Instituto Superior Técnico

Aprendizagem Automática

Laboratório 6 - Evaluation and Generalization

Autores:

Diogo Moura - nº 86976

Diogo Alves - nº 86980

Turno:

Terça 17h-18h30m

21 de Dezembro de 2019

Introdução

O objetivo deste trabalho é, dados dois datasets, cada um com o respetivo conjunto de treino e de teste, obter modelos que consigam classificar os dados o melhor possível. Para tal é necessário testar vários modelos diferentes e estudar os resultados obtidos recorrendo a diferentes indicadores.

Introdução Teórica

Os classificadores utilizados foram os seguintes:

- SVM polynomial

  Uma máquina de vetores de suporte (SVM) é um classificador que divide os dados em classes linearmente separáveis, não no espaço original (input space), mas sim num outro espaço (feature space). Os inputs são mapeados no espaço de features, onde são classificados usando um hiperplano como fronteira. Como o mapeamento é na maioria das vezes não linear, os hiperplanos no espaço de features correspondem a fronteiras não lineares no espaço de inputs.

  Em seguida apresenta-se um esboço do funcionamento deste classificador:

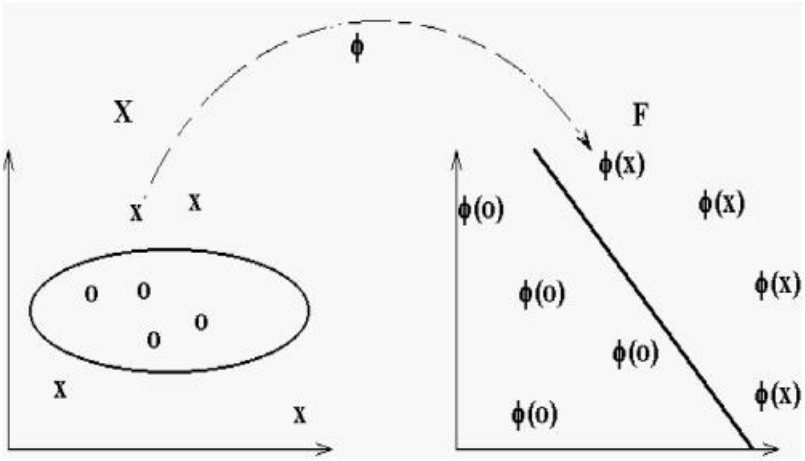

Figura 1: Esboço do funcionamento do classificador SVM

  Para o caso específico do SVM polinomial, o mapeamento é feito através do seguinte kernel

$$K(x, y) = (x \cdot y + a)^p \tag{1}$$

  onde o valor de a utilizado foi 0 e o grau do polinómio p é selecionado na estimação de hiperparâmetros.

- SVM RBF

  Para o SVM RBF, o kernel utilizado é o seguinte:

$$K(x, y) = e^{-\gamma ||x-y||^2} \tag{2}$$

  onde o valor de γ é também selecionado na estimação de hiperparâmetros.

- K Nearest Neighbours

  O classificador K Nearest Neighbours é um dos classificadores mais simples: dado um input, limita-se a fazer a contagem do número de pontos em cada classe, de entre os k pontos mais próximos do input pertencentes ao conjunto de treino e classifica o input na classe com maior frequência. Neste caso, o valor de k é selecionado na estimação de hiperparâmetros.

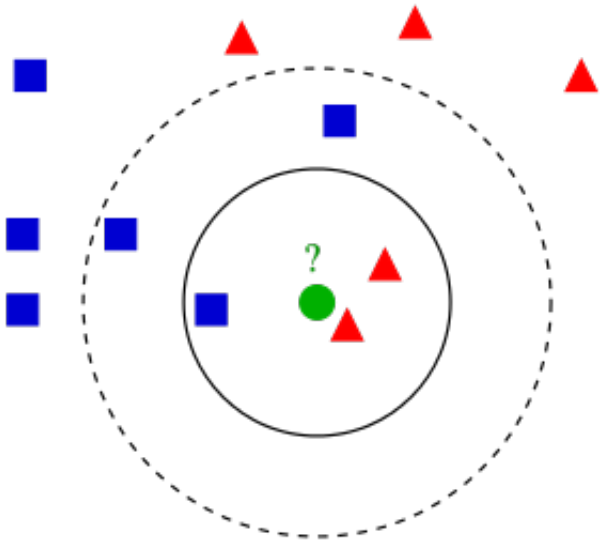

Figura 2: Esboço do classificador K Nearest Neighbour. Para k = 3 o input seria classificado na classe vermelha e para k = 5 seria classificado na azul.

- Multinomial Naive Bayes

  Os classificadores de Naive Bayes são uma família de classificadores probabilísticos baseados em aplicar o Teorema de Bayes assumindo independência entre features. Assim, o resultado do classificador é dado por:

$$\hat{y} = \underset{\omega \in \Omega}{argmax} P(\omega_j) \prod_{i=1}^{n} p(x_i|\omega_j) \tag{3}$$

  A probabilidade da classe ($P(\omega_j)$) é dada pelo quociente entre o número de amostras do conjunto de treino pertencentes à classe ω_j pelo número total de amostras do conjunto de treino. Já a probabilidade $p(x_i|\omega_j)$ depende do modelo usado. Para o modelo multinomial, esta probabilidade é dada por

$$p(x_i|\omega_j) = \theta_{ji} = \frac{N_{ji} + \alpha}{N_j + \alpha n} \tag{4}$$

  em que N_{ji} é o número de vezes que a feature x_i aparece numa amostra da classe ω_j no conjunto de treino, e N_j é o somatório de todos os N_{ji} em i. Neste trabalho será usado LaPlace smoothing ($\alpha = 1$).

  Este modelo é mais indicado para datasets em que as features assumam valores discretos.

- Gaussian Naive Bayes

  No modelo gaussiano do classificador Naive Bayes, a probabilidade $p(x_i|\omega_j)$ é dada pela expressão

$$p(x_i|\omega_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{(x_i - \mu_j)^2}{2\sigma_j^2}\right) \tag{5}$$

onde os parâmetros μ_j e σ_j são, respectivamente, a média e a variância do conjunto de features, na classe j.

Este modelo é mais adequado quando trabalhamos com features que podem assumir valores reais.

- Decision Tree

  Uma árvore de decisão é um classificador cujo modelo de decisão é constituído por nós e folhas. A cada nó corresponde uma feature, e de cada nó saem ramos que correspondem ao diversos valores que essa feature pode assumir. Cada folha contém o valor da classe predita pelo classificador, quando os valores das features são as correspondentes ao caminho desde a raiz da árvore até à folha.

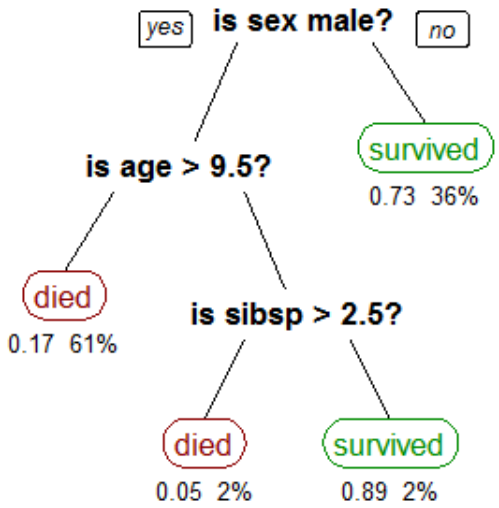

Figura 3: Exemplo de uma árvore de decisão

- Multi-Layer Perceptron

  Um Multi-Layer perceptron (MLP) é uma classe de redes neuronais com "feedforward". O MLP utiliza uma técnica de aprendizagem supervisionada chamada de propagação retroativa ("backpropagation") para treino.

  As suas múltiplas camadas permitem-no distinguir de um percetrão. Permite distinguir data que não é linearmente separável, já que pode utilizar funções de activação não lineares (i.e. tanh).
  Cada percetrão i está ligado a uma unidade j da camada seguinte através de um peso w_{ij}.

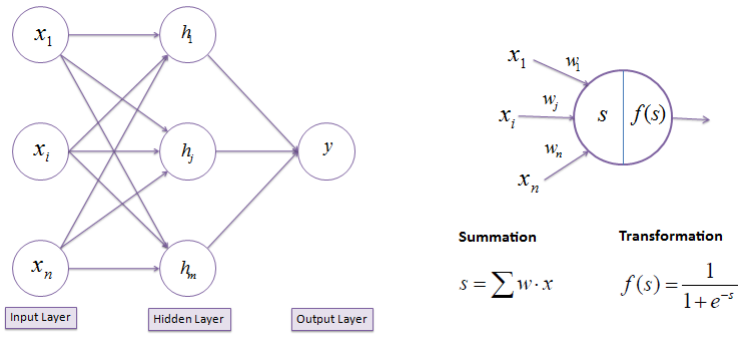

Figura 4: Percetrão Multi-camada (à esquerda) e Percetrão com função de ativação logística (à direita)

Utilizámos o Modo Mini-batch para atualizar os pesos da rede($w_{ij}^{t+1} = w_{ij}^{t} + \Delta w_{ij}^{t}$), o qual em cada época utiliza um subconjunto das amostras do conjunto de treino:

$$\Delta w_{ij} = -\eta \frac{dR}{dw_{ij}} = -\frac{\eta}{n} \sum_{k}^{N} \frac{dL(y^{(k)}, \hat{y}^{(k)})}{dw_{ij}} \tag{6}$$

Para calcular a perda é utilizado o algoritmo da propagação retroativa.

$$\frac{dL}{dw_{ij}} = z_i \epsilon_j \tag{7}$$

Em datasets não balanceados, mudámos o peso (”bias”) inicial dos percetrões e o peso atribuído a cada classe. O ”bias”correto a atribuir a cada percetrão pode ser derivado usando a equação que se segue, tendo em conta a função logística de output, em que *pos* equivale ao número de amostras do conjunto de treino classificadas em 1, *neg* às classificadas em 0 e total ao número total de amostras do conjunto de treino:

$$p_0 = pos/(pos + neg) = 1/(1 + e^{-w_0}) \iff w_0 = -log(1/p_0 - 1) \Rightarrow w_0 = log(pos/neg) \tag{8}$$

Se partirmos da premissa que cada classe deve possuir o mesmo peso no treino do classificador, o peso a atribuir a cada classe é dado pela seguinte expressão:

$$peso_{classe0} = \frac{1}{2} \times \frac{total}{neg} \tag{9}$$

$$peso_{classe1} = \frac{1}{2} \times \frac{total}{pos} \tag{10}$$

Para o caso do dataset 2, apenas no caso de treino com redes neuronais, normalizámos as features utilizando a função StandardScaler da biblioteca scikit-learn, já que o classificador apresentou melhorias consideráveis na accuracy e balanced accuracy. Esta função muda a média de cada feature para 0 e o desvio-padrão para 1.

Os indicadores utilizados para avaliar a performance dos classificadores são os seguintes:

- Accuracy

A accuracy (ou precisão) corresponde ao cálculo da fração de classificações corretas:

$$\text{accuracy}(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} 1(\hat{y}_i = y_i) \tag{11}$$

- Balanced Accuracy

  A balanced accuracy (ou precisão ponderada) evita estimativas de performance inflacionadas em datasets desbalanceados. Corresponde à accuracy onde cada amostra é pesada de forma inversamente proporcional ao número de amostras na classe a que pertence:

$$\text{balanced-accuracy}(y, \hat{y}, w) = \frac{1}{\sum \hat{w}_i} \sum_i 1(\hat{y}_i = y_i)\hat{w}_i \tag{12}$$

  onde o peso é dado por:

$$\hat{w}_i = \frac{w_i}{\sum_j 1(y_j = y_i)w_j} \tag{13}$$

  Para datasets equilibrados, a accuracy e balanced accuracy serão equivalentes.

- Confusion Matrix

  A matriz de confusão é uma matriz em que cada linha representa a classe predita e cada coluna representa a verdadeira classe. Assim, cada elemento da matriz (C_{ij}) corresponde ao número de amostras que pertencem à classe w_i e foram classificadas na classe $\hat{w}_j$. A matrix de confusão de um classificador com 100% de accuracy apenas teria valores diferentes de 0 na diagonal principal.

- Curva Roc

  Curva que mostra a taxa de true positives nas ordenadas e a de false positives a thresholds de classificação diferentes. Esta curva é útil dado que mostra, como alterando o threshold do output se altera a performance do classificador.

$$True\_Positives\_Rate = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{14}$$

$$False\_Positives\_Rate = \frac{FalsePositives}{FalsePositives + TrueNegatives} \tag{15}$$

- AUROC (Area under the ROC Curve)

  Uma métrica que considera todos os thresholds possíveis. A área sob curva ROC é a probabilidade de que um classificador estar mais confiante de que uma amostra positiva escolhida aleatoriamente é positiva do que uma amostra negativa escolhida aleatoriamente é positiva. Métrica útil sobretudo no treino com datasets não balanceados.

Para os classificadores em que não é necessário estimar hiperparâmetros (Multinomial Naive Bayes, Gaussian Naive Bayes e Decision Tree), foi utilizado o conjunto de treino para treinar o classificador e o conjunto de teste para avaliar a sua performance. Para os restantes classificadores (SVM polinomial, SVM RBF e Nearest Neighbour), em que é preciso estimar hiperparâmetros, o conjunto de treino foi dividido em dois conjuntos disjuntos: (X_train, y_train) e (X_validation, y_validation), em que o conjunto de validação tem 30% do tamanho do conjunto original. Depois, foi usado o conjunto (X_train, y_train) para treinar o classificador e o conjunto (X_validation, y_validation) para testar a performance para cada valor do hiperparâmetro e foi escolhido o valor que deu origem à máxima precisão no conjunto de validação. Com este escolhido, foi utilizado o conjunto de treino original para treinar o classificador e o conjunto de teste para avaliar a performance.

Análise de Resultados

Classifier / Evalutor	SVM polynomial (Degree=1)	SVM RBF ($\gamma = 0.0023$)	Nearest Neighbours ($neighbours = 10$)	Multinomial Naive Bayes	Gaussian Naive Bayes	Decision Tree
Accuracy	0.765	0.709	0.652	0.513	0.565	0.687
Balanced Accuracy	0.771	0.713	0.657	0.511	0.588	0.682
Confusion Matrix	$\begin{bmatrix} 94 & 14 \\ 40 & 82 \end{bmatrix}$	$\begin{bmatrix} 84 & 24 \\ 43 & 79 \end{bmatrix}$	$\begin{bmatrix} 79 & 29 \\ 51 & 71 \end{bmatrix}$	$\begin{bmatrix} 52 & 56 \\ 56 & 66 \end{bmatrix}$	$\begin{bmatrix} 103 & 5 \\ 95 & 27 \end{bmatrix}$	$\begin{bmatrix} 65 & 43 \\ 29 & 93 \end{bmatrix}$

Tabela 1: Tabela com precisões e matrizes de confusão para os diferentes métodos utilizados para o Dataset 1

Classifier / Evalutor	SVM polynomial (Degree=4)	SVM RBF ($\gamma = 2.31 \times 10^{-8}$)	Nearest Neighbours ($neighbours = 5$)	Multinomial Naive Bayes	Gaussian Naive Bayes	Decision Tree
Accuracy	0.838	0.869	0.838	0.758	0.657	0.788
Balanced Accuracy	0.820	0.817	0.812	0.762	0.715	0.731
Confusion Matrix	$\begin{bmatrix} 24 & 7 \\ 9 & 59 \end{bmatrix}$	$\begin{bmatrix} 21 & 10 \\ 3 & 65 \end{bmatrix}$	$\begin{bmatrix} 23 & 8 \\ 8 & 60 \end{bmatrix}$	$\begin{bmatrix} 24 & 7 \\ 17 & 51 \end{bmatrix}$	$\begin{bmatrix} 27 & 4 \\ 30 & 38 \end{bmatrix}$	$\begin{bmatrix} 18 & 13 \\ 8 & 60 \end{bmatrix}$

Tabela 2: Tabela com precisões e matrizes de confusão para os diferentes métodos utilizados para o Dataset 2

Em relação ao primeiro dataset, a maior precisão obtida foi de cerca de 76% com o classificador SVM linear (polinomial de ordem 1). Os restantes classificadores tiveram precisões inferiores, sendo a mais baixa de 51% para o classificador Multinomial Naive Bayes. Sendo que o conjunto de teste do dataset 1 tem quase o mesmo número de exemplos para ambas as classes (122 para a classe 1 e 108 para a classe 0), os valores obtidos para a precisão ponderada são bastante próximos dos obtidos para a precisão, para cada classificador. Observando as matrizes de confusão, verificamos que a maioria dos classificadores (SVM polinomial, SVM RBF, Nearest neighbour e Gaussian Naive Bayes) tiveram maior precisão na classificação dos exemplos da classe 0, sendo o algoritmo mais eficaz a fazê-lo o Gaussian Naive Bayes, que classificou corretamente 103 dos 108 exemplos da classe 0. No entanto, este classificador também classificou na classe 0 95 dos 122 exemplos da classe 1, daí a sua baixa precisão. Os restantes classificadores (Multinominal Naive Bayes e Decision Tree) classificaram corretamente mais exemplos da classe 1.

Para o segundo dataset, verificamos que a maior precisão (cerca de 87%) é dada pelo classificador SVM RBF com $\gamma \approx 2.31 * 10^{-8}$, no entanto, como o conjunto de teste é desproporcional (existem 68 exemplos da classe 1 e apenas 31 da classe 0), não é este o classificador que apresenta maior precisão ponderada, mas sim o SVM polinomial com grau 4, que apresenta uma precisão ponderada de 82%. Verificamos que os valores das precisões dos vários classificadores variam entre 66% e 87%, no entanto os valores das precisões ponderadas variam apenas entre 72% e 82%. Mais uma vez, isto deve-se ao facto de o conjunto de teste ser desproporcional.

É possível verificar através das matrizes de confusão que os classificadores SVM polinomial, Nearest Neighbour e SVM RBF, que foram os que obtiveram maiores precisões ponderadas, foram também aqueles que tiveram menos erros de classificação.

É curioso observar que, de forma semelhante ao dataset 1, o classificador Gaussian Naive Bayes foi aquele que obteve maior número de exemplos corretamente classificados na classe 0, mas em contrapartida foi também o que obteve maior número de exemplos que pertenciam à classe 1 incorretamente classificados na classe 0. Isto mostra que este classificador tende a classificar uma maior percentagem de dados em uma classe específica.

Verificamos também que o classificador Multinomial Naive Bayes é o classificador com pior performance no dataset 1, o que já era de esperar, visto que este classificador é mais adequado para datasets em que as features assumam valores discretos e no dataset 1 existem features que assumem valores reais. Para o dataset 2, isto também se verifica, mas como existem valores das features repetidas no conjunto de treino e de teste, este classificador consegue obter maior accuracy para o dataset 2.

Para o MLP, relativamente ao Dataset 1, a rede neuronal apenas apresentou pior performance que os classificadores SVM.

	Dataset 1	Dataset 2
Accuracy	0.757	0.879
Balanced Accuracy	0.763	0.868
Confusion Matrix	$\begin{bmatrix} 94 & 14 \\ 42 & 80 \end{bmatrix}$	$\begin{bmatrix} 26 & 5 \\ 7 & 61 \end{bmatrix}$

Tabela 3: Tabela com precisões e matrizes de confusão para o Multi-Layer Perceptron para os Datasets 1 e 2

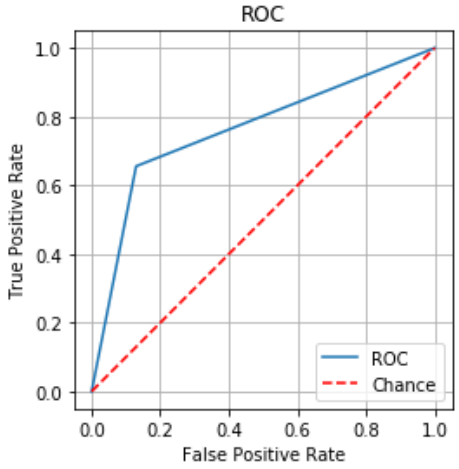

Figura 5: Curva ROC para o dataset 1

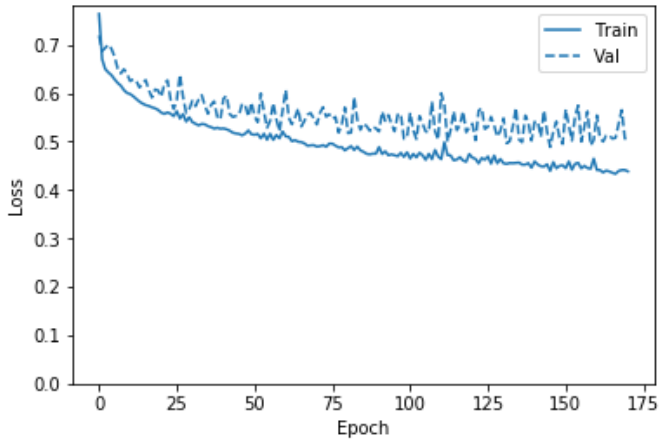

Figura 6: Evolução da Loss para o dataset 1

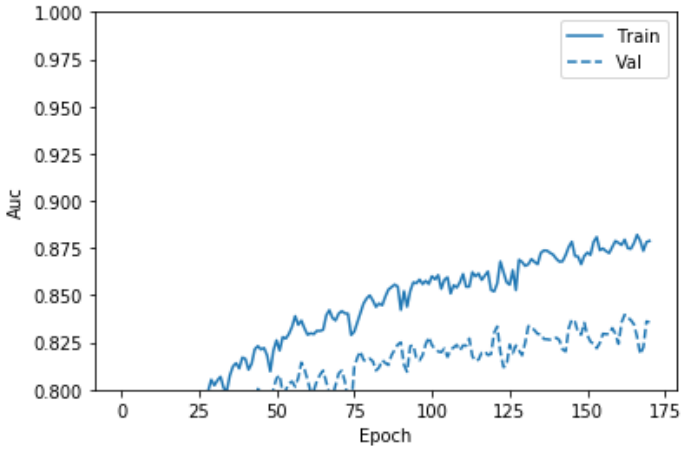

Figura 7: Evolução da AUC para o dataset 1

Para o Dataset 2, conseguimos obter melhores resultados do que qualquer outro classificador. Neste dataset não existe balanceamento dos dados, pelo que tivemos que atribuir pesos às classes. Para além disso tivemos que normalizar as features de entrada para obter resultados em que a accuracy fosse superior a 80% e atribuir pesos iniciais (bias) aos percetrões, diferentes de 0, conforme o que foi referido na secção teórica deste relatório, para um classificador binário. A accuracy foi 1% superior ao SVM RVF e a balanced accuracy 6% superior. Deste modo, este classificador apresentou melhorias significativas não só quanto à accuracy, mas também quanto à balanced accuracy, o que mostra que cumpriu a sua função de balancear as classes.

O código para implementar estas funcionalidades foi o seguinte:

```
#scale the features
scaler=StandardScaler()
dataset2_xtrain=scaler.fit_transform(dataset2_xtrain)
dataset2_xtest=scaler.fit_transform(dataset2_xtest)
#calculate the frequency of appearance of each class, based on the train dataset
aux=dataset2_ytrain.ravel()
neg,pos=np.bincount(aux.astype(int))
```

```
total=neg+pos
#Set the correct Weights for the classes
weight_for_0=(1/neg)*total/2.0
weight_for_1=(1/pos)*total/2.0
class_weight={0:weight_for_0, 1:weight_for_1}
#Set the correct initial biases for the perceptrons
initial_bias=np.log([pos/neg])
```

Ainda assim, nem sempre os pesos do classificador convergiram para o mesmo valor, devido a outros fatores de aleatoriedade no treino do modelo. Por essa razão, guardámos os pesos que obtivemos, para que sejam obtidos valores de accuracies reproduzívies e consistentes com o nosso.

```
weighted_model.load_weights('weights', by_name=False)

weighted_history=weighted_model.fit(x_train,
                    y_train, validation_data=(x_val,y_val), epochs=EPOCHS,
                    batch_size=BATCH_SIZE, verbose=0, callbacks=[early_stopping]
                    ,class_weight=class_weight)

weighted_model.save_weights('weights')
```

De notar ainda que o parâmetro utilizado para monitorizar a Early Stopping, ao invés de ser a validation loss como para o dataset 1, foi antes utilizada a validation accuracy, uma vez que o dataset não está balanceado.

```
early_stopping=tf.keras.callbacks.EarlyStopping(
          monitor='val_auc',
          verbose=1,
          patience=10,
          mode='max',
          restore_best_weights=True)
```

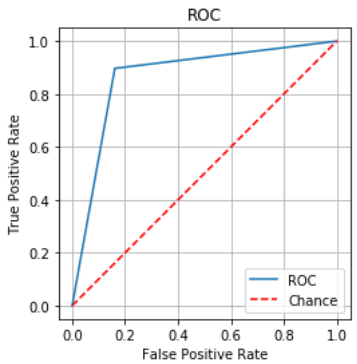

Figura 8: Curva ROC para o dataset 2, utilizando um classificador com pesos para as classes

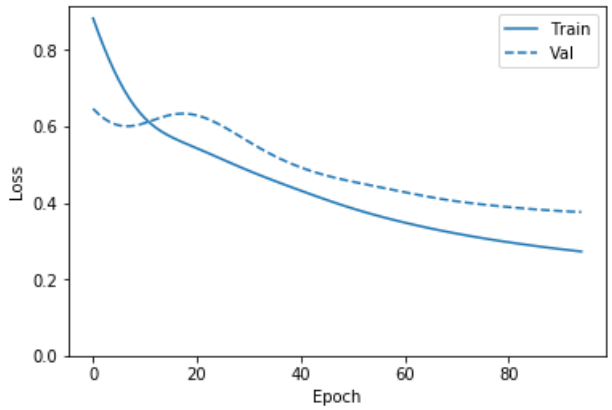

Figura 9: Evolução da Loss para o dataset 2, utilizando um classificador com pesos classes

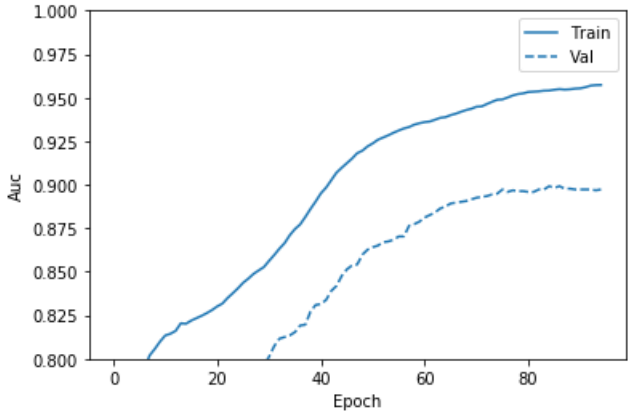

Figura 10: Evolução da AUC para o dataset 2, utilizando um classificador com pesos para as classes

Ficheiros

No ficheiro "mlp.py"está contido o código para a classificação para os 2 datasets com Multi-Layer Perceptron e nos ficheiros "weights.data-00000-of-00002", "weights.data-00001-of-00002"e "weights.index"os pesos para a rede neuronal para o 2º dataset.

No ficheiro "classifiers.py"está contido o código para a classificação para os 2 datasets para os restantes classificadores, isto é, SVM polynomial, SVM RBF, k-Nearest Neighbours, Multinomial Naive-Bayes, Gaussian Naive-Bayes e Decision Tree.