Aprendizagem Automática - Trabalho 1

Autores:
Filipe Alfaiate - l 43315
Vasco Barnabé - l 42819

30 de Dezembro de 2020

1 Introdução/Objetivos

Foi proposto a realização de um projeto cujo objetivo consiste em implementar em Python 3 uma classe geradora de árvores de decisão com "pruning". Esta classe deve ser parcialmente compatível com as classes dos classificadores do sklearn e deve implementar uma árvore de decisão de acordo com o algoritmo ID3, seguindo-se de pruning com o método REP (Reduced Error Pruning).

Esta classe deve cumprir previamente alguns requisitos, tais como:

- ter uma parametrização para decidir a medida de pureza (opção entre gini, entropia, e erro) e indicação se terá ou não pruning, por exemplo, DecisionTreeREPrune(criterion='gini', prune=True);
- ter um método fit(x,y) que gera uma árvore de decisão em função dos dados de treino x e y, em que X é um array bidimensional com conjunto de dados para o treino com a dimensão (nrSamples, nrFeatures) e y é um array unidimensional com as classes de cada exemplo, com a dimensão (nrSamples);
- no método fit, a geração da árvore deverá ser feita com 75 por cento dos dados de treino e o pruning (Reduced Error Pruning) deverá ser feito com 25 por cento.
- ter um método score(x, y) que devolve o valor da exatidão (accuracy) com o conjunto de dados de teste x e y.

Classificadores baseados em árvores de decisão são caracterizados por utilizarem os dados de treino para construírem um modelo de classificação, que uma vez encontrado e testado, estará pronto para ser utilizado na categorização de qualquer objeto. Pode-se, também, a cada novo objeto que se quer classificar utilizar-se os dados de treino para verificar quais são os objetos na base de dados que mais se assemelham ao novo objeto que se quer classificar, e assim o objeto será classificado dentro da classe mais comum a que pertencem os objeticos mais similares.

Árvores de decisão são a base de alguns classificadores que pertencem á área da inteligência artificial, mais especificamente, á área da aprendizagem automática. Isto deve-se ao facto de lhes ser possível aprender através de exemplos para posteriormente classificar novos objetos.

Existem vários algoritmos disponíveis para a construção de uma árvore de decisão, e como já foi referido, o algoritmo utilizado nesta implementação foi o ID3, tendo este a finalidade de avaliar a informação contida nos atributos segundo a sua **entropia**. Tal como pedido, esta implementação é também funcional caso se pretenda ter **gini** ou **erro** como medida de pureza.

2 Implementação

Uma vez que o objetivo deste projeto consistiu, como já foi referido, em implementar uma classe geradora de árvores de decisão com "pruning", a implementação foi dividida em duas categorias, e por isso, em dois ficheiros distintos. Assim, um deles consiste na implementação da árvore de decisão com "pruning"(**DecisionTree**), criando-se uma classe **Node**, enquanto que no outro se dá o tratamento de dados(**main**).

Uma vez que já foi especificado o comportamento esperado deste projeto, segue-se uma descrição sucinta para cada método implementado (por cada ficheiro), de modo a justificar a criação do mesmo e a elucidar sobre a sua utilidade no programa:

DecisionTree:

- **def init (self, data)**: método com a responsabilidade de criar um nó (construtor);
- **def insert(self, datakid)**: função que guarda os filhos de um nó;
- **def inserir(self, i, datakid)**: método responsável por pegar num filho de um nó, torná-lo num arco e colocar um novo filho;
- **def inser(self, i, datakid, data)**: método que seleciona um determinado filho de um nó e altera o seu conteúdo;
- **def substituicao(self, data)**: este método tem como função reescrever o conteúdo de um nó e tornar os seus filhos vazios;
- **def leaf(self)**: função que verifica se um determinado nó é uma folha ou não;
- **def kidsLeaf(self)**: função que verifica se os filhos de um determinado nó são folhas ou não;
- **def remove(self, data)**: método criado para a remoção de um nó;
- **def printArvore(self)**: método que mostra a árvore completa;
- **def printFilhos(self)**: função auxiliar da função **printArvore**, ajudando esta na apresentação da árvore, printando os filhos de um nó;

  NOTA: estes dois últimos métodos apresentados funcionam, de uma certa forma, recursivamente entre si, uma vez que ambos fazem a invocação um do outro, criando assim um ciclo de invocações entre ambos para que se dê a apresentação completa da árvore gerada.

main:

- **def classes(ydata)**: este método tem como função retornar o número de classes existentes e uma lista com todas elas (cada classe só aparece uma vez);
- **def fancyclasses(xdata, x)**: recebendo um data, este método retorna todos os elementos não repetidos de uma coluna especificada do data em questão, utilizando o método **classes** como auxiliar;
- **def nrNnrP(ydata, array)**: função que devolve uma lista com o número de vezes que cada classe surge;

- **def probabilidade(array)**: método criado para calcular as probabilidades que serão inseridas na fórmula (que calcula a entropia, por exemplo) de modo a facilitar os cálculos, principalmente no cálculo dos logaritmos;

- **def entropy(array)**: se a medida de pureza escolhida for a **entropia**, este método é invocado e retorna o valor da entropia calculado;

- **def gini(array)**: se a medida de pureza for **gini**, é este o método invocado para efetuar os cálculos;

- **def taxa erro(array)**: método invocado caso a medida de pureza escolhida for a **taxa de erro**. Retorna assim, a taxa de erro;

- **def impurezaatributo(impurezas, ydata, count)**: método que calcula a impureza de cada atributo;

- **def Gain(xdata, ydata, arrayclasses, impureza, argimpureza)**: método responsável por percorer todo o data que lhe é fornecido e, invocando alguns dos métodos acima especificados, conseguir calcular e retornar listas com o **gain** e a **impureza** de cada atributo.

- **def auxiliarvalidationfuntion(indexi, indexk, zdata)**: método auxiliar da função **validationfunction** que seleciona uma linha da coluna fornecida como argumento; Após o método seguinte completar todos os seus ciclos, esta função retorna uma lista com validações;

- **def validationfuntion(i, j, k, zdata, array)**: esta função é invocada pelo método anterior, para que todas as linhas daquela coluna em questão sejam percorridas de modo a estabelecer comparações entre elas e a linha selecionada pelo método anterior. Esta função é uma função recursiva, saltando de linha em linha cada vez que é invocada, retornando um array apenas com elementos booleanos (True se o elemento e a classe da linha selecionada por esta função forem iguais ao elemento e classe da linha selecionada pelo método anterior, e False caso esta condição não se verifique).

- **def percorrervalidacao(array)**: método que retorna um booleano, de modo a que se todos os elementos do array fornecido como argumento forem True, o método retorna True e o programa avança para a decisão de colocar o atributo em questão como nó. Caso exista algum elemento que seja False, este método retorna imediatamente False e, consequentemente, o programa avança para a validação (ou não) do próximo atributo como nó;

- **def fit(xdata, ydata, atributos)**: método que cria um novo data compilando os argumentos xdata e ydata num só data. Este método invoca ainda o método **fitaux**, método responsável pela criação da árvore;

- **def fitaux(xdata, ydata, zdata, atributos, argimpureza, poda)**: Como já foi referido, este é o método principal para a construção da Árvore de decisão. Este método vai executar inúmeras verificações e escolhas, desde a decisão sobre qual a raíz da árvore à decisão de quais serão os seus filhos, e os filhos desses filhos, até ao fim da construção da árvore. Ao longo desta construção, este método é também responsável pela recriação do data a ser analisado, de modo a que elementos que já tenham sido analisados e utilizados para a construção até aquele momento sejam eliminados, pois já não são necessários. Por fim, este método retorna o nó raíz da árvore gerada, pois através deste, utilizando a classe **Node**, é possível percorrer e apresentar toda a árvore de decisão criada;

- **def PercorrerArvore(xtest, ytest, atributos, raiz)**: este método indica qual a linha que vai testar na matriz teste (uma vez que já recebe como argumento dados de teste). Retorna por fim uma lista com elementos do tipo booleano, sendo esses elementos True se o elemento da resposta obtida (pela árvore gerada) for igual ao elemento de teste, e False caso o acontecimento anterior não se verifique;

- **def PercorrerArvoreAuxiliar(xtest, ytest, atributos, raiz, teste)**: função invocada pelo método **PercorrerArvore** que é responsável por percorrer a árvore gerada e efetuar comparações para verificar se os nós e os ramos são correspondentes aos dados de teste;

- **def scoreaux(xtrain, ytrain, atributos, raiz)**: método responsável por fornecer parâmetros ao método **score**, tais como as respostas obtidas(True or False) e o número de vezes que cada resposta é obtida, para que o score possa ser calculado e retornado;

- **def score(respostas, calculoerro)**: método que retorna a probabilidade de acerto;

- **def pruning(xtrain, ytrain, atributos, raiz)**: método responsável por aplicar pruning à árvore gerada;

- **def pruningaux(xtrain, ytrain, atributos, raiz)**: método recursivo auxiliar na aplicação do pruning;

  NOTA: estes dois últimos métodos acima apresentados não estão completamente especificados uma vez que também não foram implementados a 100 por cento, tendo sido esta uma dificuldade encontrada durante a realização deste projeto, aplicar a poda à Árvore de Decisão gerada.

- **def DataSplit(data)**: método responsável pelo carregamento e organização de todo o data que é inicialmente necessário para que se dê início à construção da Árvore de Decisão. É neste método que é efetuada a divisão entre dados de treino e dados de teste, usando a função *train-test-split()* do sklearn.

Por fim, foi ainda criado um menú com o qual o utilizador pode interagir e efetuar escolhas de modo a que a Árvore de Decisão seja criada sobre os dados que este pretende, com a medida de pureza pretendida e ainda ser aplicado, se assim o desejar, após a criação da Árvore de Decisão, pruning.

3 Dificuldades

Durante a realização deste projeto que consistiu na criação, em Python 3, de uma Classe geradora de Árvores de Decisão seguida de "pruning", surgiram diversas dificuldades. A primeira dificuldade encontrada resume-se na linguagem sobre a qual foi desenvolvido o projeto, uma vez que esta foi a primeira vez que nós, developers, desenvolvemos um programa com esta magnitude nesta linguagem, surgindo também constantemente diversos erros de sintaxe resultantes da nossa familiarização com outras linguagens e da falta dela com Python 3.

Posteriormente, uma das maiores dificuldades encontradas e não solucionada, foi a criação do método **fit** recebendo apenas dois argumentos, um array bidimensional com o conjunto de dados para o treino e um array unidimensional com as classes de cada exemplo. Nesta implementação o método **fit** recebe três argumentos, sendo dois deles os solicitados, e um terceiro argumento que corresponde a um array com todos os atributos existentes. Este problema pode ser facilmente solucionado fazendo o carregamento do data pretendido(os atributos) dentro deste método, mas uma vez que todo o restante data foi carregado e organizado no método **DataSplit**, foi decidido manter uma atitude fiel e carregar este data junto do restante.

Além disso, a maior dificuldade identificada dentro do núcleo do programa foi verificar se a partir de um certo nó já existente era possível avançar diretamente para uma classe(esse nó ter um filho folha) ou se teria que se continuar a avaliar atributos até se chegar a uma conclusão sobre a qual classe pertenceria aquele exemplo. Após alguma pesquisa e bastante debugging, esta adversidade foi utrapassada com sucesso.

No que toca às medidas de pureza e às diferenças entre si durante o funcionamento do programa, foi identificado que por vezes, quando o cunjunto de dados a ter analisado é muito grande e a medida de pureza escolhida é a taxa de erro, o programa entra num loop infinito e não termina. Quando o conjunto de dados analisado é relativamente pequeno, não surge qualquer problema durante o funcionamento do programa, segundo os testes realizados.

Por fim, a última dificuldade encontrada foi a aplicação de "pruning"à Árvore de Decisão gerada, pois surgiram complicações aquando foi pretendido sujeitar à poda os restantes filhos de um nó, tendo sido apenas possível realizar esta ação para o primeiro filho(o filho mais á esquerda) de um nó. Após várias tentativas, esta dificuldade permaneceu e não foi possível solucionar o problema.

4 Conclusão

Fazendo uma breve retrospetiva, com o desenvolvimento deste projeto foi-nos possível aprofundar conhecimentos sobre uma linguagem de programação sobre a qual nunca tínhamos trabalhado, Python 3, bem como tudo o que diz respeito à implementação de uma **Tree** e ao tratamento de dados para a criação da mesma. Uma vez que este trabalho foi realizado no âmbito da Inteligência Artificial, mais especificamente, Machine Learning, a árvore implementada constitui características especiais, sendo assim apelidada de Árvore de Decisão(de acordo com o algoritmo ID3) seguida de pruning com o método REP.

Surgiram ao logo do desenvolvimento, tal como já foi referido, diversas adversidades, tendo sido algumas delas superadas e outras não.

Relativamente às três medidas de pureza que podem ser utilizadas(**entropia**, **gini**, e **taxa de erro**), verificou-se que em termos de eficácia, todas as medidas são bastante equivalentes, conseguindo apresentar percentagens de acerto bastante elevadas, quando a árvore gerada não é sujeita a pruning. Quando esta é sujeita a tal, verifica-se uma eficácia substancialmente menor. Contudo, esta conclusão da nossa parte acaba por ser inconclusiva, uma vez que não foi possível implementar corretamente o método pruning que transformaria a árvore e, consequentemente, resultaria numa percentagem de acerto diferente da obtida quando é pretendida uma árvore de decisão com pruning.

Por fim, há que referir que foram aplicados vários dos conhecimentos obtidos nas aulas, nomeadamente na contrução da árvore de acordo com o algoritmo ID3 apresentado e em todas as decisões relativas à medida de pureza e aos cálculos a esta associado. Foi possível, assim, criar uma classe geradora de Árvores de Decisão que, dependendo dos dados fornecidos para a construção da Árvore, da medida de pureza pretendida e dos exemplos avaliados, retorna ao utilizador uma resposta, uma determinada percentagem de acerto.