Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Системы обработки информации и управления

Отчет по рубежному контролю №2

«Методы построения моделей машинного обучения»

по дисциплине «Технологии машинного обучения»

Студент ИУ5-65Б _____ Д.А. Шиленок
(Группа) (Подпись, дата) (И.О.Фамилия)

Преподаватель _____ Ю.Е. Гапанюк
(Подпись, дата) (И.О.Фамилия)

Москва

2025

Ход выполнения

Шиленок Даниил Андреевич ИУ5-65Б

Вариант 20

Группа	Метод №1	Метод №2
ИУ5-65Б	Метод опорных векторов	Градиентный бустинг

Задание. Для заданного набора данных (по Вашему варианту) постройте модели классификации или регрессии (в зависимости от конкретной задачи, рассматриваемой в наборе данных). Для построения моделей используйте методы 1 и 2 (по варианту для Вашей группы). Оцените качество моделей на основе подходящих метрик качества (не менее двух метрик). Какие метрики качества Вы использовали и почему? Какие выводы Вы можете сделать о качестве построенных моделей? Для построения моделей необходимо выполнить требуемую предобработку данных: заполнение пропусков, кодирование категориальных признаков, и т.д.

Для данного датасета построим модель для решения задачи классификации.

В качестве целевого признака выберем koi_disposition (CANDIDATE, FALSE POSITIVE, CONFIRMED).

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, f1_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.impute import SimpleImputer

data = pd.read_csv('cumulative.csv')
```

```
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9564 entries, 0 to 9563
Data columns (total 50 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   rowid               9564 non-null   int64  
 1   kepid               9564 non-null   int64  
 2   kepoi_name          9564 non-null   object 
 3   kepler_name         2294 non-null   object 
 4   koi_disposition     9564 non-null   object 
 5   koi_pdisposition    9564 non-null   object 
 6   koi_score           8054 non-null   float64
 7   koi_fpflag_nt       9564 non-null   int64  
 8   koi_fpflag_ss       9564 non-null   int64  
 9   koi_fpflag_co       9564 non-null   int64  
 10  koi_fpflag_ec       9564 non-null   int64  
 11  koi_period          9564 non-null   float64
 12  koi_period_err1     9110 non-null   float64
 13  koi_period_err2     9110 non-null   float64
 14  koi_time0bk         9564 non-null   float64
 15  koi_time0bk_err1    9110 non-null   float64
 16  koi_time0bk_err2    9110 non-null   float64
 17  koi_impact          9201 non-null   float64
 18  koi_impact_err1     9110 non-null   float64
 19  koi_impact_err2     9110 non-null   float64
...
 49  koi_kepmag          9563 non-null   float64
dtypes: float64(39), int64(6), object(5)
memory usage: 3.6+ MB
None
```

```python
# Удаление ненужных столбцов
columns_to_drop = [
    'rowid', 'kepid', 'kepoi_name', 'kepler_name',
    'koi_tce_delivname', 'koi_tce_plnt_num',
    'koi_score', 'koi_pdisposition', 'koi_fpflag_nt', 'koi_fpflag_ss',
    'koi_fpflag_co', 'koi_fpflag_ec'
]
data = data.drop(columns=[col for col in columns_to_drop if col in data.columns])
```

```python
# Обработка пропусков
# Сначала проверим, какие столбцы полностью пустые
empty_cols = data.columns[data.isna().all()].tolist()
if empty_cols:
    print(f"Удаляем полностью пустые столбцы: {empty_cols}")
    data = data.drop(columns=empty_cols)
```

```
Удаляем полностью пустые столбцы: ['koi_teq_err1', 'koi_teq_err2']
```

```python
# Числовые признаки
numeric_cols = data.select_dtypes(include=['float64', 'int64']).columns.tolist()
numeric_cols = [col for col in numeric_cols if col != 'koi_disposition']
# Категориальные признаки
categorical_cols = data.select_dtypes(include=['object']).columns.tolist()

# Заполнение пропусков только в тех столбцах, где есть хотя бы одно значение
numeric_cols_to_impute = [col for col in numeric_cols if data[col].notna().any()]
if numeric_cols_to_impute:
    imputer_num = SimpleImputer(strategy='median')
    data[numeric_cols_to_impute] = imputer_num.fit_transform(data[numeric_cols_to_impute])

categorical_cols_to_impute = [col for col in categorical_cols if data[col].notna().any()]
if categorical_cols_to_impute:
    imputer_cat = SimpleImputer(strategy='most_frequent')
    data[categorical_cols_to_impute] = imputer_cat.fit_transform(data[categorical_cols_to_impute])

# Кодирование целевой переменной
label_encoder = LabelEncoder()
data['koi_disposition'] = label_encoder.fit_transform(data['koi_disposition'])
```

```python
# Разделение на признаки и целевую переменную
X = data.drop(columns=['koi_disposition'])
y = data['koi_disposition']
# Разделение на train и test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=21)

# Масштабирование числовых признаков
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```python
# Метод опорных векторов (SVM)
svm = SVC(probability=True)
svm.fit(X_train, y_train)
y_pred_svm = svm.predict(X_test)

# Градиентный бустинг
gb = GradientBoostingClassifier(random_state=21)
gb.fit(X_train, y_train)
y_pred_gb = gb.predict(X_test)

# Оценка качества
print("Метод опорных векторов:")
print("Accuracy:", accuracy_score(y_test, y_pred_svm))
print("F1-score:", f1_score(y_test, y_pred_svm, average='weighted'))

print("\nГрадиентный бустинг:")
print("Accuracy:", accuracy_score(y_test, y_pred_gb))
print("F1-score:", f1_score(y_test, y_pred_gb, average='weighted'))
```

```
Метод опорных векторов:
Accuracy: 0.7349712493465761
F1-score: 0.7014390173044057

Градиентный бустинг:
Accuracy: 0.7877679038159958
F1-score: 0.7790239010057025
```

В данной задаче многоклассовой классификации были выбраны следующие метрики для оценки моделей:

- **Accuracy** – показывает долю правильно классифицированных объектов от общего числа. Эта метрика является интуитивно понятной и хорошо отражает качество модели при равномерном распределении классов.
- **F1-мера** – объединяет precision и recall в одну метрику, позволяя учитывать как ложные срабатывания, так и пропущенные классы. Особенно актуальна при наличии дисбаланса между классами.

Обе метрики дают обоснованное представление о качестве модели в различных сценариях распределения классов.

Модель SVM продемонстрировала умеренное качество классификации. Хотя она показала приемлемую точность (около 73,5%), значение F1-меры ниже, что может свидетельствовать о сложностях с корректной классификацией некоторых классов, особенно при наличии дисбаланса.

Модель градиентного бустинга превзошла SVM по обеим метрикам. Она достигла точности почти 79% и более высокой F1-меры (около 77,9%), что говорит о лучшей способности обобщать и учитывать различия между классами.