Tuning de hiperparâmetros

Neste primeiro código utilizamos o método de Tuning de Hiperparâmetro, na qual é uma técnica fundamental na área de inteligência artificial (IA) que visa otimizar o desempenho de um modelo de aprendizado de máquina. Chegamos a conclusão que a combinação de seleção de atributos e tuning de hiperparâmetros foi essencial para melhorar a detecção de fraudes, apesar do desbalanceamento das classes. As técnicas aplicadas reduziram a complexidade do modelo e ajustaram seus parâmetros para encontrar um equilíbrio entre precisão e recall. O F1-Score macro e a matriz de confusão foram fundamentais para medir o impacto dessas melhorias, superando a limitação de métricas como a acurácia em cenários desbalanceados.

Nós começamos com um modelo de árvore de decisão padrão, que interpreta bem as relações entre as variáveis. A acurácia pode ser alta, mas as fraudes podem não ser detectadas corretamente, já que o modelo pode favorecer a classe majoritária (transações normais).

1. Primeiro treinamento (sem otimização):

```
# 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)
```

Aqui, a Árvore de Decisão é treinada com todos os atributos e usada para fazer previsões no conjunto de teste.

2. Treinamento com atributos selecionados:

```
# treinando e avaliando o modelo com os atributos selecionados antes do tuning
clf = DecisionTreeClassifier(random_state=1)
clf.fit(X_train_selected, y_train)
y_pred_selected = clf.predict(X_test_selected)
```

Após a seleção de atributos com SelectKBest, a Árvore de Decisão é treinada novamente, mas com um conjunto reduzido de atributos.

3. Após o tuning de hiperparâmetros:

```
# treinando o modelo com os melhores hiperparâmetros
best_clf = grid_search.best_estimator_
best_clf.fit(X_train_selected, y_train)
y_pred_tuned = best_clf.predict(X_test_selected)
```

O modelo final de Árvore de Decisão é treinado com os melhores hiperparâmetros encontrados pela técnica de GridSearchCV.

Também utilizamos a técnica de SelectKBest, que escolhe os atributos mais relevantes com base em sua correlação com a variável-alvo (fraude ou não). Isso reduz a complexidade do modelo, eliminando ruído e melhorando o foco nas variáveis mais importantes para a identificação de fraudes. Isso pode aumentar o desempenho em termos de precisão, especialmente se os atributos selecionados forem altamente correlacionados com fraudes.

```
# seleção de atributos com SelectKBest
selector = SelectKBest(f_classif, k=10)  # selecionando os 10 atributos mais relevantes
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)
```

Por último, usamos o GridSearchCV para ajustar hiperparâmetros da Árvore de Decisão, como max_depth, min_samples_split e min_samples_leaf. Isso permite encontrar a combinação de parâmetros que maximiza o desempenho do modelo, melhorando a capacidade do modelo de generalizar, evitando o overfitting e melhorando o equilíbrio entre precisão e recall, principalmente para a classe minoritária (fraudes).

```
[ ]  # tuning de hiperparâmetros com GridSearchCV
     # definindo o espaço de hiperparâmetros para a árvore de decisão
     param_grid = {
         'max_depth': [None, 10, 20, 30, 40, 50],
         'min_samples_split': [2, 5, 10, 15],
         'min_samples_leaf': [1, 2, 5, 10]
     }

     # usando GridSearchCV para encontrar os melhores hiperparâmetros
     grid_search = GridSearchCV(
         estimator=DecisionTreeClassifier(random_state=1),
         param_grid=param_grid,
         scoring='accuracy',
         cv=5,
         n_jobs=-1
     )

     # realizando a busca com os dados de treino (usando atributos selecionados)
     grid_search.fit(X_train_selected, y_train)
```

https://colab.research.google.com/drive/1b1aVyP9AVpSQNF3Am9NdtGeaaDq1_Pkp#scrollTo=cGEcBCjxdObK