UNIVERSIDADE DE ÉVORA

Disciplina: Aprendizagem Automática Docente: Teresa Gonçalves
Trabalho realizado por: Pedro Emílio nº52649 e Luís Carvalho nº51817

Introdução

No âmbito da UC de Aprendizagem Automática foi-nos proposta a realização de um trabalho que consistiu na implementação e análise de dois algoritmos de classificação, sendo eles o KNN (K-Nearest Neighbors) e Naive Bayes, com a integração no ambiente scikit-learn, estas implementações devem ser compatíveis com a substituição por outros algoritmos de classificação.

Desenvolvimento KneighborsClassUE

Na classe **KNeighborsClassUE**:
__init__: Esta função é utilizada para definir os parâmetros iniciais de um objeto dessa classe quando ele é criado, sendo o **k** o número de vizinhos a serem considerados (com valor de 3 se não for fornecido) e o **p** representa a distância Euclidiana (com valor de 2 se este não for fornecido).
fit: Esta função é utilizada para treinar o modelo, sendo o **X** o conjunto de dados de treino (ou seja, os atributos) e sendo o **Y** as etiquetas que correspondem aos dados de treino.
predict: Esta função é utilizada para realizar previsões com base nos k-vizinhos mais próximos ao ponto de teste fornecido e o seu funcionamento é o seguinte:
-Para cada ponto de teste em **X**, a função calcula a distância entre esse ponto e todos os pontos no conjunto de treino utilizando a distância Euclidiana.
-De seguida os índices dessas distâncias ordenadas são armazenados na variável **indicesOrdenados**, representando os índices dos vizinhos mais próximos em ordem crescente de distância.
-A função conta as classes dos k vizinhos mais próximos e armazena essas informações em classes e **contagemClasses**.
-Sendo classe mais frequente entre os vizinhos mais próximos é escolhida e adicionada à lista **etiquetasPrevistas**.
-Por fim a lista final de etiquetas previstas é retornada.
Score: Esta função avalia a precisão do modelo através da comparação entre as previsões com as etiquetas verdadeiras e fornecendo a proporção de previsões corretas em relação ao total de exemplos no conjunto de teste, o seu funcionamento é o seguinte:
-Primeiramente é criada a variável **previsoesCorretas** que serve para contar o número de previsões corretas.

-De seguida garantimos que a variável **X** tenha pelo menos duas dimensões. Isto é necessário para lidar com casos em que **X** pode ser uma matriz unidimensional.
-A variável **Y** vai ser isimilar à **X**, garantindo que **y** tenha pelo menos duas dimensões e uma coluna.
-A seguir obtemos as previsões do modelo através da variável **previsao** que usa a função **prever** e que garante que tenha pelo menos duas dimensões e uma coluna.
-Depois é usado um loop for, que percorre cada elemento em y e compara com a previsão correspondente, se a previsão for igual à etiqueta verdadeira, incrementa **previsoesCorretas**.
-Por fim a função retorna a precisão, que é calculada dividindo o número de previsões corretas pelo número total de exemplos no conjunto de teste.

Desenvolvimento do NBayesClassUE

Na classe **NBayesClassUE**:
__init__: Esta função é utilizada para inicializar um objeto da classe, definindo o parâmetro alpha que será usado para suavizar as estimativas de probabilidades durante o processo de treinamento do modelo.
fit: Esta função treina o modelo de classificação calculando as probabilidades a priori e condicionais suavizadas com base nos dados de treino **X** e **y**. Sendo depois essas probabilidades armazenadas no objeto para serem usadas posteriormente nas previsões.
O seu funcionamento é o seguinte:
-Primeiro é “criada” a variável **self.classesUnicas** que armazena um conjunto das classes únicas presentes no conjunto de etiquetas **y**.
-De seguida é “criada” a variável **self.probabilidadesClasse** que inicializa uma lista que vai ser usada para armazenar as probabilidades calculadas.
-Depois criamos um **loop for** que é usado para calcular as probabilidades para cada classe, iterando sobre as classes únicas e calculando a probabilidade suavizada para cada uma com base na contagem de instâncias no conjunto de dados.
-A seguir é criado um segundo **loop for** que percorre cada atributo no conjunto de dados e calcula as probabilidades condicionais para cada valor único desse atribut, sendo a suavização aplicada usando o parâmetro alpha.
-Por fim as probabilidades condicionais suavizadas são armazenadas na lista **self.probabilidadesClasse**.
predict: Esta função faz previsões para um conjunto de instâncias com base nas probabilidades calculadas durante o treino do modelo, sendo escolhida a classe com a maior probabilidade para cada instância.
O seu funcionamento é o seguinte:
-Começamos por cirar uma lista chamada **previsoes** que armazenada as previsões para cada instância em X.
-Depois criamos um **loop for** que percorre cada instância (conjunto de atributos) em X.
-As variáveis **probabilidadeMaxima** e **classeMaiorProbabilidade** são utilizadas para rastrear a maior probabilidade e a classe correspondente.
-De seguida criamos um segundo **loop for** que percorre cada classe única presente no modelo.
-A seguir criamos um terceiro **loop for** que percorre cada atributo na instância atual.

-A variável **prob** é atualizada multiplicando pela probabilidade condicional do atributo dado a classe.
-Se a probabilidade atual for maior que a **probabilidadeMaxima**, atualiza **probabilidadeMaxima** e **classeMaiorProbabilidade** com os valores atuais.
-Oresultado final é a classe com a maior probabilidade para a instância atual, e esta classe é adicionada à lista de previsões.
-Por fim a função retorna a lista de previsões para todas as instâncias em **X**.

Conclusão

Análise Crítica - KneighborsClassUE

No ficheiro **iris.csv**:

O modelo KNN apresenta um desempenho consistente, com uma precisão em torno de 94.74%, para a maioria das configurações de k e p. O aumento de k de 1 para 9 promove um ligeiro aumento no desempenho, o que indica que considerar mais vizinhos pode melhorar a generalização do modelo.
O parâmetro p não parece ter um impacto muito grande no desempenho, já que não há mudanças notáveis ao variar seu valor entre 1 e 2.

No ficheiro **rice.csv**:

Os resultados indicam um desempenho geralmente bom, o aumento de K de 1 para 9 não parece resultar em mudanças muito significativas no desempenho, significando que o modelo tem uma estabilidade relativa na capacidade de generalização para diferentes quantidades de vizinhos considerados.
Comparando os valores de p, não há uma diferença substancial no desempenho entre os casos em que p = 1 e p = 2. Sendo observado que o modelo é mais sensível à variação de K do que de p para este conjunto de dados específico.

No ficheiro **wdbc.csv**:

Os resultados indicam uma redução no desempenho do modelo KNN em comparação com os casos anteriores, com precisões, a variar entre 67.83% e 83.92%. À medida que K aumenta, observamos que o desempenho tende a diminuir, indicando uma possível dificuldade do modelo em generalizar para um maior número de vizinhos.
A variação de p parece ter um impacto limitado no desempenho, com as precisões sendo relativamente consistentes entre os casos p = 1 e p = 2. A precisão de 67.83% para K = 9, p = 2 sugere que o modelo pode sofrer de *overfitting*.

Análise Crítica - NBayesClassUE

Ficheiro **rice.csv**:

Não há sinais claros de “overfitting” ou “underfitting”. O modelo parece bem ajustado.

Ficheiro **iris.csv**:

Não há sinais claros de “overfitting” ou “underfitting”. O modelo parece bem ajustado.

Ficheiro **wdbc.csv**:

Há evidências de “overfitting”, pois o desempenho no conjunto de teste é consideravelmente mais baixo que no conjunto de treino. O modelo pode estar a ajustar-se mais aos dados de treino.