Выполнил: Гун Янь

Группа: ИУ5И-24М

GradientBoostingClassifier

LogisticRegression

```
# 安装必要的库
!pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
```

```
# 导入必要的库
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_20newsgroups
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score
```

```
# 加载20 Newsgroups数据集
newsgroups = fetch_20newsgroups(subset='all', categories=['rec.sport.hockey', 'sci.space'], shuffle=True, random_state=42)
X, y = newsgroups.data, newsgroups.target
```

```
# 将数据划分为训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```


```
# 加载20 Newsgroups数据集
newsgroups = fetch_20newsgroups(subset='all', categories=['rec.sport.hockey', 'sci.space'], shuffle=True, random_state=42)
X, y = newsgroups.data, newsgroups.target
```

```
# 将数据划分为训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```


```
# 使用CountVectorizer进行特征向量化
count_vectorizer = CountVectorizer()
X_train_counts = count_vectorizer.fit_transform(X_train)
X_test_counts = count_vectorizer.transform(X_test)
```

```
# 使用TfidfVectorizer进行特征向量化
tfidf_vectorizer = TfidfVectorizer()
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

[7]
```
# 使用CountVectorizer和GradientBoostingClassifier进行训练和预测
gbc_count = GradientBoostingClassifier(random_state=42)
gbc_count.fit(X_train_counts, y_train)
y_pred_gbc_count = gbc_count.predict(X_test_counts)
print("GradientBoostingClassifier with CountVectorizer")
print(classification_report(y_test, y_pred_gbc_count))
print("Accuracy:", accuracy_score(y_test, y_pred_gbc_count))
```

```
GradientBoostingClassifier with CountVectorizer
              precision    recall  f1-score   support

           0       0.99      0.97      0.98       202
           1       0.97      0.99      0.98       196

    accuracy                           0.98       398
   macro avg       0.98      0.98      0.98       398
weighted avg       0.98      0.98      0.98       398

Accuracy: 0.9798994974874372
```

[8]
```
# 使用TfidfVectorizer和GradientBoostingClassifier进行训练和预测
gbc_tfidf = GradientBoostingClassifier(random_state=42)
gbc_tfidf.fit(X_train_tfidf, y_train)
y_pred_gbc_tfidf = gbc_tfidf.predict(X_test_tfidf)
print("\nGradientBoostingClassifier with TfidfVectorizer")
print(classification_report(y_test, y_pred_gbc_tfidf))
print("Accuracy:", accuracy_score(y_test, y_pred_gbc_tfidf))
```

```
GradientBoostingClassifier with TfidfVectorizer
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       202
           1       0.97      0.98      0.98       196

    accuracy                           0.98       398
   macro avg       0.98      0.98      0.98       398
weighted avg       0.98      0.98      0.98       398

Accuracy: 0.9798994974874372
```

```
[9]  #  使用CountVectorizer和LogisticRegression进行训练和预测
     lr_count  =  LogisticRegression(random_state=42,  max_iter=1000)
     lr_count.fit(X_train_counts,  y_train)
     y_pred_lr_count  =  lr_count.predict(X_test_counts)
     print("\nLogisticRegression  with  CountVectorizer")
     print(classification_report(y_test,  y_pred_lr_count))
     print("Accuracy:",  accuracy_score(y_test,  y_pred_lr_count))
```

```
LogisticRegression with CountVectorizer
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       202
           1       0.99      1.00      1.00       196

    accuracy                           1.00       398
   macro avg       1.00      1.00      1.00       398
weighted avg       1.00      1.00      1.00       398

Accuracy: 0.9974874371859297
```

```
              0       1.00      1.00      1.00       202
              1       0.99      1.00      1.00       196

       accuracy                           1.00       398
      macro avg       1.00      1.00      1.00       398
   weighted avg       1.00      1.00      1.00       398

   Accuracy: 0.9974874371859297
```

```
[10] #  使用TfidfVectorizer和LogisticRegression进行训练和预测
     lr_tfidf  =  LogisticRegression(random_state=42,  max_iter=1000)
     lr_tfidf.fit(X_train_tfidf,  y_train)
     y_pred_lr_tfidf  =  lr_tfidf.predict(X_test_tfidf)
     print("\nLogisticRegression  with  TfidfVectorizer")
     print(classification_report(y_test,  y_pred_lr_tfidf))
     print("Accuracy:",  accuracy_score(y_test,  y_pred_lr_tfidf))
```

```
   LogisticRegression with TfidfVectorizer
                 precision    recall  f1-score   support

              0       1.00      1.00      1.00       202
              1       0.99      1.00      1.00       196

       accuracy                           1.00       398
      macro avg       1.00      1.00      1.00       398
   weighted avg       1.00      1.00      1.00       398

   Accuracy: 0.9974874371859297
```

```
   LogisticRegression with TfidfVectorizer
                 precision    recall  f1-score   support

              0       1.00      1.00      1.00       202
              1       0.99      1.00      1.00       196

       accuracy                           1.00       398
      macro avg       1.00      1.00      1.00       398
   weighted avg       1.00      1.00      1.00       398

   Accuracy: 0.9974874371859297
```

```
[11] #  比较结果并得出结论
     results  =  {
            "GradientBoostingClassifier  with  CountVectorizer":  accuracy_score(y_test,  y_pred_gbc_count),
            "GradientBoostingClassifier  with  TfidfVectorizer":  accuracy_score(y_test,  y_pred_gbc_tfidf),
            "LogisticRegression  with  CountVectorizer":  accuracy_score(y_test,  y_pred_lr_count),
            "LogisticRegression  with  TfidfVectorizer":  accuracy_score(y_test,  y_pred_lr_tfidf)
     }
```

```
[12] best_method  =  max(results,  key=results.get)
     print("\n最佳方法:  ",  best_method,  "准确率为",  results[best_method])
```

```
   最佳方法:  LogisticRegression with CountVectorizer 准确率为 0.9974874371859297
```

Выводы:

1.Судя по результатам эксперимента, комбинация CountVectorizer и LogisticRegression показала наилучшие результаты, достигнув показателя

точности 0,9974874371859297.Это показывает, что в данной конкретной задаче классификации текста использование метода частотной векторизации слов и классификатора логической регрессии может обеспечить очень высокую эффективность классификации.

2.Дополнительные предложения

Настройка модели：

Гиперпараметры логической регрессионной модели (такие как параметр регуляризации C) могут быть дополнительно скорректированы с целью повышения производительности.

Дополнительная разработка функций：

Попробуйте другие методы выбора объектов и разработки, такие как биграммы, чтобы получить больше текстовой информации.

Перекрестная проверка：

Используйте перекрестную проверку, чтобы обеспечить стабильность и обобщение модели.

Больше данных：

Попробуйте ввести больше категорий или выборок данных, чтобы убедиться в расширяемости модели.С помощью этих дальнейших шагов вы сможете еще больше повысить производительность классификации текста для различных наборов данных и задач.