FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Aprendizagem Automática

2018/2019

Assignment 1

Relatório

Turno Prático: P2

Realizado por:

45679, Diogo Silvério
47525, Pedro Xavier

Professores:

Ludwig Krippahl
Joaquim Ferreira da Silva

20 de Outubro de 2018

1. Introdução

O objetivo deste trabalho é parametrizar, treinar e comparar o desempenho dos seguintes classificadores: Regressão Logística, *K-Nearest Neighbours* e Naïve Bayes. Para o fazer iremos utilizar um problema de classificação de notas verdadeiras ou falsas.

No conjunto de dados fornecido, cada linha corresponde a uma nota bancária e cada nota é caracterizada por 4 *features* que correspondem respetivamente à variância, à assimetria, à curtose da imagem *Wavelet* Transformada e à entropia da imagem da nota bancária, e por fim um rótulo da nota (um inteiro com valores 0 ou 1), para distinguir entre notas bancárias verdadeiras e notas bancarias falsas. Não se sabe a que valor numérico correspondem as notas verdadeiras ou falsas.

2. Pré processamento de dados

Os dados são lidos, baralhados de forma a evitar relações que possam existir entre eles (Ex: ordenação) e separados em duas matrizes: uma matriz com as 4 *features* e uma com as classes. Depois os dados da matriz de *features* são redimensionados através de estandardização de forma a que todo o conjunto tenha uma distribuição com uma média $\mu = 0$ e um desvio padrão $\sigma = 1$.

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

A estandardização apenas se aplica a dados cujo significado não seja afectado pela alteração dos valores, que é o caso das *features* dos dados do problema. Não faz sentido aplicar estandardização às classes pois estas apenas indicam se uma nota é ou não verdadeira.

Por fim, os dados são divididos em dois conjuntos: o conjunto de treino e o conjunto de teste. O conjunto de treino permitirá treinar e optimizar os parâmetros de cada classificador enquanto o conjunto de teste permitirá estimar o erro verdadeiro de cada classificador.

3. Explicação dos classificadores

Regressão Logística

A Regressão Logística é um classificador que faz uma análise preditiva, baseada numa variável dependente e binária. Apesar de ser uma regressão pode ser utilizada como classificador de forma a obter um hiperplano que separe as diferentes classes. A regressão logística é utilizada para explicar a relação entre uma variável binária e uma ou mais variável nominais, ordinais, de intervalo, ou de nível de razão. Neste caso, é utilizada de forma a estimar a probabilidade de uma nota ser verdadeira ou falsa através das suas *features*.

Para avaliar a Regressão Logística foi utilizado o Brier Score que mede o erro quadrático entre a probabilidade prevista pela hipótese e a verdadeira classe do ponto. Poderia ter sido utilizada a função score do *sklearn* mas o Brier Score oferece uma curva mais suave.

A Regressão Logística pode ser regularizada através do parâmetro C para ajustar o treino do classificador de forma a mitigar *overfitting* pois este parâmetro permite suavizar a curvatura da regressão. Se a regressão tiver uma curvatura muito acentuada é possível que o discriminante esteja demasiado perto dos dados, ou seja, podemos estar perante um caso de *overfitting*.

Este parâmetro vai ser optimizado através de 5-Fold Cross-Validation e o melhor valor será aquele que oferecer menor erro de validação.

K-Nearest Neighbours

Ao contrário de Regressão Logística que é um tipo de e*ager learning*, o K-Nearest Neighbours é um tipo de *lazy learning*.

Com *eager learning*, temos uma classe de hipóteses e um modelo que representa essa classe com parâmetros, que é treinado através do ajuste desses mesmos parâmetros. Durante a fase de treino tenta-se extrair o máximo de informação do conjunto de treino e depois guarda-se apenas o modelo de treino. Como uma resposta a uma *query* feita ao modelo é baseada nos seus parâmetros, não é necessário utilizar o conjunto de treino e por isso pode ser descartado.

Com *lazy learning*, não existe uma fase de treino do modelo, apenas é guardado o conjunto de treino. Quando é feita uma *query* ao classificador, é computada uma resposta baseada nos dados de treino.

O K-Nearest Neighbours calcula a distância ou diferença, através de uma função de distância, entre as *features* de todos os pontos fornecidos e as do ponto da *query* que se pretende classificar. De seguida, estes valores são ordenados de forma ascendente e o ponto é classificado de acordo com a moda das classes dos primeiros K vizinhos, os mais próximos ou idênticos. Quanto menor for o valor da resposta mais próximos ou idênticos são os pontos, e vice-versa.

Existem diversas funções de distâncias que podem ser utilizadas, mas a escolhida foi a distância de Minkowski com $p = 2$, ou seja a distância Euclidiana, pois os dados do problema são contínuos.

No caso do K-Nearest Neighbours, o parâmetro que se quer optimizar é o K, que representa o número de vizinhos a utilizar numa *query*. Em geral, um menor valor de K significa que o ponto vai ser classificado de acordo com uma vizinhança local, a sua classificação é afectada apenas por pontos próximos, enquanto que um maior valor de K significa que o ponto vai ser classificado de acordo com uma vizinhança global, a sua classificação é afectada por pontos mais longínquos.

Este parâmetro vai ser optimizado através de 5-fold cross-validation e o melhor valor será aquele que oferecer menor erro de validação.

Naïve Bayes

Um classificador Bayes é um classificador probabilístico baseado na regra de Bayes, que pode ser escrita da seguinte forma no contexto do nosso problema, em que c é uma classe e x é um vector de *features*:

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

Como $p(X = x)$ é independente de c, então:

$$p(C = c|X = x) \propto p(C = c)p(X = x|C = c)$$

E como:

$$p(C = c)p(X = x|C = c) = p(C = c, X = x)$$

Apenas precisamos de calcular a probabilidade conjunta de todas as combinações de classes e *features*:

$$C^{Bayes} = \underset{c \in \{0,1,\dots,N\}}{argmax} p(C = c, X = x)$$

Este classificador é ideal pois minimiza a probabilidade de classificar erradamente um ponto, mas geralmente não é prático calcular todas as combinações de classes e *features*.

No classificador Naïve Bayes vamos assumir que as *features* são condicionalmente independentes de uma dada classe o que simplifica o cálculo das probabilidades conjuntas:

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

Para evitar problemas de *overflow* ou *underflow* é aplicado um logaritmo:

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

Assim obtemos o nosso classificador Naïve Bayes:

$$C^{Bayes} = \underset{c \in \{0,1,\dots,N\}}{argmax} \ln p(C_k) + \sum_{j=1}^{N} \ln p\left(x_j \middle| C_k\right)$$

Implementação

Este classificador foi implementado através de duas funções: *NaiveBayes_train* e *NaiveBayes_predict*.

A função *NaiveBayes_train* começa por dividir as classes em classe das notas verdadeiras e classe das notas falsas, e calcula a probabilidade *a priori* de cada uma, $p(C_0)$ e $p(C_1)$. Depois divide as *features* em 8 conjuntos, cada um correspondente a uma *feature* e uma classe diferente. Cria 8 Kernel Densities utilizando a classe *KernelDensity* da biblioteca *sklearn*, cada um para os conjuntos anteriormente criados que guardam os seus respectivos valores através da função *fit*. Por fim retorna as probabilidades *a priori* das classes e os 8 *kernels*.

A função *NaiveBayes_predict* calcula o logaritmo da distribuição de probabilidade de cada *feature* por classe através dos 8 *kernels* anteriormente criados na função *NaiveBayes_train* com o auxílio da função *score_samples*. Depois executa a classificação encontrando o máximo entre a soma do logaritmo da probabilidade classe a priori e dos logaritmos das distribuições de *features* das duas classes.

As distribuições são calculadas de forma não paramétrica utilizando Estimadores de Densidade Kernel Gaussianos pois os nossos dados são contínuos e porque as distribuições gaussianas ajustam-se facilmente a qualquer tipo de dados. Os KDEs Gaussianos têm um parâmetro h que determina a largura da curva da distribuição. Se o h for muito grande, teremos uma distribuição que não curva o suficiente, *underfitting*, mas se for muito pequeno curvará demais, *overfitting*.

Este parâmetro vai ser optimizado através de 5-Fold Cross-Validation e o melhor valor será aquele que oferecer menor erro de validação.

4. Resultados

Os erros de validação e os valores óptimos para os parâmetros C, N, h foram calculados através de 5-Fold Cross Validation. Os erros dos classificadores Regressão Logística e K-Nearest Neighbours foram medidos com o auxílio da função *score* das respectivas classes e o erro do Naïve Bayes foi medido através da função *accuracy_score* da biblioteca *sklearn*.

Como os dados são baralhados antes de serem repartidos, obtemos diferentes conjuntos de treino e de teste em diferentes execuções e por isso é normal obter resultados diferentes. Foram então executados 7 testes para aumentar a veracidade dos resultados.

Os parâmetros foram escolhidos de acordo o menor erro de validação, ou seja, quanto menor o erro de validação, melhor é o parâmetro.

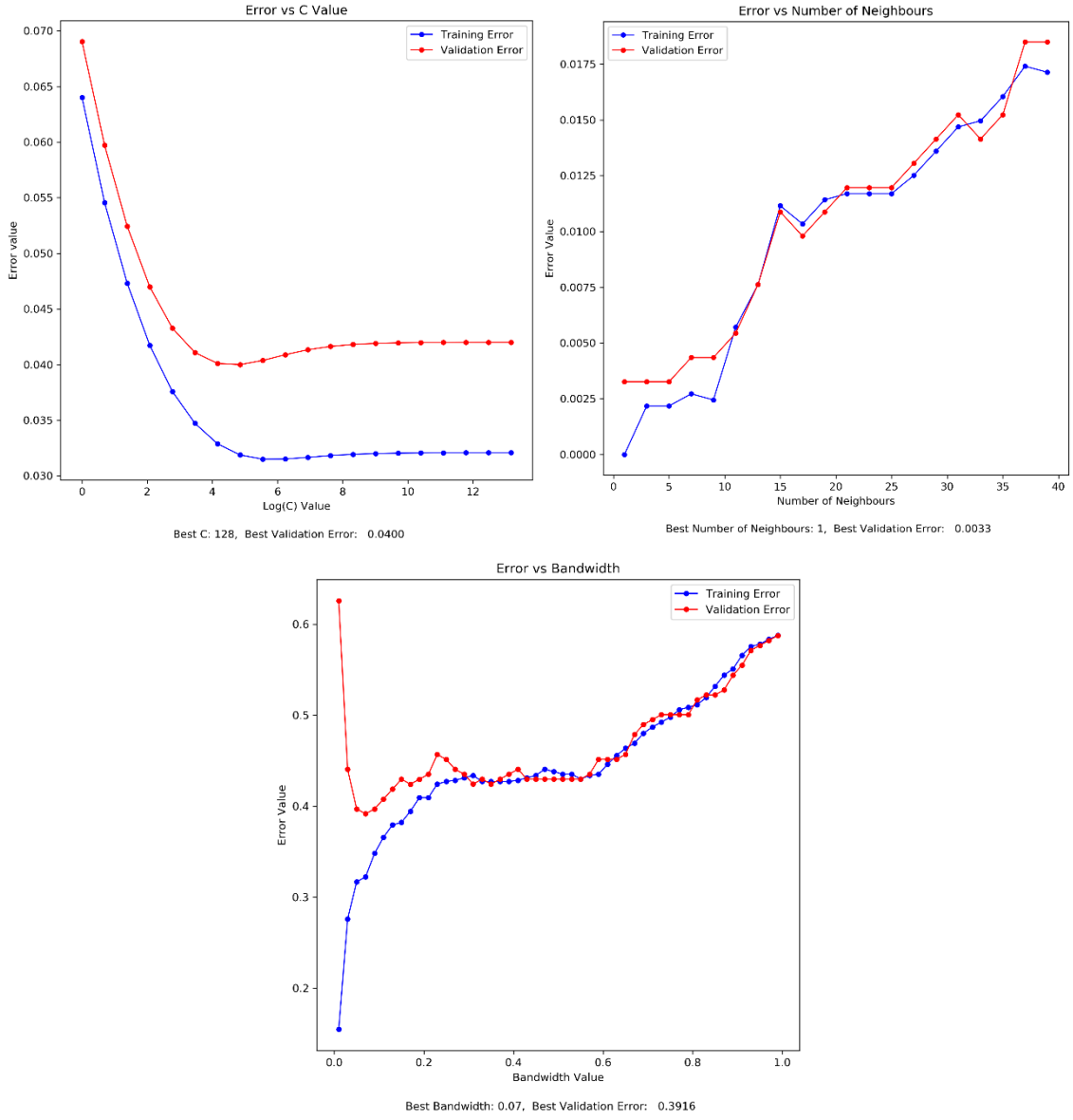

Figura 1 – Gráficos do erro de treino e validação em função dos diferentes parâmetros para uma execução.

Depois de obtidos os melhores parâmetros, foi então medido o erro de teste de cada classificador no conjunto de teste criado no início, de forma a obter um estimador do erro verdadeiro. É importante medir este erro num conjunto diferente daquele usado para treino para evitar criar um *bias* na estimação do erro verdadeiro, erro calculado no conjunto de todos os dados do universo.

A tabela seguinte mostra os erros de teste e os melhores parâmetros obtidos dos diferentes classificadores em diferentes execuções.

Testes		1	2	3	4	5	6	7
Logistic Regression	Test Error	0.0335	0.0354	0.0416	0.0377	0.0319	0.0218	0.0359
	Best C	512	512	128	128	256	64	128
K-Nearest Neighbours	Test Error	0.0	0.0022	0.0022	0.0011	0.0	0.0	0.0011
	Best N	1	1	3	1	5	1	9
Naïve Bayes	Test Error	0.3482	0.3428	0.3863	0.37	0.3101	0.3483	0.3754
	Best h	0.09	0.15	0.11	0.07	0.11	0.09	0.09

Para comparar os três classificadores, foi usado o teste de McNemar:

$$\frac{(|e_{01} - e_{10}| - 1)^2}{e_{01} + e_{10}} \approx X_1^2$$

Sendo e_{01} o número de pontos mal classificados pelo primeiro classificador, mas bem classificados pelo segundo e e_{10} o número de pontos bem classificados pelo primeiro classificador e mal classificados pelo segundo. O teste aproxima uma distribuição chi-quadrado com grau de liberdade 1 o que significa que se o seu valor for maior do que 3.84, podemos rejeitar a hipótese de que os dois classificadores têm desempenhos idênticos.

Testes	1	2	3	4	5	6	7
McNemar Logistic vs KNN	0.57143	1.5	1.(3)	2.28571	0.16667	2.5	4.16667
McNemar Logistic vs Naïve Bayes	29.25714	18.58065	18.58065	15.56757	33.06522	22.66667	22.75556
McNemar KNN vs Naïve Bayes	32.23684	32.23684	26.03571	28.03333	40.02381	37.20930	37.02564

5. Conclusão

A tabela anterior permite-nos concluir que os classificadores Regressão Logística e K-Nearest Neighbours apresentam melhor desempenho nesta tarefa do que o classificador Naïve Bayes com grau de confiança de 95%, pois ambos apresentam valores maiores que 3.84. Mas não se pode rejeitar a hipótese de que a Regressão Logística e o K-Nearest Neighbours têm desempenhos idênticos pois os valores do teste de McNemar são mais pequenos do que 3.84.