Universidade do Minho
Mestrado em Engenharia Informática

Dados e Aprendizagem Automática

Conceção e otimização de modelos de Machine Learning

2021/22

Grupo 35

Carolina Vila Chã - PG47100
Joel Martins - PG47347
Sofia Santos - A89615
Carlos Ferreira - PG47087

Conteúdo

Introdução

Com este trabalho prático, pretende-se que sejam aplicados os conhecimentos adquiridos ao longo da UC de Dados e Aprendizagem Automática. Mais especificamente, devemos explorar, modelar e analisar dois *datasets* distintos. Um destes *datasets*, fornecido pela equipa docente, consiste em informação acerca do tráfego rodoviário de acordo com as condições climatéricas ou com a altura do dia, por exemplo. O outro dataset, escolhido por nós, detalha as várias características de uma música, como o número de batidas por minuto ou a duração, de acordo com o seu género.

O nosso objetivo principal é, a partir do primeiro *dataset*, sermos capazes de prever o fluxo de tráfego com base na restante informação que temos acerca da via e do ambiente ao seu redor. Com o segundo *dataset*, o objetivo é também o de prever, mas neste caso de prever o género de uma música com base nas suas outras características.

Para atingir estes objetivos, deveremos recorrer a algoritmos e modelos de Machine Learning, mais especificamente modelos de Aprendizagem Supervisionada, utilizando a linguagem de programação Python e as suas bibliotecas próprias para lidar com este tipo de problemas.

Metodologia

A metodologia escolhida para abordar os problemas pretendidos é bastante semelhante a metodologia dada nas aulas **SEMMA** e consistiu resumidamente em 5 passos:

- Carregar o *dataset* a ser trabalhado.
- Explorar o *dataset* recorrendo a vários gráficos.
- Modificar o *dataset* de forma a prepará-lo para a próxima fase.
- Modelar usando o *dataset* diversos modelos que nos foram expostos durante o semestre.
- Avaliar qual dos modelos obteve resultados mais precisos.

Durante a fase de modelação foram aplicados diversos testes com diferentes hiper-parâmetros para cada modelo de modo a otimizar ao máximo a precisão dos resultados obtidos.

Datasets

1 Dataset do trânsito

Podemos resumir o dataset do trânsito, relativo ao trânsito na cidade do Porto, com a seguinte tabela, cujos dados foram obtidos com a função `data.info()`, onde `data` é a variável na qual foi carregado o dataset.

Coluna	Descrição	Tipo de dados	Valores não nulos
city_name	Nome da cidade.	Texto	6812
record_date	Data na qual o registo foi feito.	Texto	6812
AVERAGE_SPEED_DIFF	Diferença entre a velocidade real dos carros e a velocidade máxima que poderiam atingir se não se registasse trânsito.	Texto	6812
AVERAGE_FREE_FLOW_SPEED	Velocidade máxima média que os carros poderiam atingir se não se registasse trânsito.	Numérico	6812
AVERAGE_TIME_DIFF	Diferença entre o tempo que os carros demoram a percorrer um dado conjunto de ruas e o tempo que demorariam se não se registasse trânsito.	Numérico	6812
AVERAGE_FREE_FLOW_TIME	Tempo médio que os carros demorariam a percorrer um dado conjunto de ruas se não se registasse trânsito.	Numérico	6812
LUMINOSITY	Nível de luminosidade na via no momento do registo.	Texto	6812
AVERAGE_TEMPERATURE	Valor médio da temperatura de acordo com o momento do registo.	Numérico	6812
AVERAGE_ATMOSP_PRESSURE	Valor médio da pressão atmosférica de acordo com o momento do registo.	Numérico	6812
AVERAGE_HUMIDITY	Valor médio da humidade de acordo com o momento do registo.	Numérico	6812
AVERAGE_WIND_SPEED	Valor médio da velocidade do vento de acordo com o momento do registo.	Numérico	6812
AVERAGE_CLOUDINESS	Valor médio da percentagem de nuvens de acordo com o momento do registo.	Texto	4130
AVERAGE_PRECIPITATION	Valor médio da precipitação de acordo com o momento do registo.	Numérico	6812
AVERAGE_RAIN	Avaliação qualitativa da precipitação de acordo com o momento do registo.	Texto	563

Aqui, a coluna cujos valores pretendemos prever é a coluna *AVERAGE_SPEED_DIFF*.

A primeira coisa em que reparamos é que todos os registos dizem respeito à mesma cidade, logo poderemos remover a coluna *city_name* do dataset, pois não nos irá ajudar a prever nada.

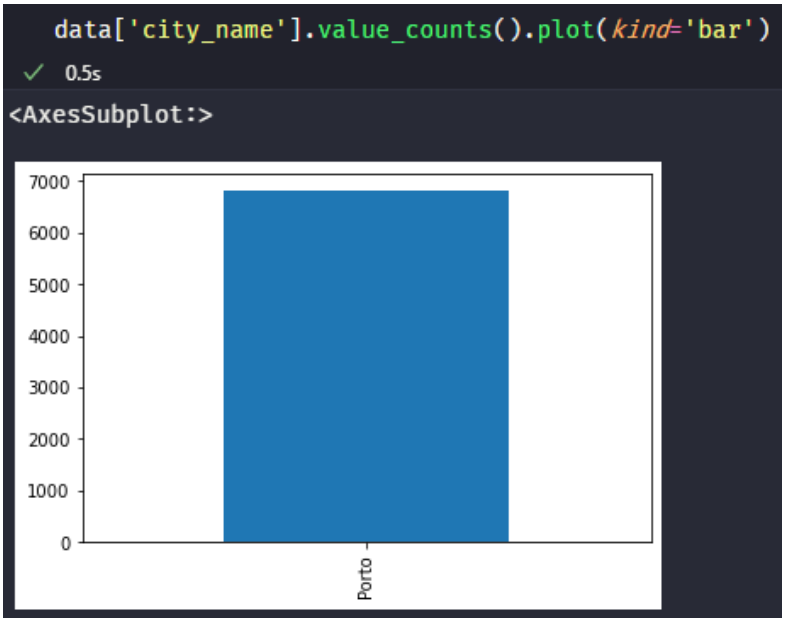


Figura 1: Valores armazenados na coluna *city_name*.

De forma semelhante, o valor da coluna *AVERAGE_PRECIPITATION* é sempre zero, logo podemos remover também esta coluna.

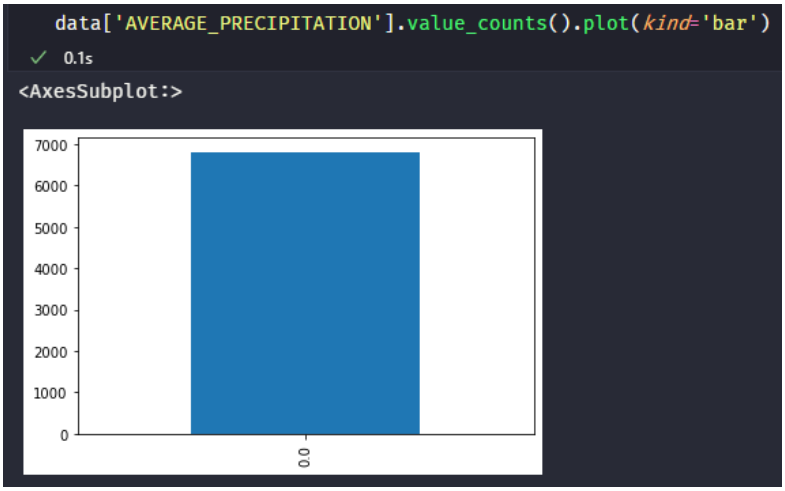


Figura 2: Valores armazenados na coluna *AVERAGE_PRECIPITATION*.

De seguida, pretendemos tratar as colunas com valores em falta. Para cada uma destas colunas (*AVERAGE_CLOUDINESS* e *AVERAGE_RAIN*) experimentámos remover as colunas e preencher os valores em falta, e reparámos que obtíamos um modelo melhor se preenchêssemos os valores em falta, ao invés de remover as colunas, mesmo para a coluna *AVERAGE_RAIN*, onde a maioria dos valores estão em falta.

Para os valores em *AVERAGE_RAIN*, começámos por converter o tipo da coluna para inteiros, agrupando valores semelhantes como "chuva leve"e "chuvisco e chuva fraca"num só valor, e atribuindo valores inteiros ordenados de acordo com a "quantidade"de chuva. Depois, assumimos que, quando o valor de *AVERAGE_RAIN* é nulo, significa que não estaria a chover no momento do registo. Desta forma, preenchemos os valores nulos com um valor que representa nenhuma precipitação.

Para a coluna *AVERAGE_CLOUDINESS*, aplicámos o mesmo primeiro passo (agrupar valores semelhantes e converter para inteiros ordenados de acordo com a percentagem de nuvens), mas em vez de substituir valores nulos por um determinado valor, interpolámos os valores em falta. Para tal, ordenámos os registos do dataset por data (a coluna *record_date*) e a partir deste novo dataset ordenado, preenchemos os valores nulos de *AVERAGE_CLOUDINESS* com base nos valores próximos não nulos. Este método foi o que nos pareceu fazer mais sentido, já que a quantidade de nuvens do céu tende a manter-se semelhante ou a alterar-se de forma linear ao longo do

tempo.

As colunas *LUMINOSITY* e *AVERAGE_SPEED_DIFF*, apesar de não apresentarem valores em falta, não representam um tipo numérico. Como tal, fizemos uma conversão usando o mesmo método usado nas duas colunas anteriores.

A última coluna que nos falta abordar é a coluna *record_date*. A partir de uma data, podemos extrair vários valores úteis, como o mês, o dia da semana ou a hora. Todos estes valores podem afetar o trânsito de uma cidade, logo devemos extraí-los e colocá-los em colunas distintas. Todo este processo pode ser resumido pelas seguintes instruções:

```
data['Month'] = data.record_date.dt.month
data['Day'] = data.record_date.dt.day
data['Hour'] = data.record_date.dt.hour
data['Day_Name'] = data.record_date.dt.day_name()
data['Year'] = data.record_date.dt.year

data['Day_Part']=data['Hour'].apply(daypart)

data['isWeekend']=data['Day_Name']\
    .apply(lambda x : 1 if x in ['Saturday','Sunday','Friday'] else 0)

# Só queremos valores numéricos na nossa tabela,
# então vamos converter o Dia da Semana e a Data
data['Day_Name'] = data['Day_Name']\
    .apply(lambda x: ['Sunday','Monday','Tuesday','Wednesday',
    'Thursday','Friday','Saturday'].index(x))
data['record_date'] = data['record_date']\
    .apply(lambda x : time.mktime(x.to_pydatetime().timetuple()))
```

Agora que temos o nosso dataset sem valores nulos e sem colunas não numéricas, podemos analisar bem os dados. Reparamos que as colunas *AVERAGE_ATMOSP_PRESSURE* e *AVERAGE_TIME_DIFF* possuem *outliers*, no primeiro caso 2 valores abaixo de 990 e no segundo caso um valor acima de 275. Removemos assim estes valores.

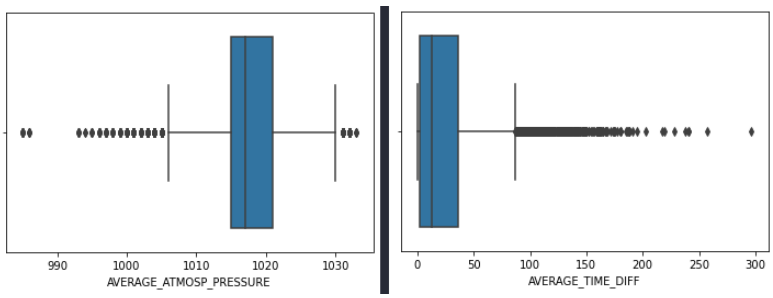


Figura 3: *Outliers* encontrados em duas colunas.

O resto da análise do dataset pode ser encontrada no nosso ficheiro Python. Para manter este relatório curto e conciso, decidimos apenas colocar aqui os aspetos mais importantes e que levaram a que alterássemos o dataset.

O último passo deste processo consiste em repartir o dataset final em dois datasets `X` e `y`, sendo que o primeiro contém as nossas colunas independentes e o segundo a coluna cujos valores pretendemos prever (*AVERAGE_SPEED_DIFF*).

2 Dataset de Musica

Nesta secção será abordada uma pequena interpretação e explicação das modificações feitas ao dataset escolhido, não sendo feita uma explicação tão extensa como feita no dataset dado.

2.1 Interpretação

```
[6]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50005 entries, 0 to 50004
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   instance_id       50000 non-null  float64
 1   artist_name       50000 non-null  object
 2   track_name        50000 non-null  object
 3   popularity        50000 non-null  float64
 4   acousticness      50000 non-null  float64
 5   danceability      50000 non-null  float64
 6   duration_ms       50000 non-null  float64
 7   energy            50000 non-null  float64
 8   instrumentalness  50000 non-null  float64
 9   key               50000 non-null  object
 10  liveness          50000 non-null  float64
 11  loudness          50000 non-null  float64
 12  mode              50000 non-null  object
 13  speechiness       50000 non-null  float64
 14  tempo             50000 non-null  object
 15  obtained_date     50000 non-null  object
 16  valence           50000 non-null  float64
 17  music_genre       50000 non-null  object
dtypes: float64(11), object(7)
memory usage: 6.9+ MB
```

Figura 4: Colunas do dataset Música

O que se deseja prever a partir dos nossos modelos é o género da musica (coluna 17) e assume 10 valores distintos: 'Electronic', 'Anime', 'Jazz', 'Alternative', 'Country', 'Rap', 'Blues', 'Rock', 'Classical', 'Hip-Hop'.

Todas as outras colunas existem como caracteristicas para calcular o genero da musica, por exemplo:

- popularidade (coluna 3) ,o quão popular é uma musica que varia entre 0 e 100.
- duração (coluna 6), qual o tempo que uma música demora em milissegundos.

- discurso (coluna 13), deteta a presença de palavras numa música, varia entre 0 e 1

- entre muitas outras ...

A descrição completa de cada coluna encontra-se no ficheiro *.ipynb* em comentário.

O dataset encontra-se balanceado como podemos ver pela seguinte figura e contêm aproximadamente 45 mil entradas, o que apesar de diminuir a eficiência na construção de modelos é uma grande vantagem em Machine Learning.

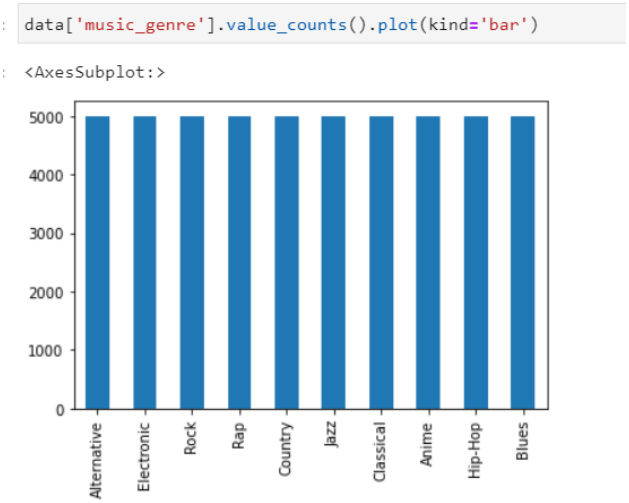


Figura 5: Dataset Música equilibrado

2.2 Modificação

O método e identificação de como cada modificação é feita pode ser observada mais uma vez no ficheiro *.ipynb* entregado, aqui será só realizada uma enumeração de cada modificação e razão por trás dessa mesma.

- Eliminação das colunas *track_name*, *obtained_date* e *instance_id* porque são colunas que consideramos redundantes porque não apresentam nenhuma informação que permite ao modelo inferir o género da musica.

- Eliminar entradas com atributos nulos, não sendo preciso usar outro método mais complexo devido a grande quantidade de dados no dataset.
- Transformar todas as colunas do tipo objeto em inteiro, usando factorize para o *artist_name*, transformando strings em *floats* para o *tempo* ou aplicando uma função para todas as outras colunas.
- Eliminar *outliers* usando gráficos do tipo *boxplot* para os identificar.
- Dividir o dataset em teste e treino como realizado nas aulas.
- Mais tarde para Deep Learning normalizar todo o dataset.

Modelos

1 Dataset do trânsito

Inicialmente, começámos por desenvolver o nosso modelo com base nos modelos apresentados nas aulas iniciais de DAA, nomeadamente o Decision Tree Classifier e o Support Vector Classifier.

Apesar destes modelos não serem tão avançados como uma rede neuronal, por exemplo, fomos capazes de obter valores de *accuracy* relativamente elevados.

Usando um Decision Tree Classifier com *cross-validation*, somos capazes de obter uma *accuracy* de 0.76.

```
clf=DecisionTreeClassifier(criterion='gini',max_depth=10,random_state=2021)
scores=cross_val_score(clf,X,y2,cv=10)
print(scores)
print("Result: %0.2f accuracy with a standart deviation of %0.2f" % (scores.mean(),scores.std()))
✓ 0.3s
[0.75294118 0.73970588 0.75147059 0.75       0.76470588 0.75147059
 0.7628866  0.76435935 0.77908689 0.75994109]
Result: 0.76 accuracy with a standart deviation of 0.01
```

Figura 6: Valor da *accuracy* obtida com um Decision Tree Classifier e com cross-validation.

Com o Support Vector Classifier, este valor cai para 0.32, mesmo usando os valores ótimos de C e de gamma, obtidos utilizando Grid Search.

```
clf=DecisionTreeClassifier(criterion='gini',max_depth=10,random_state=2021)
scores=cross_val_score(clf,X,y2,cv=10)
print(scores)
print("Result: %0.2f accuracy with a standart deviation of %0.2f" % (scores.mean(),scores.std()))
```
✓ 0.3s

```
[0.75294118 0.73970588 0.75147059 0.75       0.76470588 0.75147059
 0.7628866  0.76435935 0.77908689 0.75994109]
Result: 0.76 accuracy with a standart deviation of 0.01
```

Figura 7: Valor da *accuracy* obtida com um Decision Tree Classifier e com cross-validation.

Decidimos recorrer então a uma rede neuronal, mais especificamente um modelo Keras Classifier com 3 camadas, definido do seguinte modo:

```
def build_model(activation='relu',learning_rate=0.01):
    model=Sequential()
    model.add(Dense(16,input_dim=18,activation=activation))
    model.add(Dense(8,activation=activation))
    model.add(Dense(5,activation='softmax'))
    model.compile(
        loss='sparse_categorical_crossentropy',
        optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
        metrics=['accuracy'])
    return model
```

Figura 8: Função responsável por construir o modelo.

```
TUNING_DICT = {
    'activation' : ['relu'],
    'learning_rate' : [0.00001,0.00005,0.0001,0.005]
}

kf=KFold(n_splits=5,shuffle=True,random_state=RANDOM_SEED)

model = KerasClassifier(build_fn=build_model,epochs=40,batch_size=5)
grid_search = GridSearchCV(estimator=model,
                           param_grid = TUNING_DICT,
                           cv=kf,
                           scoring='accuracy',
                           refit='True',
                           verbose=1)
grid_search.fit(X_scaled,y,validation_split=0.2,verbose=1)
```

Figura 9: Construção do modelo, usando um Keras Classifier e Grid Search.

Com este modelo, obtemos um valor de *accuracy* de 0.77. Apesar de não parecer muito maior do que o resultado obtido com a Decision Tree, não podemos ter apenas este valor em conta, visto que é possível obter um valor de *accuracy* muito alto nesta primeira fase e obter um valor mais baixo com novos valores de teste, no caso de haver *overfitting*.

Para além destes modelos, usámos ainda um Random Forest Classifier. De todos, este foi o que nos deu o maior valor de *accuracy*, 0.79.

```
#Construção dos Modelos
rf=RandomForestClassifier(n_estimators=1000,random_state=2021,max_depth=100)

#Treino dos Modelos
rf.fit(train_X,train_y)
```

Figura 10: Construção do modelo, usando um Random Forest Classifier.

```
#Observar a precisão do modelo
print(classification_report(test_y,predictionsForest))
```

✓ 0.2s

	precision	recall	f1-score	support
0	0.85	0.89	0.87	211
1	0.67	0.64	0.65	138
2	0.82	0.79	0.80	183
3	0.68	0.87	0.76	89
4	1.00	0.64	0.78	59
accuracy			0.79	680
macro avg	0.80	0.76	0.77	680
weighted avg	0.80	0.79	0.79	680

Figura 11: Relatório de classificação do modelo.

2 Dataset da Música

Nesta secção será apresentados o tipo de modelos desenvolvidos para o dataset escolhido sendo feita novamente de forma menos especifica do que realizada para o outro dataset.

O código realizado na construção destes modelos é semelhante ao dataset do trânsito que por sua vez é semelhante ao dado nas aulas sendo a principal diferença a forma como se avaliam os modelos.

Para avaliar cada modelo foi usado o dataset teste originado anteriormente tirando assim partido da função *classification_report* para calcular a *precision* e *accuracy* de cada.

Em especifico os modelos desenvolvidos foram:

1. **Decistion Tree**

   Construido com profundidade máxima de 10.

   Usando *cross-hold-validation* para calcular a adequação do modelo.

2. **Support Vector Machine**

   Calculando os melhores hiper-parametros usando grid.

   Usando *cross-hold-validation* para calcular adequação do modelo.

3. **Logistic Regression**

   Sendo necessário diminuir o tamanho do dataset e aumentar o máximo de iterações.

4. **Deep Learning**

   Calculando os melhores hiper-parâmetros usando grid.

   Testando a rede com diferentes topologias.

   Testando o modelo com diferentes épocas e *batch_size*.

   Usando o gráfico de aprendizagem para avaliar o modelo.

5. **Random Forest Classifier**

   Desenvolvido porque **Decision Tree** demonstrou bons resultados e este é considerado uma versão melhorada.

   Usando 2000 estimadores.

Resultados

1 Dataset do trânsito

Tal como referido na secção anterior, os melhores modelos para este dataset aparentaram ser o modelo construído com um Random Forest Classifier e o modelo com um Keras Classifier.

Usando como exemplo o modelo do Random Forest Classifier, de forma a prever o valor do trânsito, criamos um novo modelo, mas neste usamos como dados de treino todo o dataset fornecido e como dados de teste o ficheiro de teste fornecido.

Com este novo modelo, ao fazer a previsão para os dados de teste, podemos prever como será o tráfego rodoviário de acordo com as características apresentadas.

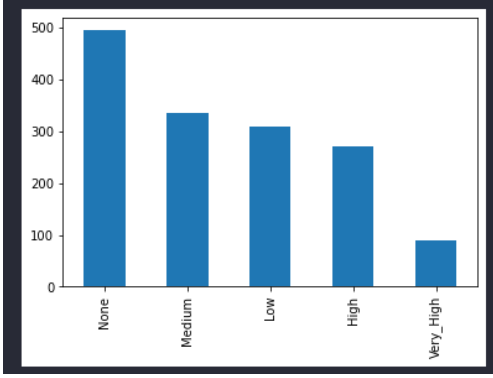
```
rfModel=RandomForestClassifier(n_estimators=2000,random_state=2021)
rfModel.fit(X,y2)
```
✓ 15.7s
```
RandomForestClassifier(n_estimators=2000, random_state=2021)
```
```
predictionsRF=rfModel.predict(data2)
predictionsRF[:10]
```
✓ 0.6s
```
array([0, 1, 0, 3, 1, 2, 2, 2, 0, 2])
```
```
result2 = pd.DataFrame(data={'Speed_Diff': predictionsRF})
result2.index = np.arange(1, len(result) + 1)
result2['Speed_Diff'] = result2['Speed_Diff'].apply(intToSpeed)
result2['Speed_Diff'].value_counts().plot(kind='bar')
```
✓ 0.1s
```
<AxesSubplot:>
```


Figura 12: Resultado da aplicação do nosso modelo aos dados de teste.

O último passo consiste em armazenar este novo dataset com as previsões num ficheiro CSV para participar no torneio.

O mesmo processo foi feito para o modelo do Keras Classifier, sendo que submetemos para o torneio os dois ficheiros gerados, de forma a apurar qual destes é o melhor para o cenário do torneio.

2 Dataset da Música

Em relação ao dataset da música escolhido pelo grupo, os melhores valores de precisão observados no dataset de teste foram de 80% em vários modelos como **Decision Tree**, **Deep Learning** e **Random Forest**.

Os outros 2 modelos testados, **Support Vector Machine** e **Logistic Regression**, apresentaram valores limitados de 27% e 50%, respetivamente.

Foi usado um gráfico de distribuição do género da música pelo espaço para tentar perceber a razão pela qual **SVM** tinha tão pouca compatibilidade e foi então confirmado que a falta de padrões de distribuição juntamente com o tamanho do dataset poderá ser a razão pela qual o modelo apresenta tão pouca precisão.

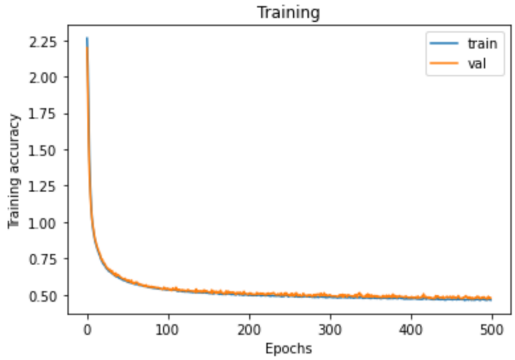


Figura 13: Gráfico de aprendizagem **Deep Learning** do dataset música.

	precision	recall	f1-score	support
0	0.98	0.97	0.98	510
1	0.82	0.97	0.89	476
2	0.72	0.73	0.72	513
3	0.97	0.98	0.97	509
4	0.88	0.91	0.90	498
5	0.54	0.54	0.54	498
6	0.86	0.88	0.87	501
7	0.93	0.78	0.85	458
8	0.74	0.70	0.72	520
9	0.58	0.56	0.57	517
accuracy			0.80	5000
macro avg	0.80	0.80	0.80	5000
weighted avg	0.80	0.80	0.80	5000

Figura 14: Relatório de classificação para modelo **Random Forest**

Conclusões

Em relação ao dataset do trânsito, estamos bastante satisfeitos com o nosso trabalho, visto que fomos capazes de cumprir todos os requisitos e conseguimos ainda ficar em 5º lugar no torneio, algo do qual estamos muito orgulhosos! Existem alguns aspetos que poderiam ser melhorados, caso contrário teríamos ficado em primeiro lugar, mas independentemente da pontuação no torneio, fomos capazes de criar um modelo que responde ao problema em questão de forma eficaz.

Quanto ao dataset da música, também acreditamos ter feito um bom trabalho, apesar de neste caso a qualidade do trabalho ser mais subjetiva, visto que não temos nenhum torneio como ponto de referência. De qualquer das formas, também aqui conseguimos desenvolver um modelo capaz de responder à questão que colocámos no momento da escolha deste dataset.

Um aspeto que tentámos sempre ter em conta em ambos os cenários é o *overfitting*. É muito fácil criar um modelo com mais de 90% de *accuracy* para os dados de treino, mas que num teste real falha miseravelmente. Deste modo, tivemos sempre como objetivo desenvolver modelos que sejam capazes de fazer o que foram criados para fazer, em qualquer tipo de situação e cenário, e não apenas num cenário de treino.