{}\ (12.0 - 7.0) / (10\ \hat{}\ (12.0 - 7.0) + 1.0) + 1.0 * 10\ \hat{}\ (5.98 - 7.0) / (10\ \hat{}\ (5.98 - 7.0) + 1.0) = 10.58$$

- Carga Negativa (Aspartic e Glutamic acids, Cysteine e Tyrosine):

$$1.0 * 10\ \hat{}\ (7.0 - 3.55) / (10\ \hat{}\ (7.0 - 3.55) + 1.0) + 3.0 * 10\ \hat{}\ (7.0 - 4.05) / (10\ \hat{}\ (7.0 - 4.05) + 1.0) + 6.0 * 10\ \hat{}\ (7.0 - 4.45) / (10\ \hat{}\ (7.0 - 4.45) + 1.0) + 0.0 * 10\ \hat{}\ (7.0 - 9.0) / (10\ \hat{}\ (7.0 - 9.0) + 1.0) + 2.0 * 10\ \hat{}\ (7.0 - 10.0) / (10\ \hat{}\ (7.0 - 10.0) + 1.0) = 9.98$$

A carga da proteína é igual à diferença entre a carga positiva e negativa:

$$10.58 - 9.98 = 0.60$$

Como a diferença entre as cargas é positiva, ao pH anterior deve ser somado 1.0 e repete-se o processo até ser obtida uma carga negativa. Caso a diferença entre as cargas fosse negativa o processo seria o oposto, ou seja, subtrair-se-ia 1.0 ao pH até que a carga fosse positiva.

Cálculo da carga da proteína quando sujeita a um pH = **8.0**:

$$10.05 - 10.02 = 0.03$$

Cálculo da carga da proteína quando sujeita a um pH = **9.0**:

$$9.55 - 10.18 = -0.63$$

Ao valor de pH que fez com que tivesse sido obtida uma carga negativa (pH = 9.0) soma-se o valor do pH da iteração anterior (pH = 8.0) e divide-se por 2:

$$(9.0 + 8.0) / 2 = 8.5$$

Calcula-se novamente o valor da carga para um dado valor de pH, atualizando o valor do pH em função da iteração anterior (à exceção da iteração 1 que deve usar o pH calculado no passo 4) e repete-se o processo até que a carga esteja próxima de zero (carga < 0.0001)

Cálculo da carga da proteína quando sujeita a um pH = **8.5**:

$$9.88 - 10.06 = -0.18$$

Cálculo da carga da proteína quando sujeita a um pH = **8.25**:

$$9.97 - 10.03 = -0.06$$

…

Cálculo da carga da proteína quando sujeita a um pH = **8.09**:

$$10.02 - 10.02 = 0{,}00$$

Cálculo do peso molecular

$$5 * 89.09 + 0 * 121.15 + 3 * 133.10 + 6 * 147.13 + 1 * 165.19 + 6 * 75.07 + 1 * 155.16 + 4 * 131.18 + 5 * 146.19 + 1 * 131.18 + 1 * 149.21 + 1 * 132.12 + 3 * 115.13 + 1 * 146.15 + 5 * 174.20 + 3 * 105.09 + 4 * 119.12 + 5 * 117.15 + 0 * 204.23 + 2 * 181.20 - (57 - 1) * 18.015 = 7268.90 - 1008.86 = \mathbf{6260.04}$$

Tamanho da Sequência

57

Frequência absoluta de cada aminoácido

{'A': **5**, 'C': **0**, 'D': **3**, 'E': **6**, 'F': **1**, 'G': **6**, 'H': **1**, 'I': **4**, 'K': **5**, 'L': **1**, 'M': **1**, 'N': **1**, 'P': **3**, 'Q': **1**, 'R': **5**, 'S': **3**, 'T': **4**, 'V': **5**, 'W': **0**, 'Y': **2**}

O resultado da representação vetorial da sequência acima exemplificada é o seguinte:

-0.76,0.06, 8.09,6260.04,57,5,0,3,6,1,6,1,4,5,1,1,1,3,1,5,3,4,5,0,2,GroES chaperonin family

Nota: A *label* "GroES chaperonin family" não pode ser usada como *target* de um modelo de *machine learning* uma vez que estes apenas suportam *arrays* de valores numéricos portanto, foi usada a classe LabelEncoder [132] que dispõe de um método (transform) que faz o *encode* dos *targets* num valor entre 0 e n_classes -1 e um método para fazer a transformação inversa (invers_transform). A Figura 61 ilustra a transformação para 3 famílias usando o LabelEncoder.

Figura 61 Exemplo da aplicação do sklearn.preprocessing.LabelEncoder