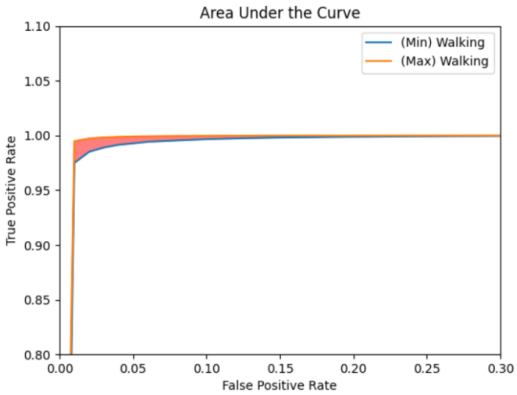
Universidade da Beira Interior
Departamento de Informática

P2 - Inteligência Artificial: *Human Activity Recognition*



Elaborado por:
Tiago Barreiros - a46118

Orientador:
Professor Doutor Hugo Pedro Proença

9 de janeiro de 2023, Covilhã

Agradecimentos

Primeiramente, gostaria de agradecer ao meu orientador Prof. Doutor Hugo Pedro Proença por ser sempre justo e fiel às suas ideias e aquando a proposta deste tema.

Apesar do, notável e significativo, aumento de dificuldade e durabilidade deste novo desafio, a sua orientação foi crucial no desenvolvimento do mesmo, bem como os métodos e as estratégias lecionadas em aula, a forma didática e prática como foram facultadas, foi muito importante para o meu desenvolvimento profissional e pessoal, pois senti que melhorei como pessoa e como futuro profissional.

É importante realçar que este trabalho é fruto de uma expressiva investigação pessoal, uma significativa quantidade de testes implementados, bem como a extensa quantidade de horas, "processador"e recursos do nosso computador **pessoal** investidos na realização deste projeto.

Como no trabalho anterior, gostaria de realçar as aprendizagens obtidas no primeiro ano, como, por exemplo, a escrita de documentos em LaTeX, lecionada na Unidade Curricular (UC) - Laboratórios de Programação (LP), regida pelo Prof. Doutor Pedro M. Inácio, a quem devo bastante reconhecimento e sou igualmente grato.

Por fim, um especial agradecimento à minha família que está neste momento a investir e a acreditar em mim. Da minha parte um muito obrigado!

Resumo

A Inteligência Artificial pode ser vista a partir de três disciplinas, Psicologia, através da modelagem cognitiva, Filosofia e Engenharia Informática, devido à logística, matemática e lógica.

Formular o problema:
- Um estado que contém todas as informações necessárias para prever os efeitos de uma ação e determinar se um estado satisfaz o objetivo;
- Assumimos que um conjunto de Operadores que definem um estado como válido, pode ser modificado de forma a transformar um outro estado válido, mas ligeiramente melhor;
- Entre o conjunto dos estados possíveis, existem dois tipos que são particularmente importantes:
- Estado inicial: O estado onde a pesquisa começa;
- *Goal State*: O estado a ser alcançado, ou seja, que satisfaça plenamente os objetivos do problema.
- No nosso contexto, a solução é a sequência de ações que visam transformar o estado inicial no *goal state*. - Analisar o conjunto de estados possíveis;
- Definir as estruturas de dados necessárias para representar um estado;
- Definir operações;
- Definir o estado inicial e expandir até que uma solução seja alcançada;
- Estado inicial;
- operador;
- estados.

Como resolvi o problema?

1º passo: Começo dos testes;

2º passo: Seleção -> escolho quais os elementos que passam para a próxima geração;

3º passo: *crossover* é utilizado para produzir novos elementos, após a seleção dos "pais";

4º passo: Mutação, cada posição dos elementos recentemente gerados será mutada, respeitando uma certa probabilidade.

Pude concluir que dá muito mais gosto trabalhar em IA, do que estudar, pena ser só para 3 valores.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Excertos de Código

Palavras-Chave

Inteligência Artificial, *Machine Learning Model*, Python, Rede Neuronal, *ROC curves*.

Acrónimos

AUC *Area Under the ROC curve*

CSV *Comma-separated values*

EI Engenharia Informática

IA Inteligência Artificial

IDE *Integrated Development Environment*

LP Laboratórios de Programação

NN *Neural Networks*

ROC *Receiver Operating Characteristic Curve*

UBI Universidade da Beira Interior

UC Unidade Curricular

Capítulo

1

Introdução

1.1 Âmbito e Enquadramento

Encontra-se a analisar um relatório, elaborado no 1º semestre do 3.º ano da Licenciatura em Engenharia Informática (EI) da Universidade da Beira Interior (UBI).

Este foi desenvolvido no contexto da Unidade Curricular (UC) Inteligência Artificial (IA) da UBI, através de pesquisas pessoais e material pedagógico providenciado pelo Professor da própria UC.

O presente relatório abrange uma das áreas mais importantes nos tempos que decorrem, esta consiste no estudo e prática da IA para construir, planear, desenvolver, manter e retirar dados, informações, num determinado *software*. Estas tarefas exigem inteligência humana, como reconhecimento de padrões, raciocínio lógico e diversos tipos de aprendizagem.

A busca pelo verdadeiro significado de gradiente, este é usado para treinar *Neural Networks* (NN), as quais se baseiam no cérebro humano. Durante o seu treino, os pesos das conexões entre os neurónios são ajustados de acordo com o gradiente da função de perda, que mede o quão longe o *Output* está do resultado desejado.

1.2 Motivação

Como mencionei anteriormente, IA é uma das áreas mais interessantes da informática, porém, a meu ver, das mais complicadas. A quantidade de dados a ser analisados, muitas vezes não reflete a qualidade dos mesmos, o cansaço e a forma exaustiva que os procuramos pode tornar-se um pesadelo.

Contudo, continuo a defender que a IA tem um papel fundamental nos dias de hoje. Atualmente, ainda não se conseguiu chegar ao que designamos "IA Forte", mas os agentes da família de IA Fraca desempenham um papel muito importante, pois são muito, muito bons, numa certa especificação.

Soluções corporativas, IA e robôs com linguagens personalizadas para processamento de dados, operação, análise estatística, funcionamento, análise analítica e segurança cibernética estão previstos fazerem parte do nosso futuro. Antecipa-se o surgimento de novos processos que exigem a competência de engenheiros com fortes conhecimentos neste área, pois vai-se procurar suporte, soluções integradas, orientação técnica e ideias inovadoras.

Além disso, este trabalho foi realizado no âmbito das metas definidas pelo Prof. Doutor Hugo Pedro Proença, como gerente da UC - Inteligência Artificial, como tal considero essa a minha principal motivação.

1.3 Objetivos

O principal objetivo deste projeto é criar um "*software*"capaz de reconhecer um leque de atividades humanas, analisar um conjunto de dados e através de uma sequência de medições, determinar a real atividade.

Pretendia-se o uso de algoritmos dedicados à IA, como a Validação Cruzada K-fold para dividir e analisar conjuntos de treino, validação e teste, Normalização, modelos de *Machine Learning* para prever as atividades, NN, curvas *Receiver Operating Characteristic Curve* (ROC), respetivas Matrizes de Confusão e *Area Under the ROC curve* (AUC).

Defini como objetivos pessoais melhorar a minha capacidade de aprender, raciocinar e interagir com a IA, para me tornar mais flexível na resolução dos problemas, de forma a tornar o meu uso da IA mais eficaz.

Não esquecer que este tipo de projetos são essenciais para o nosso futuro como engenheiros informáticos. Capacita-nos com mais e melhores ferramentas para o mundo profissional, promovendo a nossa elasticidade mental. Por sua vez, esta é fundamental na adaptação a qualquer desafio que possamos enfrentar, quer a nível académico, profissional e pessoal.

1.4 Abordagem

A maioria das decisões tomadas foi influenciada pela revisão de todo o material pedagógico que o Professor facultou, uma vez que os alunos não tinham qualquer tipo de experiência prévia na matéria. Contudo, foi realizada uma pesquisa extensa em fontes externas a esse material, de modo a complementar a matéria dada pelo Professor.

Comecei por criar uma função que lê-se os dados do ficheiro `.csv` facultado pelo professor, o qual contem todos os dados das atividades humanas a ser analisadas ao longo deste projeto. Quando lido, parti para a criação das instâncias, onde recebia esses dados e juntava, `x`, `y` e `z` de 20 em 20 linhas, mais `id` e `atividade`.

Quando terminado o passo anterior, se tudo correr bem, cerca de 15s seria o tempo despendido neste processo. Na criação dos conjuntos K-fold, dividi em k (k=10) o ficheiro e juntei a maioria para o conjunto de treino e o resto para teste/validação. Deu-me cada fold +/- 400mb, onde os de teste e treino demoraram cerca de 18m. Concluindo 10 conjuntos de treino e 10 conjuntos de teste para os 36`ids`, três `ids` por fold, onde os restantes vão para os folds que tem menos `ids` para ficar equilibrado.

Prontos os conjuntos, chegou a altura de normalizar cada uma das interações, onde verifico qual é o `x`, `y`, `z`, mínimo e máximo, do conjunto treino e procedo ao cálculo da normalização.

Depois de mais ou menos ajustes, os parâmetros da Rede Neuronal iam ser desafiantes. Comecei por abrir os conjuntos de teste e treino todos, criei uma função que recebe o conjunto e vai retirar o tipo de atividade, em busca do objetivo, criar 10 redes, 10 *fits*. Alterei inúmeras vezes os parâmetros do `MLPClassifier`, andei em busca da melhor sequência, da melhor curva ROC e consequente AUC.

No final, calculei a soma de todos os dados das Matrizes de Confusão por cada atividade e cada fold, as curvas ROC por cada atividade, de forma a forma um "rio", quanto mais curto for esse "rio", mais próximos são os resultados. A *accuracy* dos dados, depende da topologia da rede, mas independente do uso de mais ou menos Neurónios, a verdadeira pergunta é nós percebermos quantos segundos são necessários para obtermos essa diferença.

1.5 Organização do Documento

De modo a refletir o trabalho que foi feito, este documento encontra-se estruturado da seguinte forma:

1. O primeiro capítulo – **Introdução** – refere o projeto que nos foi proposto, o seu âmbito e Enquadramento, a Motivação para a sua realização, os objetivos e a respetiva organização do documento.

2. O segundo capítulo – **Estado da Arte, Desenvolvimento e Aplicações** – descreve as ferramentas utilizadas na realização deste projeto;

3. O terceiro capítulo – **Desenvolvimento e Implementação** – apresenta as dependências que existem no programa e os detalhes de implementação;

4. O quarto capítulo – **Testes e Validação** – explicita todos os Testes e Aprimoramentos realizados;

5. O quinto capítulo – **Conclusões e Trabalho Futuro** – conclusão do projeto desenvolvido, incluindo considerações sobre o desempenho e trabalho futuro por realizar.

Capítulo

2

Estado da Arte, Desenvolvimento e Aplicações

2.1 Introdução

Este capítulo apresenta as aplicações e ferramentas usadas no desenvolvimento deste projeto, os seus detalhes de implementação e uma breve descrição da plataforma PyCharm. O capítulo divide-se do seguinte modo:

- a secção 2.2, **Overleaf**, introdução à plataforma de escrita;
- a secção 2.3, **GitHub**, caracteriza fielmente a plataforma;
- a secção 2.4, **PyCharm**, define as caraterísticas da principal ferramenta usada ao longo do projeto.

2.2 Overleaf

O Overleaf é uma plataforma de colaboração *online* para escrita de documentos científicos, que inclui recursos para facilitar a criação de artigos, relatórios, apresentações e muito mais.

É especialmente útil para escritores científicos, mas igualmente para nós, estudantes, pois permite-nos trabalhar no mesmo documento, juntos, em tempo real, compartilhar ideias e receber *feedback*. Também oferece integração com várias bases de dados e repositórios de código, o que facilita a incorporação de Citações, Referências e *Linkagem*.

Foi essencial na redação e embelezamento do Relatório Final, pois possui características que se destacam, entre as quais:

- Capacidade de auto-completar funções;
- Compila o trabalho várias vezes, com o objetivo de estar sempre atualizado e não se perder dados;
- Na eventualidade de existirem erros, indica os mesmos.

Figura 2.1: Representação do logótipo do Overleaf.

2.3 GitHub

O GitHub é um serviço de gestão de código-fonte e colaboração *online*. Serve para armazenar e gerir projetos de código aberto e privado.

Permite que os utilizadores criem repositórios de código, façam o seu controlo de versões, adicionem colaboradores aos projetos e façam o rastreamento de problemas e tarefas. Também fornece ferramentas para revisão de código, integração contínua e integração com outras ferramentas de desenvolvimento de *software*.

Além disso, o GitHub é popular para desenvolvedores compartilharem e descobrirem novos projetos de código aberto, uma plataforma comunitária para colaboração e desenvolvimento de *software* em equipa.

Foi fundamental neste projeto, pois possibilitou-me retornar a versões anteriores e evitou-me trabalho acrescido.

Figura 2.2: Representação do logótipo do *GitHub*.

2.4 PyCharm

O PyCharm é um *Integrated Development Environment* (IDE) que fornece uma série de ferramentas para facilitar o processo de escrita e depuração de código Python. Para o projeto esta ferramenta teve uma presença crucial, permitiu desenvolver de forma eficiente todo o trabalho e executar todos os testes.

Em suma, algumas das suas principais características incluem:

- Suporte a várias versões do Python;
- oferece sugestões de código enquanto digita, o que pode ajudar a economizar tempo e evitar erros;
- possui uma ferramenta de depuração integrada;
- integração com o Git e outras ferramentas de controlo de versões.

Figura 2.3: Representação do logótipo do PyCharm.

Capítulo

3

Desenvolvimento e Implementação

3.1 Introdução

Garantir a qualidade de uma IA é importante, pois melhora a experiência do utilizador, evita problemas e erros, aumenta a segurança, a confiabilidade e maior retorno do investimento, desempenho e precisão.

Este capítulo descreve cada abordagem ao decorrer do projeto. O capítulo divide-se do seguinte modo:

- a secção 3.2, **Dependências**, explica as dependências do projeto;
- a secção 3.3, **Detalhes de Implementação**, caracteriza e descreve cada processo desenvolvido neste trabalho;

3.2 Dependências

O programa apresenta algumas dependências como os *import´s* e bibliotecas utilizadas:

- `import csv` - é usado para ler e escrever arquivos no formato *Comma-separated values* (CSV);
- `import random` - fornece funções para gerar números aleatórios;
- `from sklearn.metrics import roc curve, auc, accuracy score, confusion matrix` - importa quatro funções:
  1. **roc curve:** calcula a curva ROC;
  2. **auc:** calcula a área sob a curva ROC;

3. **accuracy score:** calcula a precisão;
4. **onfusion matrix:** calcula a matriz de confusão.

- `from sklearn.neural network import MLPClassifier` - implementa uma rede neural artificial de várias camadas;
- `from sklearn.preprocessing import LabelEncoder, OneHotEncoder` - usadas para pré-processar dados de entrada antes de usá-los para treinar um modelo;
- `import matplotlib.pyplot as plt` - é usado para criar gráficos e plots em Python.

3.3 Detalhes de Implementação

3.3.1 Instâncias

Depois da abertura do ficheiro `.csv`, a base de guardei os dados é tal como a mencionada pelo professor nas aulas práticas, representada no excerto de código 3.3.1.

Processo:

- ler 20 em 20 linhas, juntando os valores de `x`, `y` e `z` na mesma linha, ficando por exemplo:

  x,y,z,x,y,z,x,y,z...,id, atividade

Alguns dados:

- demoraram aproximadamente 15 segundos a ser criadas.

```
[1, Walking, x, y, z]
[1, Walking, x, y, z]
[..., ..., ..., ..., ...]
[1, Walking, x, y, z]
```

Excerto de Código 3.1: Modelo das Instâncias.

3.3.2 Validação Cruzada K-fold

É um dos métodos estatísticos mais utilizados, pois é fácil de compreender e de implementar para estimar o desempenho de modelos. Fornece estimativas que são comparáveis a outros métodos mais sofisticados, como, por exemplo, o Bootstrapping.

A função `def kfoldsets(dados, k)` divide em k folds o ficheiro onde cada fold pode ter várias instâncias com diferentes `ids`, mas um `id` não pode estar em mais do que um fold.

Regra Geral:

- Os dados disponíveis são divididos aleatoriamente em K grupos de amostras;
- O modelo é ajustado "K"vezes, usa um grupo como conjunto de teste e os restantes (k-1), como dados de treino;
- O desempenho é obtido para o conjunto de teste;
- O desempenho final é dado pelo valor médio dos valores de desempenho "K".

Meu Processo:

- declarar variáveis;
- percorrer os 10 conjuntos;
- percorrer o tamanho de cada conjunto e dar `random` nos `ids`:
  1. existem 36 `ids`;
  2. se dividir 36 / 10 = 3.6 que arredonda para 3
  3. três `ids` por fold;
  4. os que faltam vão para os folds que tem menos `ids` para ficar equilibrado;
  5. detetar de modo aleatório, pois o objetivo é concluir a atividade que ele está a exercer.
- ciclo for que percorre os 10 folds, se o tamanho do fold for menor que o primeiro, o primeiro passa a ser o menor, representado no excerto de código 3.3.2;
- adiciona o `id` ao fold com menos instâncias;
- o meu *index* atual vai o conjunto teste e os restantes de treino;
- para cada iteração criar 10 folds de Treino e 10 folds de Teste;

Alguns dados:

- demoraram cerca de 18 minutos a gerar os 10 folds de Teste e Treino.

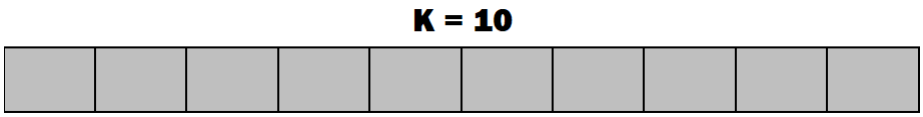


Figura 3.1: Representação dos dados divididos em 10 k-folds.

- cada fold tem +/- 400mb

```
for j in ids:

    menor = 0

    primeiro = len(k_sets[0])

    # ciclo for que percorre os k folds, se o tamanho do fold for
        menor que o primeiro, o primeiro passa a ser o menor
    for i in range(len(k_sets)):
        if len(k_sets[i]) < primeiro:
            menor = i
            primeiro = len(k_sets[i])

    # adiciona o id ao fold com menos instancias
    for aux in dados:
        if int(aux[-2]) == j:
            k_sets[menor].append(aux)
```

Excerto de Código 3.2: Verifica se os ids estão todos no fold.

3.3.3 Normalização

A normalização ajuda a equilibrar os conjuntos de dados para que as características tenham um impacto similar no modelo.

Processo:

- para cada uma das interações verificar qual é o x, y, z, mínimo e máximo do conjunto, processo representado no excerto de código 3.3.3.

```
for i in treino:
    for j in range(0, len(i)- 2, 3):
        # se o valor de x for maior que o maximo de x, o maximo de x
            passa a ser o valor de x
        if float(i[j]) > max_x:
            max_x = float(i[j])
```

```
            # se o valor de y for maior que o maximo de y, o maximo de y
                passa a ser o valor de y
            if float(i[j + 1]) > max_y:
                max_y = float(i[j + 1])

            # se o valor de z for maior que o maximo de z, o maximo de z
                passa a ser o valor de z
            if float(i[j + 2]) > max_z:
                max_z = float(i[j + 2])

            # se o valor de x for menor que o minimo de x, o minimo de x
                passa a ser o valor de x
            if float(i[j]) < min_x:
                min_x = float(i[j])

            # se o valor de y for menor que o minimo de y, o minimo de y
                passa a ser o valor de y
            if float(i[j + 1]) < min_y:
                min_y = float(i[j + 1])

            # se o valor de z for menor que o minimo de z, o minimo de z
                passa a ser o valor de z
            if float(i[j + 2]) < min_z:
                min_z = float(i[j + 2])
```

Excerto de Código 3.3: Máximos e minimos do conjunto de treino.

Resumo do conceito de *Machine Learing* a reter:

O procedimento experimental para aprender/avaliar modelos de aprendizagem de máquinas é sensível. Uma das regras de ouro na aprendizagem é que os dados devem ser divididos em três subconjuntos desajustados:

- Conjunto de **Treino**, este é o conjunto de instâncias utilizadas para se ajustar aos parâmetros da hipotese;
- Conjunto de **Teste**, fornece uma avaliação imparcial de um modelo final;
- Conjunto de **Validação**, fornece uma avaliação imparcial do desempenho de um modelo durante o processo de aprendizagem, ao mesmo tempo que afina os parâmetros do modelo, por exemplo, limiar de aceitação/rejeição.

3.3.4 Rede Neuronal

Entramos, portanto, na parte mais interessante do desenvolvimento deste projeto. Nesta, foram dedicadas imensas horas, dias, muitos recursos do com-

putador e paciência, pois de testes para testes as horas variam imenso, mas a diferença dos resultados não.

Após criar todo o processo de NN, através das bibliotecas referidas na secção 3.2, abrir todos os conjuntos de teste e treino, comecei a executar testes.

O custo de uma solução é a soma de todos os movimentos (ou seja, operações) entre os estados inicial e objetivo. No entanto, na maioria das vezes, uma solução pode ser alcançada de muitas maneiras diferentes e de acordo com um número diferente de etapas.

O problema é: Como podemos encontrar a solução ótima?

A procura de todas as soluções possíveis não é viável, ou porque existe um número potencialmente infinito de estados, ou por esse número ser, simplesmente, demasiado elevado. Em problemas de pesquisa/otimização, estamos interessados em encontrar uma parametrização para o nosso modelo que forneça uma boa solução. Isto envolve uma pesquisa na parametrização na granularidade (função `MLPClassifier`), e encontrar as configurações que maximizam/minimizam o "caminho"para a solução desejada.

Conceitos lógicos que segui:

- Um estado que contém todas as informações necessárias para prever os efeitos de uma ação e determinar se um estado satisfaz o objetivo;
- Assumimos que um conjunto de Operadores que definem um estado como válido, **pode ser modificado de forma a transformar um outro estado válido, mas ligeiramente melhor**;
- Entre o conjunto dos estados possíveis, existem dois tipos que são particularmente importantes:
    1. **Estado inicial:** O estado onde a pesquisa começa;
    2. ***Goal State*:** O estado a ser alcançado, ou seja, que satisfaça plenamente os objetivos do problema.

3.3.4.1 Função `MLPClassifier`

Os argumentos da função que usei para criar um objeto dessa classe são os seguintes:

- **`solver`:** especifica o algoritmo utilizado para otimizar os pesos do MLP. As opções incluem 'lbfgs', 'sgd' e 'adam';
- **`activation`:** especifica a função de ativação para as camadas escondidas e de saída. As opções incluem 'identity', 'logistic', 'tanh' e 'relu';

- **`alpha`:** usei sempre a (1e-5);

- **`hidden layer sizes`:** especifica o número de neurónios nas camadas escondidas do MLP. Por exemplo, se você quiser criar um MLP com duas camadas escondidas, cada uma com 100 neurónios, pode passar o argumento (100, 100);

- **`max iter`:** especifica o número máximo de iterações para o algoritmo de otimização;

- **`verbose`:** sempre a *true* para poder receber um *feedback* do processo;

- **`batch size`:** especifica o tamanho do *batch*;

- **`learning rate`:** especifica a taxa de aprendizagem;

- **`learning rate init`:** especifica a taxa de aprendizagem inicial;

- **`shuffle`:** especifica se os dados de treino devem ser misturados antes;

- **`tol`:** especifica o limite de tolerância.

Os valores da **`tol`**, **`alpha`**, **`max iter`**, **`verbose`:** e **`hidden layer sizes`**, mantiveram-se iguais em todos os seguintes testes. Na tabela 3.1, pode encontrar o respetivo valor e a sua razão de existência.

Parâmetro	**Valor**	**Razão**
`tol`	1e-4	Não considerei significante ter uma diferença mais baixa
`alpha`	1e-5	Valor padrão, decidi não alterar
`max iter`	2000	como nunca superou as 1800 iterações, deixei a 2000
`verbose`	*true*	para poder receber um *feedback* do processo
`hidden layer sizes`	15	durante estes testes usei sempre apenas 15 neurónios

Tabela 3.1: Tabela de valores fixos nos testes da função `MLPClassifier`.

Testes no Solver

Representados e especificados na tabela 3.2, número de iterações e comentários.

Parâmetro	Iterações	Ac.max	Ac.min	Comentário
`adam`	313	0.99	0.89	muito idêntica ao `sgd`, mas largura do "rio"mais curta
`sgd`	714	0.99	0.90	melhor, mas fez mais iterações, largura do "rio"maior
`lbfgs`	1300	0.98	0.79	pior

Tabela 3.2: Tabela resumo dos Testes no parâmetro Solver.

Sendo o parâmetro `sgd` o que me apresentou, ligeiramente, o melhor resultado, decidi continuar os testes com ele. O atual função `MLPClassifier`, encontra-se da seguinte forma:

```
# MLPClassifier usado para classificar os dados
        NN = MLPClassifier(solver='sgd', alpha=1e-5, hidden_layer_sizes
            =15, max_iter=2000, verbose=True)
```

Excerto de Código 3.4: Função MLPClassifier após os primeiros testes.

Testes no Activation com solver='sgd'

Representados e especificados na tabela 3.3, número de iterações e comentários. Não fiquei satisfeito com os resultados, então testei o melhor resultado do Activation com o **solver='adam'** e conclui que seria uma solução, ligeiramente, melhor.

Parâmetro	Iterações	Ac.max	Ac.min	Comentário
`identity`	60	—	—	pior
`logistic`	885	—	—	algumas accuricys abaixo de 80%
`logistic adam`	420	0.99	0.92	melhor opção
`tanh`	374	—	—	segundo pior
`relu`	735	—	—	mau e o "rio"é o maior

Tabela 3.3: Tabela resumo dos Testes no parâmetro Activation.

Testes no batch size com solver='adam' e Activation='logistic'

Representados e especificados na tabela 3.4, número de iterações e comentários.

O valor do batch size pode afetar a desempenho e o tempo de treino do seu modelo. Em geral, um valor mais alto do batch size pode resultar treino rápido, mas pode originar um levar a resultados menos precisos. Um valor mais baixo do batch size pode levar a um treino mais lento, mas pode resultar em modelos mais precisos.

Usei alguns valores que considerei como comuns, após alguma pesquisa.

Parâmetro	Iterações	Ac.max	Ac.min	Comentário
100	233	0.99	0.89	—
15	224	—	—	—
32	243	—	—	—
64	448	—	—	—
128	253	—	—	Melhorou em todos os aspetos
256	369	—	—	—
512	640	—	—	—

Tabela 3.4: Tabela resumo dos Testes no parâmetro batch size.

Testes no learning rate com tudo o resto

Representados e especificados na tabela 3.5, número de iterações e comentários.

Parâmetro	Iterações	Ac.max	Ac.min	Comentário
adaptive	529	—	—	o melhor
constant	562	—	—	nada de especial
invscaling	243	—	—	alcança mais rapido um pico

Tabela 3.5: Tabela resumo dos Testes no parâmetro learning rate.

Testes no learning rate init com tudo o resto

Representados e especificados na tabela 3.6, número de iterações e comentários.

Se o learning rate for muito alto, o modelo pode "saturar"e não conseguir convergir para uma solução ótima. Se for muito baixo, o treino pode ser muito lento.

Usei alguns valores que considerei como comuns, após alguma pesquisa.

Parâmetro	Iterações	Ac.max	Ac.min	Comentário
0.1	15	—	—	péssimo
0.01	168	—	—	nada de especial
0.001	676	—	—	melhorou ligeiramente
0.0001	887	—	—	—

Tabela 3.6: Tabela resumo dos Testes no parâmetro learning rate init.

Testes no shuffle com tudo o resto

Fiz o teste com o valor "true", fez 396 iterações e não houve nenhuma alteração significativa, decidi ignorar.

3.3.4.2 Testes no Validation fraction

Representados e especificados na tabela 3.7, número de iterações e comentários.

A validação é um processo que ajuda a avaliar o desempenho do seu modelo durante o treino, para evitar o overfitting (quando o modelo tem um desempenho muito bom nos dados de treino, mas um desempenho mau nos dados de teste).

O valor de validation fraction deve ser um número entre 0 e 1. Um valor mais alto significa que mais dados serão usados para validação, se usar muitos dados para validação, terei menos dados disponíveis para o treino do modelo, o que pode afetar o desempenho do mesmo.

Parâmetro	Iterações	Ac.max	Ac.min	Comentário
0.1	462	—	—	—
0.2	485	—	—	—
0.3	366	—	—	—
0.8	334	—	—	—
0.99	—	—	—	—

Tabela 3.7: Tabela resumo dos Testes no parâmetro validation fraction.

3.3.4.3 Testes Finais

Função `MLPClassifier` final, representada no excerto de código 3.3.4.3.

```
NN = MLPClassifier(solver='adam', activation='logistic', alpha=1e-5,
    hidden_layer_sizes=(15,15), max_iter=2000, verbose=True, batch_size
    =128, learning_rate='adaptive', learning_rate_init=0.001)
```

Excerto de Código 3.5: Função MLPClassifier Final.

Dados estatísticos de tempo e iterações na tabela 3.15, imagens ilustrativas das curvas ROC finais e "rio"**??**.

Concluí que, claramente, existem muitas (infinitas) soluções potenciais para o problema. Encontrar o melhor modelo é um problema de otimização, cada linha tem uma configuração diferente, mas qual delas é a melhor de todas?

Ao conceber uma rede neuronal, há diferentes parametrizações que têm de ser escolhidas, com a possibilidade de determinar a eficácia do sistema. O

Neurónios	Iterações	Tempo	Comentário
15	—	+/-30m	—
(15,15)	1653	+/-34m, 1h20	—
(15,15,15)	—	56m	—
60	804	1h24	—
120	731	2h10	—

Tabela 3.8: Tabela resumo dos Testes Finais no parâmetro hidden layer sizes.

número de neurónios nas camadas de entrada/saída pode variar, um número demasiado curto pode não ser suficiente para modelar a superfície desejada.

Um valor demasiado elevado pode levar a um sobre ajustamento, Na prática, fiz testes...

No caso de Redes Neuronais com várias camadas, a equação, a função de custo e os derivados correspondentes não foram fáceis de obter. As funções de transferência de todos os nós demorava algum tempo. Uma parte fundamental de qualquer problema de otimização é perceber como podemos distinguir entre duas soluções potenciais, ou seja, podemos dizer que **"a solução A é melhor que a solução B"**?

O termo otimização refere-se a encontrar os parâmetros de um modelo para obter os melhores valores de saída. Na prática, refere-se à maximização/-minimização de uma função objetivo, através da variação dos valores possíveis dos parâmetros. Existem múltiplas correlações entre parâmetros

3.3.5 Curva ROC e AUC

Assumimos que a melhor ação depende apenas do estado atual do ambiente e do comportamento do agente inteligente.

AUC deriva da Curva ROC (é o mais importante para determinar a "qualidade"do resultado), esta segunda exprime a relação entre a atividade "verdadeira"e a atividade "falsa".

Um bom resultado é por exemplo: Vamos supor que temos **True Positive Rate = 50%** para **15% False Positive Rate**, mas seria melhor **TruePositive Rate = 90%** para **15% False Positive Rate**

Nunca posso dizer que tenho um acerto de 70% na AUC, é lógica a razão de não poder, mas uma AUC tem sempre mais de 50%.

0.3 é o desvio padrão da curva de ROC.

3.3.6 Valores de AUC, Testes Finais

A curva ROC reporta todas as parametrizações de desempenho possíveis do nosso modelo. Ao compara dois modelos, o melhor teria a curva ROC acima do outro a maioria das vezes.

Valores de AUC comparáveis: AUC = 1, seria o sistema perfeito, pois seria obtido um desempenho ótimo em todos os desempenhos possíveis. AUC = 0.5 é o pior sistema possível.

Para duas camadas de 15 neurónios cada uma, obtive:

Atividade	**Iterações**	***Accuracy***
Downstairs	511	0.93 +/- 0.03
Jogging	872	0.94 +/- 0.03
Sitting	743	0.99 +/- 0.03
Standing	678	0.99 +/- 0.03
Upstairs	419	0.90 +/- 0.03
Walking	685	0.85 +/- 0.03

Tabela 3.9: Tabela resumo dos Testes Finais e Valores de AUC.

3.3.7 Resultados Matriz de Confusão

Conhecida como Matriz de Erros. esta tabela resume o desempenho do modelo, fornecendo mais informações que o simples valor de "precisão". Esta tabela é fiel aos gráficos finais apresentados neste relatório, mas não invalida todo o trabalho de pesquisa exercido ao longo deste projeto.

A tabela contem duas filas e duas colunas, representam o número de falsos positivos, falsos negativos, verdadeiros positivos e verdadeiros negativos. Cada linha corresponde a um resultado previsto e cada coluna corresponde à realidade.

		Valor Predito	**Valor Predito**
		Sim	**Não**
Real	**Sim**	954155	18858
Real	**Não**	90137	9865

Tabela 3.10: Soma dos Resultados da Matriz de Confusão da atividade "Downstairs".

		Valor Predito	**Valor Predito**
		Sim	**Não**
Real	**Sim**	693066	55444
Real	**Não**	52396	272109

Tabela 3.11: Soma dos Resultados da Matriz de Confusão da atividade "Jogging".

		Valor Predito	**Valor Predito**
		Sim	**Não**
Real	**Sim**	1011719	1433
Real	**Não**	16244	43619

Tabela 3.12: Soma dos Resultados da Matriz de Confusão da atividade "Sitting".

		Valor Predito	**Valor Predito**
		Sim	**Não**
Real	**Sim**	1019727	5104
Real	**Não**	16259	31925

Tabela 3.13: Soma dos Resultados da Matriz de Confusão da atividade "Standing".

		Valor Predito	**Valor Predito**
		Sim	**Não**
Real	**Sim**	902808	47647
Real	**Não**	103950	18610

Tabela 3.14: Soma dos Resultados da Matriz de Confusão da atividade "Upstairs".

		Valor Predito	**Valor Predito**
		Sim	**Não**
Real	**Sim**	569241	85873
Real	**Não**	132311	285590

Tabela 3.15: Soma dos Resultados da Matriz de Confusão da atividade "Walking".

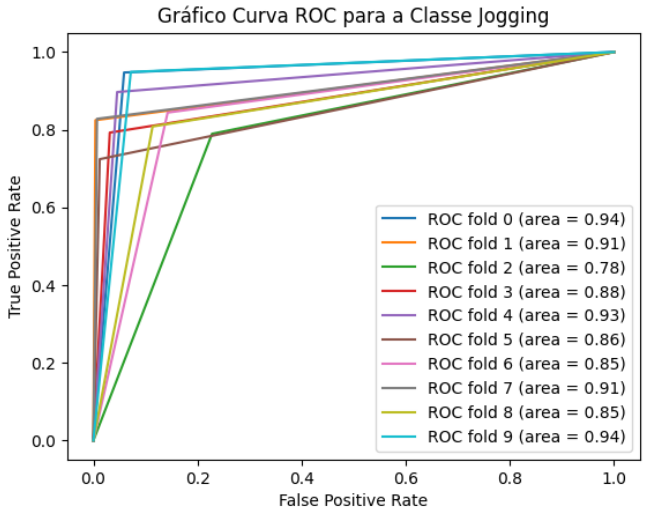


Figura 3.2: Representação da curva ROC da atividade "Jogging"final.

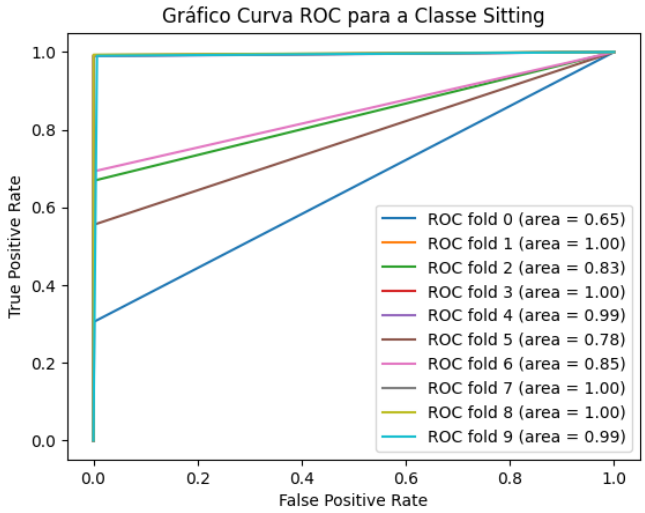


Figura 3.3: Representação da curva ROC da atividade "Sitting"final.

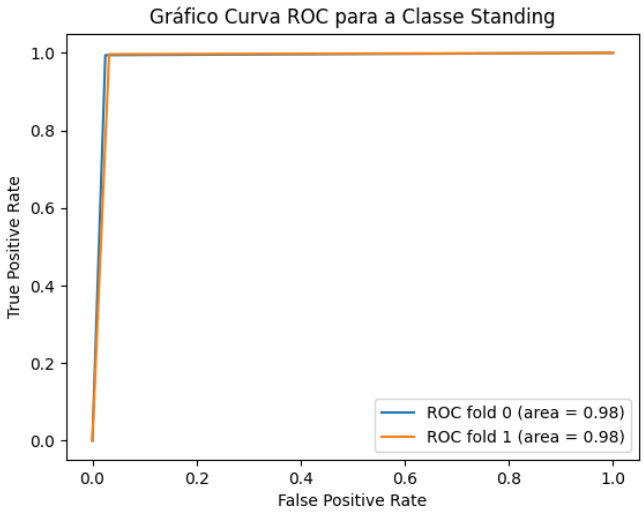


Figura 3.4: Representação da curva ROC da atividade "Sitting"com duas camadas de 15 neurónios em dois folds.

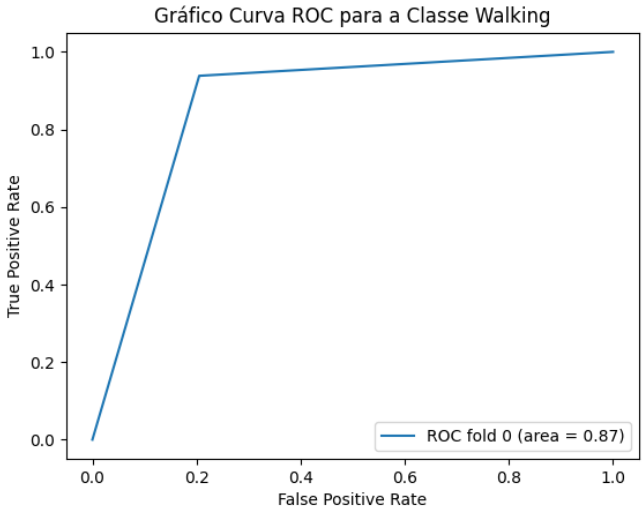


Figura 3.5: Representação da curva ROC da atividade "Walking"com uma camada de 60 neurónios em apenas um fold.

Capítulo

4

Conclusões e Trabalho Futuro

4.1 Conclusões Principais

Ao longo deste trabalho, após inúmeros dias há espera de uma simples resposta, de um simples *output*, pude concluir que o que absorvi mais deste trabalho foram os conceitos que o próprio induziu transmitir.

Não interessa (como quem diz) a topologia da rede, mais neurónios menos neurónios, o importante é nós percebermos quantos segundos são necessários para obtermos uma diferença significativa nos resultados. Todos nós analisamos em função do tempo 0.1, 0.5, 1, então e 1.5, 2... é importante saber como os dados se comportam em função do tempo. Como conclui, um número demasiado curto de "parâmetros"pode não ser suficiente para modelar a solução desejada, mas um valor demasiado alto pode levar a um sobre-ajustamento.

Em relação à área (AUC), nunca posso dizer que acerto 80%, 90%, pois a procura de todas as soluções possíveis não é viável, ou porque existe um infinito potencial número de estados, ou por esse número ser simplesmente demasiado elevado. Assumimos que a melhor ação depende apenas do estado atual do ambiente e do comportamento do agente inteligente.

Os resultados são dados num cálculo ("media +/- desvio padrão) que representam os valores de desempenho, por exemplo, "0.7 +/- 0.02"significa que se espera que o modelo tenha um bom desempenho 70% das vezes, com variações "típicas"de mais ou menos 2%.

Claramente, existem muitas (infinitas) soluções potenciais para o problema. Encontrar o melhor modelo é um problema de otimização, cada linha tem uma configuração diferente, mas qual delas é a melhor de todas?

O termo otimização refere-se a encontrar os parâmetros de um modelo

para obter os melhores valores de saída. Na prática, refere-se à maximização/-minimização de uma função objetivo, através da variação dos valores possíveis dos parâmetros. Existem múltiplas correlações entre parâmetros

Por fim, e citando a Lei de Parsimony, "é mais provável que soluções mais simples sejam corretas do que soluções complexas", ao comparar as hipóteses concorrentes para resolver um problema, deve-se selecionar a solução mais simples. O mesmo também disse: "Não devem ser usadas mais coisas do que as necessárias", que, em termos práticos, afirma que modelos simples devem (em caso de eficácia comparável), ser preferidos aos mais complexos.

Foi com que o trabalhei para obter os resultados finais, preferi rodar duas camadas de 15 neurónios, demorou-me por volta de 34m por cada fold, enquanto que a usar 60 neurónios numa camada apenas, demoraria 1h30, os resultados não se mostravam significativamente diferentes.

4.2 Trabalho Futuro

Num trabalho futuro, pretendo anexar todos os gráficos que foram fruto dos testes que executei, como forma de permitir uma apreciação visual e crítica, por parte do exterior.

O agente inteligente deve melhorar a capacidade de raciocinar sobre uma potencial ação antes de a executar, melhorar a análise dos estados, estabelecer objetivos próprios e inferir novos conhecimentos

Atualmente, os modelos revolucionários baseados em quadros de aprendizagem profunda têm um número enorme de parâmetros, por exemplo, a rede VGG-16, proposta em 2014, tem 138.000.000 parâmetros, seria aumentar a análise dos dados neste projeto.