ЛАБОРАТОРНАЯ РАБОТА 1

```
# -*- coding: utf-8 -*-
"""lab1.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1IMuTD5ilqEAKmXvaFYdT6yfmWL2wbn0B

# Лабораторная работа №1
"""

from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import (train_test_split, GridSearchCV,
                                     RandomizedSearchCV,
StratifiedKFold)
from sklearn.metrics import (classification_report, roc_auc_score,
                             roc_curve, auc, precision_recall_curve,
                             average_precision_score)
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler, label_binarize
from sklearn.ensemble import RandomForestClassifier

from imblearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE, RandomOverSampler
from imblearn.under_sampling import TomekLinks, ClusterCentroids

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random

random.seed(42)
np.random.seed(42)

"""# Wine Quality dataset

В данной лабораторной работе мы взяли датасет, оценивающий качество красных вин. Данные, представленные по ссылке ниже, содержат два варианта: один для красных вин, второй для белых. Собственно, единственное отличие лишь в сортах винограда для изготовления

Подробнее по ссылке - https://archive.ics.uci.edu/dataset/186/wine+quality
"""
```

```
URL = 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv'

df = pd.read_csv(URL, delimiter=';')
df.head()

df.info()

X = df.drop(columns=['quality']).values
y = df['quality'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

print(f'Объем обучающей выборки X_train: {X_train.shape} | y_train: {y_train.shape}')
print(f'Объем тестовой выборки X_test: {X_test.shape} | y_test: {y_test.shape}')
print(f'Соотношение данных равно {y_train.shape[0] / y.shape[0]:.4f} / {y_test.shape[0] / y.shape[0]:.4f}')

"""# Построение моделей"""

cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
scoring = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro', 'roc_auc_ovr']

classes = np.unique(y_train)
y_test_bin = label_binarize(y_test, classes=classes)

"""## Дерево решений"""

tree_grid = {
    'ccp_alpha': [0.005, 0.01, 0.015, 0.02, 0.025, 0.03, 0.035, 0.2, 0.8],
    'criterion': ['entropy', 'gini']
}

tree = GridSearchCV(
    estimator=DecisionTreeClassifier(random_state=42),
    param_grid=tree_grid,
    scoring=scoring,
    refit='roc_auc_ovr',
    cv=cv,
    n_jobs=-1,
    verbose=1
)
```

```
tree.fit(X_train, y_train)

print(f"Лучшие параметры: {tree.best_params_}")
print(f"Лучший ROC-AUC (macro, OVR) на обучающих данных:
{tree.best_score_:.4f}")

metrics_df = pd.DataFrame(tree.cv_results_)[[
    'mean_test_accuracy',
    'mean_test_precision_macro',
    'mean_test_recall_macro',
    'mean_test_f1_macro',
    'mean_test_roc_auc_ovr'
]]

metrics_df = metrics_df.sort_values(by='mean_test_roc_auc_ovr',
ascending=False).iloc[0, :]
metrics_df.name = 'Metric'

display(metrics_df)

"""Выше были метрики по обучающим данным. Теперь пойдут метрики на
тестовых данных"""

clf = tree.best_estimator_
y_pred = clf.predict(X_test)
y_proba = clf.predict_proba(X_test)

print("\nClassification report:")
print(classification_report(y_test, y_pred, zero_division=np.nan))

roc_auc = roc_auc_score(y_test_bin, y_proba, multi_class='ovr',
average='macro')
print(f"ROC-AUC (macro, OVR): {roc_auc:.4f}")

plt.figure(figsize=(10, 8))

for i, class_label in enumerate(classes):
    fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_proba[:, i])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'Класс {class_label} (AUC =
{roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--', label='Случайный выбор (AUC = 0.5)')
plt.plot([0, 0, 1], [0, 1, 1], 'k:', label='Идеальная производительность
(AUC = 1.0)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-кривые (One-vs-Rest)')
plt.legend(loc='lower right')
plt.tight_layout()
plt.show()
```

```
plt.figure(figsize=(12, 8))

for i, class_label in enumerate(classes):
    precision, recall, _ = precision_recall_curve(y_test_bin[:, i],
y_proba[:, i])
    ap = average_precision_score(y_test_bin[:, i], y_proba[:, i])
    plt.plot(recall, precision, label=f'Класс {class_label} (AUC =
{ap:.2f})')

plt.plot([0, 1], [0.5, 0.5], 'k--', label='Случайный выбор (AUC = 0.5)')
#plt.plot([0, 1], [1, 1], 'k:', label='Идеальная производительность (AUC
= 1.0)')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall кривые (One-vs-Rest)')
plt.legend(loc='best')#, bbox_to_anchor=(1, 1))
plt.tight_layout()
plt.show()

features = df.columns[:-1]

plt.figure(figsize=(15, 8))
ax = plt.gca()
annot = plot_tree(clf, feature_names=features, max_depth=2, fontsize=7,
filled=True, ax=ax)
plt.show()

"""## Машина опорных векторов"""

svm_grid = {
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'gamma': ['scale', 'auto', 1, 10, 0.1, 0.01],
    'coef0': [0, 1, 2, 5, 10, 0.1, 0.01],
    'degree': [2, 3, 4, 5],
    'C': [0.01, 0.1, 0.5, 1, 2, 10, 20]
}

svm = RandomizedSearchCV(
    estimator=SVC(probability=True, random_state=42),
    param_distributions=svm_grid,
    scoring=scoring,
    refit='roc_auc_ovr',
    cv=cv,
    n_jobs=-1,
    random_state=42,
    verbose=1
)
svm.fit(X_train, y_train)
```

```
print(f"Лучшие параметры: {svm.best_params_}")
print(f"Лучший ROC-AUC (macro, OVR) на обучающих данных:
{svm.best_score_:.4f}")

metrics_df = pd.DataFrame(svm.cv_results_)[[
    'mean_test_accuracy',
    'mean_test_precision_macro',
    'mean_test_recall_macro',
    'mean_test_f1_macro',
    'mean_test_roc_auc_ovr'
]]

metrics_df = metrics_df.sort_values(by='mean_test_roc_auc_ovr',
ascending=False).iloc[0, :]
metrics_df.name = 'Metric'

display(metrics_df)

clf = svm.best_estimator_
y_pred = clf.predict(X_test)
y_proba = clf.predict_proba(X_test)

print("\nClassification report:")
print(classification_report(y_test, y_pred, zero_division=np.nan))

roc_auc = roc_auc_score(y_test_bin, y_proba, multi_class='ovr',
average='macro')
print(f"ROC-AUC (macro, OVR): {roc_auc:.4f}")

plt.figure(figsize=(10, 8))

for i, class_label in enumerate(classes):
    fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_proba[:, i])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'Класс {class_label} (AUC =
{roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--', label='Случайный выбор (AUC = 0.5)')
plt.plot([0, 0, 1], [0, 1, 1], 'k:', label='Идеальная производительность
(AUC = 1.0)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-кривые (One-vs-Rest)')
plt.legend(loc='lower right')
plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 8))

for i, class_label in enumerate(classes):
```

```
    precision, recall, _ = precision_recall_curve(y_test_bin[:, i],
y_proba[:, i])
    ap = average_precision_score(y_test_bin[:, i], y_proba[:, i])
    plt.plot(recall, precision, label=f'Класс {class_label} (AUC =
{ap:.2f})')

plt.plot([0, 1], [0.5, 0.5], 'k--', label='Случайный выбор (AUC = 0.5)')
#plt.plot([0, 1], [1, 1], 'k:', label='Идеальная производительность (AUC
= 1.0)')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall кривые (One-vs-Rest)')
plt.legend(loc='best')#, bbox_to_anchor=(1, 1))
plt.tight_layout()
plt.show()

"""## Случайный лес"""

# Commented out IPython magic to ensure Python compatibility.
# %%time
#
# rf_grid = {
#     'criterion': ['gini', 'entropy'],
#     'n_estimators': [200, 400, 600],
#     'max_features': [0.3, 0.5, 0.7, 1.],
#     'max_depth': [1, 3, 5]
# }
#
# rf = GridSearchCV(
#     estimator=RandomForestClassifier(random_state=42, n_jobs=-1),
#     param_grid=rf_grid,
#     scoring=scoring,
#     refit='roc_auc_ovr',
#     cv=cv,
#     n_jobs=-1,
#     verbose=1
# )
# rf.fit(X_train, y_train)
#
# print(f"Лучшие параметры: {rf.best_params_}")
# print(f"Лучший ROC AUC (macro, OVR) на обучающих данных:
{rf.best_score_:.4f}")

metrics_df = pd.DataFrame(rf.cv_results_)[[
    'mean_test_accuracy',
    'mean_test_precision_macro',
    'mean_test_recall_macro',
    'mean_test_f1_macro',
    'mean_test_roc_auc_ovr'
]]
```

```
metrics_df = metrics_df.sort_values(by='mean_test_roc_auc_ovr',
ascending=False).iloc[0, :]
metrics_df.name = 'Metric'

display(metrics_df)

clf = rf.best_estimator_
y_pred = clf.predict(X_test)
y_proba = clf.predict_proba(X_test)

print("\nClassification report:")
print(classification_report(y_test, y_pred, zero_division=np.nan))

roc_auc = roc_auc_score(y_test_bin, y_proba, multi_class='ovr',
average='macro')
print(f"ROC-AUC (macro, OVR): {roc_auc:.4f}")

plt.figure(figsize=(10, 8))

for i, class_label in enumerate(classes):
    fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_proba[:, i])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'Класс {class_label} (AUC =
{roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--', label='Случайный выбор (AUC = 0.5)')
plt.plot([0, 0, 1], [0, 1, 1], 'k:', label='Идеальная производительность
(AUC = 1.0)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-кривые (One-vs-Rest)')
plt.legend(loc='lower right')
plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 8))

for i, class_label in enumerate(classes):
    precision, recall, _ = precision_recall_curve(y_test_bin[:, i],
y_proba[:, i])
    ap = average_precision_score(y_test_bin[:, i], y_proba[:, i])
    plt.plot(recall, precision, label=f'Класс {class_label} (AUC =
{ap:.2f})')

plt.plot([0, 1], [0.5, 0.5], 'k--', label='Случайный выбор (AUC = 0.5)')
#plt.plot([0, 1], [1, 1], 'k:', label='Идеальная производительность (AUC
= 1.0)')

plt.xlabel('Recall')
plt.ylabel('Precision')
```

```
plt.title('Precision-Recall кривые (One-vs-Rest)')
plt.legend(loc='best')#, bbox_to_anchor=(1, 1))
plt.tight_layout()
plt.show()

"""# Обогащение выборки

## OverSampling

### Random Over Sampler
"""

# Commented out IPython magic to ensure Python compatibility.
# %%time
#
# tree_pipeline = Pipeline([
#     ("ros", RandomOverSampler(random_state=42)),
#     ("tree", DecisionTreeClassifier(random_state=42))
# ])
#
# tree_pipe_grid = {
#     'tree__ccp_alpha': [0.005, 0.01, 0.015, 0.02, 0.025, 0.03, 0.035,
0.2, 0.8],
#     'tree__criterion': ['entropy', 'gini']
# }
#
# tree_ros = GridSearchCV(
#     estimator=tree_pipeline,
#     param_grid=tree_pipe_grid,
#     scoring=scoring,
#     refit='roc_auc_ovr',
#     cv=cv,
#     n_jobs=-1,
#     verbose=1
# )
# tree_ros.fit(X_train, y_train)
#
# print(f"Лучшие параметры: {tree_ros.best_params_}")
# print(f"Лучший ROC-AUC (macro, OVR) на обучающих данных:
{tree_ros.best_score_:.4f}")

metrics_df = pd.DataFrame(tree_ros.cv_results_)[[
    'mean_test_accuracy',
    'mean_test_precision_macro',
    'mean_test_recall_macro',
    'mean_test_f1_macro',
    'mean_test_roc_auc_ovr'
]]

metrics_df = metrics_df.sort_values(by='mean_test_roc_auc_ovr',
ascending=False).iloc[0, :]
```

```
metrics_df.name = 'Metric'

display(metrics_df)

# Commented out IPython magic to ensure Python compatibility.
# %%time
#
# svm_pipeline = Pipeline([
#     ("ros", RandomOverSampler(random_state=42)),
#     ("svm", SVC(probability=True, random_state=42))
# ])
#
# svm_pipe_grid = {
#     'svm__kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
#     'svm__gamma': ['scale', 'auto', 1, 10, 0.1, 0.01],
#     'svm__coef0': [0, 1, 2, 5, 10, 0.1, 0.01],
#     'svm__degree': [2, 3, 4, 5],
#     'svm__C': [0.01, 0.1, 0.5, 1, 2, 10, 20]
# }
#
# svm_ros = RandomizedSearchCV(
#     estimator=svm_pipeline,
#     param_distributions=svm_pipe_grid,
#     scoring=scoring,
#     refit='roc_auc_ovr',
#     cv=cv,
#     n_jobs=-1,
#     verbose=1
# )
# svm_ros.fit(X_train, y_train)
#
# print(f"Лучшие параметры: {svm_ros.best_params_}")
# print(f"Лучший ROC AUC (macro, OVR) на обучающих данных:
{svm_ros.best_score_:.4f}")

metrics_df = pd.DataFrame(svm_ros.cv_results_)[[
    'mean_test_accuracy',
    'mean_test_precision_macro',
    'mean_test_recall_macro',
    'mean_test_f1_macro',
    'mean_test_roc_auc_ovr'
]]

metrics_df = metrics_df.sort_values(by='mean_test_roc_auc_ovr',
ascending=False).iloc[0, :]
metrics_df.name = 'Metric'

display(metrics_df)

# Commented out IPython magic to ensure Python compatibility.
# %%time
```

```
#
# rf_pipeline = Pipeline([
#     ("ros", RandomOverSampler(random_state=42)),
#     ("rf", RandomForestClassifier(random_state=42, n_jobs=-1))
# ])
#
# rf_pipe_grid = {
#     'rf__criterion': ['gini', 'entropy'],
#     'rf__n_estimators': [200, 400, 600],
#     'rf__max_features': [0.3, 0.5, 0.7, 1.],
#     'rf__max_depth': [1, 3, 5]
# }
#
# rf_ros = GridSearchCV(
#     estimator=rf_pipeline,
#     param_grid=rf_pipe_grid,
#     scoring=scoring,
#     refit='roc_auc_ovr',
#     cv=cv,
#     n_jobs=-1,
#     verbose=1
# )
# rf_ros.fit(X_train, y_train)
#
# print(f"Лучшие параметры: {rf_ros.best_params_}")
# print(f"Лучший ROC AUC (macro, OVR) на обучающих данных:
{rf_ros.best_score_:.4f}")

metrics_df = pd.DataFrame(rf_ros.cv_results_)[[
    'mean_test_accuracy',
    'mean_test_precision_macro',
    'mean_test_recall_macro',
    'mean_test_f1_macro',
    'mean_test_roc_auc_ovr'
]]

metrics_df = metrics_df.sort_values(by='mean_test_roc_auc_ovr',
ascending=False).iloc[0, :]
metrics_df.name = 'Metric'

display(metrics_df)

"""### SMOTE"""

# Commented out IPython magic to ensure Python compatibility.
# %%time
#
# tree_pipeline = Pipeline([
#     ("smote", SMOTE(random_state=42)),
#     ("tree", DecisionTreeClassifier(random_state=42))
# ])
```

```
#
# tree_pipe_grid = {
#     'tree__ccp_alpha': [0.005, 0.01, 0.015, 0.02, 0.025, 0.03, 0.035,
0.2, 0.8],
#     'tree__criterion': ['entropy', 'gini']
# }
#
# tree_smote = GridSearchCV(
#     estimator=tree_pipeline,
#     param_grid=tree_pipe_grid,
#     scoring=scoring,
#     refit='roc_auc_ovr',
#     cv=cv,
#     n_jobs=-1,
#     verbose=1
# )
# tree_smote.fit(X_train, y_train)
#
# print(f"Лучшие параметры: {tree_smote.best_params_}")
# print(f"Лучший ROC-AUC (macro, OVR) на обучающих данных:
{tree_smote.best_score_:.4f}")

metrics_df = pd.DataFrame(tree_smote.cv_results_)[[
    'mean_test_accuracy',
    'mean_test_precision_macro',
    'mean_test_recall_macro',
    'mean_test_f1_macro',
    'mean_test_roc_auc_ovr'
]]

metrics_df = metrics_df.sort_values(by='mean_test_roc_auc_ovr',
ascending=False).iloc[0, :]
metrics_df.name = 'Metric'

display(metrics_df)

# Commented out IPython magic to ensure Python compatibility.
# %%time
#
# svm_pipeline = Pipeline([
#     ("smote", SMOTE(random_state=42)),
#     ("svm", SVC(probability=True, random_state=42))
# ])
#
# svm_pipe_grid = {
#     'svm__kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
#     'svm__gamma': ['scale', 'auto', 1, 10, 0.1, 0.01],
#     'svm__coef0': [0, 1, 2, 5, 10, 0.1, 0.01],
#     'svm__degree': [2, 3, 4, 5],
#     'svm__C': [0.01, 0.1, 0.5, 1, 2, 10, 20]
# }
```

```
#
# svm_smote = RandomizedSearchCV(
#     estimator=svm_pipeline,
#     param_distributions=svm_pipe_grid,
#     scoring=scoring,
#     refit='roc_auc_ovr',
#     cv=cv,
#     n_jobs=-1,
#     verbose=1
# )
# svm_smote.fit(X_train, y_train)
#
# print(f"Лучшие параметры: {svm_smote.best_params_}")
# print(f"Лучший ROC AUC (macro, OVR) на обучающих данных:
{svm_smote.best_score_:.4f}")

metrics_df = pd.DataFrame(svm_smote.cv_results_)[[
    'mean_test_accuracy',
    'mean_test_precision_macro',
    'mean_test_recall_macro',
    'mean_test_f1_macro',
    'mean_test_roc_auc_ovr'
]]

metrics_df = metrics_df.sort_values(by='mean_test_roc_auc_ovr',
ascending=False).iloc[0, :]
metrics_df.name = 'Metric'

display(metrics_df)

# Commented out IPython magic to ensure Python compatibility.
# %%time
#
# rf_pipeline = Pipeline([
#     ("smote", SMOTE(random_state=42)),
#     ("rf", RandomForestClassifier(random_state=42, n_jobs=-1))
# ])
#
# rf_pipe_grid = {
#     'rf__criterion': ['gini', 'entropy'],
#     'rf__n_estimators': [200, 400, 600],
#     'rf__max_features': [0.3, 0.5, 0.7, 1.],
#     'rf__max_depth': [1, 3, 5]
# }
#
# rf_smote = GridSearchCV(
#     estimator=rf_pipeline,
#     param_grid=rf_pipe_grid,
#     scoring=scoring,
#     refit='roc_auc_ovr',
#     cv=cv,
```

```
#     n_jobs=-1,
#     verbose=1
# )
# rf_smote.fit(X_train, y_train)
#
# print(f"Лучшие параметры: {rf_smote.best_params_}")
# print(f"Лучший ROC AUC (macro, OVR) на обучающих данных:
{rf_smote.best_score_:.4f}")

metrics_df = pd.DataFrame(rf_smote.cv_results_)[[
    'mean_test_accuracy',
    'mean_test_precision_macro',
    'mean_test_recall_macro',
    'mean_test_f1_macro',
    'mean_test_roc_auc_ovr'
]]

metrics_df = metrics_df.sort_values(by='mean_test_roc_auc_ovr',
ascending=False).iloc[0, :]
metrics_df.name = 'Metric'

display(metrics_df)

"""## UnderSampling

### Tomek Links
"""

# Commented out IPython magic to ensure Python compatibility.
# %%time
#
# tree_pipeline = Pipeline([
#     ("tomek", TomekLinks()),
#     ("tree", DecisionTreeClassifier(random_state=42))
# ])
#
# tree_pipe_grid = {
#     'tree__ccp_alpha': [0.005, 0.01, 0.015, 0.02, 0.025, 0.03, 0.035,
0.2, 0.8],
#     'tree__criterion': ['entropy', 'gini']
# }
#
# tree_tomek = GridSearchCV(
#     estimator=tree_pipeline,
#     param_grid=tree_pipe_grid,
#     scoring=scoring,
#     refit='roc_auc_ovr',
#     cv=cv,
#     n_jobs=-1,
#     verbose=1
# )
```

```
# tree_tomek.fit(X_train, y_train)
#
# print(f"Лучшие параметры: {tree_tomek.best_params_}")
# print(f"Лучший ROC-AUC (macro, OVR) на обучающих данных:
{tree_tomek.best_score_:.4f}")

metrics_df = pd.DataFrame(tree_tomek.cv_results_)[[
    'mean_test_accuracy',
    'mean_test_precision_macro',
    'mean_test_recall_macro',
    'mean_test_f1_macro',
    'mean_test_roc_auc_ovr'
]]

metrics_df = metrics_df.sort_values(by='mean_test_roc_auc_ovr',
ascending=False).iloc[0, :]
metrics_df.name = 'Metric'

display(metrics_df)

# Commented out IPython magic to ensure Python compatibility.
# %%time
#
# svm_pipeline = Pipeline([
#     ("tomek", TomekLinks()),
#     ("svm", SVC(probability=True, random_state=42))
# ])
#
# svm_pipe_grid = {
#     'svm__kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
#     'svm__gamma': ['scale', 'auto', 1, 10, 0.1, 0.01],
#     'svm__coef0': [0, 1, 2, 5, 10, 0.1, 0.01],
#     'svm__degree': [2, 3, 4, 5],
#     'svm__C': [0.01, 0.1, 0.5, 1, 2, 10, 20]
# }
#
# svm_tomek = RandomizedSearchCV(
#     estimator=svm_pipeline,
#     param_distributions=svm_pipe_grid,
#     scoring=scoring,
#     refit='roc_auc_ovr',
#     cv=cv,
#     n_jobs=-1,
#     verbose=1
# )
# svm_tomek.fit(X_train, y_train)
#
# print(f"Лучшие параметры: {svm_tomek.best_params_}")
# print(f"Лучший ROC AUC (macro, OVR) на обучающих данных:
{svm_tomek.best_score_:.4f}")
```

```
metrics_df = pd.DataFrame(svm_tomek.cv_results_)[[
    'mean_test_accuracy',
    'mean_test_precision_macro',
    'mean_test_recall_macro',
    'mean_test_f1_macro',
    'mean_test_roc_auc_ovr'
]]

metrics_df = metrics_df.sort_values(by='mean_test_roc_auc_ovr',
ascending=False).iloc[0, :]
metrics_df.name = 'Metric'

display(metrics_df)

# Commented out IPython magic to ensure Python compatibility.
# %%time
#
# rf_pipeline = Pipeline([
#     ("tomek", TomekLinks()),
#     ("rf", RandomForestClassifier(random_state=42, n_jobs=-1))
# ])
#
# rf_pipe_grid = {
#     'rf__criterion': ['gini', 'entropy'],
#     'rf__n_estimators': [200, 400, 600],
#     'rf__max_features': [0.3, 0.5, 0.7, 1.],
#     'rf__max_depth': [1, 3, 5]
# }
#
# rf_tomek = GridSearchCV(
#     estimator=rf_pipeline,
#     param_grid=rf_pipe_grid,
#     scoring=scoring,
#     refit='roc_auc_ovr',
#     cv=cv,
#     n_jobs=-1,
#     verbose=1
# )
# rf_tomek.fit(X_train, y_train)
#
# print(f"Лучшие параметры: {rf_tomek.best_params_}")
# print(f"Лучший ROC AUC (macro, OVR) на обучающих данных:
{rf_tomek.best_score_:.4f}")

metrics_df = pd.DataFrame(rf_tomek.cv_results_)[[
    'mean_test_accuracy',
    'mean_test_precision_macro',
    'mean_test_recall_macro',
    'mean_test_f1_macro',
    'mean_test_roc_auc_ovr'
]]
```

```
metrics_df = metrics_df.sort_values(by='mean_test_roc_auc_ovr',
ascending=False).iloc[0, :]
metrics_df.name = 'Metric'

display(metrics_df)

"""### Cluster Centroids"""

# Commented out IPython magic to ensure Python compatibility.
# %%time
#
# tree_pipeline = Pipeline([
#     ("cluster", ClusterCentroids(random_state=42)),
#     ("tree", DecisionTreeClassifier(random_state=42))
# ])
#
# tree_pipe_grid = {
#     'tree__ccp_alpha': [0.005, 0.01, 0.015, 0.02, 0.025, 0.03, 0.035,
0.2, 0.8],
#     'tree__criterion': ['entropy', 'gini']
# }
#
# tree_cluster = GridSearchCV(
#     estimator=tree_pipeline,
#     param_grid=tree_pipe_grid,
#     scoring=scoring,
#     refit='roc_auc_ovr',
#     cv=cv,
#     n_jobs=-1,
#     verbose=1
# )
# tree_cluster.fit(X_train, y_train)
#
# print(f"Лучшие параметры: {tree_cluster.best_params_}")
# print(f"Лучший ROC-AUC (macro, OVR) на обучающих данных:
{tree_cluster.best_score_:.4f}")

metrics_df = pd.DataFrame(tree_cluster.cv_results_)[[
    'mean_test_accuracy',
    'mean_test_precision_macro',
    'mean_test_recall_macro',
    'mean_test_f1_macro',
    'mean_test_roc_auc_ovr'
]]

metrics_df = metrics_df.sort_values(by='mean_test_roc_auc_ovr',
ascending=False).iloc[0, :]
metrics_df.name = 'Metric'

display(metrics_df)
```

```
# Commented out IPython magic to ensure Python compatibility.
# %%time
#
# svm_pipeline = Pipeline([
#     ("cluster", ClusterCentroids(random_state=42)),
#     ("svm", SVC(probability=True, random_state=42))
# ])
#
# svm_pipe_grid = {
#     'svm__kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
#     'svm__gamma': ['scale', 'auto', 1, 10, 0.1, 0.01],
#     'svm__coef0': [0, 1, 2, 5, 10, 0.1, 0.01],
#     'svm__degree': [2, 3, 4, 5],
#     'svm__C': [0.01, 0.1, 0.5, 1, 2, 10, 20]
# }
#
# svm_cluster = RandomizedSearchCV(
#     estimator=svm_pipeline,
#     param_distributions=svm_pipe_grid,
#     scoring=scoring,
#     refit='roc_auc_ovr',
#     cv=cv,
#     n_jobs=-1,
#     verbose=1
# )
# svm_cluster.fit(X_train, y_train)
#
# print(f"Лучшие параметры: {svm_cluster.best_params_}")
# print(f"Лучший ROC AUC (macro, OVR) на обучающих данных:
{svm_cluster.best_score_:.4f}")

metrics_df = pd.DataFrame(svm_cluster.cv_results_)[[
    'mean_test_accuracy',
    'mean_test_precision_macro',
    'mean_test_recall_macro',
    'mean_test_f1_macro',
    'mean_test_roc_auc_ovr'
]]

metrics_df = metrics_df.sort_values(by='mean_test_roc_auc_ovr',
ascending=False).iloc[0, :]
metrics_df.name = 'Metric'

display(metrics_df)

# Commented out IPython magic to ensure Python compatibility.
# %%time
#
# rf_pipeline = Pipeline([
#     ("cluster", ClusterCentroids(random_state=42)),
```

```
#     ("rf", RandomForestClassifier(random_state=42, n_jobs=-1))
# ])
#
# rf_pipe_grid = {
#     'rf__criterion': ['gini', 'entropy'],
#     'rf__n_estimators': [200, 400, 600],
#     'rf__max_features': [0.3, 0.5, 0.7, 1.],
#     'rf__max_depth': [1, 3, 5]
# }
#
# rf_cluster = GridSearchCV(
#     estimator=rf_pipeline,
#     param_grid=rf_pipe_grid,
#     scoring=scoring,
#     refit='roc_auc_ovr',
#     cv=cv,
#     n_jobs=-1,
#     verbose=1
# )
# rf_cluster.fit(X_train, y_train)
#
# print(f"Лучшие параметры: {rf_cluster.best_params_}")
# print(f"Лучший ROC AUC (macro, OVR) на обучающих данных:
{rf_cluster.best_score_:.4f}")

metrics_df = pd.DataFrame(rf_cluster.cv_results_)[[
    'mean_test_accuracy',
    'mean_test_precision_macro',
    'mean_test_recall_macro',
    'mean_test_f1_macro',
    'mean_test_roc_auc_ovr'
]]

metrics_df = metrics_df.sort_values(by='mean_test_roc_auc_ovr',
ascending=False).iloc[0, :]
metrics_df.name = 'Metric'

display(metrics_df)

"""# Стратификация"""

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
stratify=y, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

print(f'Объем обучающей выборки X_train: {X_train.shape} | y_train:
{y_train.shape}')
```

```
print(f'Объем тестовой выборки X_test: {X_test.shape} | y_test:
{y_test.shape}')
print(f'Соотношение данных равно {y_train.shape[0] / y.shape[0]:.4f} /
{y_test.shape[0] / y.shape[0]:.4f}')

"""## Дерево решений"""

tree_grid = {
    'ccp_alpha': [0.005, 0.01, 0.015, 0.02, 0.025, 0.03, 0.035, 0.2,
0.8],
    'criterion': ['entropy', 'gini']
}

tree = GridSearchCV(
    estimator=DecisionTreeClassifier(random_state=42),
    param_grid=tree_grid,
    scoring=scoring,
    refit='roc_auc_ovr',
    cv=cv,
    n_jobs=-1,
    verbose=1
)
tree.fit(X_train, y_train)

print(f"Лучшие параметры: {tree.best_params_}")
print(f"Лучший ROC-AUC (macro, OVR) на обучающих данных:
{tree.best_score_:.4f}")

metrics_df = pd.DataFrame(tree.cv_results_)[[
    'mean_test_accuracy',
    'mean_test_precision_macro',
    'mean_test_recall_macro',
    'mean_test_f1_macro',
    'mean_test_roc_auc_ovr'
]]

metrics_df = metrics_df.sort_values(by='mean_test_roc_auc_ovr',
ascending=False).iloc[0, :]
metrics_df.name = 'Metric'

display(metrics_df)

clf = tree.best_estimator_
y_pred = clf.predict(X_test)
y_proba = clf.predict_proba(X_test)

print("\nClassification report:")
print(classification_report(y_test, y_pred, zero_division=np.nan))

roc_auc = roc_auc_score(y_test_bin, y_proba, multi_class='ovr',
average='macro')
```

```
print(f"ROC-AUC (macro, OVR): {roc_auc:.4f}")

plt.figure(figsize=(10, 8))

for i, class_label in enumerate(classes):
    fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_proba[:, i])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'Класс {class_label} (AUC =
{roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--', label='Случайный выбор (AUC = 0.5)')
plt.plot([0, 0, 1], [0, 1, 1], 'k:', label='Идеальная производительность
(AUC = 1.0)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-кривые (One-vs-Rest)')
plt.legend(loc='lower right')
plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 8))

for i, class_label in enumerate(classes):
    precision, recall, _ = precision_recall_curve(y_test_bin[:, i],
y_proba[:, i])
    ap = average_precision_score(y_test_bin[:, i], y_proba[:, i])
    plt.plot(recall, precision, label=f'Класс {class_label} (AUC =
{ap:.2f})')

plt.plot([0, 1], [0.5, 0.5], 'k--', label='Случайный выбор (AUC = 0.5)')
#plt.plot([0, 1], [1, 1], 'k:', label='Идеальная производительность (AUC
= 1.0)')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall кривые (One-vs-Rest)')
plt.legend(loc='best')#, bbox_to_anchor=(1, 1))
plt.tight_layout()
plt.show()

features = df.columns[:-1]

plt.figure(figsize=(15, 8))
ax = plt.gca()
annot = plot_tree(clf, feature_names=features, max_depth=2, fontsize=7,
filled=True, ax=ax)
plt.show()

"""## Машина опорных векторов"""

svm_grid = {
```

```
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'gamma': ['scale', 'auto', 1, 10, 0.1, 0.01],
    'coef0': [0, 1, 2, 5, 10, 0.1, 0.01],
    'degree': [2, 3, 4, 5],
    'C': [0.01, 0.1, 0.5, 1, 2, 10, 20]
}

svm = RandomizedSearchCV(
    estimator=SVC(probability=True, random_state=42),
    param_distributions=svm_grid,
    scoring=scoring,
    refit='roc_auc_ovr',
    cv=cv,
    n_jobs=-1,
    random_state=42,
    verbose=1
)
svm.fit(X_train, y_train)

print(f"Лучшие параметры: {svm.best_params_}")
print(f"Лучший ROC-AUC (macro, OVR) на обучающих данных:
{svm.best_score_:.4f}")

metrics_df = pd.DataFrame(svm.cv_results_)[[
    'mean_test_accuracy',
    'mean_test_precision_macro',
    'mean_test_recall_macro',
    'mean_test_f1_macro',
    'mean_test_roc_auc_ovr'
]]

metrics_df = metrics_df.sort_values(by='mean_test_roc_auc_ovr',
ascending=False).iloc[0, :]
metrics_df.name = 'Metric'

display(metrics_df)

clf = svm.best_estimator_
y_pred = clf.predict(X_test)
y_proba = clf.predict_proba(X_test)

print("\nClassification report:")
print(classification_report(y_test, y_pred, zero_division=np.nan))

roc_auc = roc_auc_score(y_test_bin, y_proba, multi_class='ovr',
average='macro')
print(f"ROC-AUC (macro, OVR): {roc_auc:.4f}")

plt.figure(figsize=(10, 8))

for i, class_label in enumerate(classes):
```

```
    fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_proba[:, i])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'Класс {class_label} (AUC =
{roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--', label='Случайный выбор (AUC = 0.5)')
plt.plot([0, 0, 1], [0, 1, 1], 'k:', label='Идеальная производительность
(AUC = 1.0)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-кривые (One-vs-Rest)')
plt.legend(loc='lower right')
plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 8))

for i, class_label in enumerate(classes):
    precision, recall, _ = precision_recall_curve(y_test_bin[:, i],
y_proba[:, i])
    ap = average_precision_score(y_test_bin[:, i], y_proba[:, i])
    plt.plot(recall, precision, label=f'Класс {class_label} (AUC =
{ap:.2f})')

plt.plot([0, 1], [0.5, 0.5], 'k--', label='Случайный выбор (AUC = 0.5)')
#plt.plot([0, 1], [1, 1], 'k:', label='Идеальная производительность (AUC
= 1.0)')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall кривые (One-vs-Rest)')
plt.legend(loc='best')#, bbox_to_anchor=(1, 1))
plt.tight_layout()
plt.show()

"""## Случайный лес"""

# Commented out IPython magic to ensure Python compatibility.
# %%time
#
# rf_grid = {
#     'criterion': ['gini', 'entropy'],
#     'n_estimators': [200, 400, 600],
#     'max_features': [0.3, 0.5, 0.7, 1.],
#     'max_depth': [1, 3, 5]
# }
#
# rf = GridSearchCV(
#     estimator=RandomForestClassifier(random_state=42, n_jobs=-1),
#     param_grid=rf_grid,
#     scoring=scoring,
```

```
#     refit='roc_auc_ovr',
#     cv=cv,
#     n_jobs=-1,
#     verbose=1
# )
# rf.fit(X_train, y_train)
#
# print(f"Лучшие параметры: {rf.best_params_}")
# print(f"Лучший ROC AUC (macro, OVR) на обучающих данных:
{rf.best_score_:.4f}")

metrics_df = pd.DataFrame(rf.cv_results_)[[
    'mean_test_accuracy',
    'mean_test_precision_macro',
    'mean_test_recall_macro',
    'mean_test_f1_macro',
    'mean_test_roc_auc_ovr'
]]

metrics_df = metrics_df.sort_values(by='mean_test_roc_auc_ovr',
ascending=False).iloc[0, :]
metrics_df.name = 'Metric'

display(metrics_df)
```

ЛАБОРАТОРНАЯ РАБОТА 2

```
# -*- coding: utf-8 -*-
"""lab2.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1juKy1KyIuONLO_89mwDlXzl11mNp8Y7A

# Лабораторная работа №2
"""

!pip install -q catboost optuna

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd

import kagglehub
from kagglehub import KaggleDatasetAdapter

from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier, HistGradientBoostingClassifier, RandomForestClassifier
from sklearn.model_selection import train_test_split, StratifiedKFold, GridSearchCV, cross_val_score
from sklearn.preprocessing import RobustScaler, OneHotEncoder
from sklearn.metrics import classification_report, f1_score

from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier, Pool
import lightgbm as lgbm
import shap
import optuna

import warnings
warnings.filterwarnings('ignore')

import random
random.seed(42)
np.random.seed(42)

"""# Heart Failure Prediction Dataset
11 clinical features for predicting heart disease events.
"""
```

```
df = kagglehub.load_dataset(
    KaggleDatasetAdapter.PANDAS,
    "fedesoriano/heart-failure-prediction",
    "heart.csv"
)

df.head()

df.info()

df.describe(include=['number'])

numerical_cols = ['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak',
'HeartDisease']
categorical_cols = ['Sex', 'ChestPainType', 'RestingECG',
'ExerciseAngina', 'ST_Slope', 'FastingBS']

"""## Займемся обработкой категориальных данных

### Sex
"""

df['Sex'] = df['Sex'].apply(lambda x: 1 if x == 'M' else 0)

df['Sex'].value_counts()

"""### ChestPainType
TA: Typical Angina

ATA: Atypical Angina

NAP: Non-Anginal Pain

ASY: Asymptomatic
"""

df['ChestPainType'].value_counts()

ohe = OneHotEncoder(drop='first', sparse_output=False)
chest_pain_types = ohe.fit_transform(df[['ChestPainType']])
chest_pain_types = pd.DataFrame(chest_pain_types,
columns=ohe.get_feature_names_out())
chest_pain_types

ohe.categories_

df = pd.concat([df, chest_pain_types], axis=1)
df = df.drop(columns=['ChestPainType'])
df.head()

"""### RestingECG
```

```
Normal: Normal

ST: having ST-T wave abnormality (T wave inversions and/or ST elevation
or depression of > 0.05 mV)

LVH: showing probable or definite left ventricular hypertrophy by Estes'
criteria
"""

resting = pd.get_dummies(df['RestingECG'], prefix='RestingECG',
drop_first=True, dtype='float64')
df = pd.concat([df, resting], axis=1)
df = df.drop(columns=['RestingECG'])
df.head()

"""### ExerciseAngina
Y: Yes

N: No
"""

df['ExerciseAngina'] = df['ExerciseAngina'].apply(lambda x: 1 if x ==
'Y' else 0)

df['ExerciseAngina'].value_counts()

"""### ST_Slope

Up: upsloping

Flat: flat

Down: downsloping
"""

slope = pd.get_dummies(df['ST_Slope'], prefix='ST_Slope',
drop_first=True, dtype='float64')
df = pd.concat([df, slope], axis=1)
df = df.drop(columns=['ST_Slope'])
df.head()

"""# Итог"""

df.head()

df.info()

plt.figure(figsize=(10, 8))
ax = sns.pairplot(df[numerical_cols], kind="scatter",
hue="HeartDisease", markers=["o", "s", "D"], palette="Set2")
```

```
ax.fig.suptitle("Correlation plot", y=1.03)
plt.show()

plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap="Blues", annot_kws={"size": 7})
plt.show()

height = df['HeartDisease'].value_counts().tolist()
bars = ('Heart Disease', 'Normal')
x_pos = np.arange(len(bars))

container = plt.bar(x_pos, height, color=(0.5, 0.1, 0.5, 0.6))
plt.bar_label(container, padding=3)

plt.title('Heart Disease Bar Plot')
plt.xlabel('categories')
plt.ylabel('values')
plt.xticks(x_pos, bars)

plt.ylim(0, max(height) * 1.1)

plt.show()

"""# Подготовка данных"""

X = df.drop(columns=['HeartDisease'])
y = df['HeartDisease']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
random_state=42)

scaler = RobustScaler().set_output(transform="pandas")
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

results: dict[str, dict[str, float]] = {}
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

"""# Random Forest"""

# Commented out IPython magic to ensure Python compatibility.
# %%time
#
# rf_grid = {
#     'criterion': ['gini', 'entropy'],
#     'n_estimators': [200, 400, 600],
#     'max_features': [0.3, 0.5, 0.7, 1.],
#     'max_depth': [1, 3, 5]
# }
#
# rf = GridSearchCV(
```

```
#     estimator=RandomForestClassifier(random_state=42, n_jobs=-1),
#     param_grid=rf_grid,
#     scoring='f1_macro',
#     cv=cv,
#     n_jobs=-1,
#     verbose=1
# )
# rf.fit(X_train, y_train)
#
# print(f"Лучшие параметры: {rf.best_params_}")

print(classification_report(y_train, rf.predict(X_train)))

print(classification_report(y_test, rf.predict(X_test)))

importances = rf.best_estimator_.feature_importances_
pd.DataFrame(importances, columns=['value'],
index=rf.best_estimator_.feature_names_in_).sort_values('value',
ascending=False)

report = classification_report(y_test, rf.predict(X_test),
output_dict=True)
results['Random Forest'] = {k: v for k, v in report['macro avg'].items()
if k != 'support'}

"""# AdaBoost"""

# Commented out IPython magic to ensure Python compatibility.
# %%time
#
# ada_grid = {
#     'n_estimators': [200, 400, 600],
#     'learning_rate': [0.3, 0.5, 0.7, 1.],
# }
#
# ada = GridSearchCV(
#     estimator=AdaBoostClassifier(random_state=42),
#     param_grid=ada_grid,
#     scoring='f1_macro',
#     cv=cv,
#     n_jobs=-1,
#     verbose=1
# )
# ada.fit(X_train, y_train)
#
# print(f"Лучшие параметры: {ada.best_params_}")

print(classification_report(y_train, ada.predict(X_train)))

print(classification_report(y_test, ada.predict(X_test)))
```

```
report = classification_report(y_test, ada.predict(X_test),
output_dict=True)
results['AdaBoost'] = {k: v for k, v in report['macro avg'].items() if k
!= 'support'}

"""# Gradient Boosting"""

def objective_gb(trial):
    params = {
        "n_estimators": trial.suggest_int("n_estimators", 50, 500),
        "learning_rate": trial.suggest_float("learning_rate", 0.005,
0.2, log=True),
        "max_depth": trial.suggest_int("max_depth", 3, 10),
        "subsample": trial.suggest_float("subsample", 0.5, 1.0),
        "min_samples_split": trial.suggest_int("min_samples_split", 2,
20),
        "min_samples_leaf": trial.suggest_int("min_samples_leaf", 1,
20),
        "max_features": trial.suggest_categorical("max_features",
["sqrt", "log2", None]),
        "random_state": 42
    }
    model = GradientBoostingClassifier(**params)
    score = cross_val_score(model, X_train, y_train, cv=cv,
scoring='f1_macro').mean()

    return score

study = optuna.create_study(direction='maximize')
study.optimize(objective_gb, n_trials=30)

gb = GradientBoostingClassifier(**study.best_params)
gb.fit(X_train, y_train)

print(classification_report(y_train, gb.predict(X_train)))

print(classification_report(y_test, gb.predict(X_test)))

report = classification_report(y_test, gb.predict(X_test),
output_dict=True)
results['Gradient Boosting'] = {k: v for k, v in report['macro
avg'].items() if k != 'support'}

explainer = shap.TreeExplainer(gb)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)

explainer = shap.Explainer(gb, X_test)
shap_values = explainer(X_test, check_additivity=False)
shap.plots.waterfall(shap_values[0])
```

```
"""# HistGradientBoosting

"""

def objective_hgb(trial):
    params = {
        "max_iter": trial.suggest_int("max_iter", 100, 1500),
        "learning_rate": trial.suggest_float("learning_rate", 0.005,
0.2, log=True),
        "max_leaf_nodes": trial.suggest_int("max_leaf_nodes", 15, 255),
        "max_depth": trial.suggest_int("max_depth", 3, 15),
        "min_samples_leaf": trial.suggest_int("min_samples_leaf", 10,
100),
        "l2_regularization": trial.suggest_float("l2_regularization",
1e-8, 10.0, log=True),
        "max_bins": trial.suggest_int("max_bins", 50, 255),
        "early_stopping": True,
        "n_iter_no_change": 15,
        "random_state": 42
    }
    model = HistGradientBoostingClassifier(**params)
    score = cross_val_score(model, X_train, y_train, cv=cv,
scoring='f1_macro').mean()

    return score

study = optuna.create_study(direction='maximize')
study.optimize(objective_hgb, n_trials=30)

hgb = HistGradientBoostingClassifier(**study.best_params)
hgb.fit(X_train, y_train)

print(classification_report(y_train, hgb.predict(X_train)))

print(classification_report(y_test, hgb.predict(X_test)))

report = classification_report(y_test, hgb.predict(X_test),
output_dict=True)
results['Hist Gradient Boosting'] = {k: v for k, v in report['macro
avg'].items() if k != 'support'}

explainer = shap.TreeExplainer(hgb)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)

explainer = shap.Explainer(hgb, X_test)
shap_values = explainer(X_test, check_additivity=False)
shap.plots.waterfall(shap_values[0])

"""# XGBoost"""
```

```python
def objective_xgb(trial):
    train_x, val_x, train_y, val_y = train_test_split(X_train, y_train,
test_size=0.2, random_state=42)

    params = {
        "n_estimators": trial.suggest_int("n_estimators", 500, 2000),
        "learning_rate": trial.suggest_float("learning_rate", 1e-3, 0.1,
log=True),
        "max_depth": trial.suggest_int("max_depth", 3, 12),
        "min_child_weight": trial.suggest_int("min_child_weight", 1,
20),
        "subsample": trial.suggest_float("subsample", 0.5, 1.0),
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0.5,
1.0),
        "gamma": trial.suggest_float("gamma", 1e-8, 1.0, log=True), #
Контроль разрастания дерева
        "lambda": trial.suggest_float("lambda", 1e-8, 10.0, log=True), #
L2
        "alpha": trial.suggest_float("alpha", 1e-8, 10.0, log=True),   #
L1
        "tree_method": "hist", # Ускорение обучения
        "early_stopping_rounds": 100,
        "n_jobs": -1
    }

    model = XGBClassifier(**params)
    model.fit(
        train_x, train_y,
        eval_set=[(val_x, val_y)],
        verbose=False
    )

    preds = model.predict(val_x)
    score = f1_score(val_y, preds, average="macro")

    return score

study = optuna.create_study(direction='maximize')
study.optimize(objective_xgb, n_trials=30)

xgb = XGBClassifier(**study.best_params)
xgb.fit(
    X_train, y_train,
    eval_set=[(X_test, y_test)],
    verbose=False
)

print(classification_report(y_train, xgb.predict(X_train)))

print(classification_report(y_test, xgb.predict(X_test)))
```

```
report = classification_report(y_test, xgb.predict(X_test),
output_dict=True)
results['XGBoost'] = {k: v for k, v in report['macro avg'].items() if k
!= 'support'}

explainer = shap.TreeExplainer(xgb)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)

explainer = shap.Explainer(xgb, X_test)
shap_values = explainer(X_test, check_additivity=False)
shap.plots.waterfall(shap_values[0])

"""# LightGBM"""

def objective_lgb(trial):
    train_x, val_x, train_y, val_y = train_test_split(X_train, y_train,
test_size=0.2)

    params = {
        "n_estimators": trial.suggest_int("n_estimators", 500, 2000),
        "learning_rate": trial.suggest_float("learning_rate", 1e-3, 0.1,
log=True),
        "max_depth": trial.suggest_int("max_depth", 3, 15),
        "min_child_samples": trial.suggest_int("min_child_samples", 5,
100),
        "feature_fraction": trial.suggest_float("feature_fraction", 0.5,
1.0),
        "bagging_fraction": trial.suggest_float("bagging_fraction", 0.5,
1.0),
        "bagging_freq": trial.suggest_int("bagging_freq", 1, 10),
        "lambda_l1": trial.suggest_float("lambda_l1", 1e-8, 10.0,
log=True),
        "lambda_l2": trial.suggest_float("lambda_l2", 1e-8, 10.0,
log=True),
        "verbosity": -1,
        "n_jobs": -1,
        "random_state": 42
    }

    model = LGBMClassifier(**params)

    model.fit(
        train_x, train_y,
        eval_set=[(val_x, val_y)],
        callbacks=[lgbm.early_stopping(stopping_rounds=100),
lgbm.log_evaluation(period=0)]
    )

    preds = model.predict(val_x)
    score = f1_score(val_y, preds, average="macro")
```

```
    return score

study = optuna.create_study(direction='maximize')
study.optimize(objective_lgb, n_trials=30)

lgb = LGBMClassifier(**study.best_params)
lgb.fit(
    X_train, y_train,
    eval_set=[(X_test, y_test)],
    callbacks=[lgbm.early_stopping(stopping_rounds=100),
lgbm.log_evaluation(period=0)]
)

print(classification_report(y_train, lgb.predict(X_train)))

print(classification_report(y_test, lgb.predict(X_test)))

report = classification_report(y_test, lgb.predict(X_test),
output_dict=True)
results['LightGBM'] = {k: v for k, v in report['macro avg'].items() if k
!= 'support'}

explainer = shap.TreeExplainer(lgb)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)

explainer = shap.Explainer(lgb, X_test)
shap_values = explainer(X_test, check_additivity=False)
shap.plots.waterfall(shap_values[0])

"""# CatBoost"""

def objective_cat(trial):
    train_x, val_x, train_y, val_y = train_test_split(X_train, y_train,
test_size=0.2, random_state=42)

    params = {
        'iterations': trial.suggest_int('iterations', 100, 2000),
        'learning_rate': trial.suggest_float('learning_rate', 0.001,
0.3, log=True),
        'depth': trial.suggest_int('depth', 4, 10),
        'l2_leaf_reg': trial.suggest_float('l2_leaf_reg', 1e-8, 10.0,
log=True),
        'bootstrap_type': trial.suggest_categorical('bootstrap_type',
['Bayesian', 'Bernoulli', 'MVS']),
        'random_strength': trial.suggest_float('random_strength', 1e-8,
10.0, log=True),
        'od_type': 'Iter',
        'od_wait': 50,
        'verbose': False,
```

```
        'random_state': 42
    }

    model = CatBoostClassifier(**params)

    model.fit(
        train_x, train_y,
        eval_set=(val_x, val_y),
        early_stopping_rounds=100,
        use_best_model=True
    )

    preds = model.predict(val_x)
    score = f1_score(val_y, preds, average="macro")

    return score

study = optuna.create_study(direction='maximize')
study.optimize(objective_cat, n_trials=30)

cat = CatBoostClassifier(**study.best_params)
cat.fit(
    X_train, y_train,
    eval_set=(X_test, y_test),
    early_stopping_rounds=100,
    use_best_model=True
)

print(classification_report(y_train, cat.predict(X_train)))

print(classification_report(y_test, cat.predict(X_test)))

report = classification_report(y_test, cat.predict(X_test),
output_dict=True)
results['CatBoost'] = {k: v for k, v in report['macro avg'].items() if k
!= 'support'}

explainer = shap.TreeExplainer(cat)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)

explainer = shap.Explainer(cat, X_test)
shap_values = explainer(X_test, check_additivity=False)
shap.plots.waterfall(shap_values[0])

"""# Результаты"""

res = pd.DataFrame.from_dict(results)
display(res)
```