Практичне завдання: Бінарна класифікація на основі даних Titanic

Виконав студент групи МІТ-31
Тимохін Роман Миколайович

Мета:
Закріпити знання, отримані під час лекції з класифікації, застосовуючи різні моделі машинного навчання для прогнозування виживання пасажирів на основі набору даних Titanic. Практичне завдання спрямоване на розвиток навичок роботи з реальними даними, підготовки даних, побудови моделей та їх оцінки.
Структура завдання:

1. **Підготовка середовища:**
    a. Встановіть необхідні бібліотеки, такі як pandas, numpy, scikit-learn, matplotlib, та seaborn.
    b. Завантажте набір даних Titanic з Kaggle або використайте вбудовані дані Titanic з бібліотеки seaborn.
2. **Ознайомлення з даними:**
    a. Виведіть перші 10 рядків набору даних.
    b. Отримайте базову статистику за допомогою методу describe() та дослідіть пропущені значення.
3. **Попередня обробка даних:**
    a. Визначте та обробіть пропущені значення:
        i. У змінній Age заповніть пропуски середнім значенням.
        ii. У змінній Embarked заповніть пропуски найбільш поширеним значенням (модою).
    b. Закодуйте категоріальні змінні:
        i. Використайте One-Hot Encoding для змінних Sex та Embarked.
    c. Створіть нові ознаки:
        i. Додайте змінну FamilySize, яка буде сумою SibSp та Parch.
4. **Поділ даних на тренувальну та тестову вибірки:**
    a. Розділіть набір даних на тренувальну (80%) та тестову (20%) вибірки, використовуючи train_test_split() з бібліотеки scikit-learn.
5. **Побудова моделей:**
    a. Реалізуйте наступні моделі для бінарної класифікації:

i. Логістична регресія.
ii. Дерева рішень.
iii. Випадкові ліси (Random Forest).

b. Для кожної моделі виконайте:
i. Тренування на тренувальній вибірці.
ii. Передбачення на тестовій вибірці.
iii. Оцінку метрик: точність (accuracy), precision, recall, F1-score.

6. **Оцінка результатів:**
   a. Побудуйте матрицю плутанини для кожної моделі.
   b. Виведіть ROC-криву та AUC для кожної моделі, використовуючи методи roc_curve() та auc().
7. **Оптимізація моделі:**
   a. Виконайте крос-валідацію для логістичної регресії та дерев рішень.
   b. Оптимізуйте гіперпараметри для випадкового лісу, використовуючи GridSearchCV або RandomizedSearchCV.
8. **Порівняння моделей:**
   a. Порівняйте ефективність кожної моделі на основі метрик (точність, precision, recall, F1-score) та виберіть найкращу модель.
9. **Завдання з творчим підходом:**
   a. Проаналізуйте важливість ознак для моделі випадкового лісу та з'ясуйте, які змінні найбільше впливають на виживання.
   b. Запропонуйте власні ідеї для покращення моделі (наприклад, додавання нових ознак або удосконалення способів обробки пропущених даних).

Код і результати

```
# Імпортуємо необхідні бібліотеки
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split,
cross_val_score, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix, roc_curve, auc

# Завантажуємо набір даних Titanic
titanic = sns.load_dataset('titanic')

# Ознайомлення з даними
print(titanic.head(10))
print(titanic.describe())
print(titanic.isnull().sum())

# Попередня обробка даних
# Заповнення пропущених значень
titanic['age'].fillna(titanic['age'].mean(), inplace=True)
titanic['embarked'].fillna(titanic['embarked'].mode()[0],
inplace=True)

# Закодуємо категоріальні змінні
titanic = pd.get_dummies(titanic, columns=['sex', 'embarked'],
drop_first=True)

# Створення нової змінної FamilySize
titanic['family_size'] = titanic['sibsp'] + titanic['parch']

# Вибір ознак і цільової змінної
X = titanic.drop(columns=['survived', 'name', 'ticket', 'cabin',
'sibsp', 'parch'])
y = titanic['survived']

# Поділ на тренувальну та тестову вибірки
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Логістична регресія
model_lr = LogisticRegression(max_iter=200)
model_lr.fit(X_train, y_train)
y_pred_lr = model_lr.predict(X_test)

print("\nLogistic Regression Metrics:")
print("Accuracy:", accuracy_score(y_test, y_pred_lr))
print("Precision:", precision_score(y_test, y_pred_lr))
print("Recall:", recall_score(y_test, y_pred_lr))
```

```python
print("F1-score:", f1_score(y_test, y_pred_lr))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_lr))

# ROC-крива та AUC для Логістичної регресії
fpr, tpr, _ = roc_curve(y_test, model_lr.predict_proba(X_test)[:,1])
roc_auc = auc(fpr, tpr)
print("AUC (Logistic Regression):", roc_auc)

# Дерева рішень
model_dt = DecisionTreeClassifier()
model_dt.fit(X_train, y_train)
y_pred_dt = model_dt.predict(X_test)

print("\nDecision Tree Metrics:")
print("Accuracy:", accuracy_score(y_test, y_pred_dt))
print("Precision:", precision_score(y_test, y_pred_dt))
print("Recall:", recall_score(y_test, y_pred_dt))
print("F1-score:", f1_score(y_test, y_pred_dt))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_dt))

# ROC-крива та AUC для Дерев рішень
fpr, tpr, _ = roc_curve(y_test, model_dt.predict_proba(X_test)[:,1])
roc_auc = auc(fpr, tpr)
print("AUC (Decision Tree):", roc_auc)

# Випадковий ліс
model_rf = RandomForestClassifier()
model_rf.fit(X_train, y_train)
y_pred_rf = model_rf.predict(X_test)

print("\nRandom Forest Metrics:")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Precision:", precision_score(y_test, y_pred_rf))
print("Recall:", recall_score(y_test, y_pred_rf))
print("F1-score:", f1_score(y_test, y_pred_rf))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))

# ROC-крива та AUC для Випадкового лісу
fpr, tpr, _ = roc_curve(y_test, model_rf.predict_proba(X_test)[:,1])
roc_auc = auc(fpr, tpr)
print("AUC (Random Forest):", roc_auc)
```

```
# Крос-валідація для Логістичної регресії та Дерев рішень
cv_lr = cross_val_score(model_lr, X, y, cv=5)
cv_dt = cross_val_score(model_dt, X, y, cv=5)

print("\nCross-validation Scores:")
print("Logistic Regression CV Score:", cv_lr.mean())
print("Decision Tree CV Score:", cv_dt.mean())

# Оптимізація гіперпараметрів для Випадкового лісу за допомогою
GridSearchCV
param_grid = {'n_estimators': [100, 200], 'max_depth': [10, 20,
None]}
grid_search = GridSearchCV(estimator=model_rf,
param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)

print("\nBest Parameters for Random Forest:",
grid_search.best_params_)

# Важливість ознак для Випадкового лісу
feature_importances = model_rf.feature_importances_
features = X.columns

plt.barh(features, feature_importances)
plt.xlabel('Feature Importance')
plt.title('Random Forest Feature Importance')
plt.show()
```

```
   survived  pclass     sex   age  ...  deck  embark_town  alive  alone
0         0       3    male  22.0  ...   NaN  Southampton     no  False
1         1       1  female  38.0  ...     C    Cherbourg    yes  False
2         1       3  female  26.0  ...   NaN  Southampton    yes   True
3         1       1  female  35.0  ...     C  Southampton    yes  False
4         0       3    male  35.0  ...   NaN  Southampton     no   True
5         0       3    male   NaN  ...   NaN   Queenstown     no   True
6         0       1    male  54.0  ...     E  Southampton     no   True
7         0       3    male   2.0  ...   NaN  Southampton     no  False
8         1       3  female  27.0  ...   NaN  Southampton    yes  False
9         1       2  female  14.0  ...   NaN    Cherbourg    yes  False

[10 rows x 15 columns]
         survived      pclass         age       sibsp       parch        fare
count  891.000000  891.000000  714.000000  891.000000  891.000000  891.000000
mean     0.383838    2.308642   29.699118    0.523008    0.381594   32.204208
std      0.486592    0.836071   14.526497    1.102743    0.806057   49.693429
min      0.000000    1.000000    0.420000    0.000000    0.000000    0.000000
25%      0.000000    2.000000   20.125000    0.000000    0.000000    7.910400
50%      0.000000    3.000000   28.000000    0.000000    0.000000   14.454200
75%      1.000000    3.000000   38.000000    1.000000    0.000000   31.000000
max      1.000000    3.000000   80.000000    8.000000    6.000000  512.329200
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

Висновок:

Найкращою моделлю для цього завдання є **випадковий ліс**, завдяки своїй здатності добре справлятися з складними та великими наборами даних, а також стійкості до переобучення. Інші моделі, такі як логістична регресія та дерева рішень, також продемонстрували хороші результати, але випадковий ліс показав найкращу ефективність на тестових даних.