

Seleção de atributos

Seleção de atributos, ou *feature selection*, é o processo de escolher apenas as variáveis mais relevantes para um modelo de machine learning, removendo aquelas que são irrelevantes ou redundantes. Com isso, o modelo se torna mais simples e rápido, ganha precisão ao eliminar ruídos que poderiam confundir suas previsões e reduz o risco de overfitting, pois menos variáveis irrelevantes diminuem a chance de o modelo "memorizar" ruídos específicos do conjunto de treino.

Técnicas comuns para seleção de atributos incluem métodos estatísticos, como correlação e análise de variância, além de algoritmos como árvores de decisão, que ajudam a identificar as variáveis mais importantes para as previsões.

```
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score
```

Essas linhas importam bibliotecas do scikit-learn para separar dados, selecionar atributos, usar classificadores e avaliar o modelo. Vamos entender cada importação:

1. `train_test_split`: do módulo `model_selection`, é uma função que divide os dados em conjuntos de treino e teste. Isso permite treinar o modelo em um conjunto e testá-lo em outro, ajudando a avaliar sua capacidade de generalização.
2. `SelectKBest` e `f_classif`: ambos do módulo `feature_selection`, são usados juntos para selecionar as melhores variáveis. `SelectKBest` aplica um teste estatístico (nesse caso, `f_classif`) para escolher as variáveis mais relevantes com base em um critério específico. O `f_classif` realiza um teste F para análise de variância entre classes, útil para classificação.
3. `RandomForestClassifier` e `DecisionTreeClassifier`: ambos são modelos de classificação. O `RandomForestClassifier` cria uma floresta de várias árvores de decisão e faz uma média dos resultados, sendo robusto e eficaz em dados variados. O `DecisionTreeClassifier` cria uma única árvore de decisão, mais simples e rápida, mas que pode ter desempenho inferior em comparação com a floresta.
4. `accuracy_score`, `confusion_matrix` e `f1_score`: do módulo `metrics`, são métricas de avaliação para o modelo. `accuracy_score` mede a precisão geral; `confusion_matrix` mostra os verdadeiros e falsos positivos e negativos; e `f1_score` é a média harmônica de precisão e recall, importante para dados desbalanceados.

```
import pandas as pd

# carregando o dataset
df = pd.read_csv('/content/creditcard.csv')

# informações básicas sobre o dataset
print(df.info())
print(df.describe())
```

Aqui, estamos importando o `pandas`, uma biblioteca essencial para manipulação e análise de dados, e carregando nosso dataset de transações de cartão de crédito (`creditcard.csv`) usando `pd.read_csv()`.

Primeiro, importamos pandas como pd para facilitar o uso dos métodos dessa biblioteca. Em seguida, usamos pd.read_csv() para ler o arquivo creditcard.csv e armazená-lo em um DataFrame chamado df, que é uma estrutura de dados organizada, semelhante a uma planilha.

Com df.info(), podemos ver informações básicas do dataset, como o número de linhas e colunas, tipos de dados e se há valores ausentes. Já com df.describe(), obtemos estatísticas descritivas das colunas numéricas, como média, desvio padrão, valores mínimo e máximo e os quartis, o que nos ajuda a entender melhor a distribuição dos dados e a identificar outliers ou anomalias.

Aqui, estamos separando as variáveis independentes e dependente, dividindo o dataset em conjuntos de treino e teste, e treinando um modelo com todos os atributos para avaliar seu desempenho inicial.

1. Definindo X e y: Primeiro, definimos as variáveis independentes (X) e a variável dependente (y). Usamos df.drop('Class', axis=1) para remover a coluna 'Class' do conjunto X (essa coluna indica se uma transação é uma fraude ou não, com 1 para fraude e 0 para normal). Assim, X contém todas as colunas, exceto 'Class', e y é definida como df['Class'], que será a variável que queremos prever. A opção axis=1 em drop indica que estamos removendo uma coluna (e não uma linha).

```
# definindo as variáveis independentes (X) e a variável dependente (y)
X = df.drop('Class', axis=1) # 'Class' é a coluna de fraudes (1 para fraude, 0 para normal)
y = df['Class']
# obs: quando você usa axis=1 em uma função do pandas, está dizendo à função para realizar a operação
# ao longo de todas as colunas
```

2. Dividindo os dados: Com train_test_split(X, y, test_size=0.2, random_state=42), dividimos o dataset em conjuntos de treino e teste. Usamos test_size=0.2 para reservar 20% dos dados para o teste e o restante para o treino. O random_state=42 garante que a divisão seja a mesma toda vez que o código é executado, tornando os resultados consistentes.

```
# dividindo em conjuntos de treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

3. Treinando o modelo com todos os atributos: Aqui, instanciamos um classificador de árvore de decisão (DecisionTreeClassifier), configurando random_state=1 para controlar a aleatoriedade. Com clf.fit(X_train, y_train), treinamos o modelo usando o conjunto de treino.

```
# treinando e avaliando o modelo com todos os atributos
clf = DecisionTreeClassifier(random_state=1)
clf.fit(X_train, y_train)
y_pred_all = clf.predict(X_test)
```

4. Fazendo previsões e avaliando: Em seguida, usamos clf.predict(X_test) para fazer previsões no conjunto de teste e armazenamos os resultados em y_pred_all. Para avaliar o desempenho, usamos as métricas:
 - a. Acurácia: accuracy_score(y_test, y_pred_all), que mede a taxa de previsões corretas.
 - b. Matriz de Confusão: confusion_matrix(y_test, y_pred_all), que mostra os verdadeiros e falsos positivos e negativos.
 - c. F1 Score: f1_score(y_test, y_pred_all), que é uma média harmônica entre precisão e recall, muito importante para avaliar dados desbalanceados como fraudes.

```

print("Desempenho com todos os atributos:")
print("Acurácia:", accuracy_score(y_test, y_pred_all))
print("Matriz de Confusão:")
print(confusion_matrix(y_test, y_pred_all))
print("F1 Score:", f1_score(y_test, y_pred_all))

```

Esse processo nos ajuda a entender o desempenho inicial do modelo com todas as variáveis do dataset, que depois poderá ser comparado ao desempenho com variáveis selecionadas.

```

# selecionando os 8 melhores atributos com base no teste F
selector = SelectKBest(score_func=f_classif, k=8)
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)

# exibindo as pontuações de cada atributo
scores = selector.scores_
feature_scores = list(zip(X.columns, scores))
feature_scores.sort(key=lambda x: x[1], reverse=True) # ordenando atributos por relevância

print("Pontuação dos atributos:")
for feature, score in feature_scores:
    print(f"{feature}: {score}")

```

Nessa parte, estamos aplicando uma técnica de seleção de atributos para identificar os 8 mais relevantes com base em um teste estatístico, o teste F.

1. Selecionando os 8 melhores atributos: Criamos um seletor `SelectKBest` com `score_func=f_classif` e `k=8`. O `f_classif` aplica um teste F para avaliar a correlação de cada atributo com a variável alvo, enquanto `k=8` indica que queremos manter os 8 atributos mais relevantes. Depois, usamos `fit_transform(X_train, y_train)` para treinar o seletor nos dados de treino e transformar `X_train` para incluir apenas os 8 melhores atributos. O mesmo seletor é então aplicado aos dados de teste com `transform(X_test)`, para garantir que o conjunto de teste corresponda ao de treino em termos de variáveis.
2. Exibindo as pontuações dos atributos: Obtemos as pontuações dos atributos com `selector.scores_` e associamos cada pontuação ao respectivo nome de atributo usando `zip(X.columns, scores)`. Em seguida, ordenamos essa lista de pares em ordem decrescente de pontuação (com `sort(key=lambda x: x[1], reverse=True)`), o que permite identificar quais atributos são mais relevantes para prever fraudes.
3. Exibindo os resultados: Por fim, exibimos o nome e a pontuação de cada atributo em ordem de relevância. Essa lista ajuda a visualizar quais atributos têm maior impacto na variável alvo, o que pode influenciar as escolhas de modelagem e ajudar na interpretação dos resultados do modelo.

```

# escolhendo os melhores atributos para o modelo
# exemplo: vamos manter os 8 atributos mais relevantes
selected_features = [feature for feature, score in feature_scores[:8]]
X_train_top = X_train[selected_features]
X_test_top = X_test[selected_features]

# treinando e avaliando o modelo com os atributos selecionados
clf = DecisionTreeClassifier()
clf.fit(X_train_top, y_train)
y_pred_selected = clf.predict(X_test_top)

print("\nDesempenho com atributos selecionados:")
print("Acurácia:", accuracy_score(y_test, y_pred_selected))
print("Matriz de Confusão:")
print(confusion_matrix(y_test, y_pred_selected))
print("F1 Score:", f1_score(y_test, y_pred_selected))

```

Aqui, estamos aplicando os 8 atributos mais relevantes ao modelo de árvore de decisão e avaliando seu desempenho, comparando com o modelo anterior que usou todos os atributos.

1. Escolhendo os melhores atributos: Compreendendo a lista `feature_scores`, que contém o nome e a pontuação de cada atributo, selecionamos os 8 atributos mais relevantes. Usamos uma compreensão de lista para armazenar os nomes desses atributos em `selected_features`, escolhendo apenas aqueles com as 10 maiores pontuações (`feature_scores[:8]`).
2. Criando conjuntos de treino e teste com atributos selecionados: Usamos `X_train[selected_features]` e `X_test[selected_features]` para criar novos DataFrames `X_train_top` e `X_test_top` que contêm apenas os 8 atributos escolhidos. Esses conjuntos são mais enxutos e focados nas variáveis mais impactantes.
3. Treinando e avaliando o modelo: Com `DecisionTreeClassifier(random_state=1)`, treinamos uma nova árvore de decisão usando apenas os atributos selecionados. `clf.fit(X_train_top, y_train)` ajusta o modelo nos dados de treino, e `clf.predict(X_test_top)` faz previsões no conjunto de teste, armazenadas em `y_pred_selected`.
4. Avaliação: Avaliamos o modelo filtrado usando as mesmas métricas: acurácia, matriz de confusão e F1 Score. Comparando esses resultados com o desempenho do modelo com todos os atributos, conseguimos ver se a seleção de atributos melhorou ou manteve a performance, além de reduzir a complexidade do modelo.

Conclusão

A técnica de seleção de atributos no código ajudou a simplificar o modelo, reduzindo a dimensionalidade e tornando-o mais rápido e menos propenso ao overfitting. A redução de atributos irrelevantes pode melhorar o desempenho, especialmente em métricas como o F1 Score, que é crucial para dados desbalanceados, como fraudes. A matriz de confusão também pode mostrar melhorias na identificação de fraudes. Embora a seleção de atributos tenha potencial para melhorar o desempenho, a eficácia depende da qualidade dos dados e da relevância dos atributos.

Caso o desempenho não seja ideal, outras abordagens, como algoritmos mais avançados e técnicas de balanceamento, podem ser exploradas.

Antes:

```
Desempenho com todos os atributos:  
Acurácia: 0.9261363636363636  
Matriz de Confusão:  
[[245  12]  
 [ 14  81]]  
F1 Score: 0.8617021276595744
```

Depois:

```
Desempenho com atributos selecionados:  
Acurácia: 0.9289772727272727  
Matriz de Confusão:  
[[243  14]  
 [ 11  84]]  
F1 Score: 0.8704663212435233
```

https://colab.research.google.com/drive/1QdWCfJp26U510p-9OzfXzBN-hPqEsIeI#scrollTo=khV_zcx2AMzs