FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Aprendizagem Automática

17/18

Assignment 1

Report

Turno P1

Prof. Ludwig Krippahl

Ana Simão 42978

Pedro Vieira 42610

Índice

1. Introdução

O intuito deste trabalho é comparar a performance de diversas técnicas estatísticas de forma a consolidar a implementação e o funcionamento próprio de cada uma delas. Para tal é apresentada uma situação "real" onde é fornecido um conjunto de dados referente à autenticação de notas bancárias e ao qual é suposto serem aplicadas três técnicas distintas: Regressão Logística, K-Nearest Neighbours e Naive Bayes.

Os dados fornecidos estão organizados de forma a que cada linha corresponde a uma nota bancária, e cada nota possui cinco valores distintos, separados por vírgulas, onde 4 deles são características (variância, assimetria e curtose da imagem Wavelet transformada e entropia da imagem da nota bancária), que de agora em diante vão ser referidas como *features*, e o quinto valor que indica a classe de cada nota, sendo que o valor 0 representa as notas reais e 1 representa as notas falsas.

2. Classificadores

2.1. Logistic Regression

A regressão logística é um método de aprendizagem supervisionada que apesar de ser uma regressão é utilizado como um classificador, visto que a sua variável resposta Y é binária e não contínua. O que a regressão logística vai fazer é estimar a probabilidade de um dado ponto, tendo em conta as suas *features*, pertencer a uma classe e irá traçar uma linha que separa as duas classes, ou seja, os pontos nessa linha apresentam uma probabilidade equivalente de pertencer tanto a uma classe como a outra. Para tal é necessário aplicar a seguinte função logística:

$$g(\vec{x}, \widetilde{w}) = \frac{1}{1 + e^{-(\vec{w}^T \vec{x} + w_0)}}$$

Foi também utilizado um parâmetro de regularização de forma a reduzir o problema do *overfitting*, para tal foi utilizado o método de *cross validation* (mais detalhes da secção 3);

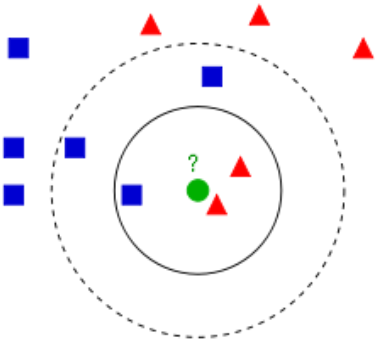
Como forma de proceder à otimização do parâmetro C, usamos o método do *cross validation* e tendo em conta que, como foi referido anteriormente, a regressão logística estima as probabilidades de uma nota pertencer a uma classe, foi tomada a opção de calcular e utilizar o *Brier score* (que mede o erro quadrático entre a probabilidade prevista pela hipótese em questão e a classe a que o dado pertence de facto) ao invés de utilizar a *accuracy* ou mais precisamente o erro dado por 1 - *accuracy* (medida utilizada nos outros classificadores) para definir qual o

valor ótimo para o parâmetro C visto que o *Brier score* devolve valores com uma variação mais suave.

Para aplicar a função logística e calcular a regressão foi usado o *LogisticRegression* da biblioteca *sklearn*.

2.2. K-Nearest Neighbors

Contrariamente à regressão logística, o classificador *K-Nearest Neighbors* utiliza uma estratégia de *lazy learning* na medida em que, quando este recebe os dados de treino, não treina um modelo/*hypothesis class* através do ajustamento de parâmetros. Ao invés disso, tendo em conta que o *K-NN* também é um método de aprendizagem supervisionada, simplesmente guarda todos as entradas do *data set*, incluindo as respetivas classes, e posteriormente, aquando uma *query*, tentará prever a classe de um novo *item* desconhecido olhando para todo o *data set* e devolvendo as classes dos *K* dados que mais se assemelham a este, sendo que a resposta será a classe dominante desses *K* pontos.



Este processo de semelhança é calculado através de uma função de distância que é aplicada a todos os pontos previamente fornecidos como base e verifica a diferença entre as *features* desses pontos e as do novo ponto que se pretende classificar: quanto mais pequeno o valor devolvido pela função mais parecidos são os pontos, e vice-versa. De seguida, estes valores são ordenados de forma ascendentes e as classes das primeiras *K* entradas são usadas para a previsão final.

Existem múltiplas funções de distância, como a *Manhattan distance*, a *Euclidean distance* e a *Hamming distance*. No entanto, como as *features* do *data set* fornecido representam valores contínuos, a *Hamming distance* foi excluída logo à partida, pois esta deverá ser usada apenas quando os valores são categóricos devido ao facto de apenas contabilizar o número de *features* diferentes entre dois pontos. Assim, para este projeto, usou-se a fórmula de *Minkowski* (quando *p=1* representa a *Manhattan distance* e quando *p=2* representa a *Euclidean distance*) no *K-NN*:

$$D_{x,x'} = \sqrt[p]{\sum_{d} |x_d - x'_d|^p}$$

De forma a encontrar o melhor número de vizinhos a ser usado pelo classificador, o grupo variou o parâmetro *K* de 1 a 39, excluindo números pares pois poderia introduzir um comportamento não determinístico perante a existência de um empate nas classes dos vizinhos, e para cada um desses valores foi

aplicado o método de *cross validation* para encontrar o *validation error* mais baixo e, por sua vez, o valor do *K* associado a esse erro de forma a ser utilizado, numa fase final, com o *test set*.

Por fim, o grupo fez uso do *KNeighborsClassifier* da biblioteca *sklearn*, com a distância euclidiana, para implementar este classificador e a estimativa dos erros foi calculada através da *accuracy*, nomeadamente *1 - accuracy* (*accuracy* essa que é dada pelo método *score* do objeto descrito).

2.3. Naïve Bayes

O classificador *Naïve Bayes* é baseado no teorema de *Bayes*:

$$p(C = c|X = x) = \frac{p(C = c)p(X = x|C = c)}{p(X = x)}$$

No entanto, no *Naïve* é assumido que todas as *features* são independentes dada uma qualquer classe C. O que resulta na seguinte expressão:

$$p(x_n|C_k, x_1, x_2, ..., x_{n-1}) = p(x_n|C_k)$$

Desta forma, é possível simplificar o cálculo da distribuição de probabilidade conjunta:

$$p(C_k, x_1, x_2, ..., x_n) = p(C_k) \prod_{j=1}^{N} p(x_j|C_k)$$

Paralelamente, de forma a evitar *overflow*, ou *underflow*, com o produto das probabilidades condicionadas, é possível alterar a fórmula anterior para utilizar o somatório de logaritmos:

$$\ln p(C_k, x_1, x_2, ..., x_n) = \ln p(C_k) + \sum_{j=1}^{N} \ln p(x_j|C_k)$$

Então é possível definir a fórmula do classificador *Naïve Bayes:*

$$C^{\text{Naïve Bayes}} = \underset{k \in \{0,1,\ldots,K\}}{\operatorname{argmax}} \ln p(C_k) + \sum_{j=1}^{N} \ln p(x_j | C_k)$$

Resultando numa classificação dos valores de *x* (*features*) segundo a classe com a maior probabilidade.

2.3.1. Implementação

De forma a implementar o classificador *Naïve Bayes* foi criada uma função, denominada *calc_bayes*, que recebe como argumentos o *training_set*, os índices do *set* que serão utilizados para treinar e validar, obtidos através do *StratifiedKFold* para *cross validation*, e a *bandwidth* a ser utilizada pelo *Kernel Density Estimator*.

De seguida, dividimos o *training_set* em 2 *sets*, nomeadamente *class_0_train* e *class_1_train*, que contêm apenas notas verdadeiras e falsas respetivamente, e para cada um destes conjuntos, calculamos a *priori probability* e populamos duas listas com estes valores como *default*:

$$p(C_{nota\ verdadeira}) = ln(\frac{\#notas\ verdadeiras}{\#total\ de\ notas}) \qquad p(C_{nota\ falsa}) = ln(\frac{\#notas\ falsas}{\#total\ de\ notas})$$

Posteriormente, para todas as *features* em cada classe, foi criado um *KDE* com um *kernel* gaussiano (perfazendo 8 no total), utilizando a classe *KernelDensity* do *sklearn*. Cada *KDE* foi ajustado com os dados de todos os pontos de treino da respetiva *feature*, obtendo assim o logaritmo da distribuição de probabilidade dessa feature através da função *score_samples* dando como *input* todos os valores dessa *feature* no *validation set*. Assim, as somas dos respetivos valores obtidos dos *KDE* aquando a execução do *score_samples* serão guardadas nas listas acima descritas, somando à respetiva *prior probability* anteriormente calculada.

Tendo os dados do *Naïve Bayes* para ambas as classes, as notas utilizadas para validação serão classificadas comparando os logaritmos das probabilidades em ambas classes, escolhendo a que tem um maior valor. Realizando o mecanismo descrito para todas as notas, obtém-se a uma lista com as classificações finais de todos os pontos do *validation set*.

O erro é calculado através da *accuracy*, nomeadamente *1 - accuracy*, usando o método *accuracy_score* do *sklearn* que irá comparar a previsão obtida e as classes reais, para os mesmos pontos, presentes no ficheiro do *data set*.

Por fim, o parâmetro *bandwidth* recebido como argumento no *calc_bayes* é otimizado (através de *cross validation*), variando-o de 0.01 a 1 com um salto de 0.02 em cada iteração. Esta otimização da *bandwith* é importante pois vai influenciar o peso que é dado a cada ponto, ou seja, uma *bandwidth* com um valor

baixo (*undersmoothing*) vai considerar apenas os pontos mais próximos para o cálculo da densidade e uma com um valor mais alto, ou *oversmoothing*, vai incluir pontos mais distantes.

3. Detalhes de implementação

Antes da implementação dos classificadores anteriormente explanados, foi necessário proceder a um tratamento do *data set* de forma a garantir a maior veracidade possível para os resultados obtidos.

Inicialmente foi carregado o conteúdo do ficheiro de dados que foi transposto para uma matriz. Posto isto, redimensiona-se os dados através do método de *standardization* onde a cada ponto é subtraído o valor da média e dividimos então o resultado pelo desvio padrão, isto resulta numa distribuição com uma média de 0 e um desvio padrão de 1 o que permite uniformizar os valores das várias features, visto que cada uma é medida na sua escala própria. Este processo pode ser executado sem corromper os dados pois não existe nenhuma feature cuja a medida seja em tempo ou distância, bem como também não existe nenhuma feature com a mesma escala de medida do que outra, logo a *standardization* não irá alterar relações significativas entre os dados**.**

$$x_{new} = \frac{x - \mu(X)}{\sigma(X)}$$

Após executar a função *stardardize* é necessário dividir o *data set* em dois conjuntos distintos, o conjunto de treino (*training set*) e o conjunto de teste (*test set)*. O *training_set* vai ser utilizado, em todos os classificadores, para otimizar o valor de cada parâmetro (o valor que minimiza o *training error)* e o *test_set* vai ser utilizado para obter uma estimativa do *true error* visto que os dados presentes no *test set* nunca foram utilizados por nenhum dos classificadores para ajustar o seu comportamento e portanto é um conjunto de dados completamente "novo" e inalterado para os classificadores. Para dividir o conjunto de dados original é usada a função *train_test_split* da biblioteca *sklearn* que vai fazer um *shuffle* aos dados e colocar dois terços no conjunto de treino e um terço no conjunto de teste, tudo isso de forma estratificada segundo a classe de cada nota bancária.

Ao longo do documento foi várias vezes referido que o método de *cross-validation* era utilizado para a otimização de um parâmetro, para isso é necessário dividir o *training set* num número N de *folds* (neste projeto N = 5) e para cada hipótese (para cada valor do parâmetro) vamos usar N - 1 *folds* para treino (sendo a média destes valores o *training error*) e o fold que ficou de fora vai ser utilizado para calcular o *validation error*. Este processo de treino com N-1 folds e validação com o fold que ficou vai ser repetido de forma a que todos os N folds

fiquem de fora numa iteração, feito isso, é calculada a média dos N *training error* e dos N *validation error*. A média do *validation error* será depois utilizada para comparar as execuções das diferentes hipóteses e escolher o melhor valor para o parâmetro.

4. Resultados

Execução	1	2	3	4	5	6	7	8	9	10
Logistic Regression Error	0.00875	0.01094	0.01532	0.00656	0.00875	0.00875	0.00875	0.00875	0.01532	0.01313
K-Nearest Neighbors Error	0.0	0.00219	0.00438	0.0	0.0	0.0	0.00438	0.0	0.00219	0.0
Naïve Bayes Error	0.0372	0.05689	0.07659	0.08753	0.05033	0.06127	0.05252	0.0744	0.06565	0.07002
McNemar's test entre Logistic Regression e K-NN	2.25	1.5	1.77778	1.33333	2.25	2.25	0.16667	2.25	3.125	4.16667
McNemar's test entre K-NN e Naïve Bayes	15.05882	23.04	31.0303	38.025	21.04348	26.03571	18.375	32.02941	27.03448	30.03125
McNemar's test entre Logistic Regression e Naïve Bayes	6.85714	13.7931	19.18421	30.13953	12.96	16.53125	12.89286	23.36111	17.92593	17.36111

Como forma de fazer uma comparação entre os três classificadores foi aplicado o teste de McNemar. Neste teste são comparados dois métodos de cada vez e o resultado é obtido através da aplicação da seguinte fórmula:

$$\frac{(|e_{01} - e_{10}| - 1)^2}{e_{01} + e_{10}} \approx \chi_1^2$$

Seja e_{01} o número de exemplos que o primeiro classificador classifica erroneamente, mas o segundo classifica corretamente e e_{10} é o número de exemplos primeiro classificador classifica corretamente, mas o segundo classifica erroneamente. Vai ser calculada a diferença entre e_{01} e e_{10} , que dividida pelo total de exemplos segue uma distribuição de qui-quadrado com um grau de liberdade um.

O termo *-1* é um termo de correção de continuidade porque as contagens de erro são discretas e a distribuição X^2_1 é contínua. Se o valor for maior que 3.84, deduzir que a hipótese não é nula, ou seja, que os dois classificadores não tem o mesmo resultado de execução, com 95% de confiança.

Tal como apresentado na tabela supra, foram executadas 10 execuções independentes e posteriormente foi calculada a média dos valores de cada uma das situações. É importante proceder à execução dos diferentes métodos mais do que uma vez pois os resultados obtidos estão dependentes da forma como os dados no *data set* são divididos inicialmente. É também relevante notar que os valores apresentados para os classificadores na tabela acima foram alcançados utilizando o *set* de teste (referido no ponto anterior), e portanto foi calculada a estimativa do *true error* em cada execução.

De realçar que o *test error* é medido como a fração de classificações incorretas, que é equivalente a *1 - accurancy* (a *accuracy* é obtida através do método *score* do *sklearn*). Isto ocorre em todos os classificadores de forma a possibilitar uma comparação justa entre as várias performances, ou seja, na regressão logística esta medida também é utilizada para calcular o *test error* e o *Brier score* é apenas utilizado para medir o erro na otimização do parâmetro no *cross validation*.

	Média Aritmética
Logistic Regression Error	0.0105
K-Nearest Neighbors Error	0.00131
Naïve Bayes Error	0.06324
McNemar's test entre Logistic Regression e K-NN	2.10694
McNemar's test entre K-NN e Naïve Bayes	26.17034
McNemar's test entre Logistic Regression e Naïve Bayes	17.10062

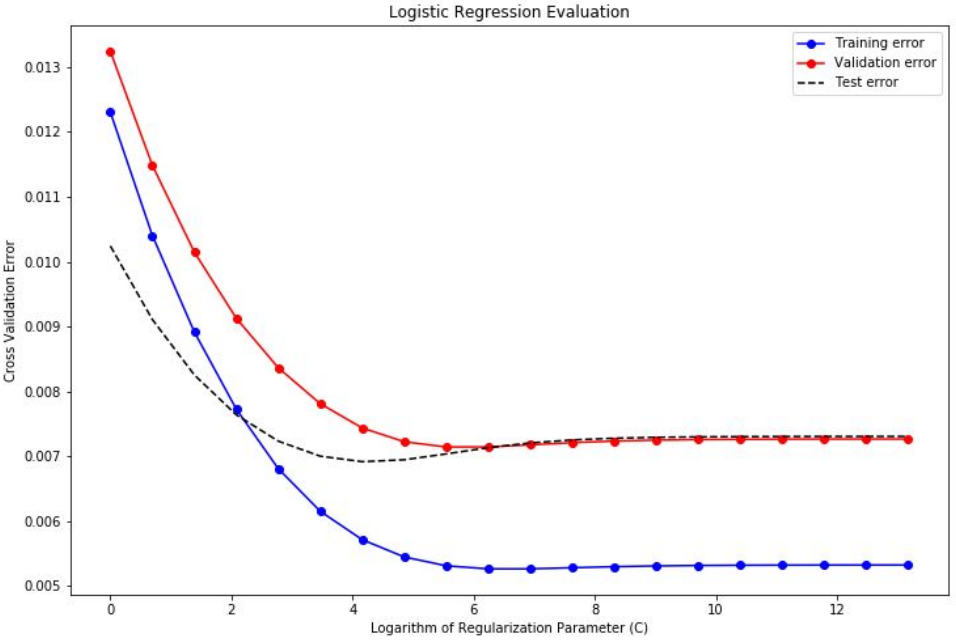


Fig. 1: Exemplo de execução da Regressão Logística variando o parâmetro C

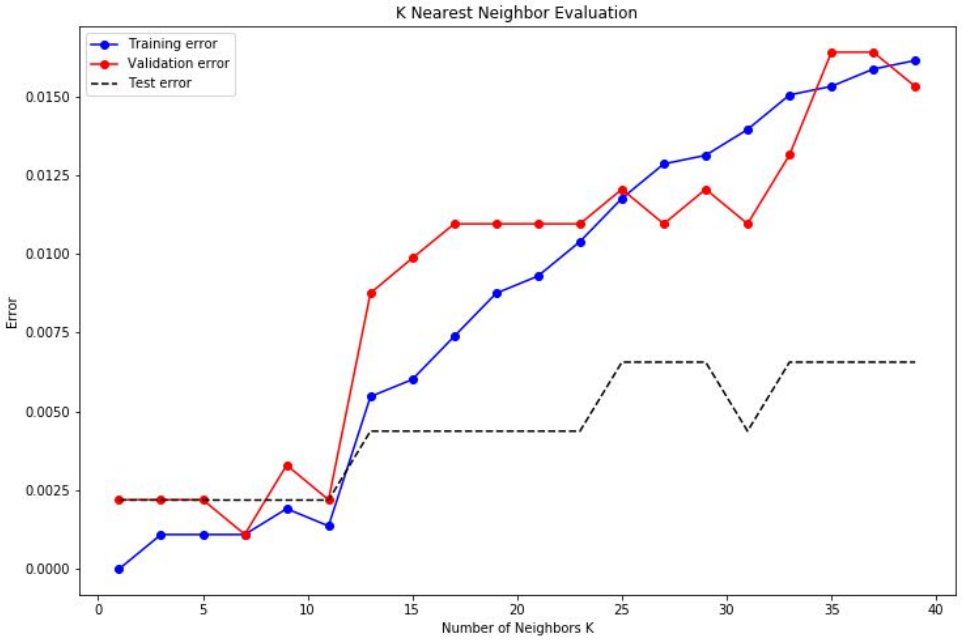


Fig. 2: Exemplo de execução do K-Nearest Neighbor variando o parâmetro K

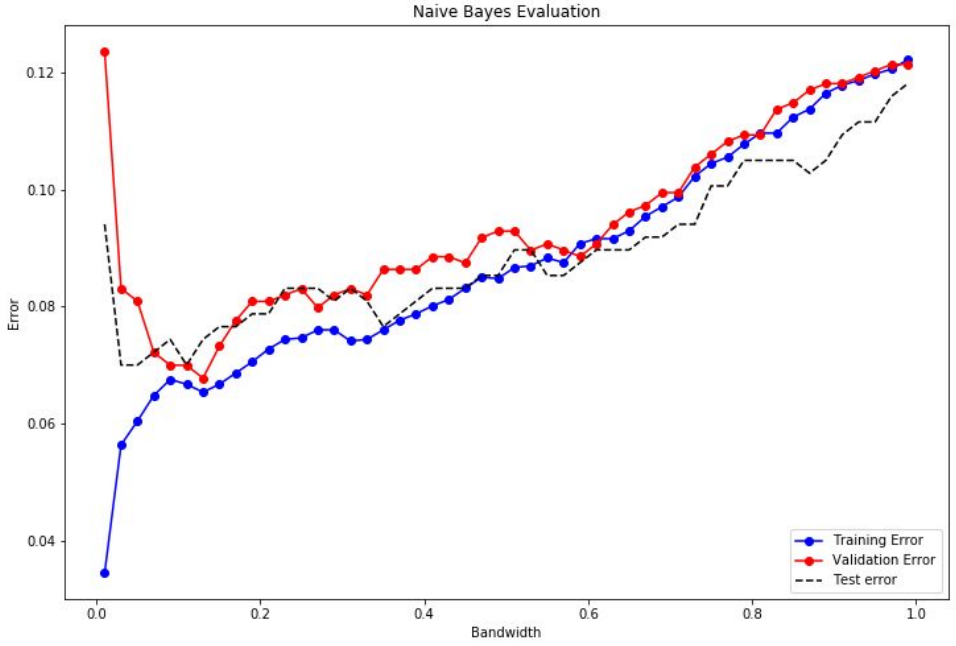


Fig. 3: Exemplo de execução do Naive Bayes variando o parâmetro de bandwidth

5. Conclusão

Analisando os dados da tabela referente às médias aritméticas das diferentes execuções, é possível concluir (segundo os valores do teste de *McNemar*) com um grau de confiança de 95% que o classificador *K-Nearest Neighbors* tem uma performance melhor do que o *Naive Bayes* e que o classificador *Logistic Regression* é também ele melhor do que o *Naive Bayes*, ou seja, o classificador *Naive Bayes* é o pior dos três classificadores apresentados. Quanto à comparação entre o *LogisticRegression* e o *K-Nearest Neighbors*, como o resultado é superior a 1.32, podemos concluir que *LogisticRegression* é melhor que *K-Nearest Neighbors* apenas com um grau de confiança de 75%.

Para finalizar gostaríamos apenas de referir que achámos o projecto de grande valor pois permitiu aplicar quase todos os conhecimentos teóricos adquiridos na cadeira bem como exigiu um estudo mais cuidadoso e aprofundado desses mesmos conhecimentos.