Análise de algoritmos de classificação em dados de previsão do tempo

Victor Hugo Claudio Gomes

Universidade Federal do Tocantins

Ciência da Computação

Carlos Alberto de Souza Parente Rodrigues

28/06/2023

Resumo

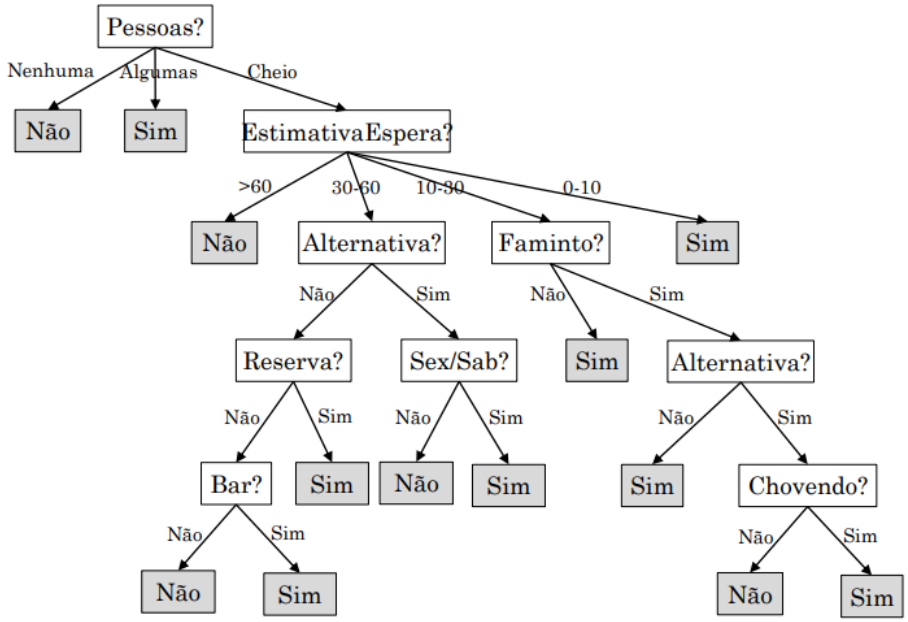
Este artigo compara três algoritmos de classificação amplamente utilizados: DecisionTreeClassifier, KNeighborsClassifier e GaussianNB. O DecisionTreeClassifier constrói um modelo de classificação com base em uma estrutura em forma de árvore, o KNeighborsClassifier classifica os pontos de dados com base nas classes de seus vizinhos mais próximos, e o GaussianNB utiliza a teoria de Bayes e assume distribuição Gaussiana nos atributos. A escolha do algoritmo deve ser baseada nas características específicas do conjunto de dados e nos requisitos do problema.

Palavras-chave: algoritmos de classificação

Metodologia

Decision Tree Classifier (Árvore de Decisão)

O DecisionTreeClassifier é um algoritmo de classificação baseado em árvore de decisão. Ele cria um modelo de classificação representado por uma estrutura em forma de árvore, na qual cada nó interno representa uma decisão baseada em um atributo e cada folha representa uma classe ou uma decisão final.



O processo de construção da árvore começa com um único nó, que contém todos os dados de treinamento. Em cada passo da construção, o algoritmo seleciona o melhor atributo para dividir os dados com base em uma métrica de qualidade. A ideia é encontrar a divisão que maximize a pureza dos subconjuntos resultantes.

Uma vez que o atributo de divisão é escolhido, a árvore é expandida adicionando nós filhos correspondentes aos possíveis valores desse atributo. O processo de seleção do melhor atributo e expansão dos nós é repetido recursivamente para cada subconjunto até que uma condição de parada seja atendida, como atingir uma profundidade máxima pré definida ou quando não há mais atributos disponíveis para dividir.

Durante a fase de treinamento, a árvore é ajustada aos dados de treinamento, capturando as relações e padrões presentes nos atributos. Após a construção da árvore, na fase de classificação, os dados de teste são percorridos descendo pela estrutura da árvore com base nos valores dos atributos. Cada nó interno é responsável por fazer uma decisão com base em um atributo, enquanto as folhas representam as classes preditas.

Uma das vantagens do DecisionTreeClassifier é sua interpretabilidade, pois a estrutura da árvore pode ser visualizada e compreendida facilmente. Além disso, ele é capaz de lidar com dados categóricos e numéricos, bem como com conjuntos de dados grandes. No entanto, é importante notar que as árvores de decisão podem ser propensas ao overfitting se não forem devidamente controladas por parâmetros, como a profundidade máxima da árvore ou o número mínimo de amostras necessárias para criar um nó.

K-Neighbors Classifier

O KNeighborsClassifier é um algoritmo de classificação baseado em vizinhos mais próximos. Ele classifica um ponto de dados com base nas classes de seus k vizinhos mais próximos. O valor de k é um parâmetro definido pelo usuário.

O funcionamento do KNeighborsClassifier é relativamente simples. Durante a fase de treinamento, o algoritmo armazena os dados de treinamento no espaço multidimensional. Em seguida, na fase de classificação, quando um novo ponto de dados precisa ser classificado, o algoritmo calcula a distância entre esse ponto e todos os outros pontos de treinamento.

As distâncias mais comumente usadas são a distância Euclidiana e a distância de Manhattan, embora outras medidas também possam ser utilizadas. Uma vez que as distâncias são calculadas, o algoritmo seleciona os k pontos de treinamento mais próximos ao ponto a ser classificado.

Em seguida, com base nas classes desses k vizinhos mais próximos, o KNeighborsClassifier atribui ao ponto a classe mais frequente entre eles. Em caso de empate, pode-se adotar uma estratégia de votação ponderada, onde os vizinhos mais próximos têm maior peso na decisão.

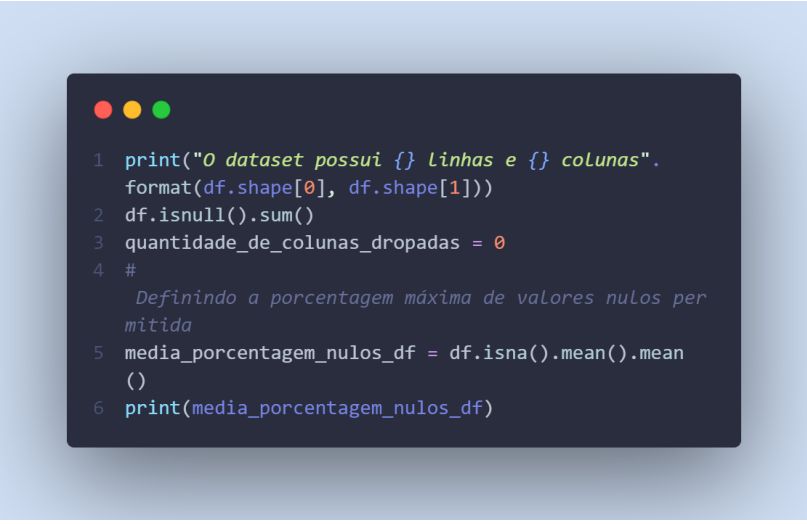
Uma das características do KNeighborsClassifier é que ele não realiza um treinamento explícito no sentido convencional de ajustar um modelo aos dados. Em vez disso, ele armazena os dados de

treinamento para serem usados durante a classificação. Isso permite que o algoritmo se adapte a conjuntos de dados em constante mudança sem a necessidade de re-treinamento.

Uma das vantagens do KNeighborsClassifier é sua simplicidade de implementação e interpretabilidade. Além disso, ele é adequado para problemas nos quais a estrutura dos dados é importante, como clusters ou regiões densamente povoadas. No entanto, o desempenho do algoritmo pode ser afetado pela escala dos atributos e pode ser computacionalmente intensivo para conjuntos de dados grandes, já que requer o cálculo das distâncias entre todos os pontos de treinamento e o novo ponto a ser classificado.

Aplicação

Após a aquisição do conjunto de dados, é conduzida uma etapa de pré-processamento para limpar as informações desnecessárias. Nesse contexto, um procedimento inicial é a avaliação da média de valores nulos presentes no conjunto de dados. Com base nessa porcentagem, é possível identificar as colunas que possuem um número de valores nulos menor ou igual a esse limiar preestabelecido e, posteriormente, remover essas colunas do conjunto de dados, uma vez que elas não serão consideradas para análise subsequente. Segue o código que realiza esse processo:



Após conhecer reconhecer essa porcentagem ele irá passar por esse algoritmo para apagar os valores e apresenta isso como retorno:

```python
for coluna in df.columns:
    # Calculando a porcentagem de valores nulos na coluna
    porcentagem_nulos = df[coluna].isnull().sum() / len(df)
    print(f"Coluna {coluna} não eliminada porcentagem de nulos: {porcentagem_nulos:.2%} de valores nulos.")
    # Verificando se a porcentagem de valores nulos na coluna é maior ou igual ao limite máximo permitido
    if porcentagem_nulos >= media_porcentagem_nulos_df:
        # Imprimindo mensagem informando que a coluna será eliminada
        print(f"Coluna {coluna} será eliminada por conter {porcentagem_nulos:.2%} de valores nulos.")
        # Removendo a coluna do DataFrame
        df.drop(columns=[coluna], inplace=True)
        # Calculando quantas colunas foram dropadas do dataframe
        quantidade_de_colunas_dropadas = quantidade_de_colunas_dropadas + 1
print("O dataset possui {} linhas e {} colunas".format(df.shape[0], df.shape[1]))
```

```
0.10259745694319072
Coluna Date não eliminada porcentagem de nulos: 0.00% de valores nulos.
Coluna Location não eliminada porcentagem de nulos: 0.00% de valores nulos.
Coluna MinTemp não eliminada porcentagem de nulos: 1.02% de valores nulos.
Coluna MaxTemp não eliminada porcentagem de nulos: 0.87% de valores nulos.
Coluna Rainfall não eliminada porcentagem de nulos: 2.24% de valores nulos.
Coluna Evaporation não eliminada porcentagem de nulos: 43.17% de valores nulos.
Coluna Evaporation será eliminada por conter 43.17% de valores nulos.
Coluna Sunshine não eliminada porcentagem de nulos: 48.01% de valores nulos.
Coluna Sunshine será eliminada por conter 48.01% de valores nulos.
Coluna WindGustDir não eliminada porcentagem de nulos: 7.10% de valores nulos.
Coluna WindGustSpeed não eliminada porcentagem de nulos: 7.06% de valores nulos.
Coluna WindDir9am não eliminada porcentagem de nulos: 7.26% de valores nulos.
Coluna WindDir3pm não eliminada porcentagem de nulos: 2.91% de valores nulos.
Coluna WindSpeed9am não eliminada porcentagem de nulos: 1.21% de valores nulos.
Coluna WindSpeed3pm não eliminada porcentagem de nulos: 2.11% de valores nulos.
Coluna Humidity9am não eliminada porcentagem de nulos: 1.82% de valores nulos.
Coluna Humidity3pm não eliminada porcentagem de nulos: 3.10% de valores nulos.
Coluna Pressure9am não eliminada porcentagem de nulos: 10.36% de valores nulos.
Coluna Pressure9am será eliminada por conter 10.36% de valores nulos.
Coluna Pressure3pm não eliminada porcentagem de nulos: 10.33% de valores nulos.
Coluna Pressure3pm será eliminada por conter 10.33% de valores nulos.
Coluna Cloud9am não eliminada porcentagem de nulos: 38.42% de valores nulos.
Coluna Cloud9am será eliminada por conter 38.42% de valores nulos.
Coluna Cloud3pm não eliminada porcentagem de nulos: 40.81% de valores nulos.
Coluna Cloud3pm será eliminada por conter 40.81% de valores nulos.
Coluna Temp9am não eliminada porcentagem de nulos: 1.21% de valores nulos.
Coluna Temp3pm não eliminada porcentagem de nulos: 2.48% de valores nulos.
Coluna RainToday não eliminada porcentagem de nulos: 2.24% de valores nulos.
Coluna RainTomorrow não eliminada porcentagem de nulos: 2.25% de valores nulos.
```

Assim que terminar essa exclusão dessas colunas ele continua com outros procedimentos, como o de exclusão ou tratamento, como retirar valores que são escritos para números, dropar outras que não fazem valor real para a aplicação, por mais que elas estejam completos.

Após isso irá realizar um processo de normalização dos resultados para ficar fácil de utilizar os algoritmos, o algoritmo MinMaxScaler para esse processo. O MinMaxScaler é uma técnica de pré-processamento que normaliza dados numéricos para um intervalo específico, como 0 e 1, por meio de transformações lineares. Ele mapeia o valor mínimo para 0 e o valor máximo para 1, comprimindo a

escala dos atributos numéricos e facilitando o treinamento de algoritmos de aprendizado de máquina. Após isso ele passa esse resultado para uma variável que separa os valores entre treino e predição e já inicia os processos de treinamento utilizando os modelos citados acima, com isso ele plota a matriz de confusão e a acurácia que o modelo utilizado resultou, segue o codigo abaixo para analise do codigo:

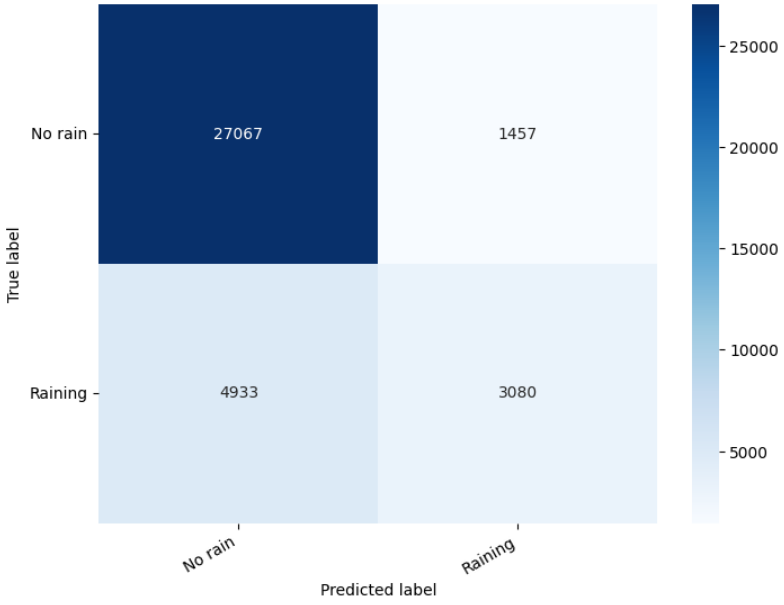
```
modelos = [DecisionTreeClassifier(), KNeighborsClassifier()]
for modelo in modelos:

    X = df[['Humidity3pm','Rainfall','RainToday']]
    y = df[['RainTomorrow']]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

    modelo.fit(X_train, y_train)

    y_pred = modelo.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)
    class_report = classification_report(y_test, y_pred, zero_division=0)
    print(f'_____ {modelo} _____')
    print("Matriz de confusão:\n", conf_matrix)
    print("Acurácia do modelo:", accuracy)
    # print("Revogação das Classes\n", class_report)
    classes = ['No rain', 'Raining']
    df_cm = pd.DataFrame(conf_matrix, index=classes, columns=classes)

    # Plota a matriz de confusão usando o seaborn
    plt.figure(figsize=(8, 6))
    hmap = sns.heatmap(df_cm, annot=True, fmt="d", cmap='Blues')

    # Configurações adicionais do gráfico
    hmap.yaxis.set_ticklabels(hmap.yaxis.get_ticklabels(), rotation=0, ha='right')
    hmap.xaxis.set_ticklabels(hmap.xaxis.get_ticklabels(), rotation=30, ha='right')
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Resultados

Segue abaixo as imagens dos resultados obtidos pela a aplicação dos dois metodos de classificação:

```
_____ DecisionTreeClassifier() _____
Matriz de confusão:
 [[27067  1457]
 [ 4933  3080]]
Acurácia do modelo: 0.8251087938254372
c:\Users\vhcla\faculdade\ml-2023-1-trabalho-final\venv\Lib\site-
  return self._fit(X, y)
_____ KNeighborsClassifier() _____
Matriz de confusão:
 [[25845  2679]
 [ 4365  3648]]
Acurácia do modelo: 0.8072091304704819
c:\Users\vhcla\faculdade\ml-2023-1-trabalho-final\venv\Lib\site-
  y = column_or_1d(y, warn=True)
                  GaussianNB()
```

Agora os plots gerados dos mapas de confusão da Árvore de Decisão:



Agora o plot do mapa de confusão do K-Neighbors: