Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Омский государственный технический университет»

Факультет *информационных технологий и компьютерных систем*

Кафедра *«Автоматизированные системы обработки информации и управления»*

КУРСОВОЙ ПРОЕКТ

по дисциплине ***Динамические языки программирования***

на тему ***Разработка классификатора данных об аниме***

Пояснительная записка

Шифр проекта **043-КП-09.03.04-№18-ПЗ**

Студента (ки) *Петровского Ильи Владимировича*
фамилия, имя, отчество полностью

Курс ***4*** Группа ***ПИН-201***

Направление (специальность) 09.03.04 – Программная инженерия
код, наименование

Руководитель ***старший преподаватель***
ученая степень, звание

Кабанов.А.А
фамилия, инициалы

Выполнил (а) _____
дата, подпись студента (ки)

К защите _____
дата, подпись руководителя

Проект (работа) защищен (а) с оценкой _____

Набранные баллы: _____ _____ _____
в семестре на защите Итого

Омск 2023

Задание

на выполнение курсового проекта

Задача 1. Найти набор данных (датасет) для классификации удовлетворяющий следующим условиям: более 10 000 строк, более 20 столбцов, разные типы в столбцах, обязательно наличие целевого признака (таргета).

Задача 2. Провести классификацию найденного датасета, методом к-ближайших соседей. В формате Markdown писать пояснения. Объяснить почему были выбраны именно такие гиперпараметры, была ли перекрестная проверка, и т.д.

Задача 3. Провести классификацию найденного датасета, методом машины опорных векторов. В формате Markdown писать пояснения. Объяснить почему были выбраны именно такие гиперпараметры, была ли перекрестная проверка, и т.д.

Задача 4. Провести классификацию найденного датасета, методами линеной и логистической регрессий. В формате Markdown написать пояснения. Объяснить почему были выбраны именно такие гиперпараметры, была ли перекрестная проверка, и т.д.

Задача 5. Провести классификацию найденного датасета, методами наивного Байеса. В формате Markdown написать пояснения. Объяснить почему были выбраны именно такие гиперпараметры, была ли перекрестная проверка, и т.д.

Задача 6. Провести классификацию найденного датасета, методами решающего дерева и случайного леса. В формате Markdown написать пояснения. Объяснить почему были выбраны именно такие гиперпараметры, была ли перекрестная проверка, и т.д.

Задача 7. Провести классификацию найденного датасета, методами CatBoost. В формате Markdown написать пояснения. Объяснить почему были выбраны именно такие гиперпараметры, была ли перекрестная проверка, и т.д.

РЕФЕРАТ

Пояснительная записка 42 с., 40 рис., 5 источников.

ИССЛЕДОВАНИЕ МОДЕЛЕЙ МАШИННОГО ОБУЧЕНИЯ, K-БЛИЖАЙШИХ СОСЕДЕЙ, МЕТОД ОПОРНЫХ ВЕКТОРОВ (SVM), ЛОГИСТИЧЕСКАЯ РЕГРЕССИЯ, НАИВНЫЙ БАЙЕСОВСКИЙ КЛАССИФИКАТОР, МОДЕЛЬ РЕШАЮЩЕГО ДЕРЕВА, СЛУЧАЙНЫЙ ЛЕС, CATBOOST, ОЦЕНКА ПРОИЗВОДИТЕЛЬНОСТИ, СРАВНИТЕЛЬНЫЙ АНАЛИЗ, ГИПЕРПАРАМЕТРЫ МОДЕЛЕЙ, МЕТРИКИ КАЧЕСТВА, ИНТЕРПРЕТИРУЕМОСТЬ МОДЕЛЕЙ

Объектом исследования данной работы являются различные модели машинного обучения, такие как *k*-ближайших соседей, метод опорных векторов, логистическая регрессия, наивный байесовский классификатор, модель решающего дерева, случайный лес и CatBoost.

Целью работы является использование моделей и проведение сравнительного анализа производительности различных моделей машинного обучения на основе метрик классификации. В процессе исследования будут проведены эксперименты с определением оптимальных гиперпараметров каждой модели, обучены модели на выборке данных, и оценены их показатели эффективности с использованием различных метрик.

Содержание

Введение

В современном мире использование методов машинного обучения становится все более важным и широко распространенным. Благодаря доступу к большим объемам данных и возможности эффективного их анализа, машинное обучение стало ключевым инструментом для обработки информации, прогнозирования и классификации данных в различных областях знаний.

Цель данной курсовой работы заключается в проведении анализа и сравнительного обзора различных моделей машинного обучения на основе конкретного набора данных. Работа включает в себя изучение и оценку производительности моделей, таких как *k*-ближайших соседей, метода опорных векторов, логистической регрессии, наивного байесовского классификатора, модели решающего дерева, случайного леса и CatBoost. Также проводится сопоставление результатов этих моделей.

Анализ работы моделей также включает сравнение их преимуществ и недостатков, а также выявление наиболее подходящей модели для конкретного датасета. Полученные результаты представляют ценность для областей, где необходимо делать прогнозы или классифицировать данные на основе имеющейся информации.

1 Модель k-ближайших соседей

1.1 Теория

Классификатор *k*-ближайших соседей (*k-nn*) - это метод машинного обучения, используемый для принятия решений на основе данных ближайших объектов в пространстве признаков. Он может выполнять как задачи классификации, определяя класс объекта, так и задачи регрессии, предсказывая его значение.

Основные аспекты метода *k-nn*:

1. Число соседей (*k*): Это один из важных параметров модели. Определяет количество ближайших соседей, учитываемых при принятии решения. Выбор значения *k* влияет на точность модели и её обобщающую способность.
2. Метрики расстояния: Для определения ближайших соседей необходимо использовать метрику расстояния. К примеру, Евклидово расстояние, Манхэттенское расстояние или Расстояние Чебышева - каждая из них определяет расстояние между объектами в пространстве признаков. Выбор подходящей метрики зависит от конкретной задачи и природы данных.
3. Процесс классификации объекта: После определения *k* ближайших соседей для нового объекта происходит классификация путем голосования большинства соседей. Объект относится к классу, который представлен наибольшим числом ближайших соседей.

k находит применение во многих областях, включая рекомендательные системы, классификацию текстов или изображений, анализ медицинских данных для определения паттернов или диагнозов, прогнозирование финансовых показателей и других задачах, требующих анализа данных. Важно выбирать параметры метода, исходя из специфики данных и требований задачи.

1.2 Задачи

Задача 1. Провести классификацию найденного датасета, методом k-ближайших соседей. В формате Markdown писать пояснения. Объяснить почему были выбраны именно такие гиперпараметры, была ли перекрестная проверка, и т.д.

Ход работы:

Был найден датасет с 24 столбцами и 24 905 строк. Информация о датасете представлена на рисунке 1.

```
Общая информация о данных:
Количество строк: 24905
Количество столбцов: 24

Типы данных по столбцам:
anime_id         int64
Name            object
English name    object
Other name      object
Score           object
Genres          object
Synopsis        object
Type            object
Episodes        object
Aired           object
Premiered       object
Status          object
Producers       object
Licensors       object
Studios         object
Source          object
Duration        object
Rating          object
Rank            object
Popularity       int64
Favorites        int64
Scored By       object
Members          int64
Image URL       object
dtype: object
```

Рисунок 1 – Информация о датасете

В модели используем следующие гиперпараметры:

```
# Задаем значения параметров для перебора
param_grid = {'n_neighbors': list(range(2, 11)), 'metric': ['euclidean', 'manhattan', 'chebyshev', 'minkowski']}
```

Рисунок 2 – Гиперпараметры

На рисунках 3-7 представлен код программы.

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, classification_report
```

Рисунок 4 – Импорт библиотек

```
# Разделение на признаки (X) и целевую переменную (y)
X = data.drop('Type', axis=1)
y = data['Type']
# Преобразование категориальных признаков в числовые
label_encoder = LabelEncoder()
for column in X.columns:
    if X[column].dtype == 'object':
        X[column] = label_encoder.fit_transform(X[column])
# Разделение на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Рисунок 5 – Разделение данных и нормализация

```
# Создание модели k-ближайших соседей
knn = KNeighborsClassifier()

# Задаем значения параметров для перебора
param_grid = {'n_neighbors': list(range(2, 11)), 'metric': ['euclidean', 'manhattan', 'chebyshev', 'minkowski']}

# Создание объекта Grid Search с кросс-валидацией (cv=5 для примера)
grid_search = GridSearchCV(knn, param_grid, cv=5)
# Обучение Grid Search для поиска лучших параметров
grid_search.fit(X_train, y_train)
# Получение лучших параметров и модели с этими параметрами
best_params = grid_search.best_params_
best_knn = grid_search.best_estimator_

# Получение имен из тестового набора данных
names_test = data.loc[X_test.index]['anime_id'].values

# Предсказания на тестовой выборке с использованием лучшей модели
predictions = best_knn.predict(X_test)
```

Рисунок 6 – Обучение модели

```
# Оценка точности лучшей модели
accuracy = accuracy_score(y_test, predictions)
print(f"\nЛучшие параметры: {best_params}")
print(f"Точность: {accuracy}")

# Отчет по классификации
print("\nОтчет по классификации:")
print(classification_report(y_test, predictions))
```

Рисунок 7 – Проверка модели и вывод результатов

В ходе обучения модели были выявлены лучшие гиперпараметры. На рисунке 9 представлены лучшие гиперпараметры, точность модели на реальных данных и отчёт по классификации.

```
Лучшие параметры: {'metric': 'manhattan', 'n_neighbors': 9}
Точность: 0.5775948604697851

Отчет по классификации:
              precision    recall  f1-score   support

       Movie       0.50      0.47      0.48       867
       Music       0.74      0.90      0.81       517
         ONA       0.48      0.42      0.45       683
         OVA       0.52      0.62      0.57       817
     Special       0.40      0.21      0.27       506
          TV       0.65      0.70      0.67      1576
     UNKNOWN       0.33      0.07      0.11        15

    accuracy                           0.58      4981
   macro avg       0.52      0.48      0.48      4981
weighted avg       0.56      0.58      0.56      4981
```

Рисунок 9 – Результаты

Модель демонстрирует средние параметры точности.

Модель имеет среднюю полноту (recall) это значит, что она определяет в большинстве случаев правильно.

Следовательно, значение F1-меры (f1-score) для классов Obesity, Hypertension также относительно невысоко, что указывает на слабую способность модели сбалансированно комбинировать точность и полноту для класса 1.

Общая точность модели (accuracy) составляет 0.58, что показателем выше среднего.

Макро-усредненная F1-мера (macro avg) для обеих классов составляет 0.48-0.52, что свидетельствует о нормальном гармоническом среднеметрическом

значении между точностью и полнотой для обоих классов. Однако, weighted avg F1-мера также указывает на взвешенное среднее значение метрики, где объекты каждого класса взвешиваются пропорционально их поддержке (support).

1.3 Вывод

Итак, модель нормально справляется с предсказанием объектов разных класса, и имеет незаметные проблемы, что указывает на необходимость улучшения способности модели к выявлению объектов этого класса для повышения ее общей производительности и балансировки предсказаний между классами.

Преимущества модели *k-nn*:

- Простота реализации и понимания;

- Не требует обучения на больших объемах данных перед прогнозированием;

- Эффективен для многих типов данных и может быть использован для задач классификации и регрессии.

Недостатки модели *k-nn*:

- Чувствительность к выбору числа соседей *k*;

- Требует хранения всего обучающего набора данных, что может быть ресурсоемким для больших наборов данных;

- Неэффективен для данных с большим числом признаков из-за "проклятия размерности".

2 Модель машины опорных векторов

2.1 Теория

Метод опорных векторов (англ. support vector machine, SVM) — один из наиболее популярных методов обучения, который применяется для решения задач классификации и регрессии. Основная идея метода заключается в построении гиперплоскости, разделяющей объекты выборки оптимальным способом. Алгоритм работает в предположении, что чем больше расстояние (зазор) между разделяющей гиперплоскостью и объектами разделяемых классов, тем меньше будет средняя ошибка классификатора.

Основная идея:

Максимизация зазора (Margin Maximization): SVM стремится найти оптимальную разделяющую гиперплоскость, которая максимизирует зазор между классами. Зазор представляет собой расстояние между разделяющей гиперплоскостью и ближайшими к ней точками обучающих данных, называемыми опорными векторами.

Опорные векторы (Support Vectors): Опорные векторы - это точки данных, находящиеся ближе всего к разделяющей гиперплоскости. Они играют важную роль в определении этой гиперплоскости и максимизации зазора.

Опорные векторы являются ключевыми, потому что они задают границы между классами, и изменения или удаление этих точек могут влиять на позицию разделяющей гиперплоскости.

Ядровая функция (Kernel Function): SVM использует ядровые функции для преобразования данных в пространство более высокой размерности. Это позволяет построить более сложные разделяющие гиперплоскости.

Линейное ядро (Linear Kernel): Производит линейное преобразование данных без изменения их структуры.

Радиальная базисная функция (Radial Basis Function) ядро: Одно из наиболее распространенных ядер. Преобразует данные в пространство с бесконечным числом измерений на основе радиального распределения относительно центра.

Сигмоидное ядро (Sigmoid): Преобразует данные на основе сигмоидной функции в пространство более высокой размерности.

Полиномиальное ядро (Polynomial Kernel): Полиномиальная ядровая функция преобразует данные в пространство более высокой размерности с использованием полиномиальной функции. Она представляет собой функцию, которая вычисляет степень полинома между точками данных, что позволяет строить нелинейные разделяющие гиперплоскости. Полиномиальное ядро может быть полезным при обработке данных, которые не могут быть линейно разделены в исходном пространстве признаков. Оно позволяет модели SVM строить более сложные гиперплоскости для разделения данных, представляющих собой нелинейные отношения между классами. Выбор правильной степени полинома и других параметров полиномиального ядра (degree и coef0) играет важную роль в процессе настройки модели SVM и может существенно влиять на ее производительность и способность обобщения на новые данные.

Гиперпараметры - это настройки модели, которые не могут быть изучены самой моделью и должны быть определены до начала процесса обучения. В данном случае, для SVM заданы следующие гиперпараметры:

C: Параметр регуляризации, который контролирует штраф за ошибки классификации. Меньшие значения C могут приводить к более гладким разделяющим гиперплоскостям, тогда как большие значения C позволяют модели создавать более точные разделяющие гиперплоскости.

kernel: Определяет тип ядровой функции для преобразования данных. В данном случае, определены линейное, RBF и сигмоидное ядра.

gamma: Параметр, используемый в некоторых ядровых функциях, таких как RBF. Высокие значения gamma могут привести к более сложным

разделяющим гиперплоскостям, что может привести к переобучению модели, в то время как низкие значения могут привести к недообучению.

Параметр degree: Это гиперпараметр полиномиальной ядровой функции, который определяет степень полинома. Он контролирует сложность модели, поскольку более высокие степени полинома могут создавать более сложные разделяющие поверхности. Значение по умолчанию для degree - 3.

Параметр coef0: Это свободный член в полиномиальной функции. Он контролирует, насколько сильно полиномиальная функция влияет на модель в сравнении с другими ядрами.

Эти гиперпараметры позволяют настраивать и оптимизировать производительность модели SVM в соответствии с данными и требованиями конкретной задачи.

2.2 Задачи

Задача 1. Провести классификацию найденного датасета, методом машины опорных векторов. В формате Markdown писать пояснения. Объяснить почему были выбраны именно такие гиперпараметры, была ли перекрестная проверка, и т.д.

Ход работы:

В модели используем следующие гиперпараметры:

```
param_grid = {'C': [0.01, 0.1, 1], 'kernel': ['linear', 'rbf', 'sigmoid'], 'gamma': [0.1, 1, 10]}
```

Рисунок 8 – Гиперпараметры

На рисунках 8-10 представлен код программы.

```
# Разделение на признаки (X) и целевую переменную (y)
X = data.drop('Type', axis=1)
y = data['Type']
label_encoder = LabelEncoder()
for column in X.columns:
    if X[column].dtype == 'object':
        X[column] = label_encoder.fit_transform(X[column])

# Разделение данных на тренировочный и тестовый наборы
# random_state=42 - гарантирует, что данные каждый раз будут одинакого разбиваться
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Масштабирование признаков (нормализация)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Рисунок 9 – Разделение данных и нормализация

```
grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=3)
grid.fit(X_train_scaled, y_train)

best_params = grid.best_params_
best_score = grid.best_score_
```

Рисунок 10 – Создание модели

```
print("Лучшие гиперпараметры:", grid.best_params_)
best_svc = grid.best_estimator_
y_pred = best_svc.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
print(f"Точность модели: {accuracy}")

# Получение отчета о классификации с предупреждениями
report = classification_report(y_test, y_pred, zero_division=1)
print(f'Отчет о классификации:\n{report}')
```

Рисунок 10 – Проверка модели и вывод результатов Процесс обучения

В ходе обучения модели были выявлены лучшие гиперпараметры. На рисунке 16 представлены лучшие гиперпараметры, точность модели на реальных данных и отчёт по классификации.

```
[CV 3/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.713 total time=  16.8s
[CV 4/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.726 total time=  17.0s
[CV 5/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.719 total time=  16.8s
[CV 1/5] END ..C=0.1, gamma=0.1, kernel=sigmoid;, score=0.521 total time=  11.2s
[CV 2/5] END ..C=0.1, gamma=0.1, kernel=sigmoid;, score=0.495 total time=  11.9s
[CV 3/5] END ..C=0.1, gamma=0.1, kernel=sigmoid;, score=0.499 total time=  12.0s
[CV 4/5] END ..C=0.1, gamma=0.1, kernel=sigmoid;, score=0.515 total time=  11.0s
[CV 5/5] END ..C=0.1, gamma=0.1, kernel=sigmoid;, score=0.506 total time=  12.0s
[CV 1/5] END .....C=0.1, gamma=1, kernel=linear;, score=0.713 total time=   5.9s
[CV 2/5] END .....C=0.1, gamma=1, kernel=linear;, score=0.711 total time=   6.0s
[CV 3/5] END .....C=0.1, gamma=1, kernel=linear;, score=0.710 total time=   5.9s
[CV 4/5] END .....C=0.1, gamma=1, kernel=linear;, score=0.716 total time=   6.2s
[CV 5/5] END .....C=0.1, gamma=1, kernel=linear;, score=0.712 total time=   6.0s
[CV 1/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.305 total time=  34.2s
[CV 2/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.305 total time=  34.0s
[CV 3/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.304 total time=  33.9s
[CV 4/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.304 total time=  34.0s
[CV 5/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.304 total time=  34.1s
[CV 1/5] END ....C=0.1, gamma=1, kernel=sigmoid;, score=0.345 total time=  17.5s
```

Рисунок 11- процесс обучения модели

```
Лучшие гиперпараметры: {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
Точность модели: 0.7906042963260389
Отчет о классификации:
              precision    recall  f1-score   support

       Movie       0.73      0.82      0.77       867
       Music       0.92      0.95      0.94       517
         ONA       0.64      0.63      0.63       683
         OVA       0.75      0.76      0.76       817
     Special       0.50      0.34      0.41       506
          TV       0.93      0.96      0.94      1576
     UNKNOWN       0.75      0.20      0.32        15

    accuracy                           0.79      4981
   macro avg       0.75      0.67      0.68      4981
weighted avg       0.78      0.79      0.78      4981
```

Рисунок 12 – Результаты

2.3 Вывод

Метод опорных векторов является мощным инструментом для решения задач классификации и регрессии, который хорошо работает как с линейно, так и с нелинейно разделимыми данными, в зависимости от выбранной ядровой функции и настроек модели. В данном датасете точность и параметры macro avg и weighted avg получилась больше чем в прошлом методе.

Преимущества метода опорных векторов:

- Эффективность в пространствах высокой размерности: SVM хорошо работает даже в пространствах с большим числом признаков, включая случаи, когда количество признаков превышает количество образцов.

- Хорошая обобщающая способность: Благодаря своей способности находить оптимальные разделяющие гиперплоскости, SVM обладает хорошей обобщающей способностью.

- Модификация ядровой функции: SVM позволяет использовать различные ядровые функции, такие как линейная, полиномиальная, радиальная базисная функция (RBF) и другие, для адаптации к различным типам данных.

Недостатки метода опорных векторов:

- Чувствительность к выбору ядра и параметров: Выбор подходящей ядровой функции и ее параметров может быть нетривиальной задачей и влиять на производительность модели.

- Подгонка к выбросам: SVM склонен быть чувствительным к выбросам в данных, особенно в случае несбалансированных классов.

- Вычислительная сложность: Работа с большими объемами данных может потребовать значительных вычислительных ресурсов.

3 Модель линейной регрессии

3.1 Теория

Линейная регрессия - это один из наиболее простых и широко используемых методов машинного обучения, который применяется для предсказания значений переменной на основе линейной зависимости между набором признаков и целевой переменной. Основная идея линейной регрессии заключается в том, чтобы найти линейную функцию, которая наилучшим образом описывает отношения между независимыми и зависимой переменными.

Основные концепции линейной регрессии:

1. Линейная зависимость:

Линейная регрессия предполагает, что зависимость между предсказываемой переменной (целевой) и независимыми признаками может быть аппроксимирована линейной функцией. Формула линейной регрессии имеет следующий вид:

$y = w0 + w1*x1 + w2*x2 + ... + wn*xn + \varepsilon$,

где:

y - целевая переменная, которую мы пытаемся предсказать.

x1, x2, ..., xn - независимые признаки (факторы).

w0, w1, ..., wn - коэффициенты модели (веса), которые определяют веса каждого признака.

ε - ошибка модели (шум, который не удается объяснить моделью).

2. Метод наименьших квадратов (Ordinary Least Squares, OLS):

Основной метод оценки параметров модели линейной регрессии - метод наименьших квадратов (OLS). Цель состоит в том, чтобы найти такие значения коэффициентов w0, w1, ..., wn, которые минимизируют сумму квадратов отклонений (расхождений) между фактическими и предсказанными значениями целевой переменной.

3. Оценка качества модели:

Для оценки качества предсказаний модели линейной регрессии используют различные метрики, включая:

Средняя квадратическая ошибка (Mean Squared Error, MSE): Средняя ошибка между фактическими и предсказанными значениями, возведенная в квадрат.

Коэффициент детерминации (Coefficient of Determination, R^2): Показывает, какую часть дисперсии целевой переменной объясняет модель.

4. Множественная линейная регрессия:

Множественная линейная регрессия включает более одной независимой переменной. В этом случае модель будет иметь вид:

$y = w0 + w1*x1 + w2*x2 + ... + wnxn + \varepsilon$,

где x1, x2, ..., xn - это несколько независимых переменных.

3.2 Задачи

Задача 1. Провести классификацию найденного датасета, методом линейной. В формате Markdown написать пояснения. Объяснить почему были выбраны именно такие гиперпараметры, была ли перекрестная проверка, и т.д.

Ход работы:

На рисунках 13-15 представлен остальной код программы.

```
# Преобразование категориальных признаков в числовые
label_encoder = LabelEncoder()
for column in data.columns:
    if data[column].dtype == 'object':
        data[column] = label_encoder.fit_transform(data[column])

# Удаление ненужных столбцов или подготовка признаков и целевой переменной
X = data.drop(['Type'], axis=1) # Признаки
y = data['Type']  # Целевая переменная
```

Рисунок 13 – Разделение данных и нормализация

```
# Обучение модели линейной регрессии
linear_model.fit(X_train, y_train)
```

Рисунок 14 – Обучение модели и проверка модели

```
# Модель линейной регрессии без указанных гиперпараметров
linear_model = LinearRegression()
```

Рисунок 15 – Гиперпараметры

```
Среднеквадратичная ошибка (Линейная регрессия) : 1.8925115525255056
R^2 Score (Линейная регрессия) : 0.4589586127781572
```

Рисунок 16 – Вывод результатов

3.3 Вывод

Линейная регрессия обычно используется для задач регрессии, где предсказывается непрерывное числовое значение.

Что касается кода, линейная регрессия создается и обучается, но стоит отметить, что в задаче классификации используется обычно логистическая регрессия, так как она работает лучше для предсказания вероятности принадлежности к классам.

Анализ результатов работы линейной регрессии в задаче классификации может быть затруднительным и неинформативным из-за того, что модель не подходит для таких задач. Как правило, в таких случаях для задач классификации используют модели, оптимизированные и предназначенные именно для этого, чтобы достичь лучшей производительности и интерпретируемости результатов. Поэтому у линейной регрессии я вывел среднеквадратичную ошибку и коэффициент детерминации. Результат представлен на рисунке

4 Модель логистической регрессии

4.1 Теория

Логистическая регрессия - это модель бинарной классификации, используемая для оценки вероятности принадлежности наблюдения к определенному классу. Несмотря на название, логистическая регрессия используется для задач классификации, а не регрессии.

Основные концепции:

Логистическая функция (сигмоид): Основой логистической регрессии является логистическая (или сигмоидальная) функция. Она преобразует линейную комбинацию входных признаков в вероятность принадлежности к классу 1.

Формула логистической функции:

$P(Y=1|X) = 1 / (1 + \exp(-(w0 + w1x1 + w2x2 + ... + wnxn)))$

где:

$P(Y=1|X)$ - вероятность того, что целевая переменная Y равна 1 при заданных значениях признаков X.

x1, x2, ..., xn - признаки (факторы).

w0, w1, ..., wn - коэффициенты модели (веса), которые определяют влияние каждого признака.

exp - экспоненциальная функция.

Функция потерь (Loss function): В логистической регрессии используется логарифмическая функция потерь (log loss), также известная как бинарная кросс-энтропия (binary cross-entropy). Она измеряет разницу между предсказанными и фактическими значениями и используется в процессе обучения модели для нахождения оптимальных весов.

Оценка параметров: Для оценки параметров модели, таких как веса w, используются методы оптимизации, например, метод градиентного спуска, который минимизирует функцию потерь.

Гиперпараметры:

Гиперпараметры - это настройки модели, которые не могут быть изучены самой моделью и должны быть определены до начала процесса

обучения. Для модели логистической регрессии заданы следующие гиперпараметры:

C: Параметр регуляризации (обратная сила регуляризации). Он контролирует влияние регуляризации на модель. Меньшие значения C указывают на более сильную регуляризацию. Увеличение значения C уменьшает силу регуляризации, что может привести к более точной подгонке модели под обучающие данные. Слишком большие значения C могут привести к переобучению.

penalty: Тип регуляризации, который определяет тип штрафа, применяемого к коэффициентам модели. Варианты: L1-регуляризация(Lasso) добавляет абсолютное значение весов признаков в функцию потерь. Она может привести к разреженности модели, уменьшая веса при некоторых признаках до нуля, что позволяет выполнить отбор признаков. L2- регуляризация (Ridge) добавляет квадраты весов признаков в функцию потерь. Она штрафует большие значения весов, но не обнуляет их.

solver: Это алгоритм, используемый для оптимизации функции потерь при поиске оптимальных весов модели. Варианты:

1) 'liblinear': Оптимизация основана на библиотеке LIBLINEAR. Этот solver подходит для небольших датасетов и поддерживает только 'l1' и 'l2' для регуляризации.

2) 'saga': Метод Stochastic Average Gradient. Это улучшенная версия solver'a 'sag', обычно эффективна для больших датасетов.

3) 'lbfgs': Limited-memory Broyden-Fletcher-Goldfarb-Shanno. Этот solver подходит для небольших и средних датасетов. Он использует приближенный метод второго порядка и может обрабатывать различные типы регуляризации.

Выбор solver'a зависит от размера датасета, типа регуляризации и предпочтений при работе с моделью. Например, 'liblinear' может быть предпочтительным для маленьких датасетов с L1- или L2-регуляризацией, в то время как 'lbfgs' может быть хорошим выбором для средних по размеру датасетов. 'saga' часто рекомендуется для больших наборов данных.

4.2 Задачи

Задача 1. Провести классификацию найденного датасета, методом логистической регрессий. В формате Markdown написать пояснения. Объяснить почему были выбраны именно такие гиперпараметры, была ли перекрестная проверка, и т.д.

Ход работы:

В модели используем следующие гиперпараметры:

```
# Определение параметров для перебора
parameters = {
    'penalty': ['l1', 'l2'],
    'C': [0.01, 0.1, 1.0],
    'solver': ['liblinear', 'lbfgs'],
}
```

Рисунок 17 – Гиперпараметры

На рисунках 18-26 представлен код программы.

```
# Преобразование категориальных признаков в числовые
label_encoder = LabelEncoder()
for column in data.columns:
    if data[column].dtype == 'object':
        data[column] = label_encoder.fit_transform(data[column])

# Удаление ненужных столбцов или подготовка признаков и целевой переменной
X = data.drop(['Type'], axis=1) # Признаки
y = data['Type']  # Целевая переменная
```

Рисунок 18 – Разделение данных и нормализация

```
# Создание модели логистической регрессии
logistic_model = LogisticRegression()

# Подбор лучших параметров с помощью GridSearchCV
grid_search = GridSearchCV(logistic_model, parameters, scoring='accuracy', cv=5)
grid_search.fit(X_train, y_train)
```

Рисунок 19 – Разделение данных и нормализация

```
print("Лучшие параметры:", best_params)
print("Лучшая точность:", best_accuracy)
classification_rep = classification_report(y_test, best_logistic_predictions)
```

Рисунок 20 – Проверка модели и вывод результатов

В ходе обучения модели были выявлены лучшие гиперпараметры. На рисунке 21 представлены лучшие гиперпараметры, точность модели на

реальных данных и отчёт по классификации.

```
Лучшие параметры: {'C': 1.0, 'penalty': 'l1', 'solver': 'liblinear'}
```

	precision	recall	f1-score	support
0	0.55	0.62	0.59	867
1	0.72	0.89	0.79	517
2	0.44	0.43	0.44	683
3	0.54	0.56	0.55	817
4	0.36	0.09	0.15	506
5	0.80	0.88	0.84	1576
6	0.00	0.00	0.00	15
accuracy			0.64	4981
macro avg	0.49	0.50	0.48	4981
weighted avg	0.61	0.64	0.62	4981

Рисунок 21 – Результаты

Модель демонстрирует хорошую (accuracy) в 64%, что означает, что она правильно классифицирует 64% образцов в наборе данных. Стоит отметить, что предсказание классов распределены раномерно.

4.3 Вывод

Преимущества модели логистической регрессии:

- Простота и интерпретируемость: Модель логистической регрессии - это простая и легко интерпретируемая линейная модель. Ее результаты могут быть легко объяснены, что делает ее полезной для понимания влияния различных признаков на прогнозы.

- Эффективность при линейно разделимых данных: Когда классы линейно разделимы, логистическая регрессия может демонстрировать хорошую производительность.

- Малая потребность в вычислительных ресурсах: Модель требует относительно небольших вычислительных ресурсов для обучения и предсказаний, что делает ее эффективной для применения на больших датасетах.

- Устойчивость к переобучению: При правильной настройке гиперпараметров модель логистической регрессии обычно обладает хорошей устойчивостью к переобучению на обучающих данных.

Недостатки модели логистической регрессии:

- Ограничение линейности: Модель логистической регрессии предполагает линейную разделимость между классами, что делает ее менее эффективной в задачах с нелинейными зависимостями.

- Чувствительность к выбросам: Логистическая регрессия может быть чувствительной к выбросам в данных, что может привести к искажению результатов.

- Неспособность улавливать сложные взаимосвязи между признаками: Признаки с сложными нелинейными взаимосвязями могут быть плохо предсказаны моделью логистической регрессии.

- Неэффективность при низкой линейной разделимости классов: В случае, если классы плохо разделимы линейно, модель может демонстрировать низкую точность.

5 Модель наивного Байеса

5.1 Теория

Теорема Байеса позволяет вычислить условные вероятности и предсказать вероятность принадлежности объекта к определенному классу на основе известных признаков.

Основные концепции:

Теорема Байеса: Теорема Байеса позволяет вычислить условные вероятности и предсказать вероятность принадлежности объекта к определенному классу на основе известных признаков.

Формула теоремы Байеса:

P(C|X) = (P(X|C) * P(C)) / P(X)

где:

P(C|X) - вероятность того, что объект принадлежит классу C при условии, что известны его признаки X.

P(X|C) - вероятность признаков X при условии принадлежности объекта классу C.

P(C) - априорная вероятность класса C.

P(X) - вероятность признаков X.

Наивное предположение о независимости признаков:

Модель наивного Байеса предполагает, что все признаки являются независимыми между собой при условии принадлежности к определенному классу. Это "наивное" предположение позволяет снизить вычислительную сложность модели.

Типы наивных Байесовских моделей:

- Мультиномиальный наивный Байес (Multinomial Naive Bayes): Подходит для признаков с дискретными или счетными значениями, такими как частота слов в тексте.

- Бернуллиев наивный Байес (Bernoulli Naive Bayes): Применяется к бинарным признакам, где каждый признак представляет собой булево значение.

- Гауссов наивный Байес (Gaussian Naive Bayes): Предполагает, что непрерывные признаки распределены по нормальному (гауссовому) закону.

Применение модели:

Модель наивного Байеса широко используется в задачах классификации текстов, фильтрации спама, анализе тональности текста и других областях, где требуется быстрая и эффективная классификация на основе небольшого объема данных.

Выбор конкретной вариации модели зависит от типа данных и характеристик признаков в задаче классификации.

5.2 Задачи

Задача 1. Провести классификацию найденного датасета, методами наивного Байеса. В формате Markdown написать пояснения. Объяснить почему были выбраны именно такие гиперпараметры, была ли перекрестная проверка, и т.д.

Ход работы:

Модель наивного Байеса:

В модели нет гиперпараметров.

На рисунках 22-24 представлен остальной код программы.

```
# Разделение данных на признаки и целевую переменную
X = data.drop('Type', axis=1)
y = data['Type']
# Преобразование категориальных признаков в числовые
label_encoder = LabelEncoder()
for column in data.columns:
    if data[column].dtype == 'object':
        data[column] = label_encoder.fit_transform(data[column])
```

Рисунок 22 – Разделение данных и нормализация

```
# Инициализация модели наивного Байеса
nb = GaussianNB()

# Обучение модели на обучающем наборе данных
nb.fit(X_train, y_train)

# Предсказание на тестовых данных
predictions = nb.predict(X_test)
```

Рисунок 23 – Обучение модели и проверка модели

```
# Оценка качества модели на тестовом наборе
accuracy = accuracy_score(y_test, predictions)
print(f'Точность: {accuracy}')

# Вывод отчета о классификации
print(classification_report(y_test, predictions))
```

Рисунок 24 – Вывод результатов

```
Точность: 0.5207789600481831
              precision    recall  f1-score   support

           0       0.73      0.10      0.17       867
           1       0.46      0.94      0.62       517
           2       0.30      0.55      0.39       683
           3       0.68      0.40      0.51       817
           4       0.28      0.39      0.32       506
           5       0.98      0.70      0.82      1576
           6       0.06      0.93      0.12        15

    accuracy                           0.52      4981
   macro avg       0.50      0.57      0.42      4981
weighted avg       0.67      0.52      0.52      4981
```

Рисунок 25 - Результаты

5.3 Вывод

Исходя из представленного отчета по классификации и точности модели наивного Байеса, можно сделать следующие выводы:

Модель наивного Байеса демонстрирует саммую среднию 52%. Анализируя отчет по классификации, можно выделить следующие аспекты:

Модель наивного Байеса демонстрирует относительно приемлемую производительность, но имеет проблемы в точности классификации обеих категорий.

6 Модель решающего дерева

6.1 Теория

Модель решающего дерева:

Модель решающего дерева - это деревообразная структура, которая представляет собой последовательность правил принятия решений, каждое из которых разбивает данные на более чистые подгруппы. Эта модель применяется как в задачах классификации, так и в задачах регрессии.

Основные концепции:

Разделение по признакам: Решающее дерево основывается на разделении набора данных на более мелкие подгруппы, выбирая оптимальное разделение по признакам. Цель состоит в минимизации неопределенности или увеличении чистоты (преимущественно увеличение одного класса или уменьшение дисперсии в случае регрессии) в каждой подгруппе.

Узлы и листья:

В решающем дереве каждый узел представляет собой условие на признаке, а каждый лист - прогноз или класс. Алгоритм построения дерева стремится минимизировать ошибку прогнозирования, разделяя данные на чистые подгруппы до определенной глубины или до выполнения критерия останова.

Принцип работы:

Дерево строится путем выбора признака, который лучше всего разделяет данные на основе определенного критерия (например, индекс Джини или энтропия для классификации, среднеквадратичная ошибка для регрессии). Процесс построения дерева продолжается рекурсивно для каждого полученного узла, пока не будет выполнен критерий останова.

Гиперпараметры:

max_depth: Максимальная глубина дерева. Этот параметр контролирует количество уровней в дереве.

min_samples_split: Минимальное количество выборок, необходимое для разделения узла. Если количество выборок в узле меньше этого значения, разделение прекращается.

min_samples_leaf: Минимальное количество выборок, необходимое для формирования листа. Этот параметр определяет критерий останова построения дерева.

Гиперпараметры позволяют настроить процесс построения дерева для достижения лучшей производительности и предотвращения переобучения или недообучения модели. Варьирование этих параметров позволяет найти оптимальное дерево для конкретного набора данных.

6.2 Задачи

Задача 1. Провести классификацию найденного датасета, методом решающего дерева. В формате Markdown написать пояснения. Объяснить почему были выбраны именно такие гиперпараметры, была ли перекрестная проверка, и т.д.

Ход работы:

В модели используем следующие гиперпараметры:

```
# Настройка гиперпараметров для решающего дерева
params_dt = {
    'max_depth': [3, 5, 7, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

Рисунок 26 – Гиперпараметры

На рисунках 27-29 представлен код программы.

```
# Выборка признаков и целевой переменной
features = ['Popularity', 'Favorites', 'Members', 'Episodes']
target = 'Type'

X = data[features]
y = data[target]

# Преобразование категориальных признаков в числовые значения
label_encoders = {}
for column in X.select_dtypes(include=['object']).columns:
    label_encoders[column] = LabelEncoder()
    X.loc[:, column] = label_encoders[column].fit_transform(X.loc[:, column])

# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Рисунок 27 – Разделение данных и нормализация

```
decision_tree = DecisionTreeClassifier(random_state=42)
grid_search_dt = GridSearchCV(decision_tree, params_dt, cv=5)
grid_search_dt.fit(X_train, y_train)

best_decision_tree = grid_search_dt.best_estimator_
predictions_dt_tuned = best_decision_tree.predict(X_test)
accuracy_dt_tuned = accuracy_score(y_test, predictions_dt_tuned)
```

Рисунок 28 – Обучение модели

```
decision_tree = DecisionTreeClassifier(random_state=42)
grid_search_dt = GridSearchCV(decision_tree, params_dt, cv=5)
grid_search_dt.fit(X_train, y_train)

best_decision_tree = grid_search_dt.best_estimator_
predictions_dt_tuned = best_decision_tree.predict(X_test)
accuracy_dt_tuned = accuracy_score(y_test, predictions_dt_tuned)

print("Лучшие параметры для решающего дерева:", grid_search_dt.best_params_)
print("Точность настроенных параметров решающего дерева:", accuracy_dt_tuned)
print("Classification Report of Tuned Decision Tree:")
print(classification_report(y_test, predictions_dt_tuned))
```

Рисунок 29 – Проверка модели и вывод результатов

В ходе обучения модели были выявлены лучшие гиперпараметры. На рисунке 39 представлены лучшие гиперпараметры, точность модели на реальных данных и отчёт по классификации.

```
Лушчие параметры для случайного леса: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 100}
Точность настроенных параметров случайного леса: 0.5645452720337282
Classification Report of Tuned Random Forest:
              precision    recall  f1-score   support

       Movie       0.45      0.49      0.47       867
       Music       0.35      0.36      0.35       517
         ONA       0.38      0.28      0.32       683
         OVA       0.52      0.54      0.53       817
     Special       0.35      0.28      0.31       506
          TV       0.82      0.90      0.86      1576
     UNKNOWN       0.33      0.07      0.11        15

    accuracy                           0.56      4981
   macro avg       0.46      0.42      0.42      4981
weighted avg       0.55      0.56      0.55      4981
```

Рисунок 30 – Результаты

Модель решающего дерева демонстрирует точность на уровне 56% на тестовом наборе данных.

Анализ отчета по классификации:

Precision средний означает что модель примерно одинкова оперделяет для всех классов,recall также значения усреднённые.

6.3 Вывод

Несмотря на высокую точность модели решающего дерева в определении класса недостаточную точность и низкую полноту, что указывает на потенциальную неэффективность модели в выявлении истинно положительных случаев для этого класса.

Преимущества модели решающего дерева:

- Простота интерпретации: Решающие деревья могут быть легко визуализированы и интерпретированы человеком, что делает их полезными для принятия решений.

- Универсальность: Могут работать с разными типами данных (категориальными и числовыми), а также в задачах классификации и регрессии.

Недостатки модели решающего дерева:

- Склонность к переобучению: Могут строить слишком сложные деревья, которые переобучаются к обучающим данным.

- Чувствительность к изменениям в данных: Деревья могут сильно меняться при небольших изменениях в обучающих данных, что делает их нестабильными.

- Неустойчивость к шуму: Решающие деревья могут быть чувствительны к шуму и аномалиям в данных.

7 Модель случайного леса

7.1 Теория

Случайный лес - это ансамбль (ensemble) деревьев решений, который использует метод "усреднения прогнозов" для улучшения точности и уменьшения переобучения. Он состоит из большого количества деревьев, где каждое дерево строится на основе подвыборки данных и подмножества признаков.

Основные концепции:

Бутстрэп выборка: Случайный лес использует бутстрэп выборку (подвыборки с возвращением) из обучающего набора данных для построения различных деревьев в ансамбле.

Случайный выбор признаков: Каждое дерево строится на основе случайного подмножества признаков. Это позволяет модели быть более разнообразной и уменьшает корреляцию между деревьями, что способствует улучшению обобщающей способности модели.

Процесс усреднения: Предсказания каждого дерева усредняются для получения окончательного прогноза модели. В задачах классификации используется голосование большинства, а в задачах регрессии - усреднение.

Гиперпараметры модели случайного леса:

n_estimators: Количество деревьев в случайном лесу.

max_depth: Максимальная глубина дерева в случайном лесу. Этот параметр контролирует количество уровней в каждом дереве.

min_samples_split: Минимальное количество выборок, необходимое для разделения узла в дереве.

min_samples_leaf: Минимальное количество выборок, необходимое для формирования листа в дереве.

Гиперпараметры позволяют настраивать процесс построения деревьев в случайном лесу, чтобы достичь оптимальной производительности модели.

Использование различных комбинаций значений гиперпараметров позволяет найти наилучшую модель с учетом специфики данных и задачи.

7.2 Задачи

Задача 1. Провести классификацию найденного датасета, методом решающего дерева. В формате Markdown написать пояснения. Объяснить почему были выбраны именно такие гиперпараметры, была ли перекрестная проверка, и т.д.

Ход работы:

В модели используем следующие гиперпараметры:

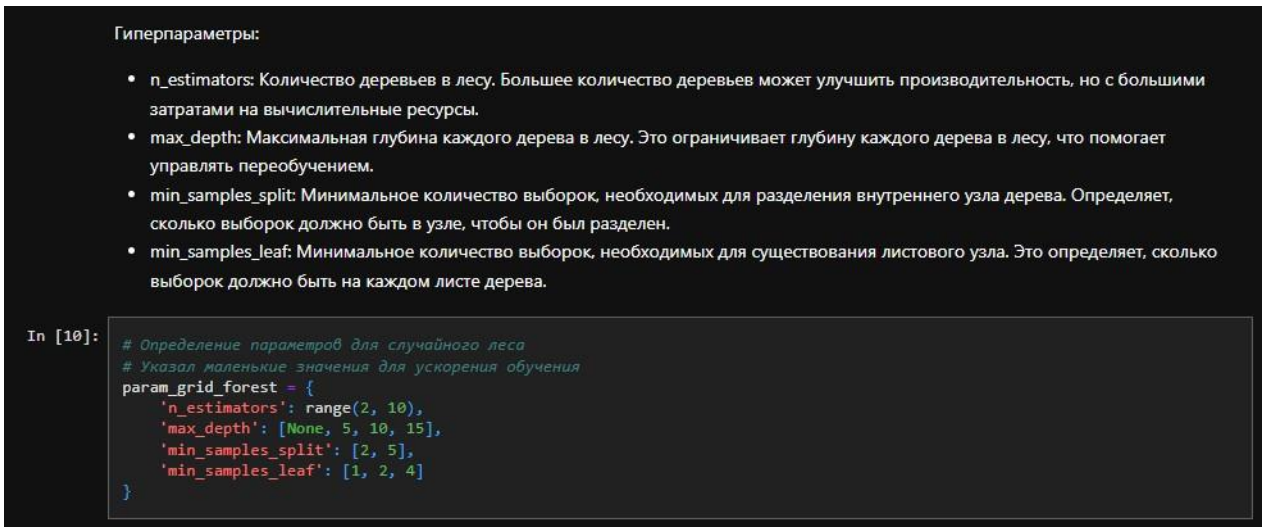

Рисунок 31 – Гиперпараметры

На рисунках 32-34 представлен код программы.

```
# Выборка признаков и целевой переменной
features = ['Popularity', 'Favorites', 'Members', 'Episodes']
target = 'Type'

X = data[features]
y = data[target]

# Преобразование категориальных признаков в числовые значения
label_encoders = {}
for column in X.select_dtypes(include=['object']).columns:
    label_encoders[column] = LabelEncoder()
    X.loc[:, column] = label_encoders[column].fit_transform(X.loc[:, column])

# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Рисунок 32 – Разделение данных и нормализация

```
random_forest = RandomForestClassifier(random_state=42)
grid_search_rf = GridSearchCV(random_forest, params_rf, cv=5)
grid_search_rf.fit(X_train, y_train)

best_random_forest = grid_search_rf.best_estimator_
predictions_rf_tuned = best_random_forest.predict(X_test)
accuracy_rf_tuned = accuracy_score(y_test, predictions_rf_tuned)
```

Рисунок 33 – Обучение модели

```
print("\nЛушчие параметры для случайного леса:", grid_search_rf.best_params_)
print("Точность настроенных параметров случайного леса:", accuracy_rf_tuned)
print("Classification Report of Tuned Random Forest:")
print(classification_report(y_test, predictions_rf_tuned))
```

Рисунок 34 – Проверка модели и вывод результатов

В ходе обучения модели были выявлены лучшие гиперпараметры. На рисунке 46 представлены лучшие гиперпараметры, точность модели на реальных данных и отчёт по классификации.

```
Лушчие параметры для случайного леса: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 100}
Точность настроенных параметров случайного леса: 0.5645452720337282
Classification Report of Tuned Random Forest:
              precision    recall  f1-score   support

       Movie       0.45      0.49      0.47       867
       Music       0.35      0.36      0.35       517
         ONA       0.38      0.28      0.32       683
         OVA       0.52      0.54      0.53       817
     Special       0.35      0.28      0.31       506
          TV       0.82      0.90      0.86      1576
     UNKNOWN       0.33      0.07      0.11        15

    accuracy                           0.56      4981
   macro avg       0.46      0.42      0.42      4981
weighted avg       0.55      0.56      0.55      4981
```

Рисунок 35 – Результаты

7.3 Вывод

Модель случайного леса с параметрами демонстрирует точность на уровне 56% на тестовом наборе данных.

Анализ отчета по классификации показывает: Модель среднюю точность , но более хорошей полнотой по всесм классам в среднем.

Преимущества модели случайного леса:

- Высокая точность: Обычно обладает высокой точностью предсказаний благодаря агрегированию множества деревьев.

- Устойчивость к переобучению: Благодаря случайному выбору подмножества данных и признаков, случайный лес обладает лучшей устойчивостью к переобучению по сравнению с одиночными деревьями.

- Способность к обработке больших объемов данных: Эффективен для работы с большими наборами данных и большим количеством признаков без предварительной обработки.

- Масштабируемость: Может эффективно работать с большим количеством деревьев, что делает его масштабируемым для параллельной обработки и ускорения обучения.

- Может оценивать важность признаков: Позволяет оценивать важность каждого признака в предсказании целевой переменной.

Недостатки модели случайного леса:

- Склонность к переобучению при неблагоприятном соотношении числа деревьев и количества объектов обучения.

- Требуется настройка гиперпараметров: Несмотря на устойчивость к переобучению, требуется настройка гиперпараметров для достижения лучшей производительности модели.

- Сложность интерпретации: При большом количестве деревьев интерпретация модели может быть затруднена из-за большого объема информации.

- Неэффективность на разреженных данных: Модель может показать неоптимальную производительность на разреженных данных (например, текстовых данных), требуя более сложной предварительной обработки.

8 Модель CatBoost

8.1 Теория

CatBoost (Categorical Boosting) - это метод градиентного бустинга, который автоматически обрабатывает категориальные признаки без необходимости их предварительного кодирования или обработки. Он является мощным алгоритмом машинного обучения для задач классификации, регрессии и ранжирования.

Основные концепции:

Обработка категориальных признаков: CatBoost проводит автоматическую обработку категориальных признаков, что позволяет использовать их напрямую в моделировании без необходимости предварительного кодирования.

Структура градиентного бустинга: Это итеративный алгоритм, который комбинирует множество слабых моделей (обычно деревьев решений), улучшая каждую новую модель путем корректировки предыдущих ошибок.

Регуляризация: CatBoost имеет встроенную регуляризацию для предотвращения переобучения модели. Параметры регуляризации, такие как l2_leaf_reg, помогают контролировать сложность модели.

Гиперпараметры:

depth: Максимальная глубина деревьев.

learning_rate: Скорость обучения, контролирующая величину шага при обновлении весов модели.

l2_leaf_reg: Коэффициент регуляризации L2 для весов листьев деревьев.

iterations: Количество итераций (деревьев), выполняемых в процессе обучения.

loss_function: Функция потерь для оптимизации.

Гибкость и удобство в работе с категориальными признаками, автоматическая обработка данных и регуляризация делают CatBoost

популярным инструментом для работы с данными, где присутствуют категориальные признаки.Выбор оптимальных гиперпараметров играет важную роль в повышении производительности модели.

8.2 Задачи

Задача 1. Провести классификацию найденного датасета, методом решающего дерева. В формате Markdown написать пояснения. Объяснить почему были выбраны именно такие гиперпараметры, была ли перекрестная проверка, и т.д.

Ход работы:

В модели используем следующие гиперпараметры:

```
# Определение параметров для CatBoost
param_grid_catboost = {
    'depth': [1, 4, 7, 10],
    'learning_rate': [0.01, 0.1, 1],
    'l2_leaf_reg': [1, 3, 5, 9],
    'iterations': [100, 200,],
    'depth': [0, 3, 6],
    'loss_function': ['MultiClass', 'Logloss']
}
```

Рисунок 36 – Гиперпараметры

На рисунках 37-39 представлен код программы.

```
# Создание модели CatBoost
catboost = CatBoostClassifier()

# Подбор оптимальных параметров с помощью перекрестной проверки для CatBoost
grid_search_catboost = GridSearchCV(catboost, param_grid_catboost, refit=True, verbose=3, cv=5)
grid_search_catboost.fit(X_train_scaled, y_train)

# Получение лучших параметров для CatBoost
best_params_catboost = grid_search_catboost.best_params_
best_score_catboost = grid_search_catboost.best_score_
```

Рисунок 37 – Разделение данных и нормализация

```python
test_score = grid_search_catboost.score(X_test_scaled, y_test)
print("Лучшие параметры для CatBoost:", best_params_catboost)
print("Лучший результат (точность) для CatBoost:", best_score_catboost)
print("Точность на тестовом наборе данных:", test_score)

best_model = grid_search_catboost.best_estimator_
predictions = best_model.predict(X_test_scaled)

# Отчет по классификации
print("\nОтчет по классификации:")
print(classification_report(y_test, predictions))
```

Рисунок 39 – Проверка модели и вывод результатов

В ходе обучения модели были выявлены лучшие гиперпараметры. На рисунке 40 представлены лучшие гиперпараметры, точность модели на реальных данных и отчёт по классификации.

```
Лучшие параметры для CatBoost: {'depth': 6, 'iterations': 100, 'l2_leaf_reg': 9, 'learning_rate': 1, 'loss_function': 'MultiCla
ss'}
Лучший результат (точность) для CatBoost: 0.9625
Точность на тестовом наборе данных: 0.95

Отчет по классификации:
              precision    recall  f1-score   support

       Movie       0.74      1.00      0.85        14
         OVA       1.00      0.82      0.90        17
     Special       0.00      0.00      0.00         2
          TV       1.00      1.00      1.00        67

    accuracy                           0.95       100
   macro avg       0.68      0.71      0.69       100
weighted avg       0.94      0.95      0.94       100
```

Рисунок 40 – Результаты

8.3 Вывод

Модель CatBoost продемонстрировала точность на уровне 95% на тестовом наборе данных, что является достаточно высоким.

Анализ отчета по классификации показывает:

Модель имеет высокую точность (0.95).

Преимущества модели CatBoost:

- Автоматическая обработка категориальных признаков: CatBoost автоматически обрабатывает категориальные данные, что позволяет использовать их напрямую без предварительного кодирования.

- Устойчивость к переобучению: Модель обладает встроенной регуляризацией и способна эффективно управлять сложностью модели, снижая вероятность переобучения.

- Высокая производительность на больших датасетах: CatBoost хорошо масштабируется и демонстрирует высокую производительность на больших наборах данных.

- Высокая точность предсказаний: Модель часто обеспечивает высокую точность в задачах классификации и регрессии благодаря эффективному использованию градиентного бустинга.

Недостатки модели CatBoost:

- Высокое время обучения: Из-за сложности модели и большого количества итераций обучение CatBoost может занимать больше времени по сравнению с некоторыми другими алгоритмами.

- Сложность интерпретации: При большом количестве деревьев и автоматической обработке категориальных данных интерпретация модели может быть затруднительной.

- Требует настройки гиперпараметров: Для достижения наилучшей производительности модели может потребоваться тщательная настройка гиперпараметров.

Заключение

Результаты анализа различных моделей машинного обучения выявили сильные и слабые стороны каждого подхода. Модель K-ближайших соседей демонстрирует хорошую общую точность и для моего дата сета она показал лучший результат по всем параметрам и всем классам.

Метод опорных векторов (SVM) обладает хорошей способностью предсказывать классы, но в то же время чувствителен к выбору гиперпараметров, что требует дополнительной тщательной настройки для достижения оптимальной производительности.

Логистическая регрессия демонстрирует высокую точность, но ограничена в моделировании нелинейных связей в данных.

Наивный Байесовский классификатор имеет неплохую точность, однако для достижения лучших результатов требует предварительной обработки данных.

Модель решающего дерева обладает хорошей интерпретируемостью, но может страдать от переобучения на некоторых типах данных.

Случайный лес показывает высокую точность, но подвержен переобучению и требует тщательной настройки гиперпараметров.

CatBoost в моём наборе данных не показал высокую точность, но выделяется своей способностью автоматической обработки категориальных данных и эффективным решением задачи классификации для данного датасета.

Выводя общие итоги, исходя из проведенного анализа и результатов каждой модели, модель K-ближайших соседей является наилучшим выбором для данного набора данных благодаря своей высокой точности и успешному решению задачи классификации.

Список используемых источников

1. Кадурин, С., Грудинин, С., & Николаев, М. (2018). Deep Learning. Бином.

2. Лукьяненко, Д. (2019). Python и машинное обучение. БХВ-Петербург.

3. Трофимов, А. (2020). Машинное обучение для текстов. ДМК Пресс.

4. Воронцов, К. (2019). Математические методы обучения по прецедентам. Издательский дом "Физико-математическая литература".

5. Кантарович, А., & Гергель, В. (2017). Методы машинного обучения в задачах анализа данных. БХВ-Петербург.

6. Зенкевич, С. (2019). Машинное обучение на Python. Питер.

7. Попов, М. (2018). Глубокое обучение. Символ-Плюс.

8. Бахтеев, О., & Бурков, А. (2020). Основы машинного обучения. Учебное пособие. БХВ-Петербург.

9. Воронцов, К. В. (2014). Математические методы обучения по прецедентам. МФТИ.

10. Юдин, Д. (2018). Практический курс машинного обучения. Питер.