Politécnico Castelo Branco
Escola Superior de Tecnologia

Escola Superior de Tecnologia
Curso de 1 º Ciclo em Engenharia Informática
1º Semestre do 2º Ano
Unidade Curricular: Inteligência Artificial
Docentes: Ana Paula Silva & Miguel magueijo
Trabalho Prático

Trabalho Prático

Treino de uma Rede Neuronal para Previsão de Atrasos em Voos

1. Objetivo

O objetivo deste trabalho é aplicar os conceitos estudados sobre agentes aprendizes através do treino de uma **rede neuronal multicamada**, com uma camada escondida, para resolver uma tarefa real de **classificação binária**.

O propósito é desenvolver um modelo capaz de prever se um voo irá sofrer atraso, com base em atributos relevantes do voo (companhia aérea, aeroporto, horário, distância, entre outros).

Os estudantes deverão:

- integrar o seu código nas estruturas fornecidas pelo docente,
- implementar o pré-processamento dos dados (incluindo codificação "one-hot" e normalização),
- treinar e testar a rede neuronal,
- analisar experimentalmente o comportamento do modelo.

2. Funcionamento

- O trabalho deverá ser realizado em grupos de três alunos.
- Deverá ser implementado utilizando a linguagem usada nas aulas práticas (Python).
- Não podem ser usadas classes que não foram introduzidas nas aulas, com exceção das necessárias para a construção dos gráficos pedidos.

3. O Problema

Neste trabalho pretendemos implementar um agente baseado em redes neuronais capaz de prever se um voo sofrerá atraso com base num conjunto de atributos: É fornecido código que define uma rede neuronal multicamada e implementa o mecanismo de treino por retro-propagação do erro. Também é fornecido código exemplificativo de como utilizar uma rede neuronal para imitar funções booleanas simples como o AND, o OR e o XOR.

Pretende-se adaptar este código para treinar uma rede capaz de receber os atributos de um voo e prever se o mesmo sofre ou não de atraso. Para o treino é fornecido um ficheiro: DataSet1.csv, com 1000 exemplos de voo.

O *dataset* inclui registos de voos simulados com informação operacional relevante para estimar a probabilidade de atraso. Cada instância corresponde a um voo. Os atributos foram selecionados de forma a evitar fuga de informação (*data leakage*) para garantir que a rede neuronal não utiliza atributos que revelem diretamente o atraso real do voo.

Os atributos são os seguintes:

1. Month

Descrição: Mês em que o voo ocorre, representado como um número inteiro entre 1 e 12.

Tipo: Numérico discreto (ordinal).

Justificação: O mês do ano influencia sazonalmente o tráfego aéreo e a probabilidade de atrasos (ex.: inverno, férias de verão).

Pré-processamento recomendado: Normalização Min–Max ou divisão por 12.

---

2. DayOfWeek

Descrição: Dia da semana em que o voo ocorre, representado como um inteiro de 1 (segunda-feira) a 7 (domingo).

Tipo: Numérico discreto (ordinal).

Justificação: Certos dias apresentam maior fluxo de passageiros e um padrão distinto de atrasos.

Pré-processamento recomendado: Normalização Min–Max ou divisão por 7.

---

3. CRSDepTime

(CRS = Computer Reservation System)

Descrição: Hora programada de partida, representada no formato inteiro HHMM.

Tipo: Numérico (temporal).

Justificação: A hora do dia está fortemente associada aos níveis de tráfego e, portanto, ao risco de atraso.

Não contém fuga de informação, pois é a hora planeada e não a real.

Pré-processamento recomendado:

Converter para minutos desde o início do dia:

$$HHMM \rightarrow 60 \times HH + MM$$

---

4. UniqueCarrier

Descrição: Código IATA da companhia aérea responsável pelo voo (por exemplo, AA, DL, UA, WN, B6).

Tipo: Categórico nominal.

Justificação: Companhias aéreas têm históricos de pontualidade diferentes, influenciando a probabilidade de atraso.

Pré-processamento recomendado: *One-hot encoding*. (é explicado mais à frente)

---

5. Origin

Descrição: Código IATA do aeroporto de origem (ex.: JFK, ATL, ORD).

Tipo: Categórico nominal.

Justificação: Aeroportos variam significativamente em tráfego, condições meteorológicas e eficiência operacional.

Pré-processamento recomendado: *One-hot encoding*.

---

6. Dest

Descrição: Código IATA do aeroporto de destino.

Tipo: Categórico nominal.

Justificação: O aeroporto de destino também tem impacto no padrão de atrasos.

Pré-processamento recomendado: *One-hot encoding*.

---

7. Distance

Descrição: Distância em milhas entre o aeroporto de origem e o destino.

Tipo: Numérico contínuo.

Justificação: Voos curtos e longos têm diferentes comportamentos de atraso (por exemplo, voos curtos são mais sensíveis ao congestionamento).

Pré-processamento recomendado: Normalização Min–Max.

---

16. Delayed

Descrição: Rótulo binário da classe:

- **0** → voo não atrasado
- **1** → voo atrasado

Tipo: Categórico binário (variável alvo).

Notas: Esta é a variável que a rede neuronal deverá aprender a prever.

O último atributo é a classe a que pertence o voo (atrasado ou não atrasado) e não deverá ser utilizado no treino. Os restantes são de dois tipos: numéricos e categóricos e deverão ser utilizados para criar as entradas para a rede. Os categóricos deverão ser transformados numa representação adequada, uma vez que as entradas de uma rede neuronal têm que ser valores numéricos.

A rede deve ter **2 neurónios de saída**, representando as classes:

Classe 0 → "não atrasado"

Classe 1 → "atrasado"

A codificação recomendada (e mais simples para treino) é:

- **Delayed = 0** → vetor alvo **1, 0**
- **Delayed = 1** → vetor alvo **0, 1**

[0.92, 0.08] → rede prevê "não atrasado"

[0.10, 0.90] → rede prevê "atrasado"

Observação: A rede poderia ter apenas uma unidade na camada de saída, mas para este trabalho vamos considerar duas classes e por isso duas unidades na camada de saída. Permite mostrar claramente a noção de ***one-hot encoding*** no rótulo e mantém consistência com muitos *frameworks* e exemplos de classificação multiclasse.

4. Código disponibilizado

Antes de iniciar o trabalho propriamente dito, o estudante deverá analisar o código fornecido, o qual está comentado de forma a descrever o seu funcionamento. Deverá ainda testar o funcionamento das funções `train_and`, `train_or` e `train_xor`. Ao fazer 'Evaluate' do ficheiro fornecido já irá observar o treino de uma rede para aprender a função lógica AND e depois a resposta do modelo treinado quando lhe são fornecidas as combinações lógicas possíveis. É importante que nesta fase os estudantes testem o código para diferentes valores de número de épocas e para as diferentes funções booleanas.

5. Construção dos Conjuntos de Treino e Teste

Numa primeira fase vamos implementar a função `build_sets` e uma função auxiliar chamada `translate` que irá ser chamada pela anterior. A função `build_sets` cria os conjuntos de treino e teste que irão ser utilizados no treino e avaliação da rede neuronal a partir dos dados armazenados no ficheiro **DataSet1.csv.**

Como já foi referido, estes ficheiros apresentam informação sobre 1000 voos. A função `build_sets` deve receber como argumento o número de exemplos que se pretende que sejam usados para os conjuntos de treino e de teste. A função deve ler cada linha e transformá-la numa lista de valores, tendo em atenção o tipo dos valores lidos. A lista é posteriormente passada como argumento à função `translate`, a qual construirá um padrão de treino no formato adequado, discutido mais abaixo. A lista resultante será armazenada numa lista de padrões, cuja ordem deve ser randomizada (consultar método `shuffle` disponível no módulo `random`). Finalmente, a função deverá devolver duas listas, o conjunto de treino e o conjunto de teste de tamanhos iguais aos valores recebidos como argumento.

A função `translate` recebe cada lista de valores e transforma-a num padrão de treino. Cada padrão é uma lista com o formato [`padrao_de_entrada, classe_do_vôo, padrao_de_saida`], em que os elementos da lista são obtidos como se explica em baixo.

Para a codificação devem ser seguidas as indicações da seguinte tabela:

Atributo	Tipo Original	Descrição	Codificação Recomendada
Month	Numérico discreto (ordinal)	Mês do voo (1–12).	Normalização Min–Max ou divisão por 12.
DayOfWeek	Numérico discreto (ordinal)	Dia da semana (1 = segunda, 7 = domingo).	Normalização Min–Max ou divisão por 7.
CRSDepTime	Numérico (temporal HHMM)	Hora *programada* de partida. Não contém atraso.	Converter HHMM → minutos; depois normalizar.
UniqueCarrier	Categórico nominal	Código da companhia aérea (AA, DL, UA, WN...).	One-hot encoding (1 neurónio por companhia).
Origin	Categórico nominal	Aeroporto de origem (JFK, LAX, ATL...).	One-hot encoding.
Dest	Categórico nominal	Aeroporto de destino.	One-hot encoding.
Distance	Numérico contínuo	Distância entre origem e destino (milhas).	Normalização Min–Max.
Delayed (classe)	Categórico binário	0 = voo não atrasado; 1 = voo atrasado.	One-hot encoding da classe: 0 → [1,0], 1 → [0,1].

Exemplo de Registo de Voo (antes da codificação)

Month = 3

DayOfWeek = 5

CRSDepTime = 1430

UniqueCarrier = "DL"

Origin = "JFK"

Dest = "ATL"

Distance = 760

Delayed = 1

A. Conversão do tempo CRSDepTime (HHMM → minutos)

CRSDepTime = 1430 → 14h30

14 × 60 + 30 = 870 minutos

B Normalização dos atributos numéricos

Suponhamos que os valores mínimos e máximos no dataset são:

Atributo	Min	Max
Month	1	12
DayOfWeek	1	7
CRSDepTime(min)	0	1439
Distance	150	3000

A normalização Min–Max:

$$x' = \frac{x - \min}{\max - \min}$$

Aplicando:

- Month_norm = (3 - 1) / 11 = 0.1818
- DayOfWeek_norm = (5 - 1) / 6 = 0.6667
- CRSDepTime_norm = 870 / 1439 ≈ 0.6046
- Distance_norm = (760 - 150) / 2850 ≈ 0.2140

C. One-hot encoding de atributos categóricos

UniqueCarrier

Conjunto de companhias no dataset (exemplo):

["AA", "B6", "DL", "UA", "WN"]

Logo:

UniqueCarrier = "DL" → [0, 0, 1, 0, 0]

Origin

Conjunto de aeroportos (exemplo):

["ATL", "DEN", "JFK", "LAX", "MIA", "ORD", "SFO", "SEA"]

Então:

Origin = "JFK" → [0, 0, 1, 0, 0, 0, 0, 0]

Dest

Dest = "ATL" → [1, 0, 0, 0, 0, 0, 0, 0]

D. Codificação da classe (saída desejada)

Usamos *one-hot*:

Delayed = 1 → [0, 1]

E. Vetor Final de Entrada da Rede Neuronal

Concatenamos:

1. atributos numéricos normalizados
2. one-hot do UniqueCarrier
3. one-hot do Origin
4. one-hot do Dest

Entrada (x):

```
[
  0.1818,      # Month_norm
  0.6667,      # DayOfWeek_norm
```

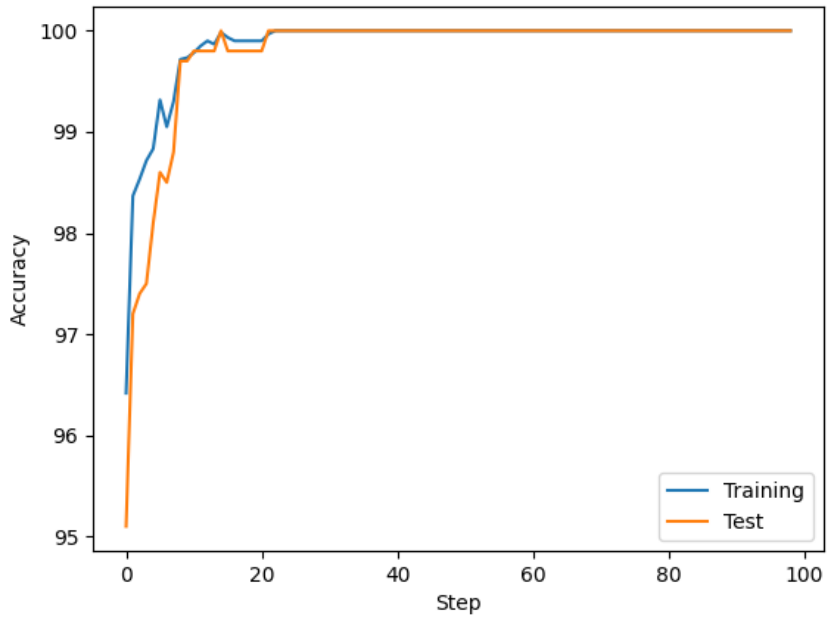
```
  0.6046,      # CRSDepTime_norm
  0.2140,      # Distance_norm
  # One-hot UniqueCarrier (DL)
  0, 0, 1, 0, 0,
# One-hot Origin (JFK)
  0, 0, 1, 0, 0, 0, 0, 0,
  # One-hot Dest (ATL)
  1, 0, 0, 0, 0, 0, 0, 0
]
```

Saída (y):

```
[0, 1]
```

6. Treino da Rede Neural

Nesta segunda fase vamos implementar três funções. A primeira função chama-se `train_flights`, recebe o conjunto de treino e o número de épocas (e outros parâmetros documentados no código) e deve criar a rede neuronal e chamar a função `iterate` para a treinar durante **o número de épocas indicado no argumento**. Esta função funciona de forma semelhante às funções de treino fornecidas no código (para o AND, OR e XOR), devendo, em cada iteração de treino, percorrer todos os padrões armazenados no conjunto de treino, chamando a função `iterate`, a qual recebe um padrão de entrada (lista dos atributos de um voo) e de saída (lista com a codificação binária da classe do voo).



Esta função deve ainda testar, no final de cada época, a rede tanto com o conjunto de treino como de teste (`test_flights`) e ir armazenando os valores retornados (estrutura de dados do tipo dicionário) em listas para no fim do treino, usando o **package matplotlib**, apresentar os gráficos da exatidão, precisão, cobertura e F1-Score da rede nos dois conjuntos, semelhante ao da figura acima.

A segunda função, `test_flights`, tem como objetivo testar o desempenho da rede treinada. Para tal deve percorrer o conjunto de exemplos recebido como argumento, e, para cada padrão, chamar a função `forward` passando como argumentos a rede treinada e a lista de atributos do estudante. De seguida, a função deve analisar a lista de saídas da rede (net['y']), chamando a terceira função a implementar (`retranslate`). Esta função deve devolver a classe do voo correspondente à saída da rede com maior valor. A função `test_flights` deve comparar o valor devolvido pela função auxiliar com a verdadeira classe do voo, armazenada no padrão de treino, calculando:

- a percentagem de respostas corretas (exatidão);
- o nº de Falsos Positivos (FP); Verdadeiros Positivos (VP), Falsos Negativos (FN), Verdadeiros Negativos (VN);
- a precisão;
- a cobertura;
- F1-Score.

Para cada padrão deve ainda imprimir uma linha semelhante à seguinte:

```
A rede prevê que o voo nº 3 não está atrasado e na realidade ele está atrasado.
```

Após a impressão das linhas com o resultado referente a cada padrão de treino, deve ainda imprimir o valor do desempenho final da rede (percentagem de respostas corretas), por exemplo:

```
Success rate: 94.12
```

Estas impressões só devem ocorrer se o parâmetro **`printing`** for **`True`**, se tal não acontecer a função deve devolver apenas um dicionário com as seguintes chaves:

```
{"exatidão": # a taxa de sucesso da rede,
 "matriz";. #uma lista com a seguinte estrutura [VP, FP, VN, FN],
"precisão": # valor da precisão,
"cobertura": #valor da cobertura",
"F1-score": #valor da F1-Score }
```

Por último, a função `run_flights`, que será a função principal do nosso programa, deverá reunir as funções anteriormente implementadas: desde a construção dos conjuntos de treino e teste, até ao treino e teste da rede. Deve receber como argumento o *dataset*, o número de épocas que irá ser usado para o treino, o número de exemplos no conjunto de treino e o número de instâncias no conjunto de teste, além dos outros parâmetros presentes no código fornecido.

7. O que deve ser entregue

Os alunos devem submeter no moodle o ficheiro .py com o código desenvolvido devidamente comentado (os nomes dos alunos devem ser colocados em comentário no início do ficheiro), assim como um ficheiro PDF com um pequeno relatório, onde apresentam os resultados obtidos no trabalho experimental realizado.

8. Trabalho Experimental

Os estudantes devem realizar um conjunto de experiências sistemáticas com a rede neuronal construída, analisando o seu comportamento e comparando desempenhos.

Cada experiência deve isolar a variável em estudo, mantendo todos os restantes parâmetros **constantes**.

Os resultados devem ser apresentados em tabelas e gráficos, acompanhados de **análise crítica**.

8.1 Parâmetros Base (por defeito)

Para todas as experiências, salvo indicação contrária, devem ser usados os seguintes valores:

- **Número de neurónios na camada escondida:** 8
- **Número de épocas:** 50
- **Taxa de aprendizagem:** 0.01
- **Proporção treino/teste:** 80% / 20%
- **Normalização:** ativa (Min–Max)
- **Conjunto completo de atributos permitidos**

8.2 Experiências Obrigatórias

8.2.1 Variação do número de neurónios na camada escondida

Testar os seguintes valores:

- 4 neurónios
- 8 neurónios
- 12 neurónios
- 20 neurónios

Parâmetros fixos nesta experiência:

- nº de épocas = **50**
- taxa de aprendizagem = **0.01**
- normalização ativa
- treino/teste = **80% / 20%**
- atributos completos

Objetivo: avaliar como a dimensão da camada escondida afeta a aprendizagem, subajuste e sobreajuste.

8.2.2 Variação do número de épocas de treino

Testar:

- 10 épocas
- 50 épocas

- 100 épocas
- 200 épocas

Parâmetros fixos nesta experiência:

- nº de neurónios escondidos = **8**
- taxa de aprendizagem = **0.01**
- normalização ativa
- treino/teste = **80% / 20%**

Objetivo: analisar convergência, estabilidade e risco de sobreajuste.

8.2.3 Impacto da normalização

Executar duas versões:

1. Com normalização Min–Max
2. Sem normalização

Parâmetros fixos nesta experiência:

- nº de neurónios = **8**
- épocas = **50**
- taxa de aprendizagem = **0.01**
- treino/teste = **80% / 20%**

Objetivo: analisar impacto da normalização dos atributos numéricos..

8.2.4 Diferentes tamanhos dos conjuntos de treino e teste

Testar:

- 70% treino / 30% teste
- 80% treino / 20% teste
- 90% treino / 10% teste

Parâmetros fixos nesta experiência:

- nº de neurónios = **8**
- épocas = **50**
- taxa de aprendizagem = **0.01**
- normalização ativa

Objetivo: analisar impacto do tamanho do conjunto de treino na capacidade de generalização.

8.2.5 Remoção seletiva de atributos

Testar as seguintes configurações:

1. Remover **Distance**
2. Remover **CRSDepTime**
3. Remover **UniqueCarrier**
4. Remover **Origin** e **Dest** simultaneamente

Parâmetros fixos nesta experiência:

- nº de neurónios = **8**
- épocas = **50**
- taxa de aprendizagem = **0.01**
- normalização ativa
- treino/teste = **80% / 20%**

Objetivo: identificar quais atributos têm maior importância para a previsão.

8.2.6 Variação da taxa de aprendizagem

Testar:

- 0.1
- 0.01
- 0.001

Parâmetros fixos nesta experiência:

- nº de neurónios = **8**
- épocas = **50**
- normalização ativa
- treino/teste = **80% / 20%**

Objetivo: estudar a influência da *learning rate* na rapidez e estabilidade da convergência.

6.2.7 Análise da matriz de confusão

Esta experiência deve ser realizada **apenas** para a melhor configuração encontrada pelos alunos.

Devem apresentar:

- matriz de confusão
- precisão, recall, F1-score
- análise dos erros (falsos positivos e falsos negativos)

Parâmetros:

- utilizar a melhor combinação identificada nas experiências anteriores.

Matriz de Confusão

A matriz de confusão compara as **previsões da rede neuronal** com as **classes reais** do conjunto de teste. Para problemas binários (como "Delay = 0 ou 1"), a matriz tem a seguinte forma:

	Previsto = 0	**Previsto = 1**
Real = 0	TN (Verdadeiros Negativos)	FP (Falsos Positivos)
Real = 1	FN (Falsos Negativos)	TP (Verdadeiros Positivos)

Significado:

- **TP**: modelo previu atraso e o voo estava atrasado
- **FP**: modelo previu atraso mas o voo NÃO estava atrasado
- **TN**: modelo previu "não atrasado" e estava realmente sem atraso
- **FN**: modelo previu "não atrasado", mas o voo estava atrasado

Precisão (Precision)

A precisão indica, entre todos os casos que o modelo previu como atrasados, quantos estavam realmente atrasados.

$$\text{Precisão} = \frac{TP}{TP + FP}$$

Valores altos significam:
poucos falsos alarmes; o modelo é "confiável" quando diz que haverá atraso.

Cobertura (Recall)

A **cobertura** (também chamada de *sensibilidade*) indica, entre todos os voos que estavam realmente atrasados, **quantos o modelo conseguiu identificar**.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Valores altos significam:

- o modelo raramente falha atrasos;
- poucos atrasos "passam despercebidos".

Exemplo numérico

Suponhamos que no teste obtivemos:

- TP = 80
- FP = 20
- TN = 70
- FN = 30

Precisão:

$$\frac{80}{80+20} = 0.80$$

O modelo acerta **80% das vezes em que afirma que há atraso**.

Recall:

$$\frac{80}{80+30} = 0.727$$

O modelo identifica **72.7% dos voos realmente atrasados**.

F1-Score

A **F1-score** é uma medida que combina a **Precisão (Precision)** e a **Cobertura (Recall)** numa única métrica. É especialmente útil em datasets **desbalanceados** ou quando queremos equilibrar:

- a capacidade do modelo de **evitar falsos positivos** (Precisão)
- a capacidade do modelo de **detetar verdadeiros positivos** (Recall)

A F1-score corresponde à **média harmónica** entre Precisão e Recall.

Fórmula da F1-score

$$F1 = 2 \times \frac{\text{Precisão} \times \text{Recall}}{\text{Precisão} + \text{Recall}}$$

Exemplo numérico

Se:

- Precisão = 0.80
- Recall = 0.727

Então:

$$F1 = 2 \times \frac{0.80 \times 0.727}{0.80 + 0.727}$$

$$F1 = 2 \times \frac{0.5816}{1.527} \approx 0.76$$

Interpretação

- Valores próximos de **1** indicam um modelo equilibrado e eficaz.
- Valores mais baixos revelam que o modelo falha demasiado em precisão, recall, ou ambos.
- A F1-score é robusta em situações onde a classe positiva é minoritária.

6.3 Organização e Apresentação dos Resultados

O relatório deve incluir:

Tabelas

- Comparação de desempenho entre configurações
- Valores de exatidão, precisão, cobertura e F1-Score em treino e teste
- Resumo das experiências

Gráficos

- Curvas da exatidão, precisão, F1-score e cobertura no final de cada época.
- Comparação entre configurações relevantes

Discussão crítica (obrigatória)

O aluno deve refletir sobre:

- qual a arquitetura mais eficaz e porquê
- se ocorreu sobreajuste ou subajuste
- importância dos atributos
- impacto da normalização
- influência da taxa de aprendizagem
- limitações do modelo e propostas de melhoria

---

9. Estrutura do Relatório

O relatório deve apresentar de forma clara, organizada e fundamentada todo o trabalho desenvolvido. Deve ser escrito de forma objetiva, usando linguagem técnica adequada e incluindo tabelas, gráficos e análise crítica dos resultados.

O relatório deve conter as seguintes secções:

Introdução

Nesta secção, o aluno deve:

- apresentar o objetivo geral do trabalho: construir, treinar e avaliar uma rede neuronal para prever atrasos em voos;
- descrever brevemente o problema abordado;
- indicar o dataset utilizados (indicar que é um dataset sintético preparado para a unidade curricular);
- apresentar uma visão global da estrutura do relatório.

Descrição do Dataset e dos Atributos

O aluno deve:

- descrever os atributos usados como entrada e a variável alvo;
- indicar o tipo de cada atributo (numérico, categórico, binário...);
- incluir a tabela dos atributos e codificações usada no pré-processamento.

Pré-processamento dos Dados

O aluno deve explicar, de forma clara:

- como transformou o atributo temporal **CRSDepTime** (HHMM → minutos → normalização);
- como normalizou os atributos numéricos (qual método usou);
- como fez o *one-hot encoding* dos atributos categóricos;
- qual a dimensão final do vetor de entrada;
- como tratou o embaralhamento e divisão treino/teste.

Um pequeno exemplo numérico deve ser incluído.

Arquitetura da Rede Neuronal

Esta secção deve incluir:

- representação da arquitetura final (ex.: input → hidden(8) → output(2));
- função de ativação usada na camada escondida;

- função de ativação da camada de saída;
- explicação da codificação da classe (*one-hot*);

Deve ser incluído um pequeno desenho/esquema da arquitetura (opcional mas recomendado).

Trabalho Experimental

O aluno deve apresentar:

- todas as experiências obrigatórias pedidas no enunciado;
- tabelas com resultados (treino e teste);
- gráficos relevantes;
- comparação entre configurações;
- identificação da melhor arquitetura e justificação.

Aqui devem ser aplicadas boas práticas de análise:

Não basta apresentar números — é necessário **interpretá-los**.

Análise da Matriz de Confusão

O aluno deve:

- apresentar a matriz de confusão para a melhor configuração;
- discutir:
    - falsos positivos;
    - falsos negativos;
    - cobertura;
    - precisão;
    - F1-Score.
- explicar porque aquela configuração é a melhor.

Discussão e Conclusões

Esta secção deve incluir:

- interpretação global dos resultados;
- limitações do modelo;
- como o modelo poderia ser melhorado (por exemplo: mais atributos, mais dados, outras arquiteturas, regularização);
- reflexão sobre o que foi aprendido.

---

10. Defesa do Trabalho Prático

O trabalho prático está sujeito a **defesa individual obrigatória**. Cada estudante deve demonstrar domínio completo sobre o trabalho que entregou, incluindo todos os aspetos de programação, pré-processamento, implementação da rede neuronal e análise dos resultados.

Durante a defesa, o estudante deve ser capaz de:

1. **Explicar qualquer parte do código implementado**, justificando as decisões tomadas e relacionando-as com o objetivo do trabalho. Não é suficiente ler ou descrever o código; o estudante deve evidenciar compreensão real do funcionamento do mesmo.
2. **Demonstrar conhecimento sobre redes neuronais artificiais**, incluindo a interpretação dos resultados obtidos, e dos fenómenos observados nas experiências (como sobreajuste, subajuste, influência dos parâmetros, importância dos atributos, etc.).
3. **Responder a questões sobre o processo experimental**, justificando diferenças de desempenho entre configurações e explicando o impacto dos parâmetros variáveis (número de neurónios, taxa de aprendizagem, normalização, entre outros).
4. **Evidenciar autoria e domínio do trabalho realizado**. Trabalhos cuja defesa revele desconhecimento substancial sobre o código, o dataset, os procedimentos ou os resultados serão considerados não válidos, podendo o estudante não obter os mínimos exigidos, independentemente da qualidade da entrega escrita.

Durante a defesa, o docente poderá solicitar ao estudante que realize pequenas modificações no código, com o objetivo de verificar a sua compreensão e autonomia na implementação.

Estas modificações serão simples e relacionadas com parâmetros ou componentes já presentes no trabalho, mas não serão divulgadas previamente para garantir que o estudante domina verdadeiramente o código que entregou.

O estudante deve, portanto, estar preparado para:

- compreender o fluxo do código e o papel de cada função;
- localizar e ajustar parâmetros relevantes;
- interpretar o impacto das alterações solicitadas;
- executar e explicar novamente os resultados após a modificação.

A incapacidade de realizar estas alterações pode indicar que o estudante não domina o trabalho apresentado, podendo resultar na reprovação.