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 финансов***

Пояснительная записка

Шифр проекта **043-КП-09.03.04-№7-ПЗ**

Студента (ки) *Зубковской Анжелики Алексеевны*
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 2024

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 46 с., 48 рис., 3 источника.

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

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.

Оглавление

Введение

В современном информационном обществе важность и применение методов машинного обучения в различных областях знаний становятся все более значимыми. Расширение доступа к данным и возможность эффективно их анализировать и использовать в различных задачах делают машинное обучение ключевым инструментом для прогнозирования, классификации и обработки информации.

Цель данной курсовой работы - провести анализ и сравнительный обзор различных моделей машинного обучения на примере конкретного набора данных. Работа включает в себя исследование и оценку производительности таких моделей, как *k*-ближайших соседей, метода опорных векторов, логистической регрессии, наивного байесовского классификатора, модели решающего дерева, случайного леса и CatBoost, а также сопоставление их результатов.

Работа также включает в себя сравнительный анализ преимуществ и недостатков каждой модели, а также итоговый вывод о наиболее подходящей модели для конкретного датасета. Полученные результаты могут быть использованы в различных областях, требующих прогнозирования или классификации на основе имеющихся данных.

1 Модель *k*-ближайших соседей

Модель *k*-ближайших соседей (*k*-Nearest Neighbors, *k*-NN) представляет собой метод обучения, используемый для решения задач классификации и регрессии. Основная идея этого метода заключается в том, что объекты принимают класс (в задаче классификации) или значение (в задаче регрессии) на основе классов или значений их ближайших соседей в пространстве признаков.

Принцип работы:

Выбор числа соседей (*k*): Одним из важных параметров модели *k*-NN является количество ближайших соседей (*k*), которые используются для принятия решения. Значение K выбирается заранее и влияет на точность и обобщающую способность модели.

Определение расстояния: Для определения ближайших соседей необходимо выбрать метрику расстояния. Ниже описаны метрики расстояния:

- Евклидово расстояние, широко используемая метрика в пространстве признаков;

- Манхэттенское расстояние, вычисляющее сумму абсолютных различий между координатами точек;

- Расстояние Чебышева, определяющее максимальную разницу между координатами точек;

- Обобщенная метрика, включающая в себя Евклидово и Манхэттенское расстояния. Параметр p определяет степень Минковского (обычно p=2 для Евклидова и p=1 для Манхэттенского).

Классификация объекта: После определения K ближайших соседей для нового объекта его классификация осуществляется путем голосования большинства соседей. Новый объект относится к классу, который представлен наибольшим количеством соседей.

Применение модели *k*-NN:

- Рекомендательные системы, основанные на истории покупок или предпочтений.

- Классификация текстов или изображений.

- Анализ медицинских данных для определения паттернов или диагнозов.

- Прогнозирование финансовых показателей и др.

Использование модели K-ближайших соседей зависит от характеристик данных, требований к производительности и конкретной задачи, для которой применяется модель.

Это представляет более подробный обзор теории модели *k*-ближайших соседей, который может быть расширен в соответствии с требованиями вашей курсовой работы.

Задача 1-2. 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). 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 19 столбцами и 537 667 строк. Датасет представлен на рисунках 1-3.

Out[2]:

ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	NAME_EDUCATION_TYPE
5065438	F	Y	N	2+ children	270000.0	Secondary / secondary special
5142753	F	N	N	No children	81000.0	Secondary / secondary special
5111146	M	Y	Y	No children	270000.0	Higher education
5010310	F	Y	Y	1 children	112500.0	Secondary / secondary special
5010835	M	Y	Y	2+ children	139500.0	Secondary / secondary special
...	...	...	...	...	...	...
5142999	M	Y	N	1 children	166500.0	Secondary / secondary special
5010773	F	N	Y	No children	135000.0	Higher education
5105601	M	N	Y	No children	180000.0	Higher education
5132833	M	Y	N	No children	220500.0	Secondary / secondary special
5135381	F	N	Y	No children	387000.0	Secondary / secondary special

Рисунок 1 – Датасет (часть 1)

Out[2]:

NAME_FAMILY_STATUS	NAME_HOUSING_TYPE	DAYS_BIRTH	DAYS_EMPLOYED	FLAG_MOBIL	FLAG_WORK_PHONE	FLAG_PHONE
Married	With parents	-13258	-2300	1	0	0
Single / not married	House / apartment	-17876	-377	1	1	1
Married	House / apartment	-19579	-1028	1	0	1
Married	House / apartment	-15109	-1956	1	0	0
Married	House / apartment	-17281	-5578	1	1	0
...	...	...	...	...	...	...
Married	With parents	-12372	-5401	1	0	1
Married	With parents	-14160	-4635	1	0	0
Married	House / apartment	-24204	-2462	1	0	0
Married	House / apartment	-22647	-3847	1	0	1
Married	House / apartment	-20082	-4979	1	0	0

Рисунок 2 – Датасет (часть 2)

FLAG_EMAIL	JOB	BEGIN_MONTHS	STATUS	TARGET
0	Managers	-6	C	0
0	Private service staff	-4	0	0
0	Laborers	0	C	0
0	Core staff	-3	0	0
0	Drivers	-29	0	0
...	...	...	...	...
0	Core staff	-8	0	0
0	Sales staff	-8	0	0
0	Private service staff	-7	0	0
0	Laborers	-1	0	0
0	Sales staff	-45	C	0

Рисунок 3 – Датасет (часть 3)

Определение столбцов:

ID - Номер клиента

CODE_GENDER - Пол клиента (F/M)

FLAG_OWN_CAR - Наличие автомобиля (Y/N)

FLAG_OWN_REALTY - Наличие недвижимости (Y/N)

CNT_CHILDREN - Количество детей (No children/1 children/..)

AMT_INCOME_TOTAL - Годовой доход

NAME_EDUCATION_TYPE – Уровень образовани ('Secondary / secondary special'/Higher education/..)

NAME_FAMILY_STATUS - Семейное положение (Married/'Single / not married'/..)

NAME_HOUSING_TYPE - Тип проживания ('House / apartment'/With parents/..)

DAYS_BIRTH - Возраст в днях

DAYS_EMPLOYED - Продолжительность работы в днях

FLAG_MOBIL - Наличие мобильного телефона (0/1)

FLAG_WORK_PHONE - Наличие рабочего телефона (0/1)

FLAG_PHONE - Наличие телефона (0/1)

FLAG_EMAIL - Наличие почты (0/1)

JOB - Работа

BEGIN_MONTHS - Рекордный месяц (Месяц для извлеченных данных является отправной точкой в обратном направлении, 0 - текущий месяц, -1 - предыдущий месяц и так далее)

STATUS - Статус (0: просрочка на 1-29 дней 1: просрочка на 30-59 дней 2: просрочка на 60-89 дней 3: просрочка на 90-119 дней 4: просрочка на 120-149 дней 5: Просроченные или безнадежные долги, списания более чем на 150 дней C: погашен в этом месяце X: Нет кредита в течение месяца)

TARGET - Целевой столбец (Пользователь с риском отмечен как "1", в противном случае - как "0")

Гиперпараметры, использующиеся в модели:

```
param_grid = {'n_neighbors': list(range(2, 11)), 'metric': ['euclidean', 'manhattan', 'chebyshev', 'minkowski']}
```

Рисунок 4 – Гиперпараметры

На рисунках 5-8 представлен код программы.

```
In [1]: from sklearn.model_selection import train_test_split, GridSearchCV #тренировочные тесты
        from sklearn.neighbors import KNeighborsClassifier #классификатор
        from sklearn.preprocessing import StandardScaler, LabelEncoder #категориальные переменные в числовой формат
        from sklearn.metrics import accuracy_score, classification_report #метрики
        import pandas as pd

        file = "credit_card_approval.csv"
        df = pd.read_csv(file)
        df.dropna(inplace=True)

In [2]: df
```

Рисунок 5 – Импорт библиотек и загрузка датасета

```
# Преобразование категориальной переменной в числовой формат
label_encoder = LabelEncoder()
df['FLAG_OWN_CAR'] = label_encoder.fit_transform(df['FLAG_OWN_CAR'])
df['FLAG_OWN_REALTY'] = label_encoder.fit_transform(df['FLAG_OWN_REALTY'])

df = df.sample(n=10000)

# Разделение на признаки (X) и целевую переменную (y)
X = df[['FLAG_OWN_CAR', 'FLAG_OWN_REALTY']]
y = df['TARGET']

# Разделение данных на тренировочный и тестовый наборы
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 6 – Преобразование данных, разделение данных и нормализация

```
param_grid = {'n_neighbors': list(range(2, 11)), 'metric': ['euclidean', 'manhattan', 'chebyshev', 'minkowski']}

# Создание модели k-ближайших соседей
knn = KNeighborsClassifier()
grid = GridSearchCV(knn, param_grid, refit=True, verbose=0, cv=5)

# Обучение модели
grid.fit(X_train_scaled, y_train)

best_params = grid.best_params_
best_knn = grid.best_estimator_
```

Рисунок 7 – Обучение модели

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

# 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, zero_division=1))
```

Рисунок 8 – 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': 'euclidean', 'n_neighbors': 10}
Точность: 0.8078333333333333

Отчет по классификации:
              precision    recall  f1-score   support

           0       0.83      0.95      0.89      4687
           1       0.64      0.29      0.39      1313

    accuracy                           0.81      6000
   macro avg       0.73      0.62      0.64      6000
weighted avg       0.78      0.81      0.78      6000
```

```
Лучшие параметры: {'metric': 'euclidean', 'n_neighbors': 2}
Точность: 0.9955

Отчет по классификации:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1991
           1       1.00      0.00      0.00         9

    accuracy                           1.00      2000
   macro avg       1.00      0.50      0.50      2000
weighted avg       1.00      1.00      0.99      2000
```

Рисунок 8 – Результаты

Интерпретация результата:

Для данного набора данных лучшие параметры метода ближайших соседей были установлены как метрика расстояния "euclidean" и количество соседей равное 2.

Для класса "0" (основной класс) точность, полнота и F1-мера составляют 100%, что указывает на то, что модель очень хорошо определяет объекты этого класса.

Для класса "1" (редкого класса) точность равна 100%, но полнота составляет всего 0%, что означает, что модель не обнаружила ни одного объекта этого класса.

Макро среднее равно 1.00 для класса 0 и 0.50 для класса 1. Значение 1.00 для класса 0 указывает на отличное качество классификации для этого класса.

Weighted avg также равен 1.00 для класса 0, что указывает на высокое качество классификации для этого класса, и близко к 0.99 для класса 1, что указывает на хорошее качество, учитывая меньшее количество представителей этого класса.

Модель достигает очень высокой общей точности (accuracy) в 99.55%, что указывает на то, что она в большинстве случаев правильно классифицирует объекты. Однако, стоит обратить внимание на следующие аспекты:

Дисбаланс классов: Наблюдается явный дисбаланс между классами 0 и 1. В то время как класс 0 имеет множественные представители, класс 1 представлен всего 9 объектами. Это может привести к искаженным результатам, особенно в контексте recall и F1-score для класса 1.

Проблема с классом 1: Модель показывает идеальную точность (precision) для класса 1, но низкие значения recall и F1-score. Это говорит о том, что модель правильно идентифицирует объекты класса 1, но упускает большое количество истинных положительных случаев, что является проблемой, особенно при рассмотрении приложений, где важна высокая полнота.

Макро и взвешенное среднее: Оба показателя позволяют понять общее качество модели без учета дисбаланса классов. В данном случае, макро и взвешенное среднее высоки, что указывает на хорошее общее качество классификации.

Вывод:

Итак, модель хорошо справляется с предсказанием объектов класса 0, но имеет заметные проблемы с классификацией объектов класса 1, что указывает на необходимость улучшения способности модели к выявлению объектов этого класса для повышения ее общей производительности и балансировки предсказаний между классами. Также это зависит от выбранного датасета, так

как в данном случае модели тяжело справится с предсказанием объектов класса 1 из-за их мизерного количества.

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;

2 Модель машины опорных векторов

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.

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.

Ход работы:

В модели используем следующие гиперпараметры:

Гиперпараметры:

C: Определяет силу регуляризации в модели SVM. gamma: Определяет влияние одного тренировочного примера на другие. kernel: Определяет тип использованного ядра.

Рисунок 10 – Гиперпараметры

На рисунках 11-14 представлен код программы.

In [1]:
```
from sklearn.model_selection import train_test_split, GridSearchCV #тренировочные тесты
from sklearn.model_selection import StratifiedKFold # Stratified K-Folds для стратифицированной кросс-валидации
from sklearn.svm import SVC #классификация методом опорных векторов
from sklearn.preprocessing import StandardScaler, LabelEncoder #категориальные переменные в числовой формат
from sklearn.metrics import accuracy_score, classification_report #метрики
import pandas as pd

file = "new_card_approval.csv"
df = pd.read_csv(file)
df.dropna(inplace=True)
```

In [2]:
```
df
```

Рисунок 11 – Импорт библиотек и загрузка датасета

```
# Инициализация StratifiedKFold для кросс-валидации с 3 фолдами.
stratified_kfold = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

# Преобразование категориальных переменных в числовой формат
label_encoder = LabelEncoder()
categorical_columns = ['CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STAT
for col in categorical_columns:
    df[col] = label_encoder.fit_transform(df[col])

df = df.sample(n=100)

# Разделение на признаки (X) и целевую переменную (y)
X = df[['CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
        'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'DAYS_BIRTH', 'DAYS_EMPLOYED']]
y = df['TARGET']

# Разделение данных на тренировочный и тестовый наборы
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 12 – Преобразование данных, разделение данных и нормализация

```
param_grid = {'C': [0.01, 0.1, 1], 'kernel': ['linear', 'rbf', 'sigmoid', 'poly'], 'gamma': [0.1, 1, 10]}
# Подбор гиперпараметров с помощью перекрестной проверки
grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=0)
grid.fit(X_train_scaled, y_train)

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

Рисунок 13 – Обучение 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 14 – Проверка модели и вывод результатов

В ходе обучения модели были выявлены лучшие гиперпараметры. На рисунке 15 представлены лучшие гиперпараметры, точность модели на реальных данных и отчёт по классификации.

```
Лучшие гиперпараметры: {'C': 0.01, 'gamma': 1, 'kernel': 'poly'}
Точность модели: 0.995
Отчет о классификации:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       197
           1       1.00      0.67      0.80         3

    accuracy                           0.99       200
   macro avg       1.00      0.83      0.90       200
weighted avg       1.00      0.99      0.99       200
```

Рисунок 15 – Результаты

Интерпретация результата:

Гиперпараметры модели указывают на то, что был выбран полиномиальный (poly) ядро для модели с параметрами C = 0.01 и gamma = 1. Эти параметры могут указывать на относительно низкую сложность модели.

Точность (accuracy) модели составляет 99.5%, что указывает на высокую общую точность классификации.

Precision (точность) для класса 0 составляет 99%, что означает, что 99% объектов, отнесенных к классу 0, действительно принадлежат этому классу.

Precision для класса 1 составляет 100%, но учитывая небольшое количество объектов класса 1, этот показатель может быть не совсем надежным.

Recall (полнота) для класса 0 также очень высок, составляет 100%, что указывает на то, что модель правильно идентифицировала все истинно принадлежащие классу 0 объекты.

Recall для класса 1 равен 67%, что может быть проблемой, особенно если важно обнаружить все объекты класса 1.

F1-score для класса 0 составляет 1.00, что указывает на отличное сбалансирование между точностью и полнотой для этого класса.

F1-score для класса 1 равен 0.80, что может указывать на некоторые проблемы в точности или полноте для этого класса.

Следует обратить внимание на следующее:

Для класса 1 присутствует низкая полнота (recall) в 67%, что означает, что модель пропускает значительное количество истинных положительных случаев для этого класса. Это может быть особенно проблематично, если класс 1 представляет собой важный класс в контексте задачи.

Для класса 1 также наблюдается низкий F1-score, что указывает на некоторые проблемы в балансе между точностью и полнотой для этого класса.

В целом, модель метода опорных векторов с указанными гиперпараметрами демонстрирует высокую общую точность, но требует улучшения в обнаружении объектов класса 1, особенно увеличения полноты этого класса.

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.

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 Модели линейной и логической регрессии

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 + ε,

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

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.

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.

Ход работы:

In [8]:
```
from sklearn.model_selection import train_test_split, GridSearchCV  # функций для разделения данных на тренировочный и тестовый н
from sklearn.linear_model import LogisticRegression  # модели логистической регрессии
from sklearn.linear_model import LinearRegression  # модели линейной регрессии
from sklearn.preprocessing import StandardScaler, LabelEncoder  # функции для масштабирования признаков и кодирования категориаль
from sklearn.metrics import accuracy_score, classification_report  # метрики: точность (accuracy), отчет о классификации
import pandas as pd

# Загрузка данных
file = "new_card_approval.csv"
df = pd.read_csv(file)
df.dropna(inplace=True)
```

In [2]:
```
df
```

Рисунок 16 – Импорт библиотек и загрузка датасета

Модель линейной регрессии:

В модели нет гиперпараметров.

Модель логической регрессии:

C, penalty, solver.

На рисунках 17-20 представлен код программы.

```
In [10]: # Преобразование категориальных переменных в числовой формат
label_encoder = LabelEncoder()
categorical_columns = ['CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STAT
for col in categorical_columns:
    df[col] = label_encoder.fit_transform(df[col])

df = df.sample(n=10000, replace=True)

# Разделение на признаки (X) и целевую переменную (y)
X = df[['CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
        'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'DAYS_BIRTH', 'DAYS_EMPLOYED']]
y = df['TARGET']

# 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 17 – Преобразование данных, разделение данных и нормализация

```
linear_model = LinearRegression()
linear_model.fit(X_train_scaled, y_train)
linear_accuracy = linear_model.score(X_test_scaled, y_test)
```

Рисунок 18 – Обучение модели и проверка модели (линейная регрессия)

```
param_grid = {
    'C': [0.01, 0.1, 1, 10],
    'penalty': ['l2'],
    'solver': ['liblinear', 'saga', 'lbfgs']
}
logistic = LogisticRegression(max_iter=1000)
grid = GridSearchCV(logistic, param_grid, refit=True, verbose=0)
grid.fit(X_train_scaled, y_train)
```

Рисунок 19 – Обучение модели и проверка модели (логическая регрессия)

```
# Оценка производительности модели логистической регрессии на тестовом наборе
best_logistic = grid.best_estimator_
logistic_accuracy = best_logistic.score(X_test_scaled, y_test)
print(f"Лучшие гиперпараметры логистической регрессии: {grid.best_params_}")
print(f"Точность модели логистической регрессии: {logistic_accuracy}")
print(f"Точность модели линейной регрессии: {linear_accuracy}")

# Предсказания на тестовой выборке с использованием лучшей модели
predictions = best_logistic.predict(X_test_scaled)
# Отчет по классификации
print("\nОтчет по классификации:")
print(classification_report(y_test, predictions))
```

Рисунок 20 – Проверка модели и вывод результатов

В ходе обучения модели были выявлены лучшие гиперпараметры. На рисунке 21 представлены лучшие гиперпараметры, точности моделей и отчёт по классификации.

```
Лучшие гиперпараметры логистической регрессии: {'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}
Точность модели логистической регрессии: 0.9745
Точность модели линейной регрессии: 0.0526870284697023

Отчет по классификации:
              precision    recall  f1-score   support

           0       0.97      1.00      0.99      1949
           1       0.00      0.00      0.00        51

    accuracy                           0.97      2000
   macro avg       0.49      0.50      0.49      2000
weighted avg       0.95      0.97      0.96      2000
```

Рисунок 21 - Результаты

Модель демонстрирует хорошую точность (accuracy) в 97%, что означает, что она правильно классифицирует 97% образцов в наборе данных. Однако стоит отметить, что при анализе отдельных классов видно неравномерное качество предсказания:

Класс 0 имеет высокие показатели precision (97%) и recall (100%), что свидетельствует о том, что модель хорошо определяет и точно классифицирует примеры этого класса.

В то время как класс 1 показывает значительно более низкую precision (0%) и recall (0%), что указывает на то, что модель более склонна к неправильной классификации примеров этого класса и пропускает значительную часть положительных случаев (не обнаруживает их в принципе).

F1-score для класса 0 составляет 0.99, что говорит о хорошем балансе между точностью и полнотой для этого класса. Однако для класса 1 F1-score равен 0.0, что означает, что модель показывает слабую способность одновременно обеспечивать как точность, так и полноту для этого класса.

Вывод:

Линейная регрессия является простым и интерпретируемым методом, который часто используется для оценки отношений между переменными, однако она может давать плохие результаты, если связь между признаками и целевой переменной нелинейна или если данные содержат выбросы или несбалансированные значения.

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.

4 Модель наивного Байеса

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.

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.

Ход работы:

In [1]:
```
from sklearn.model_selection import train_test_split, GridSearchCV  # функций для разделения данных на тренировочный и тестовый н
from sklearn.naive_bayes import GaussianNB # Байесовский классификатор (Гаусовское распределение)
from sklearn.preprocessing import StandardScaler, LabelEncoder # функции для масштабирования признаков и кодирования категориальн
from sklearn.metrics import accuracy_score, classification_report # метрики: точность (accuracy), отчет о классификации
import pandas as pd


# Загрузка данных
file = "new_card_approval.csv"
df = pd.read_csv(file)
df.dropna(inplace=True)
```

In [2]:
```
df
```

Рисунок 22 – Импорт библиотек и загрузка датасета

Модель не содержит гиперпараметров.

На рисунках 23-28 представлен остальной код программы.

```
# Преобразование категориальных переменных в числовой формат
label_encoder = LabelEncoder()
categorical_columns = ['CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STAT
for col in categorical_columns:
    df[col] = label_encoder.fit_transform(df[col])

df = df.sample(n=10000)

# Разделение на признаки (X) и целевую переменную (y)
X = df[['CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
        'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'DAYS_BIRTH', 'DAYS_EMPLOYED']]
y = df['TARGET']

# 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 23 – Преобразование данных, разделение данных и нормализация

```
# Гаусовское распределение (нет гиперпараметров => не нужна перекрёстная проверка)
naive_bayes = GaussianNB()
naive_bayes.fit(X_train_scaled, y_train)
```

Рисунок 24 – Обучение модели и проверка модели

```
# Оценка производительности модели на тестовом наборе
y_pred = naive_bayes.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
print(f"Точность модели наивного Байеса: {accuracy}")

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

Рисунок 25 – Вывод результатов

```
Точность модели наивного Байеса: 0.9675

Отчет по классификации:
              precision    recall  f1-score   support

           0       0.99      0.98      0.98      1942
           1       0.45      0.52      0.48        58

    accuracy                           0.97      2000
   macro avg       0.72      0.75      0.73      2000
weighted avg       0.97      0.97      0.97      2000
```

Рисунок 26 – Результаты

Вывод:

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

Модель наивного Байеса демонстрирует высокую точность на уровне 96,75%. Анализируя отчет по классификации, можно выделить следующие аспекты:

Для класса 0 модель имеет хорошую precision (точность) 0.99, что указывает на высокий процент правильно предсказанных объектов.

Для класса 1 модель показывает низкую точность 0.45 и полноту 0.52. Это может означать, что модель склонна классифицировать объекты как

принадлежащие к классу 1, но при этом их слишком мало по сравнению с объектами класса 0.

Исходя из этого, модель наивного Байеса демонстрирует относительно приемлемую производительность.

5 Модель решающего дерева и случайного леса

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.

Случайный лес:

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.

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.

Ход работы:

В модели **решающего дерева** используем следующие гиперпараметры:

```
param_grid_tree = {
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

Рисунок 27 – Гиперпараметры

На рисунках 28-31 представлен код программы.

In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV # функции для разделения данных на тренировочный и тестовый на
from sklearn.metrics import accuracy_score, classification_report # метрики: точность (accuracy), отчет о классификации
from sklearn.tree import DecisionTreeClassifier # классификатор дерева решений
from sklearn.ensemble import RandomForestClassifier # классификатор случайного леса
from sklearn.preprocessing import StandardScaler, LabelEncoder # функции для масштабирования признаков и кодирования категориальн

# Загрузка данных
file = "new_card_approval.csv"
df = pd.read_csv(file)
df.dropna(inplace=True)
```

In [2]:
```
df
```

Рисунок 28 – Импорт библиотек и загрузка датасета

```
# Преобразование категориальных переменных в числовой формат
label_encoder = LabelEncoder()
categorical_columns = ['CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STAT
for col in categorical_columns:
    df[col] = label_encoder.fit_transform(df[col])

df = df.sample(n=10000)

# Разделение на признаки (X) и целевую переменную (y)
X = df[['CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
        'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'DAYS_BIRTH', 'DAYS_EMPLOYED']]
y = df['TARGET']

# 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 29 – Преобразование данных, разделение данных и нормализация

```
# Создание модели решающего дерева
decision_tree = DecisionTreeClassifier()

# Поиск лучших параметров для решающего дерева
grid_search_tree = GridSearchCV(decision_tree, param_grid_tree, refit=True, verbose=0, cv=5)
grid_search_tree.fit(X_train_scaled, y_train)

# Лучшие параметры для решающего дерева
best_params_tree = grid_search_tree.best_params_
best_score_tree = grid_search_tree.best_score_

best_tree = DecisionTreeClassifier(**best_params_tree)
best_tree.fit(X_train_scaled, y_train)
```

Рисунок 30 – Обучение модели

```
accuracy_tree = best_tree.score(X_test_scaled, y_test)
print("Лучшие параметры для решающего дерева:", best_params_tree)
print("Точность решающего дерева на тестовом наборе:", accuracy_tree)

predictions = best_tree.predict(X_test_scaled)

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

Рисунок 31 – Проверка модели и вывод результатов

В ходе обучения модели были выявлены лучшие гиперпараметры. На рисунке 32 представлены лучшие гиперпараметры, точность модели на реальных данных и отчёт по классификации.

```
Лучшие параметры для решающего дерева: {'max_depth': 15, 'min_samples_leaf': 4, 'min_samples_split': 5}
Точность решающего дерева на тестовом наборе: 0.994

Отчет по классификации:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1948
           1       0.93      0.83      0.88        52

    accuracy                           0.99      2000
   macro avg       0.97      0.91      0.94      2000
weighted avg       0.99      0.99      0.99      2000
```

Рисунок 32 – Результаты

Модель решающего дерева с параметрами max_depth=15, min_samples_leaf=4 и min_samples_split=5 демонстрирует точность на уровне 99.4% на тестовом наборе данных.

Анализ отчета по классификации:

Для класса 0 модель показывает высокие значения precision (точность) 1.00 и recall (полнота) 1.00, что указывает на то, что она хорошо определяет и точно классифицирует объекты этого класса.

Однако, для класса 1 precision (точность) равна 0.93, а recall (полнота) составляет 0.83. Это означает, что модель менее точно определяет объекты класса 1.

Средневзвешенное значение F1-score составляет 0.99, что означает неплохой баланс между точностью и полнотой на общем наборе данных.

В модели **случайного леса** используем следующие гиперпараметры:

```
param_grid_forest = {
    'n_estimators': range(2, 10),
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2, 4]
}
```

Рисунок 33 – Гиперпараметры

На рисунках 34 и 35 представлен код программы.

```
# Создание модели случайного леса
random_forest = RandomForestClassifier()

# Поиск лучших параметров для случайного леса
grid_search_forest = GridSearchCV(random_forest, param_grid_forest, refit=True, verbose=0, cv=5)
grid_search_forest.fit(X_train_scaled, y_train)

# Лучшие параметры для случайного леса
best_params_forest = grid_search_forest.best_params_
best_score_forest = grid_search_forest.best_score_
best_forest = RandomForestClassifier(**best_params_forest)
best_forest.fit(X_train_scaled, y_train)
```

Рисунок 34 – Обучение модели

```
accuracy_forest = best_forest.score(X_test_scaled, y_test)
print("Лучшие параметры для случайного леса:", best_params_forest)
print("Точность случайного леса на тестовом наборе:", accuracy_forest)

predictions = best_forest.predict(X_test_scaled)

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

Рисунок 35 – Проверка модели и вывод результатов

В ходе обучения модели были выявлены лучшие гиперпараметры. На рисунке 36 представлены лучшие гиперпараметры, точность модели на реальных данных и отчёт по классификации.

```
Лучшие параметры для случайного леса: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 9}
Точность случайного леса на тестовом наборе: 0.9955

Отчет по классификации:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1943
           1       0.94      0.89      0.92        57

    accuracy                           1.00      2000
   macro avg       0.97      0.95      0.96      2000
weighted avg       1.00      1.00      1.00      2000
```

Рисунок 36 – Результаты

Модель случайного леса с параметрами max_depth=10, min_samples_leaf=1, min_samples_split=2 и n_estimators=9 демонстрирует точность на уровне 99,55% на тестовом наборе данных.

Анализ отчета по классификации показывает:

Модель обладает высокой точностью (1.00) и хорошей полнотой (1.00) в предсказании объектов класса 0. Для класса 1 точность (0.94) и полнота (0.89).

Вывод для модели решающего дерева:

Результаты указывают на потенциальную неэффективность модели в выявлении истинно отрицательных 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.

Вывод для модели случайного леса:

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.

6 Модель CatBoost

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.

Задача 7. 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 37 – Гиперпараметры

На рисунках 38-46 представлен код программы.

```
In [3]: import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV # функции для разделения данных на тренировочный и тестовый на
from sklearn.metrics import accuracy_score, classification_report # метрики: точность (accuracy), отчет о классификации
from catboost import CatBoostClassifier  # классификатор CatBoost
from sklearn.preprocessing import StandardScaler, LabelEncoder # функции для масштабирования признаков и кодирования категориальн

# Загрузка данных
file = "new_card_approval.csv"
df = pd.read_csv(file)
df.dropna(inplace=True)

In [4]: df
```

Рисунок 38 – Импорт библиотек и загрузка датасета

```
In [6]: # Преобразование категориальных переменных в числовой формат
label_encoder = LabelEncoder()
categorical_columns = ['CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STAT
for col in categorical_columns:
    df[col] = label_encoder.fit_transform(df[col])

df = df.sample(n=10000)

# Разделение на признаки (X) и целевую переменную (y)
X = df[['CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
        'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'DAYS_BIRTH', 'DAYS_EMPLOYED']]
y = df['TARGET']
# Разделение данных на тренировочный и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)

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

Рисунок 39 – Преобразование данных, разделение данных и нормализация

```
# Создание модели CatBoost
catboost = CatBoostClassifier(verbose=0)

# Подбор оптимальных параметров с помощью перекрестной проверки для CatBoost
grid_search_catboost = GridSearchCV(catboost, param_grid_catboost, refit=True, verbose=0, 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 40 – Обучение модели

Процесс обучения модели представлен на рисунке 41.

```
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 41 – Проверка модели и вывод результатов

В ходе обучения модели были выявлены лучшие гиперпараметры. На рисунке 42 представлены лучшие гиперпараметры, точность модели на реальных данных и отчёт по классификации.

```
Лучшие параметры для CatBoost: {'depth': 3, 'iterations': 100, 'l2_leaf_reg': 1, 'learning_rate': 0.1, 'loss_function': 'MultiC
lass'}
Лучший результат (точность) для CatBoost: 0.9964444444444445
Точность на тестовом наборе данных: 0.994

Отчет по классификации:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       976
           1       1.00      0.75      0.86        24

    accuracy                           0.99      1000
   macro avg       1.00      0.88      0.93      1000
weighted avg       0.99      0.99      0.99      1000
```

Рисунок 48 – Результаты

Вывод:

Модель CatBoost с параметрами depth=3, iterations=100, l2_leaf_reg=1, learning_rate=0.1 и loss_function=MultiClass продемонстрировала точность на уровне 99,6% на тестовом наборе данных.

Анализ отчета по классификации показывает:

Модель имеет высокую точность (0.99) и хорошую полноту (1.00) в предсказании объектов класса 0.

Для класса 1 точность (1.00) и полнота (0.75) остаются на относительно среднем уровне, что указывает на ограниченную способность модели определять истинно положительные случаи для этого класса.

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

Проведенная работа по анализу различных моделей машинного обучения представляет собой значимый этап в выборе оптимального алгоритма для конкретного набора данных. Рассмотрим основные выводы и рекомендации на основе результатов каждой модели:

Модель *k*-ближайших соседей: Обладает неплохой точностью, но не справляется с определением объектов класса 1. Неэффективна при работе с данными, требующими высокой интерпретируемости.

Метод опорных векторов (SVM): Проявляет хорошую способность в предсказании классов, но может быть чувствительным к выбору гиперпараметров.

Логистическая регрессия: Показывает сбалансированные результаты, но имеет ограниченные возможности в моделировании нелинейных зависимостей.

Наивный Байесовский классификатор: Демонстрирует неплохую точность, но требует предварительной обработки данных для достижения лучших результатов.

Модель решающего дерева: Обладает хорошей интерпретируемостью, но может склоняться к переобучению на некоторых данных.

Случайный лес: Обеспечивает высокую точность, но может быть склонен к переобучению и требует настройки гиперпараметров.

CatBoost: Показывает высокую точность и обладает способностью автоматически обрабатывать категориальные данные.

Исходя из проведенного анализа и результатов каждой модели, наилучшей моделью для данных этого конкретного случая может быть выбрана CatBoost, благодаря высокой точности, автоматической обработке категориальных признаков и способности эффективно справляться с задачей классификации для данного датасета.

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