

Министерство образования и науки РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Омский государственный технический университет»

Кафедра «Автоматизированные системы обработки информации и управления»

Курсовой проект
по дисциплине
«Динамические языки программирования»

Выполнил

Студент гр. ПИН-202

Сычев Д.С.

(подп., дата)

Проверил

Старший преподаватель каф. АСОИУ

Кабанов А.А.

(подп., дата)

Омск 2023

Реферат

ОТЧЕТ 28 с., 15 рис., 3 источника.

КУРСОВОЙ ПРОЕКТ, МАШИННОЕ ОБУЧЕНИЕ, PYTHON, CATBOOST

Цель курсовой работы — ознакомиться и приобрести базовые знания в области машинного обучения.

В данной работе выполнены:

1. метод k- ближайших соседей;
2. метод машины опорных векторов;
3. методы линейной и логистической регрессий;
4. метод наивного Байеса;
5. методы решающего дерева и случайного леса;
6. метод CatBoost.

СОДЕРЖАНИЕ

1. Метод к-ближайших соседей (k-NN)	5
1.1 Описание метода	5
1.2 Принцип работы	5
1.3 Применение на наборе данных	6
1.4 Преимущества и ограничения.....	11
2. Метод машины опорных векторов (SVM).....	12
2.1 Описание метода	12
2.2 Разделительная гиперплоскость и принцип работы	12
2.3 Применение на наборе данных	12
2.4 Преимущества и ограничения.....	13
3. Методы линейной и логистической регрессии	15
3.1 Описание методов	15
3.2 Принцип работы	16
3.3 Сравнительный анализ с другими методами.....	17
4. Метод наивного Байеса.....	18
4.1 Описание метода	18
4.2 Применение на наборе данных	18
4.3 Преимущества и ограничения.....	19
5.1 Описание метода	20
5.2 Применение на модельных данных	21
5.3 Сравнение с другими методами классификации	23
6. Метод CatBoost.....	24
6.1 Описание метода	24

6.2 Применение на модельных данных.....	25
Заключение	27
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	29

Введение

Машинное обучение — одна из наиболее динамично развивающихся областей, которая позволяет компьютерам обучаться на основе данных и делать прогнозы или принимать решения без явного программирования.

Цель настоящей курсовой работы заключается в ознакомлении с основами машинного обучения и рассмотрении различных методов классификации. В ходе исследования были проведены эксперименты с различными алгоритмами, такими как метод k -ближайших соседей, машина опорных векторов, линейная и логистическая регрессии, наивный Байес, а также решающее дерево и случайный лес.

В рамках данной работы представлен обзор и сравнение указанных методов классификации на простых модельных данных. Каждый из методов исследован с целью понимания их принципов работы, преимуществ и ограничений.

1. Метод k -ближайших соседей (k -NN)

1.1 Описание метода

Метод k -ближайших соседей (k -NN) относится к одному из простейших алгоритмов классификации в машинном обучении. Он основывается на принципе близости объектов: если у объекта есть соседи известного класса, то скорее всего, этот объект также принадлежит к этому классу.

1.2 Принцип работы

При классификации нового объекта метод k -NN ищет k ближайших к нему объектов в обучающем наборе данных. Затем присваивает новому объекту тот класс, который наиболее часто встречается среди его соседей.

1.3 Применение на наборе данных

Для обучения моделей был выбран датасет. Датасет представляет собой коллекцию данных, предназначенных, для анализа кредитных заявок или риска мошенничества в банковской сфере. Описание полей данных:

1. **income (доход):** Годовой доход заявителя в децильной форме. Значения в диапазоне [0.1, 0.9].
2. **name_email_similarity (схожесть имени и электронной почты):** Метрика схожести между электронной почтой и именем заявителя. Значения в диапазоне [0, 1].
3. **prev_address_months_count (количество месяцев предыдущего адреса):** Количество месяцев в предыдущем зарегистрированном адресе заявителя. Значения в диапазоне [-1, 380] месяцев (-1 - отсутствующее значение).
4. **current_address_months_count (количество месяцев текущего адреса):** Месяцы в текущем зарегистрированном адресе заявителя. Значения в диапазоне [-1, 429] месяцев (-1 - отсутствующее значение).
5. **customer_age (возраст заявителя):** Возраст заявителя в годах, округленный до десятилетия. Значения в диапазоне [10, 90] лет.
6. **days_since_request (дни с момента подачи заявки):** Количество дней, прошедших с момента подачи заявки. Значения в диапазоне [0, 79] дней.
7. **intended_balcon_amount (начальная сумма для заявки):** Начальная переведенная сумма для заявки. Значения в диапазоне [-16, 114] (отрицательные значения - отсутствующие значения).
8. **payment_type (тип кредитного плана):** Категориальное поле с 5 возможными (анонимизированными) значениями.
9. **zip_count_4w (количество заявок в пределах того же почтового индекса за последние 4 недели):** Значения в диапазоне [1, 6830].
10. **velocity_6h (скорость заявок за последние 6 часов):** Среднее количество заявок в час за последние 6 часов. Значения в диапазоне [-175, 16818].
11. **velocity_24h (скорость заявок за последние 24 часа):** Среднее количество заявок в час за последние 24 часа. Значения в диапазоне [1297, 9586].

12. **velocity_4w** (скорость заявок за последние 4 недели): Среднее количество заявок в час за последние 4 недели. Значения в диапазоне [2825, 7020].
13. **bank_branch_count_8w** (количество заявок в выбранном банковском отделении за последние 8 недель): Значения в диапазоне [0, 2404].
14. **date_of_birth_distinct_emails_4w** (количество электронных писем с одинаковой датой рождения за последние 4 недели): Значения в диапазоне [0, 39].
15. **employment_status** (статус занятости): Категориальное поле с 7 возможными (анонимизированными) значениями.
16. **credit_risk_score** (кредитный риск): Внутренний балл риска заявки. Значения в диапазоне [-191, 389].
17. **email_is_free** (бесплатная электронная почта): Бинарное поле, указывающее, является ли домен электронной почты бесплатным или платным.
18. **housing_status** (статус проживания): Категориальное поле с 7 возможными (анонимизированными) значениями.
19. **phone_home_valid** (домашний телефон действителен): Бинарное поле, указывающее на действительность предоставленного домашнего телефона.
20. **phone_mobile_valid** (мобильный телефон действителен): Бинарное поле, указывающее на действительность предоставленного мобильного телефона.
21. **bank_months_count** (возраст предыдущего счета в месяцах): Значения в диапазоне [-1, 32] месяцев (-1 - отсутствующее значение).
22. **has_other_cards** (имеются другие карты): Бинарное поле, указывающее, есть ли у заявителя другие карты от того же банка.
23. **proposed_credit_limit** (предлагаемый кредитный лимит): Предложенный кредитный лимит заявителя. Значения в диапазоне [200, 2000].
24. **foreign_request** (запрос из другой страны): Бинарное поле, указывающее, отличается ли страна происхождения запроса от страны

банка.

- 25.**source (источник)**: Категориальное поле, указывающее онлайн-источник заявки (браузер или приложение).
- 26.**session_length_in_minutes (длительность сеанса пользователя в банковском веб-сайте в минутах)**: Значения в диапазоне [-1, 107] минут (-1 - отсутствующее значение).
- 27.**device_os (операционная система устройства)**: Категориальное поле с возможными значениями: Windows, macOS, Linux, X11 или другое.
- 28.**keep_alive_session (сохранить сеанс при выходе)**: Бинарное поле, представляющее опцию пользователя по сохранению сеанса при выходе.
- 29.**device_distinct_emails (количество различных электронных адресов с использованием устройства в последние 8 недель)**: Значения в диапазоне [-1, 2] адреса (-1 - отсутствующее значение).
- 30.**device_fraud_count (количество мошеннических заявок с использованием устройства)**: Значения в диапазоне [0, 1].
- 31.**month (месяц, в котором была подана заявка)**: Значения в диапазоне [0, 7].
- 32.**fraud_bool (Целевой столбец)**: Бинарное поле, указывающее, является ли заявка мошеннической или нет.

В связи с большим объемом датасета и малом проценте положительных значений для целевого столбца, было принято решение уменьшить его объем и увеличить процент положительных результатов

```
You, 4 days ago | 1 author (You)
1 import pandas as pd
2
3 # Загрузка CSV файла
4 df = pd.read_csv('Base.csv')
5
6 # Разделение данных на классы
7 class_0_df = df[df['fraud_bool'] == 0]
8 class_1_df = df[df['fraud_bool'] == 1]
9
10 class_0_sampled = class_0_df.sample(n=10000, random_state=42)
11 class_1_sampled = class_1_df.sample(n=2000, random_state=42)
12
13 # Объединение и перемешивание нового датасета
14 new_df = pd.concat([class_0_sampled, class_1_sampled])
15 new_df = new_df.sample(frac=1, random_state=42)
16
17 # Сохранение нового датасета
18 new_df.to_csv('Base_min_shuffled.csv', index=False)
```

Рис. 1 – Скрипт для преобразования датасета


```

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

# Загрузка данных
df = pd.read_csv("./Base_min_shuffled.csv")

print(df)

```

	fraud_bool	income	name_email_similarity	prev_address_months_count	\
0	0	0.5	0.041967	11	
1	0	0.8	0.865305	-1	
2	0	0.9	0.788227	-1	
3	0	0.7	0.182315	-1	
4	0	0.9	0.718332	-1	
...
11995	1	0.4	0.369884	-1	
11996	0	0.3	0.679246	-1	
11997	0	0.8	0.784484	-1	
11998	0	0.6	0.509644	-1	
11999	0	0.6	0.302013	-1	

	current_address_months_count	customer_age	days_since_request	\
0	16	50	0.031605	
1	37	40	0.024381	
2	122	30	0.004421	
3	56	20	0.015698	
4	184	30	0.001387	
...
11995	195	50	0.004030	
11996	74	20	0.018687	
11997	69	40	0.003280	
11998	118	40	0.005140	
11999	99	40	0.026095	
...
11998	1	0	3	
11999	1	0	5	

[12000 rows x 32 columns]

Рис. 2 – Подготовка данных

На рисунке 3 представлены разделение данных и нормализация

```

# Разделение данных на признаки (X) и целевую переменную (y)
y = df["fraud_bool"]
X = df.drop("fraud_bool", axis=1)

columns_to_drop = ['payment_type', 'employment_status', 'source', 'device_os', 'housing_status']

# Удаление указанных столбцов
df = df.drop(columns=columns_to_drop)

y = df["fraud_bool"]
X = df.drop("fraud_bool", axis=1)

# Разделение на тренировочные и тестовые данные
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Стандартизация данных
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

print(f"Размер обучающего набора: {X_train.shape}")
print(f"Размер тестового набора: {X_test.shape}")

```

29]

.. Размер обучающего набора: (9600, 26)
 Размер тестового набора: (2400, 26)

Рис.3 – Разделение данных и нормализация

На рисунке 4 представлены подбор гиперпараметров и наилучшего k

Подбор гиперпараметров и обучение модели

```
knn = KNeighborsClassifier()

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

grid = GridSearchCV(knn, param_grid, refit=True, verbose=3, cv=5)
# Обучение модели
grid.fit(X_train, y_train)

best_params = grid.best_params_
best_knn = grid.best_estimator_

Fitting 5 folds for each of 36 candidates, totalling 180 fits
[CV 1/5] END ...metric=euclidean, n_neighbors=2;, score=0.829 total time= 0.0s
[CV 2/5] END ...metric=euclidean, n_neighbors=2;, score=0.840 total time= 0.1s
```

Рис. 4 – Подбор гиперпараметра и наилучшего k

На рисунке 5 представлены проверка на тестовых данных и вывод результатов

```
# Инициализация и обучение модели методом ближайших соседей
predictions = best_knn.predict(X_test)

# Предсказание на тестовых данных

# Оценка качества модели
accuracy = accuracy_score(y_test, predictions)
report = classification_report(y_test, predictions)

print(f"Accuracy: {accuracy}")
print(f"\nЛучшие параметры: {best_params}")
print("Classification Report:\n", report)
```

[31]

```
... Accuracy: 0.86125

Лучшие параметры: {'metric': 'euclidean', 'n_neighbors': 8}
Classification Report:

```

	precision	recall	f1-score	support
0	0.88	0.97	0.92	2038
1	0.60	0.24	0.34	362
accuracy			0.86	2400
macro avg	0.74	0.60	0.63	2400
weighted avg	0.84	0.86	0.83	2400

Рис.5 – Проверка на тестовых данных и вывод результатов

1.4 Преимущества и ограничения

Преимущества метода k-NN:

- Прост в реализации и понимании.
- Не требует обучения модели, пока поступают новые данные.
- Хорошо подходит для начальной оценки данных и быстрых прототипов.

Однако у метода k-NN есть и ограничения:

- Чувствителен к выбросам в данных.
- Неэффективен на больших наборах данных из-за вычислительной сложности.
- Не учитывает значимость признаков, все признаки равнозначны.

Метод k-NN полезен для первичного анализа данных, но в реальных приложениях может потребоваться более сложная модель для учета различных аспектов и повышения точности предсказаний.

2. Метод машины опорных векторов (SVM)

2.1 Описание метода

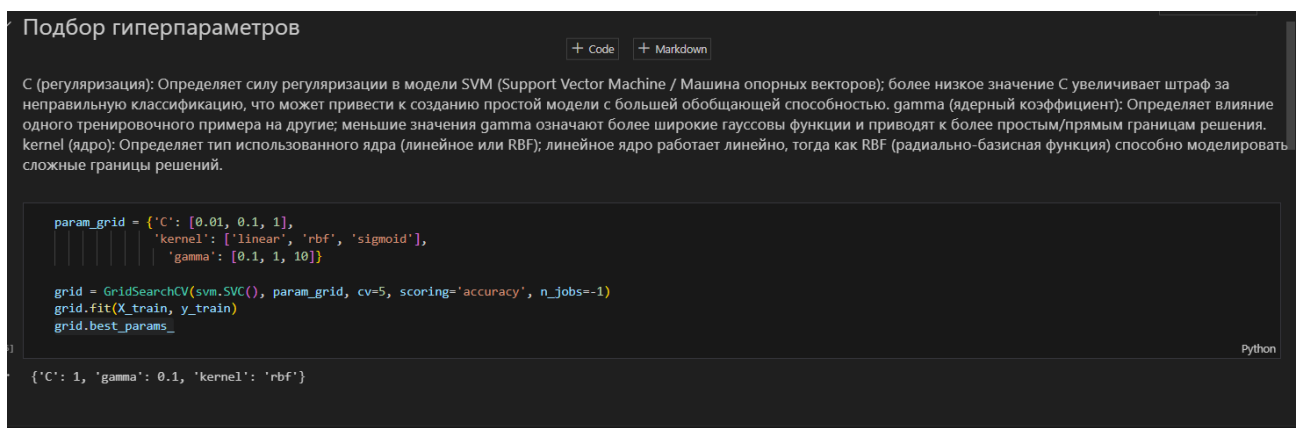
Метод машины опорных векторов (SVM) является мощным алгоритмом машинного обучения, используемым для задач классификации и регрессии. Основная идея заключается в поиске оптимальной разделительной гиперплоскости, которая максимально разделяет классы в данных.

2.2 Разделительная гиперплоскость и принцип работы

SVM строит гиперплоскость в n -мерном пространстве, где n - количество признаков. Эта гиперплоскость разделяет пространство на две части и максимизирует расстояние (зазор) между объектами разных классов, называемое отступом. Оптимальная гиперплоскость выбирается так, чтобы этот отступ был максимальным.

2.3 Применение на наборе данных

На рисунке 6 представлено описание гиперпараметров и подбор гиперпараметров с помощью перекрестной проверки



```
Подбор гиперпараметров

C (регуляризация): Определяет силу регуляризации в модели SVM (Support Vector Machine / Машина опорных векторов); более низкое значение C увеличивает штраф за неправильную классификацию, что может привести к созданию простой модели с большей обобщающей способностью. gamma (ядерный коэффициент): Определяет влияние одного тренировочного примера на другие; меньшие значения gamma означают более широкие гауссовы функции и приводят к более простым/прямым границам решения. kernel (ядро): Определяет тип использованного ядра (линейное или RBF); линейное ядро работает линейно, тогда как RBF (радиально-базисная функция) способно моделировать сложные границы решений.

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

grid = GridSearchCV(svm.SVC(), param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid.fit(X_train, y_train)
grid.best_params_

{'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
```

Рис.6 – Подбор гиперпараметров с помощью перекрестной проверки

Оценка производительности модели на тестовом наборе

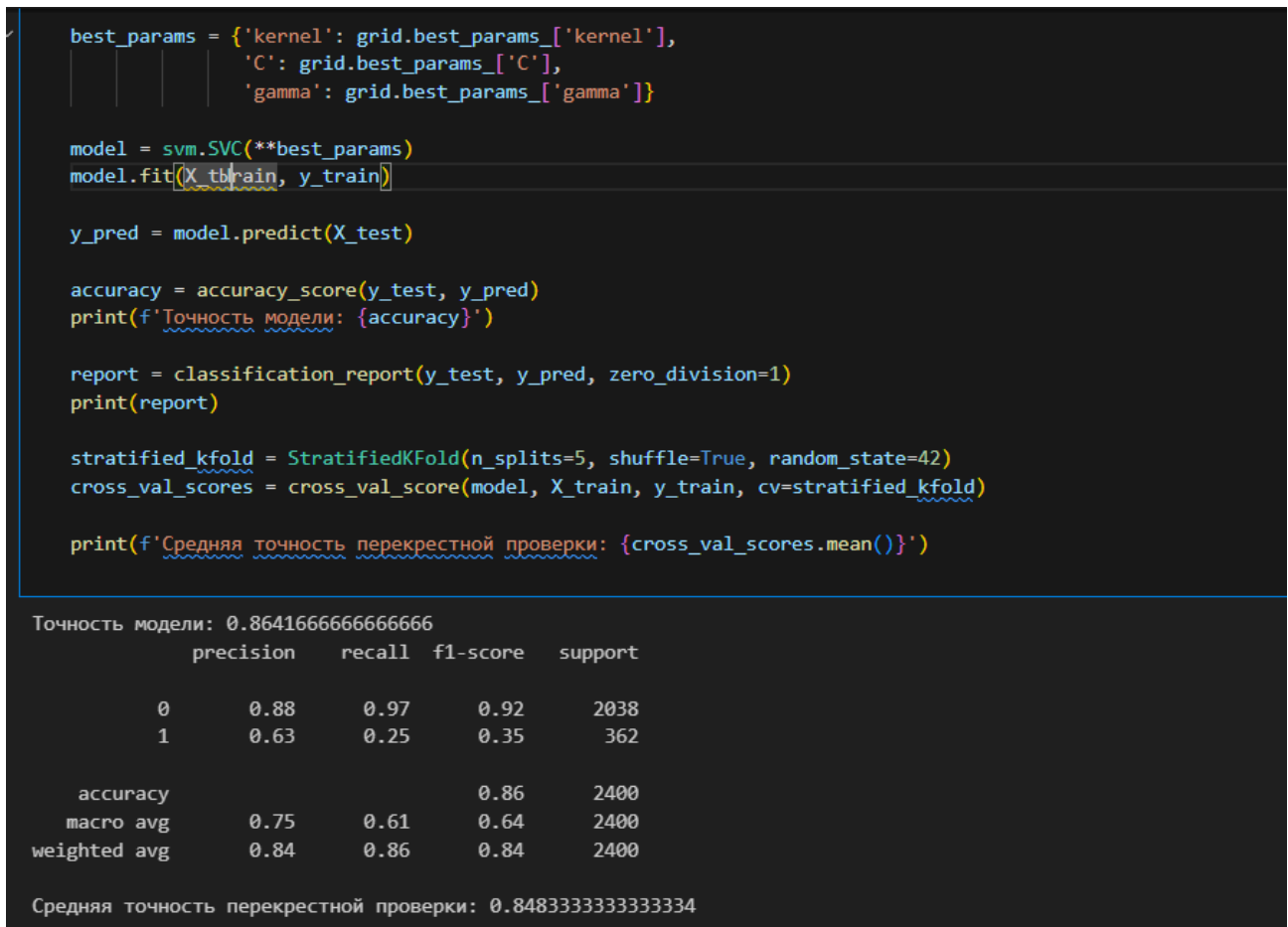


Рис. 7 - Оценка производительности модели на тестовом наборе

2.4 Преимущества и ограничения

Преимущества метода SVM:

- Эффективен в пространствах с большим количеством признаков.
- Хорошо работает в условиях разделяемых данных с помощью нелинейных ядер.
- Стабилен при обучении на небольшом наборе данных.

Однако у метода SVM есть и ограничения:

- Требуется тщательного выбора гиперпараметров и ядра.
- Неэффективен при работе с большими наборами данных из-за вычислительной сложности.
- Может быть чувствителен к выбросам в данных.

SVM - мощный алгоритм, который может быть эффективен при правильной настройке, но требует опыта для правильного применения и настройки гиперпараметров для достижения оптимальных результатов.

3. Методы линейной и логистической регрессии

3.1 Описание методов

Линейная регрессия – это метод, используемый для прогнозирования значений непрерывной зависимой переменной на основе линейной комбинации независимых переменных. Основная идея заключается в поиске линейной зависимости между предикторами и целевой переменной.

Применение линейной регрессии может быть полезным при прогнозировании численных значений, например, прогнозировании цены на недвижимость на основе её характеристик, таких как площадь, количество комнат и т.д.

Логистическая регрессия используется для задач классификации, прогнозируя вероятность принадлежности объекта к определенному классу. В отличие от линейной регрессии, логистическая регрессия применяется для бинарной или многоклассовой классификации.

3.2 Принцип работы

На рисунке 8 представлено обучение модели и подбор гиперпараметров для метода логистической регрессии

Логистическая регрессия

Описание гиперпараметров логистической регрессии

C (Обратная сила регуляризации):

Описание: C - это параметр регуляризации, который контролирует обратную силу регуляризации. Меньшие значения C увеличивают силу регуляризации. В контексте логистической регрессии, C определяет, насколько модель будет подстраиваться под тренировочные данные, минимизируя ошибку, и при этом учитывая сложность модели. Большие значения C могут привести к переобучению, если данных мало.

Penalty (Тип регуляризации):

Описание: Этот параметр определяет вид регуляризации, которую следует применять. L1-регуляризация (Lasso) добавляет штраф для суммы абсолютных значений коэффициентов, что может привести к отбору признаков. L2-регуляризация (Ridge) добавляет штраф для суммы квадратов коэффициентов, что может предотвратить большие значения коэффициентов.

Solver (Алгоритм оптимизации):

Описание: Solver - это алгоритм оптимизации, используемый для решения задачи оптимизации при подгонке модели. Различные алгоритмы могут подходить для разных задач и данных. 'liblinear' подходит для небольших наборов данных, поддерживает L1 и L2 регуляризацию. 'saga' поддерживает большие наборы данных и может обрабатывать как L1, так и L2 регуляризацию. 'lbfgs' также подходит для маленьких и средних по размеру наборов данных.

Перекрестная проверка гиперпараметров

+ Code

+ Markdown

```
param_grid = {
    'C': [0.01, 0.1, 1, 10],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear', 'saga', 'lbfgs']
}

logistic = LogisticRegression(max_iter=1000)
grid = GridSearchCV(logistic, param_grid, refit=True, verbose=1)
grid.fit(X_train, y_train)
```

Рис.8 – Логистическая регрессия

На рисунке 9 представлено обучение модели модели при помощи линейной регрессии

Линейная регрессия - это метод, используемый для предсказания значения переменной на основе линейной комбинации её признаков.

```
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
linear_accuracy = linear_model.score(X_test, y_test)
```

Рис.9 – Обучение модели методом линейной регрессии


```

# Оценка производительности модели логистической регрессии на тестовом наборе
best_logistic = grid.best_estimator_
logistic_accuracy = best_logistic.score(X_test, 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
logistic_predictions = best_logistic.predict(X_test)
linear_prediction = linear_model.predict(X_test)
print(classification_report(y_test, logistic_predictions))

# Отчет по классификации
print("\nОтчет по классификации для линейной регрессии:")
# Метрики регрессии
mse = mean_squared_error(y_test, linear_prediction)
r2 = r2_score(y_test, linear_prediction)

print("Средняя квадратичная ошибка:", mse)
print("Коэффициент детерминации (R^2):", r2)

```

```

Лучшие гиперпараметры логистической регрессии: {'C': 0.01, 'penalty': 'l2', 'solver': 'saga'}
Точность модели логистической регрессии: 0.8758333333333334
Точность модели линейной регрессии: 0.20044855024085606

```

	precision	recall	f1-score	support
0	0.89	0.97	0.93	2038
1	0.69	0.32	0.44	362
accuracy			0.88	2400
macro avg	0.79	0.65	0.68	2400
weighted avg	0.86	0.88	0.86	2400

```

Отчет по классификации для линейной регрессии:
Средняя квадратичная ошибка: 0.10240865961258802
Коэффициент детерминации (R^2): 0.20044855024085606

```

Рис. 10 – Вывод результатов

3.3 Сравнительный анализ с другими методами

Линейная и логистическая регрессии имеют свои сильные и слабые стороны по сравнению с другими методами. Линейная регрессия хорошо работает при предсказании непрерывных значений, тогда как логистическая регрессия применяется для задач классификации. Однако оба метода чувствительны к выбросам в данных и могут быть недостаточно гибкими для улавливания сложных нелинейных зависимостей. Поэтому в случае сложных данных и нелинейных связей эти методы могут оказаться менее эффективными по сравнению с другими алгоритмами, такими как деревья решений или нейронные сети.

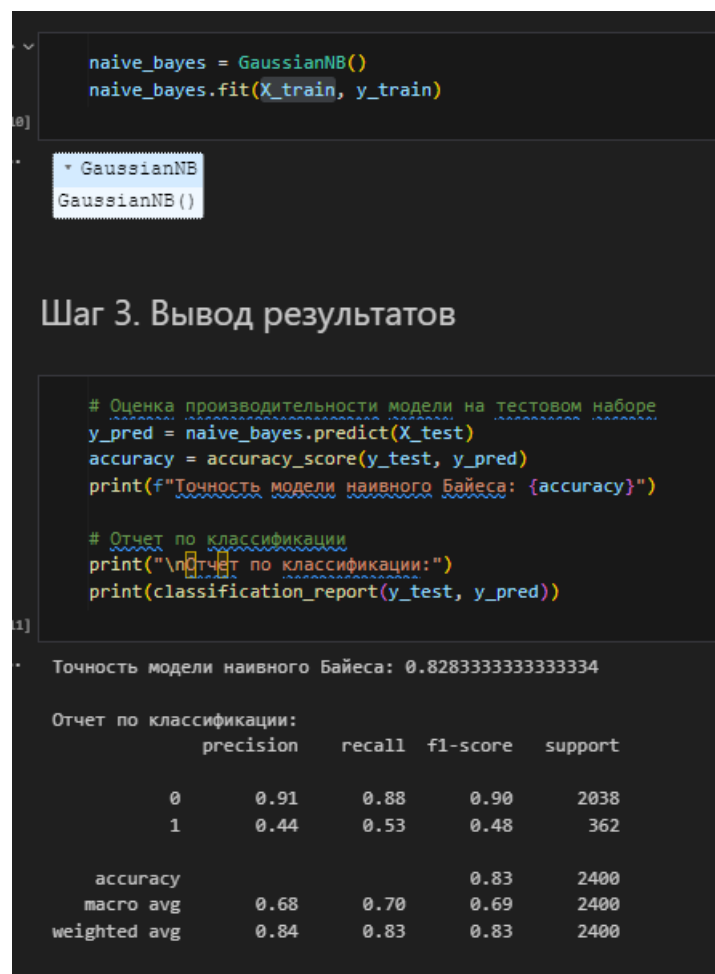
4. Метод наивного Байеса

4.1 Описание метода

Метод наивного Байеса основан на теореме Байеса и предполагает независимость между признаками. Он использует вероятностный подход к классификации и основывается на простой предпосылке, что признаки объектов независимы между собой при условии принадлежности к определенному классу.

4.2 Применение на наборе данных

На рисунке 11 представлены создание модели наивного Байеса и вывод результатов



```
naive_bayes = GaussianNB()
naive_bayes.fit(X_train, y_train)
```

▼ GaussianNB
GaussianNB()

Шаг 3. Вывод результатов

```
# Оценка производительности модели на тестовом наборе
y_pred = naive_bayes.predict(X_test)
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))
```

Точность модели наивного Байеса: 0.8283333333333334

Отчет по классификации:

	precision	recall	f1-score	support
0	0.91	0.88	0.90	2038
1	0.44	0.53	0.48	362
accuracy			0.83	2400
macro avg	0.68	0.70	0.69	2400
weighted avg	0.84	0.83	0.83	2400

Рис. 11 – Создание модели наивного Байеса и вывод результатов

4.3 Преимущества и ограничения

Преимущества метода наивного Байеса:

- Эффективен и быстр в обучении, особенно при работе с большими наборами данных.
- Хорошо работает в условиях небольшой обучающей выборки.
- Прост в реализации и понимании.

Однако у метода наивного Байеса есть и ограничения:

- Предполагает независимость признаков, что может быть не всегда верным в реальных данных.
- Может давать неоптимальные результаты в случае, когда зависимости между признаками сильны.

5. Методы решающего дерева и случайного леса

5.1 Описание метода

Решающее дерево представляет собой структуру, состоящую из узлов и листьев, которая используется для принятия решений на основе вопросов о значениях признаков. Оно строит дерево, разбивая данные на подмножества на основе определенных признаков.

Принцип работы решающего дерева заключается в выборе наилучшего признака для разделения данных на каждом узле дерева. Этот процесс повторяется, пока не будет достигнуто условие остановки, такое как максимальная глубина дерева или минимальное количество объектов в листе.

5.2 Применение на модельных данных

На рисунке 12 представлен подбор параметров и обучение методом решающего дерева

▼ Метод решающих деревьев

Метод решающих деревьев (Decision Trees) представляет собой модель машинного обучения, которая используется для задач как классификации, так и регрессии. Он строит дерево решений путем разбиения данных на подмножества с учетом определенных условий по признакам. Каждый узел в дереве представляет собой тест на признаке, каждая ветвь - результат этого теста, а каждый лист - прогноз.

Гиперпараметры решающего дерева: max_depth (максимальная глубина дерева):

Описание: Максимальная глубина дерева. Управляет тем, насколько глубоко дерево может разветвляться. Возможные значения: Любое целое положительное число или None (без ограничения глубины), min_samples_split (минимальное количество образцов для разделения узла):

Описание: Минимальное количество образцов, необходимых для разделения внутреннего узла. Большие значения помогают предотвратить переобучение. Возможные значения: Любое целое положительное число, min_samples_leaf (минимальное количество образцов в листе):

Описание: Минимальное количество образцов, необходимых для нахождения в листе. Большие значения помогают предотвратить переобучение. Возможные значения: Любое целое положительное число.

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

# 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=3, cv=5)
grid_search_tree.fit(X_train, y_train)

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

Python

Рис. 12 – Метод решающего дерева

На рисунке 13 представлены обучение решающего дерева и вывод результата

Вывод результатов обучения методом решающих деревьев

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

accuracy_tree = best_tree.score(X_test, 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)

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

[11]

... Лучшие параметры для решающего дерева: {'max_depth': 5, 'min_samples_leaf': 2, 'min_samples_split': 10}
Точность решающего дерева на тестовом наборе: 0.8570833333333333

Отчет по классификации:

	precision	recall	f1-score	support
0	0.87	0.98	0.92	2038
1	0.59	0.17	0.27	362
accuracy			0.86	2400
macro avg	0.73	0.58	0.59	2400
weighted avg	0.83	0.86	0.82	2400

Рис. 13 – Обучение и вывод результата

5.3 Сравнение с другими методами классификации

Решающее дерево и случайный лес являются эффективными методами классификации, которые обладают рядом преимуществ:

- Способны обрабатывать как категориальные, так и числовые данные.
- Позволяют визуализировать принятие решений.
- Могут обрабатывать большие объемы данных.

Однако у этих методов также есть некоторые ограничения:

- Склонны к переобучению на обучающих данных.
- Могут не улавливать сложные нелинейные зависимости.

По сравнению с линейными моделями, решающие деревья и случайный лес могут быть более гибкими в обработке сложных данных, но требуют осторожной настройки параметров для предотвращения переобучения. Их эффективность часто зависит от природы данных и особенностей задачи классификации.

6. Метод CatBoost

6.1 Описание метода

CatBoost (Categorical Boosting) – это высокоэффективная библиотека градиентного бустинга, специально разработанная для работы с категориальными признаками. Она представляет собой мощный алгоритм машинного обучения, который широко применяется в задачах классификации, регрессии и ранжирования.

6.2 Применение на модельных данных

На рисунке 14 представлена реализация метода CatBoost

l2_leaf_reg (L2 регуляризация):

Описание: Сила L2-регуляризации для листьев дерева. Возможные значения: Любое положительное число

iterations (количество итераций):

Описание: Количество деревьев в ансамбле (аналогично `n_estimators` в других моделях). Возможные значения: Любое положительное число

loss_function (функция потерь):

Описание: Функция потерь, которая оптимизируется во время обучения. Возможные значения: 'MultiClass', 'Logloss', 'CrossEntropy'

```
# 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']
}

# 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, y_train)

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

[3]

Рис.14 – Метод CatBoost

На рисунке 15 представлен вывод результата работы метода CatBoost

```
test_score = grid_search_catboost.score(X_test, 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)

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

[4]

```
... Лучшие параметры для CatBoost: {'depth': 3, 'iterations': 200, 'l2_leaf_reg': 1, 'learning_rate': 0.1, 'loss_function': 'MultiClass'}
... Лучший результат (точность) для CatBoost: 0.8670833333333334
... Точность на тестовом наборе данных: 0.87375
```

Отчет по классификации:				
	precision	recall	f1-score	support
0	0.89	0.97	0.93	2038
1	0.65	0.35	0.45	362
accuracy			0.87	2400
macro avg	0.77	0.66	0.69	2400
weighted avg	0.86	0.87	0.86	2400

Рис.15 – Вывод результата

CatBoost обладает мощными возможностями для работы с категориальными данными и является одним из популярных алгоритмов для задач машинного обучения, особенно в случаях, когда требуется обработка сложных и разнородных данных без предварительной подготовки.

Заключение

При выборе метода обучения для модели анализа кредитных заявок или оценки риска мошенничества в банковской сфере, необходимо учитывать не только характеристики датасета, но и цели моделирования. В данном случае, у нас есть набор признаков, описывающих заявителей, и целевой столбец, указывающий, является ли заявка мошеннической или нет.

Из рассмотренных методов машинного обучения следует рассмотреть несколько критериев для выбора подходящего метода:

Размер датасета: Поскольку у вас большой объем данных, методы, такие как случайный лес и градиентный бустинг (например, CatBoost), обычно хорошо справляются с большими объемами данных.

Дисбаланс классов: У вас уменьшили объем датасета и увеличили процент положительных результатов для целевого столбца. Это хороший подход для работы с дисбалансом классов, но также стоит учитывать, что не все методы хорошо справляются с несбалансированными данными.

Интерпретируемость: Если важно понимать, какие признаки оказывают влияние на решение модели, линейные методы (линейная регрессия, логистическая регрессия) могут быть более интерпретируемыми.

Скорость обучения и предсказания: Некоторые методы, такие как k-ближайших соседей, могут быть более вычислительно затратными, особенно с увеличением размера датасета.

Обработка категориальных признаков: Если у вас есть категориальные признаки, CatBoost способен обрабатывать их без предварительного кодирования.

С учетом вышеперечисленных критериев, для модели анализа кредитных заявок рекомендуется рассмотреть методы, такие как CatBoost, случайный лес или градиентный бустинг. Эти методы обладают хорошей производительностью на больших датасетах, способны работать с дисбалансом классов и могут обрабатывать категориальные признаки.

Однако, перед окончательным выбором метода, рекомендуется провести дополнительные эксперименты, включая настройку гиперпараметров и

валидацию модели на отложенной выборке, чтобы оценить ее производительность на конкретной задаче и датасете.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Основы машинного обучения, лекция 2 — метод k ближайших соседей:
<https://www.youtube.com/watch?v=X081VuXB1og&list=PLEwK9wdS5g0oCRxBzxsq9lkJkzMgzWiyg&index=2>
2. Основы машинного обучения, лекция 4 — линейная регрессия:
https://www.youtube.com/watch?v=8RAXDT_5_js&list=PLEwK9wdS5g0oCRxBzxsq9lkJkzMgzWiyg&index=24
3. Андрей Бурков. Машинное обучение без лишних слов