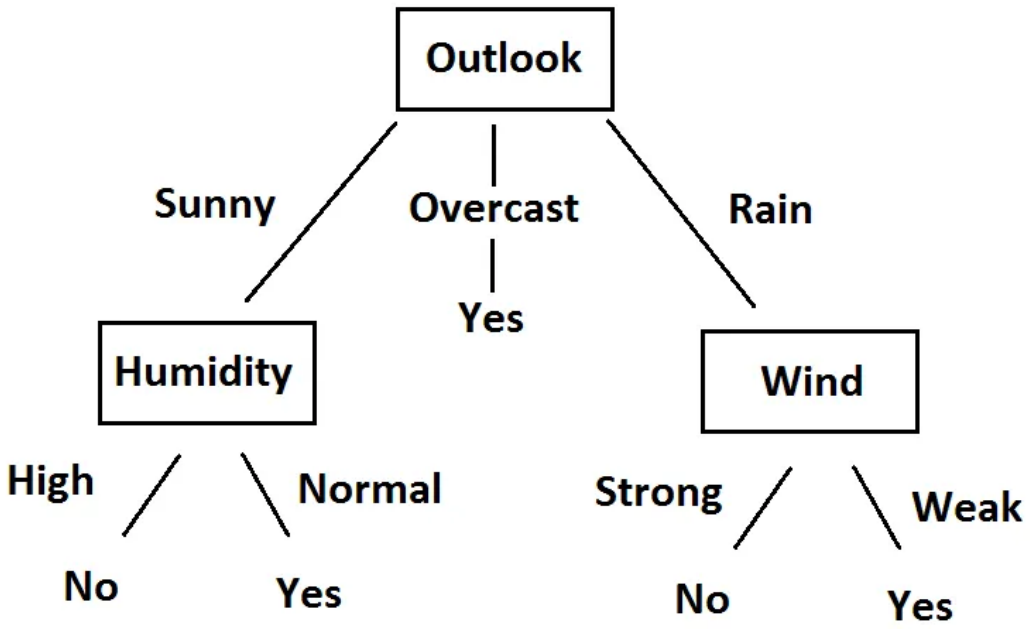
Relatório do trabalho

Aprendizagem Automática

Docentes:
Professor Luís Rato
Professora Teresa Gonçalves

Árvores de Decisão com "Pruning"



Trabalho realizado por:

Rúben Wilson nº 39837
Ludgero Teixeira nº41348

Introdução

Foi pedido a implementação de uma classe para gerar árvores de decisão com "pruning".

Sendo que esta classe terá que cumprir os seguintes requisitos:

- Ter uma parametrização para decidir a medida de pureza
(opção entre gini, entropia e erro) e indicação se terá ou não pruning, e.g. **DecisionTreeREPrune** (criterion='gini', prune=True);

- Implementação de 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 (n_samples, n_features) e y é um array unidimensional com as classes de cada exemplo, com a dimensão (n_samples);

- No método fit, a geração da árvore deverá ser feita com 75% dos dados de treino e o pruning (Reduced Error Pruning) deverá ser feito com 25%.

- Implementar um método score(x, y) que devolva o valor da exatidão (accuracy) com o conjunto de dados de teste x e y.

- Todos os atributos são nominais.

Implementação

Classes:

myDecisionTreeREPrune – Usada para guardar os dados **criterion** e **prune**

Node – Usada para construir a árvore de nós, guarda os seguintes dados:

- **Label** = nome do atributo
- **Children** = dicionário de filhos de nós
- **Leaf** = booleano para saber se é folha ou não

Entropia e Gini

Na implementação do método entropia como do gini, estes recebem o mesmo argumento "**target_col**" e retornam a entropia/ou gini do valor que pretendemos calcular.

```
def entropy(target_col):

    elementos, el_ocorrencias = np.unique(target_col,
return_counts=True)

    for i in range(len(elementos)):
        entropy = np.sum([(-el_ocorrencias[i] /
np.sum(el_ocorrencias)) * np.log2(el_ocorrencias[i] /
np.sum(el_ocorrencias))])

    return entropy
```

```
def gini(target_col):

    elementos, el_ocorrencias = np.unique(target_col,
return_counts=True)

    if len(elementos) == 1:
        gini = 0
        return gini

    gini = 1

    for i in range(len(elementos)):
        gini = gini * (el_ocorrencias[i] / np.sum(el_ocorrencias))

    return gini
```

ID3

data = os dados para os quais o algoritmo ID3 deve ser executado.
original_data = este é o conjunto de dados original
features = espaço das features do conjunto de dados.
target_attribute_name = nome do target attributes.

Este algoritmo inicialmente faz as seguintes verificações:

- Verifica se todos os valores da classe de um determinado atributo são iguais.
- Verifica se a **data** se encontra vazia.
- Verifica se existem atributos/**features.**

Caso não seja verificado nenhum destes casos, o algoritmo constrói um array com os valores de **InfoGain** para cada atributo em relação á **data** que estamos a verificar e posteriormente escolhe o melhor atributo.
Introduz esse mesmo atributo ao nó existente e de seguida faz **split** para cada valor existente do atributo escolhido como **“ best feature** ” construindo sub-árvores e chamando recursivamente o algoritmo **ID3**.
Por último adiciona esta sub-árvore ao dicionário de filhos do nó atual.

```
def ID3(data, original_data, features, target_class):

    root = Node(None)

    if len(np.unique(data[target_class])) <= 1:
        return np.unique(data[target_class])[0]

    elif len(data) == 0:
        return np.unique(original_data[target_class])[
            np.argmax(np.unique(original_data[target_class],
return_counts=True)[1])]

    elif len(features) == 0:
        root = Node(None)
        return root

    else
        item_values = [InfoGain(data, feature, target_class) for
feature in features]

        best_feature_index = np.argmax(item_values)

        best_feature = features[best_feature_index]

        root.label = best_feature

        features = [i for i in features if i != best_feature]

        for value in np.unique(data[best_feature]):

            value = value

            sub_data = data.where(data[best_feature] ==
value).dropna()

            subtree = ID3(sub_data, data, features, target_class)

            root.children[value] = subtree

        return root
```

InfoGain

Esta função recebe os 3 argumentos mencionados abaixo e tem como objetivo calcular o ganho de informação de cada atributo. Varia consoante o valor do criterion, pois se for **criterion = "*entropy*"**, calcula usando a entropia, caso contrário **criterion = "*gini*"** calcula usando o gini.

data = conjunto de dataset ao qual vamos buscar os dados.
split_attribute_name = nome do atributo para as quais vamos calcular o InfoGain.
target_name = nome da coluna da classe, decidimos usar como default o nome de "class" para a coluna target.

```
def InfoGain(data, split_attribute_name, target_name):

    total_entropy = entropy(data[target_name])

    total_gini = gini(data[target_name])

    atr, atr_ocorr = np.unique(data[split_attribute_name],
return_counts = True)

    Peso_Entropy = np.sum([(atr_ocorr[i] / np.sum(atr_ocorr)) *
entropy(data.where(data[split_attribute_name] ==
atr[i]).dropna()[target_name])

         for i in range(len(atr))])
    Peso_Gini = np.sum([(atr_ocorr[i] / np.sum(atr_ocorr)) * 2 *
gini(data.where(data[split_attribute_name] ==
atr[i]).dropna()[target_name])
         for i in range(len(atr))])

    if (classifier.criterion == "entropy"):

        Information_Gain = total_entropy - Peso_Entropy

    elif (classifier.criterion == "gini"):

        Information_Gain = total_gini - Peso_Gini

    return Information_Gain
```

Fit

Esta função recebe dois argumentos, sendo estes os dados de treino x e y, em que X é um array bidimensional com conjunto de dados para o treino com a dimensão (n_samples, n_features) e y é um array unidimensional com as classes de cada exemplo, com a dimensão (n_samples), e cria uma árvore de decisão com base no algoritmo ID3, posteriormente faz a verificação de **prune**, se é ***True*** ou ***False,*** se for ***False*** não faz nada é a árvore mantem-se, se for ***True***, devia fazer o **prune** e a árvore final devia ser a árvore podada.

```
def fit(self, x_train, y_train):

        global dtf

        dtf = pd.DataFrame(data = x_train[0:, 0:], index=[i for i in
range(x_train.shape[0])], columns=h1[0,0:])

        dtf.insert(len(dtf.columns), "class", y_train, True)

        print(dtf)

        global tree
        global target_class
        target_class = "class"

        tree = ID3(dtf, dtf, dtf.columns[:-1], target_class="class")

        print("\nPrint Tree")
        print_tree(tree, queue={})

        if (self.prune=="True"):
            print("\nDevia fazer o prune...")
```

Previs

Esta função recebe dois argumentos, sendo estes um conjunto de **queries** e a **tree** que irá servir para encontrar a melhor previsão para cada caso.
O conjunto de **queries** é um array de dicionários que guarda os casos a analisar da seguinte forma [{'coluna': 'valor' , 'coluna': 'valor' …},…], exemplo : [{'outlook': 'sunny' , temperature: cool, 'humidity': 'normal', 'windy': 'FALSE'},{…}].
Irá percorrer a árvore através dos nós até encontrar uma correspondência para a **querie** que estamos a analisar, caso tenha percorrida a arvore toda e não tenha encontrado devolve a classe mais provável.

```
def previs(query, tree):

    for key in list(query.keys()):

        if key == tree.label:
            temp = query[key]
            print(temp)
            if temp in list(tree.children.keys()):

                try:
                    result = tree.children.get(temp)

                except:
                    return maxprob(dtf, target_class)

                result = tree.children.get(temp)

                if isinstance(result, dict):
                    return previs(query, result)

                elif isinstance(result, Node):
                    return previs(query, result)

                else:
                    return result
```

Score

Esta função recebe o conjunto de dados de teste x e y, e retorna o valor da exatidão das previsões realizadas.

```
def score(self, x_test, y_test):

        data = pd.DataFrame(data=x_test[0:, 0:], index=[i for i in
range(x_test.shape[0])], columns=h1[0, 0:])

        data.insert(len(data.columns), "class", y_test, True)

        queries = data.iloc[:, :-1].to_dict(orient="records")

        previsoes = pd.DataFrame(columns=["previsoes"])

        for i in range(len(data)):
            previsoes.loc[i, "previsoes"] = previs(queries[i], tree,
1.0)

        resultado = np.sum(previsoes["previsoes"] == data["class"])
/ len(data)

        return resultado
```

Desempenho

Embora a construção do pruning não tenha sido implementada com sucesso, conseguimos obter os seguintes resultados para os casos **prune=*False*** e **criterion="*entropy*" e criterion="*gini*":**

Ficheiro	Criterion	Prune	Score (Accuracy)
weather.nominal.csv	entropy	False	100%
weather.nominal.csv	gini	False	100%
soybean.csv	entropy	False	88,65%
soybean.csv	gini	False	71,63%
vote.csv	entropy	False	93,10%
vote.csv	gini	False	93,10%
contact-lenses.csv	entropy	False	83,33%
contact-lenses.csv	gini	False	66,10%

Como podemos observar através da tabela acima, existem dois casos que apresentam valores diferentes entre os dois critérios de impureza, isto deve-se a que o algoritmo implementado para o **gini** foi pensado apenas para 2 classes e não para multiclasse, por isso ocorre esta discrepância entre os dois critérios unicamente para os casos multiclasse.

Conclusão

Com a realização deste trabalho obtivemos uma melhor noção e compreensão tanto dos conceitos como da implementação de Árvores de Decisão e do "***Prunning***" das mesmas, de algoritmos de construção como "***ID3***", métodos como a Entropia, Gini, InfoGain.
Adquirimos também conhecimento no que toca á análise de diversos conjuntos de dados e a sua compreensão.
Ao longo da realização do trabalho tivemos algumas dificuldades de desenvolvimento, mas trabalhamos sempre da melhor maneira a tentar encontrar as melhores opções a seguir para essas mesmas dificuldades.
O trabalho como mencionado anteriormente encontra-se a fazer tudo menos o ***prunning***, no entanto tentamos sempre implementar o mesmo, mas sem sucesso.
Temos a plena noção que com uma melhor preparação quer nos conteúdos teóricos, quer nos práticos teríamos conseguido alcançar melhores resultados neste trabalho.
Concluindo, apesar do trabalho não cumprir um aspeto pedido pensamos ir ao encontro do que foi pedido pelos professores.

Bibliografia

- Slides disponibilizados pelos docentes na plataforma moodle.
- https://en.wikipedia.org/wiki/Decision_tree
- https://scikit-learn.org/stable/modules/tree.html
- https://www.geeksforgeeks.org/decision-tree/
- https://sefiks.com/2017/11/20/a-step-by-step-id3-decision-tree-example/
- https://kindsonthegenius.com/blog/how-to-build-a-decision-tree-for-classification-step-by-step-procedure-using-entropy-and-gain/