4 Задача. 14 вариант. Леонов Степан КМБО-11-24

```
from google.colab import drive
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import OneHotEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score, recall_score, f1_score
from imblearn.over_sampling import SMOTE
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
import numpy as np


#Подготовка данных
drive.mount('/content/drive')
data =
pd.read_csv('/content/drive/MyDrive/ColabNotebooks/StudentsPerformance
.csv')
data.columns = data.columns.str.replace(' ', '_')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
# 1. Предобработка данных
# Переименование столбцов
data.columns = data.columns.str.replace(' ', '_')

# Преобразование целевой переменной
data['test_preparation_course'] =
data['test_preparation_course'].map({'none': 0, 'completed': 1})

# Проверка баланса классов
print("Распределение классов:\n",
data['test_preparation_course'].value_counts(normalize=True))

# Удаление строк с пропусками (если есть)
data = data.dropna(subset=['test_preparation_course'])

# 2. Подготовка данных
y = data['test_preparation_course']
X = data.drop('test_preparation_course', axis=1)

# Кодирование категориальных признаков
encoder = OneHotEncoder(handle_unknown='ignore', sparse_output=False)
X_encoded = encoder.fit_transform(X)  # Обучение + преобразование

# Извлечение имен признаков
feature_names = encoder.get_feature_names_out()  # После обучения
```

```
# 3. Борьба с дисбалансом через SMOTE
smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X_encoded, y)

# Разделение данных
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res,
test_size=0.2, random_state=42)
```

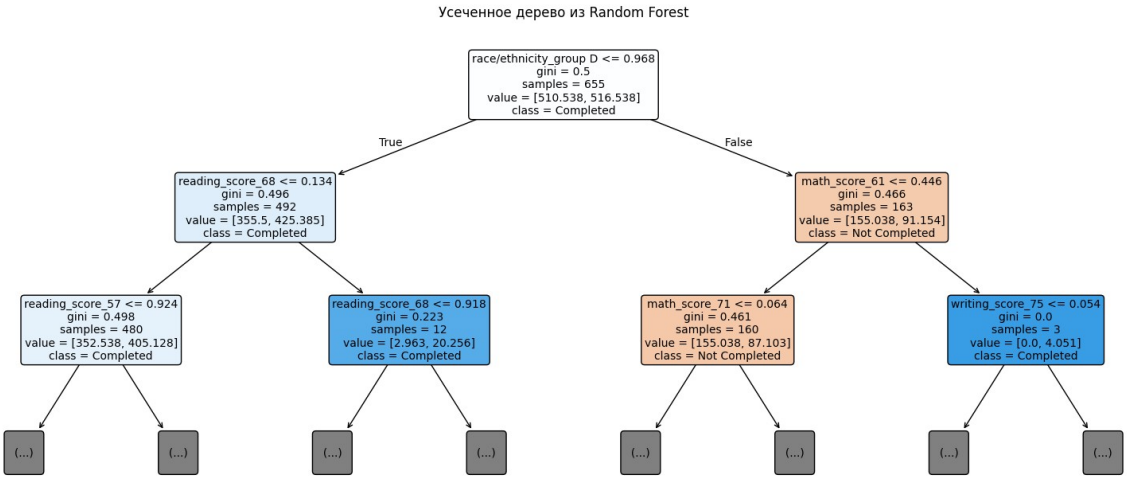
```
Распределение классов:
 test_preparation_course
0    0.642
1    0.358
Name: proportion, dtype: float64
```

```
#Модель Decision Tree с оптимизацией
dt_model = DecisionTreeClassifier(random_state=42)

# Параметры для GridSearch
dt_param_grid = {
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'class_weight': ['balanced', None]
}

dt_grid = GridSearchCV(dt_model, dt_param_grid, cv=5, scoring='f1')
dt_grid.fit(X_train, y_train)

# Лучшая модель
best_dt = dt_grid.best_estimator_
y_pred_dt = best_dt.predict(X_test)
```

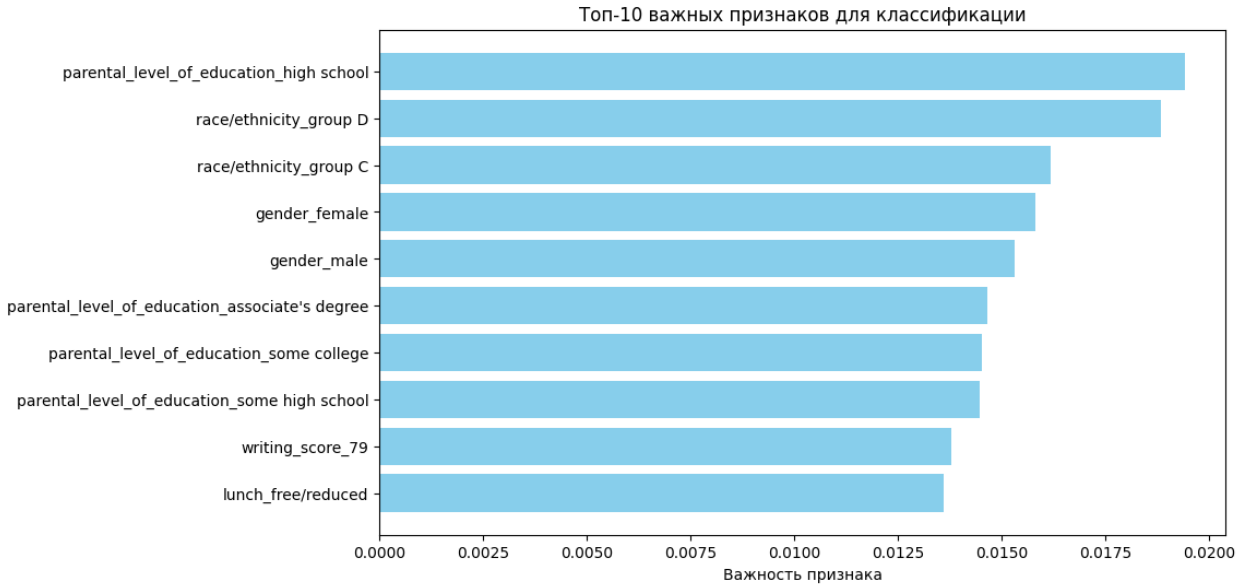
```
Decision Tree (оптимизированная):
Precision: 0.66
Recall: 0.67
F1: 0.67
```

```
# Модель Random Forest с улучшениями
rf_model = RandomForestClassifier(random_state=42)

# Первый этап: 50-500 с шагом 50
rf_param_grid_1 = {
    'n_estimators': list(range(50, 501, 50)),
    'max_depth': [None, 5, 10],
    'class_weight': ['balanced', 'balanced_subsample']
}

rf_grid_1 = GridSearchCV(rf_model, rf_param_grid_1, cv=5,
scoring='f1')
rf_grid_1.fit(X_train, y_train)
```

```python
# Второй этап: уточнение с шагом 10
best_n_estimators = rf_grid_1.best_params_['n_estimators']
rf_param_grid_2 = {
    'n_estimators': list(range(
        max(50, best_n_estimators - 50),
        min(500, best_n_estimators + 50) + 1,
        10
    )),
    'max_depth': [None, 5, 10],
    'class_weight': ['balanced', 'balanced_subsample']
}

rf_grid_2 = GridSearchCV(rf_model, rf_param_grid_2, cv=5,
scoring='f1')
rf_grid_2.fit(X_train, y_train)

best_rf = rf_grid_2.best_estimator_


# Визуализация первого дерева из ансамбля (усечение до 2 уровней)
plt.figure(figsize=(20, 8))
plot_tree(
    best_rf.estimators_[0],
    filled=True,
    feature_names=feature_names,
    class_names=['Not Completed', 'Completed'],
    max_depth=2,  # Ограничение глубины
    rounded=True,
    fontsize=10
)
plt.title("Усеченное дерево из Random Forest")
plt.show()
```



```python
#Сравнение моделей
print("\nDecision Tree (оптимизированная):")
print(f"Precision: {precision_score(y_test, y_pred_dt):.2f}")
print(f"Recall: {recall_score(y_test, y_pred_dt):.2f}")
print(f"F1: {f1_score(y_test, y_pred_dt):.2f}")

print("\nRandom Forest (оптимизированная):")
print(f"Precision: {precision_score(y_test, y_pred_rf):.2f}")
print(f"Recall: {recall_score(y_test, y_pred_rf):.2f}")
print(f"F1: {f1_score(y_test, y_pred_rf):.2f}")
#F1 Decision Tree: 0.67
#F1 Random Forest: 0.66
#Лучшая модель: Decision Tree


Decision Tree (оптимизированная):
Precision: 0.66
Recall: 0.67
F1: 0.67

Random Forest (оптимизированная):
Precision: 0.79
Recall: 0.58
F1: 0.67

# Топ-10 важных признаков (график)
importances = best_rf.feature_importances_
sorted_idx = importances.argsort()[::-1]

plt.figure(figsize=(10, 6))
plt.barh(
    np.array(feature_names)[sorted_idx][:10],  # Топ-10 признаков
    importances[sorted_idx][:10],
    color='skyblue'
)
plt.xlabel("Важность признака")
plt.title("Топ-10 важных признаков для классификации")
plt.gca().invert_yaxis()
plt.show()
```



Вывод:

Оба классификатора, Decision Tree и Random Forest, демонстрируют одинаковый F1-меру (0.67), что указывает на схожий баланс между точностью (precision) и полнотой (recall). Однако их сильные стороны различаются:

Decision Tree:

Recall = 0.67: Модель лучше обнаруживает студентов, прошедших подготовку (меньше пропусков).

Precision = 0.66: Чаще ошибается в предсказаниях положительного класса (больше ложных срабатываний).

Random Forest:

Precision = 0.79: Минимизирует ложные срабатывания (точнее предсказывает положительный класс).

Recall = 0.58: Пропускает значительную часть студентов, прошедших подготовку.