Relatório do primeiro trabalho prático de Aprendizagem Automática

Henrique Rosa – Diogo Matos
51923 – 54466

Introdução:

O presente relatório foi realizado no âmbito da unidade curricular Aprendizagem Automática integrante na Licenciatura em Engenharia Informática da Universidade de Évora, Colégio Luís António Verney.

O documento aborda de forma sucinta, todos os passos para a nossa implementação prática dos algoritmos para problemas de classificação requeridos no enunciado, respeitando a integração com o ambiente scikit-learn.

Funcionalidades:

O Código fonte desempenha as funcionalidades de resolução de problemas de classificação, para exemplos com atributos com valores tanto numéricos quanto nominais, pela implementação dos algoritmos KNN (K-nearest neighbours) e de Naïve Bayes, respectivamente.

A implementação do algoritmo KNN , com k variável, tem por base a métrica de minkowski, com p variável. Já a implementação do algoritmo de Naïve Bayes tem por base o estimador de Lidstone com, com alpha variável.

As implementações dos algoritmos para as funcionalidades acima descritas têm parametrização de natureza dinâmica (p, k, e alpha estão todas a critério do utilizador, tendo este assim controlo sobre as soluções de classificação)

Relativamente às métodos comuns às implementações dos dois algoritmos: A método fit tem a funcionalidade de guardar e converter os dados inseridos como exemplos de treino para dados operáveis pela método (por exemplo, passar o array de dataframe, proveniente da leitura do ficheiro CSV pela biblioteca pandas, para um array numpy para poder ser iterado e indexado corretamente); a método predict tem a funcionalidade de, com base nos exemplos e nas etiquetas de treino, prever, com base na lógica funcional do algoritmo, uma classificação, na forma de etiqueta, para um ou mais conjuntos de dados. A método score vai, com base na método predict, avaliar o desempenho do algoritmo, comparando as etiquetas previstas, para um ou mais conjuntos de dados, com etiquetas reais que lhe são fornecidas.

Arquitetura do código fonte:

Com base nas boas práticas de programação e desenvolvimento de software, todas as métodos presentes no código fonte estão devidamente comentadas, tanto com um bloco de comentário com os inputs e outputs da método logo acima da mesma, como com comentários de funcionamento ao longo da método em si, que explicam mais detalhadamente o funcionamento da método

Todas as métodos requeridas pelo enunciado, nomeadamente os métodos fit, predict e score, estão implementadas e funcionais, mas também foram implementadas métodos auxiliares (devidamente identificadas nos comentários), que, como o nome indica, auxiliam no desempenho das funcionalidades das métodos principais, promovendo também a facilidade de leitura do código, bem como a sua manutenção e alteração.

A implementação dos algoritmos foi dividida em dois ficheiros, cada um com a classe respectiva ao algoritmo que representa, bem como um ficheiro main, para auxiliar na realização e estruturamento dos testes dos algoritmos (um pouco mais esteticamente agradável do que ver só números a aparecer no terminal)

Ficheiros e lógica de implementação dos algoritmos:

→ main.py: Contém a lógica da leitura e tradução para data frame do conteúdo dos ficheiros CSV disponibilizados pela professora. Também corre as implementações dos algoritmos por importação dos ficheiros KNeighborsClassUE.py e NBayesClassUE.py e, de modo a visualizar o seu funcionamento com prints para o terminal.

Nos métodos contidos nos ficheiros seguintes está muitas vezes presente o parâmetro "self" nos argumentos da função, que desempenha o papel de dar ao método a referência do objeto ao qual pertence, para o correto funcionamento do código

→ KNeighborsClassUE.py: Contem a implementação do algoritmo KNN

- def __init__(self, k, p): construtor da classe, para a criação do objeto que serve como modelo de classificação, com k e p como argumentos para a lógica do algoritmo KNN e da métrica de Minkowski.
- def fit(self, X, y): Vai guardar o conjunto de exemplos de treino e das suas respectivas etiquetas, provenientes do ficheiro CSV, aquando da sua chamada no ficheiro main.py.
- def Metric(self, x1, x2): Implementa a lógica da métrica de minkowski. Este método retorna a distância entre x1 e x2 (com p=2 seria a distância euclidiana, com p=1 seria a distância de manhattan).
- def predict(self, input): Para um array de conjunto de dados inserido no argumento input, faz a previsão da classe para cada um desses conjuntos. Com base na métrica retornada da função metric, executada em loop, comparando o input com cada um dos conjuntos de dados e treino já dentro do objeto. Depois de obtidos os k vizinhos mais próximos de acordo com a métrica, coloca num array a etiqueta que está mais vezes presente no array das etiquetas dos KNN, e parte para o próximo conjunto de dados do input, retornando no fim o array com as etiquetas de cada um dos conjuntos de dados inseridos no input, por ordem.
- def score(self, X, y):por cada conjunto de dados inserido em X, vai comparar a sua respectiva etiqueta real em y com a etiqueta retornada do método predict, somando um a um somador quando são iguais e não somando quando não são, retornando depois o valor final do somador dividido pelo número de conjuntos em X, efetivamente retornando a percentagem (em valor decimal) do sucesso da previsão do modelo
- def arr_copy(arr1): Em python, tendo dois arrays arr1 e arr2, quando se quer copiar os valores de um array para outro, não é possível fazê-lo da maneira convencional: arr1=arr2, pois isto vai simplesmente passar a referência de arr1 para arr2, ou seja, se mudarmos arr2, também vamos mudar arr1, pois na prática são o mesmo array, guardado mesmo lugar, mas com nomes diferentes. Este método foi um "*workaround*" desenhado simplesmente para resolver este problema, copiando iterativamente todos os valores de arr1 para arr2 e retornando a referência de arr2.

→NBayesClassUE.py:

- def __init__(self, alpha): construtor da classe, para a criação do objeto que serve como modelo de classificação, com alpha como argumentos para a lógica do algoritmo NBayes relativamente ao estimador de Listonde.
- def fit(self, X, y): Vai guardar o conjunto de exemplos de treino e das suas respectivas etiquetas, provenientes do ficheiro CSV, aquando da sua chamada no ficheiro main.py, fazendo uma verificação se o tamanho dos inputs é o mesmo. X é o array que contém os conjuntos de valores de atributos de treino; y é o array das etiquetas de treino para os valores de X.
- def predict(self, X): Para um array de conjuntos de dados inserido no argumento X, faz a previsão da classe para cada um desses conjuntos. Com base na lógica do algoritmo Naive Bayes, este método vai guardar num array todas as probabilidades relativas a uma classe (usando o estimador de lidstone para as obter), e vai guardar esse array dentro de outro array que vai guardar todos os conjuntos de probabilidades, cada um relativo a uma classe. Depois vai fazer a multiplicação de todas as probabilidades respectivas a cada classe, e vai achar a maior, sendo a etiqueta da classe com maior probabilidade adicionada a um array de retorno, partindo depois para o próximo conjunto de dados de X, obtendo todas as classificações e retornando o array com estas, por ordem.
- def lidstone(self, nx, total, nvals): Retorna o valor da probabilidade estimada pelo estimador de lidstone, com alpha já indicado na criação do modelo, tem como inputs nx, total, e nvals necessários para o cálculo da probabilidade, como consta na fórmula do estimador indicada no enunciado.
- def score(self, X, y): Retorna a percentagem de etiquetas corretamente previstas. Recebe como entrada o array de arrays X, onde cada array contém os valores dos atributos a serem testados, e o array y com as etiquetas corretas para comparação. Inicialmente, verifica se os tamanhos dos inputs são iguais. Se forem, para cada valor em y, verifica se a etiqueta prevista pelo método predict está correta, incrementando um contador se sim. No final, a função retorna a divisão entre o número de etiquetas previstas corretamente e o tamanho dos arrays de entrada, efetivamente retornando a percentagem (em decimal) das etiquetas corretamente previstas.

Conclusão e considerações finais:

Este trabalho prático, embora desafiante, permitiu-nos aplicar os conhecimentos lecionados nas aulas, sendo-nos possível implementar tudo o que nos era requerido no enunciado, tendo em conta as boas práticas de programação.

Temos a apontar que o método predict no ficheiro NBayesClassUE.py foi, embora com sucesso, implementado de uma forma que torna um pouco confusa a leitura e percepção do código e da sua funcionalidade, pois toma uma abordagem que faz uso extensivo de várias estruturas de dados, bem como a própria lógica um pouco confusa do algoritmo de naive bayes, mas em suma, a implementação e a abordagem que escolhemos permitiu com que o método fosse implementado com sucesso e que o seu desempenho fosse satisfatório.

Depois de uma fase inicial de problemas com a definição das funções de cada método, conseguimos desenvolver uma abordagem satisfatória para a resolução dos problemas do enunciado.