Московский государственный технический университет им. Н. Э. Баумана

Курс «Технологии машинного обучения»

Отчёт по рубежному контролю №2

«Методы построения моделей машинного обучения»

Вариант № 21

Выполнил:
Фролов М. К.
группа ИУ5-64Б

Проверил:
Гапанюк Ю.Е.

Дата: 12.06.25

Подпись:

Дата:

Подпись:

2025 г.

Задание:

Для заданного набора данных https://www.kaggle.com/datasets/oreojam/formula-e-world-championship-race-results постройте модели классификации или регрессии (в зависимости от конкретной задачи, рассматриваемой в наборе данных). Для построения моделей используйте методы Линейная/логистическая регрессия и Градиентный бустинг. 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.

Ход выполнения:

```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegressor

from sklearn.metrics import (
    accuracy_score, roc_auc_score, precision_score, recall_score, f1_score,
    mean_squared_error, mean_absolute_error, r2_score
)
```

```python
def time_to_seconds(t):
    if pd.isna(t):
        return np.nan
    parts = t.split(':')
    try:
        if len(parts) == 2:
            minutes, seconds = parts
            return int(minutes) * 60 + float(seconds)
        elif len(parts) == 3:
            hours, minutes, seconds = parts
            return int(hours) * 3600 + int(minutes) * 60 + float(seconds)
        else:
            return float(t)
    except:
        return np.nan
```

```python
df = pd.read_csv('Formual_E_Raceresults.csv')

for col in ['Best', 'Started', 'Time']:
    df[f'{col}_sec'] = df[col].apply(time_to_seconds)


df['SeasonYear'] = (
    df['SeasonName']
      .str.extract(r'(\d{4})', expand=False)
      .astype(float)
)
df = df[df['SeasonYear'].notna()].copy()
df['SeasonYear'] = df['SeasonYear'].astype(int)

df['Driver'] = df['DriverFirstName'] + ' ' + df['DriverLastName']

df['podium'] = (df['Pos'] <= 3).astype(int)

features = [
    'SeasonYear',
    'Team',
    'DriverNumber',
    'Driver',
    'Best_sec',
    'Started_sec',
    'Time_sec'
]
```

```
num_feats = ['SeasonYear', 'Best_sec', 'Started_sec', 'Time_sec']
cat_feats = ['Team', 'DriverNumber', 'Driver']

numeric_transformer = SimpleImputer(strategy='median')
categorical_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer([
    ('num', numeric_transformer, num_feats),
    ('cat', categorical_transformer, cat_feats),
])
```

Классификация “подиум/не-подиум”

```
X_clf = df[features]
y_clf = df['podium']

X_train_c, X_test_c, y_train_c, y_test_c = train_test_split(
    X_clf, y_clf,
    test_size=0.2,
    stratify=y_clf,
    random_state=42
)

pipe_lr_clf = Pipeline([
    ('prep', preprocessor),
    ('scale', StandardScaler(with_mean=False)),
    ('clf',   LogisticRegression(max_iter=1000))
])

pipe_gb_clf = Pipeline([
    ('prep', preprocessor),
    ('clf',   GradientBoostingClassifier(n_estimators=200, random_state=42))
])

pipe_lr_clf.fit(X_train_c, y_train_c)
pipe_gb_clf.fit(X_train_c, y_train_c)

y_pred_lr = pipe_lr_clf.predict(X_test_c)
y_proba_lr = pipe_lr_clf.predict_proba(X_test_c)[:,1]
y_pred_gb = pipe_gb_clf.predict(X_test_c)
y_proba_gb = pipe_gb_clf.predict_proba(X_test_c)[:,1]
```

```python
metrics_clf = pd.DataFrame({
    'Model': ['LogisticRegression', 'GradientBoosting'],
    'Accuracy': [
        accuracy_score(y_test_c, y_pred_lr),
        accuracy_score(y_test_c, y_pred_gb)
    ],
    'ROC AUC': [
        roc_auc_score(y_test_c, y_proba_lr),
        roc_auc_score(y_test_c, y_proba_gb)
    ],
    'Precision': [
        precision_score(y_test_c, y_pred_lr),
        precision_score(y_test_c, y_pred_gb)
    ],
    'Recall': [
        recall_score(y_test_c, y_pred_lr),
        recall_score(y_test_c, y_pred_gb)
    ],
    'F1-score': [
        f1_score(y_test_c, y_pred_lr),
        f1_score(y_test_c, y_pred_gb)
    ]
}).set_index('Model')

print("Classification metrics:\n", metrics_clf)
```

Model	Accuracy	ROC AUC	Precision	Recall	F1-score
LogisticRegression	0.657807	0.645965	0.475610	0.393939	0.430939
GradientBoosting	0.727575	0.720672	0.630769	0.414141	0.500000

```
X_reg = df[features]
y_reg = df['Pos']

X_train_r, X_test_r, y_train_r, y_test_r = train_test_split(
    X_reg, y_reg, test_size=0.2, random_state=42
)

pipe_lr_reg = Pipeline([
    ('prep', preprocessor),
    ('scale', StandardScaler(with_mean=False)),
    ('reg',   LinearRegression())
])
pipe_gb_reg = Pipeline([
    ('prep', preprocessor),
    ('reg',   GradientBoostingRegressor(n_estimators=200, random_state=42))
])

pipe_lr_reg.fit(X_train_r, y_train_r)
pipe_gb_reg.fit(X_train_r, y_train_r)

y_pred_lr_r = pipe_lr_reg.predict(X_test_r)
y_pred_gb_r = pipe_gb_reg.predict(X_test_r)

rmse_lr = np.sqrt(mean_squared_error(y_test_r, y_pred_lr_r))
rmse_gb = np.sqrt(mean_squared_error(y_test_r, y_pred_gb_r))
```

```python
X_reg = df[features]
y_reg = df['Pos']

X_train_r, X_test_r, y_train_r, y_test_r = train_test_split(
    X_reg, y_reg, test_size=0.2, random_state=42
)

pipe_lr_reg = Pipeline([
    ('prep', preprocessor),
    ('scale', StandardScaler(with_mean=False)),
    ('reg',   LinearRegression())
])
pipe_gb_reg = Pipeline([
    ('prep', preprocessor),
    ('reg',   GradientBoostingRegressor(n_estimators=200, random_state=42))
])

pipe_lr_reg.fit(X_train_r, y_train_r)
pipe_gb_reg.fit(X_train_r, y_train_r)

y_pred_lr_r = pipe_lr_reg.predict(X_test_r)
y_pred_gb_r = pipe_gb_reg.predict(X_test_r)

rmse_lr = np.sqrt(mean_squared_error(y_test_r, y_pred_lr_r))
rmse_gb = np.sqrt(mean_squared_error(y_test_r, y_pred_gb_r))
```

```
Regression metrics:
                        RMSE       MAE        R2
Model
LinearRegression  6.129629  4.929065 -0.082406
GradientBoosting  5.778500  4.637041  0.038050
```