UNIVERSIDADE DE ÉVORA

Aprendizagem Automática

Trabalho Prático 1

2022/2023

Carlos Palma 46520
Bruno Ascenção 45460

Naive Bayes – Apredizagem Automática

Introdução

No âmbito da disciplina de Aprendizagem Automática foi-nos incumbida a implementação do algoritmo Naive Bayes para tipos nominais em Python e testá-lo em scikit-learn com um estimador suavizado, assim como a avaliação do classificador através da exatidão e precisão.

Para facilitar a implementação e testagem foi-nos fornecido 5 DataSets diferentes em formato csv.

Foi pedida a implementação dos seguintes componentes:

- A classe **NaiveBayesUevora** que para cada objeto deverá permitir a escolha de um estimador suavizado de acordo com a fórmula dada, e aceitar dados nominais na forma de strings
- O método **fit(x, y)** para gerar um classificador a partir dum conjunto de treino com etiquetas;
- O método **predict(X)** que fará predições em função dum conjunto de dados de teste;
- O método **accuracy_score(x, y)** para calcular a exatidão
- O método **precsision_score(x, y)** para calcular a precisão.

Desenvolvimento/solução(Classes e métodos)

Para o funcionamento do algoritmo, definimos vários métodos, o método **__init__(self, alpha)** é um "*construtor*", que inicia o objeto com que queremos trabalhar, recebendo o alfa que utilizaremos para calcular o suavizador, inicia também dicionários e os dados treino e teste, assim como variáveis que iremos utilizar mais tarde.

Carlos Palma 46520
Bruno Ascenção 45460

Implementámos o método **fit(self, x, y)**, que ajusta o modelo aos seus dados, recebe dados de treino, insere as probabilidades á priori nos seus dicionários e calcula as probabilidades necessárias para as predições.

Auxiliares ao método anterior temos 3 métodos, **_calc_probyesorno(self)**, **_calc_featureperclass(self)** e **_calc_featuretotal(self)**, cada um calcula o que o seu nome indica, o primeiro método calcula a probabilidade de cada atributo por classe, ou seja, calcula a probabilidade de yes ou no, o segundo calcula a probabilidade de cada atributo por classe, ou seja, calcula e guarda o numero de ocorrências de cada feature assim como calcula e guarda a probabilidade de cada feature consoante a classe e guarda indicando a classe. Finalmente o último método, probabilidade total de cada atributo, ou seja, calcula e guarda no respetivo sítio a probabilidade de cada feature ocorrer independentemente da classe.

Seguidamente implementámos um dos métodos mais importantes, o método que faz as predições, **predict(self, X)**, que calcula a probabilidade tendo em conta o conjunto de dados de teste.

Para a predição com suavizador foi utilizada a fórmula dada:

$$\hat{\theta}_i = \frac{x_i + \alpha}{N + \alpha d} \qquad (i = 1, \ldots, d),$$

Os métodos **accuracy(self,X,y)** e **precision_score(self,X,y)** permitem avaliar o modelo consoante exatidão e precisão. A acurracy e precision são calculados através das seguintes fórmulas:

- Acurracy = Número de predições corretas / Número de predições;
- Precision = Número de predições corretas / Número de predições positivas*;

*Podem ser falsos positivos!

O funcionamento do algoritmo está dependente das bibliotecas **numpy**, **pandas** e **sklearn**, que permitem a utilização de métodos que facilitaram a implementação do trabalho.

O projeto está dividido em vários ficheiros, o NaiveBayesUevora.py, que é onde o algoritmo é implementado, e contem a classe principal **NaiveBayesUevora** e é responsável por implementar a classe com os métodos pedidos. Tem também um ficheiro á parte chamado Teste.py, que é responsável por carregar os ficheiros com os dados, pedir o alpha ao utilizador e chamar os métodos de classe. Ainda tem também os ficheiros com os dados, estando todos estes na mesma pasta base do projeto.

Outputs

Outputs com dados 1:

```
45460$ python3 -u "/home/carlos/Desktop/Universidade/AA/Trabalho_1/46
520_45460/Teste.py"
Digite o alpha:0
ACCURACY:  85.71428571428571 %
PRECISION:  81.31868131868131 %
carlos@cmmp-GF63-Thin-9SC:~/Desktop/Universidade/AA/Trabalho_1/46520_
45460$ python3 -u "/home/carlos/Desktop/Universidade/AA/Trabalho_1/46
520_45460/Teste.py"
Digite o alpha:1
ACCURACY:  85.71428571428571 %
PRECISION:  81.31868131868131 %
carlos@cmmp-GF63-Thin-9SC:~/Desktop/Universidade/AA/Trabalho_1/46520_
45460$ python3 -u "/home/carlos/Desktop/Universidade/AA/Trabalho_1/46
520_45460/Teste.py"
Digite o alpha:5
ACCURACY:  90.47619047619048 %
PRECISION:  86.90476190476191 %
```

Outputs com dados 2:

```
520_45460/Teste.py"
Digite o alpha:0
ACCURACY:  85.71428571428571 %
PRECISION:  81.31868131868131 %
carlos@cmmp-GF63-Thin-9SC:~/Desktop/Universidade/AA/Trabalho_1/46520_
45460$ python3 -u "/home/carlos/Desktop/Universidade/AA/Trabalho_1/46
520_45460/Teste.py"
Digite o alpha:1
ACCURACY:  85.71428571428571 %
PRECISION:  81.31868131868131 %
carlos@cmmp-GF63-Thin-9SC:~/Desktop/Universidade/AA/Trabalho_1/46520_
45460$ python3 -u "/home/carlos/Desktop/Universidade/AA/Trabalho_1/46
520_45460/Teste.py"
Digite o alpha:5
ACCURACY:  90.47619047619048 %
PRECISION:  86.90476190476191 %
carlos@cmmp-GF63-Thin-9SC:~/Desktop/Universidade/AA/Trabalho_1/46520_
45460$
```

Conclusões e problema

A nosso ver foi um trabalho desafiante que envolveu muita pesquisa, principalmente a nível teórico e alguma destreza a nível prático.

Tivemos algumas dificuldades em entender o problema em si e em perceber quais eram as estruturas mais adequadas á resolução do problema proposto, no entanto conseguimos a implementação desejada.

Entendemos o que está por detrás deste algoritmo e a forma como o mesmo funciona e é deve ser implementado considerando este projeto bastante produtivo.