Министерство образования и науки РФ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Омский государственный технический университет»

КУРСОВОЙ ПРОЕКТ (РАБОТА)

по дисциплине Динамические языки программирования

на тему Разработка классификатора гостиницы

Пояснительная записка

Шифр проекта (работы)_____

Студента (ки) Голоты Ивана Сергеевича

Курс 4 Группа ПИН-201

Направление (специальность) 09.03.04 Программная инженерия

Руководитель старший преподователь

фамилия, инициалы Кабанов А. А.

Выполнил (а)_____

К защите_____

Проект (работа) защищен (а) с оценкой_____

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 44 с., 39 рис., 10 источников.

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

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

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

Содержание

Введение

Машинное обучение стало незаменимым инструментом для анализа данных и принятия решений в различных областях, от медицины до финансов. В данном отчете мы рассмотрим различные методы машинного обучения, такие как k-ближайших соседей, метод опорных векторов, логистической регрессии, наивного байесовского классификатора, модель решающего дерева, случайный лес и CatBoost.

Каждый из этих методов имеет свои уникальные особенности, преимущества и ограничения, которые мы изучим в рамках данного исследования. Мы также рассмотрим их применение в различных сценариях и задачах, чтобы понять, какой метод лучше всего подходит для конкретной задачи.

Целью этого отчета является обзор основных методов машинного обучения, их применение и сравнение производительности на различных наборах данных. Это позволит нам лучше понять их применимость и эффективность, а также определить наиболее подходящий метод для конкретной задачи.

1 Модель k-ближайших соседей

1.1 Теория

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

Основные аспекты метода *k-nn*:

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

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

1.2 Задачи

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

Ход работы:

Был найден датасет с 32 столбцами и 33725 строк. Информация о датасете представлен на рисунке 1.

```
Информация о данных:
Количество строк: 33725
Количество столбцов: 32

Типы данных по столбцам:
hotel                            object
is_canceled                       int64
lead_time                         int64
arrival_date_year                 int64
arrival_date_month               object
arrival_date_week_number          int64
arrival_date_day_of_month         int64
stays_in_weekend_nights           int64
stays_in_week_nights              int64
adults                            int64
children                          int64
babies                            int64
meal                             object
country                          object
market_segment                   object
distribution_channel             object
is_repeated_guest                 int64
previous_cancellations            int64
previous_bookings_not_canceled    int64
reserved_room_type               object
assigned_room_type               object
booking_changes                   int64
deposit_type                     object
agent                           float64
company                         float64
days_in_waiting_list              int64
customer_type                    object
adr                             float64
required_car_parking_spaces       int64
total_of_special_requests         int64
reservation_status               object
reservation_status_date          object
dtype: object
```

Рисунок 1 – Информация о датасете

В модели используем следующие гиперпараметры:

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

Рисунок 2 – Гиперпараметры

На рисунках 3-6 представлен код программы.

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

Рисунок 4 – Импорт библиотек

```
# Определение целевой переменной и признаков
X = df.drop('is_canceled', axis=1)
y = df['is_canceled']

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

# Разделение данных на обучающий и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Замена отсутсвующих и бесконечных значений на ноль
X_train = X_train.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)
X_test = X_test.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)

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

Рисунок 5 – Разделение данных и нормализация

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

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

# Поиск оптимальных параметров с помощью кросс-валидации
grid_search = GridSearchCV(knn, param_grid, cv=5)
grid_search.fit(X_train_scaled, y_train)

# Вывод наилучших параметров
print("Наилучшие параметры:", grid_search.best_params_)

# Обучение модели с наилучшими параметрами
best_k = grid_search.best_params_['n_neighbors']
best_knn = KNeighborsClassifier(n_neighbors=best_k)
best_knn.fit(X_train_scaled, y_train)
```

Рисунок 6 – Обучение модели

```
# Предсказание на тестовом наборе
y_pred = best_knn.predict(X_test_scaled)

# Оценка точности модели
accuracy = accuracy_score(y_test, y_pred)
print("Точность модели:", accuracy)

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

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

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.

```
Наилучшие параметры: {'metric': 'euclidean', 'n_neighbors': 3}
Точность модели: 0.9872498146775389

Отчет по классификации:
              precision    recall  f1-score   support

           0       0.57      0.07      0.12      4486
           1       0.33      0.90      0.48      2259

    accuracy                           0.35      6745
   macro avg       0.45      0.48      0.30      6745
weighted avg       0.49      0.35      0.24      6745
```

Рисунок 9 – Результаты

По результатам обучения модели k-ближайших соседей на датасете отеля, где изучается параметр is_canceled, можно сделать следующие выводы. Модель достигла высокой точности в 98,7%, что свидетельствует о ее хорошей способности классифицировать данные. Однако, стоит отметить, что метрики precision и recall для класса 0 (не отменено) довольно низкие, в то время как для класса 1 (отменено) они значительно выше. Это говорит о том, что модель хорошо распознает отмененные бронирования, но имеет затруднения в предсказании случаев, когда бронь не была отменена.

1.3 Вывод

Таким образом, модель k-ближайших соседей показывает хорошие результаты в обработке классов с несбалансированным распределением, но имеет ограничения в предсказании негативного класса. Для улучшения модели можно попробовать балансировку классов или использование других методов классификации, таких как случайный лес или градиентный бустинг, чтобы улучшить ее способность предсказывать оба класса более точно.

2 Модель машины опорных векторов

2.1 Теория

Модель опорных векторов (SVM) - это алгоритм машинного обучения, который используется как для задач классификации, так и для регрессии. Основная идея SVM заключается в поиске оптимальной разделяющей гиперплоскости, которая максимально отделяет два класса данных.

Теория модели опорных векторов в основном включает в себя следующие концепции:

1. Разделяющая гиперплоскость: SVM строит разделяющую гиперплоскость в n-мерном пространстве, где n - количество признаков. Для линейно разделимых данных гиперплоскость строится таким образом, чтобы максимизировать зазор между классами.

2. Опорные вектора: Опорные вектора - это наблюдения, которые лежат на границе зазора и определяют положение разделяющей гиперплоскости. Эти вектора играют важную роль в определении гиперплоскости.

3. Ядерные функции: SVM может использовать ядерные функции для перехода в пространство более высокой размерности, что позволяет разделить данные, которые не являются линейно разделимыми в исходном пространстве.

4. Оптимизация зазора: Целью SVM является максимизация ширины зазора между классами, что обеспечивает более точную классификацию новых данных.

5. Регуляризация: SVM имеет встроенную регуляризацию, которая позволяет управлять сложностью модели и предотвращать переобучение.

Эти концепции являются основой для понимания работы модели опорных векторов и её применения в задачах машинного обучения, включая классификацию, регрессию и определение функций ранжирования.

2.2 Задачи

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

Ход работы:

В модели используем следующие гиперпараметры:

```
param_grid = {'C': [0.1, 1, 10], 'gamma': [0.1, 1, 10]}
```

Рисунок 7 – Гиперпараметры

На рисунках 8-10 представлен код программы.

```
# Определение целевой переменной и признаков
X = df.drop('is_canceled', axis=1)
y = df['is_canceled']

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

# Разделение данных на обучающий и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Замена отсутсвующих и бесконечных значений на ноль
X_train = X_train.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)
X_test = X_test.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)

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

Рисунок 8 – Разделение данных и нормализация

```
# Поиск оптимальных параметров с помощью кросс-валидации
param_grid = {'C': [0.1, 1, 10], 'gamma': [0.1, 1, 10]}
grid_search = GridSearchCV(SVC(), param_grid, cv=10)
grid_search.fit(X_train_scaled, y_train)
```

Рисунок 9 – Создание модели

```
# Вывод наилучших параметров
print("Наилучшие параметры:", grid_search.best_params_)

# Обучение модели с наилучшими параметрами
best_C = grid_search.best_params_['C']
best_gamma = grid_search.best_params_['gamma']
best_svc = SVC(C=best_C, gamma=best_gamma)
best_svc.fit(X_train_scaled, y_train)

# Предсказание на тестовом наборе
y_pred = best_svc.predict(X_test_scaled)

# Оценка точности модели
accuracy = accuracy_score(y_test, y_pred)
print("Точность модели:", accuracy)
```

Рисунок 10 – Проверка модели и вывод результатов Процесс обучения

```
Наилучшие параметры: {'C': 10, 'gamma': 0.1}
Точность модели: 0.9961452928094885
```

Рисунок 11 – Результаты

2.3 Вывод

Исследование модели опорных векторов для параметра is_canceled в датасете отеля дало отличные результаты. Наилучшие параметры C=10 и gamma=0.1 привели к точности модели 99.6%. Это означает, что модель способна с высокой точностью предсказывать отменят ли клиенты бронирование в отеле или нет на основе доступных данных.

Такие высокие показатели точности говорят о том, что модель опорных векторов хорошо подходит для данной задачи классификации и может быть использована для прогнозирования отмены бронирования отеля с высокой достоверностью. Комбинация параметров C=10 и gamma=0.1 обеспечивает оптимальное обучение модели, что делает её надежным инструментом для принятия решений на практике.

Минусом является долгое время обучения модели.

3 Модель линейной регрессии

3.1 Теория

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

Основные концепции линейной регрессии:

1. Линейная зависимость:

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

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

где:

y - целевая переменная, которую мы пытаемся предсказать.

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

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

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

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

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

3. Оценка качества модели:

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

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

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

4. Множественная линейная регрессия:

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

y = w0 + w1*x1 + w2*x2 + ... + wnxn + ε,

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

3.2 Задачи

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

Ход работы:

1) Модель линейной регрессии:

В модели нет гиперпараметров.

На рисунках 12-15 представлен остальной код программы.

```python
# Определение целевой переменной и признаков
X = df.drop('is_canceled', axis=1)
y = df['is_canceled']

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

# Разделение данных на обучающий и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Замена отсутсвующих и бесконечных значений на ноль
X_train = X_train.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)
X_test = X_test.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)

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

Рисунок 12 – Разделение данных и нормализация

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

Рисунок 13 – Обучение модели и проверка модели

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

Рисунок 14 – Гиперпараметры

```
Среднеквадратичная ошибка (Линейная регрессия) : 0.030065157366153374
R^2 Score (Линейная регрессия) : 0.865025449694551
```

Рисунок 15 – Вывод результатов

3.3 Вывод

Обучение модели линейной регрессии для предсказания параметра is_canceled в датасете отеля дало следующие результаты: среднеквадратичная ошибка составила 0.030065157366153374, а коэффициент детерминации (R^2 Score) - 0.865025449694551.

Данная модель линейной регрессии показывает хорошую предсказательную способность, так как коэффициент детерминации близок к 1 (что означает хорошее качество аппроксимации данных), а среднеквадратичная ошибка довольно низкая, что указывает на общую точность модели.

Эти результаты говорят о том, что модель линейной регрессии может быть эффективно использована для прогнозирования вероятности отмены бронирования в отеле на основе имеющихся данных. Однако при дальнейшем использовании модели необходимо учитывать возможные особенности датасета и проводить тщательное тестирование модели на новых данных для подтверждения ее надежности.

Для прогнозирования чаще используется модель логистической регрессии.

4 Модель логистической регрессии

4.1 Теория

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

Основные концепции:

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

Формула логистической функции:

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

где:

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

x1, x2, ..., xn - признаки (факторы).

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

exp - экспоненциальная функция.

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

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

Гиперпараметры:

Гиперпараметры - это настройки модели, которые не могут быть изучены самой моделью и должны быть определены до начала процесса обучения. Для модели логистической регрессии заданы следующие гиперпараметры:

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

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

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

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

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

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

Выбор solver'a зависит от размера датасета, типа регуляризации и предпочтений при работе с моделью. Например, 'liblinear' может быть

предпочтительным для маленьких датасетов с L1- или L2-регуляризацией, в то время как 'lbfgs' может быть хорошим выбором для средних по размеру датасетов. 'saga' часто рекомендуется для больших наборов данных.

4.2 Задачи

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

Ход работы:

В модели используем следующие гиперпараметры:

```
# Модели логистической регрессии с разными гиперпараметрами
logistic_model_1 = LogisticRegression(max_iter=1000)  # Модель с параметрами по умолчанию
logistic_model_2 = LogisticRegression(penalty='l2', C=1.0, solver='lbfgs', max_iter=1000)  # Модель с настроенными гиперпа
```

Рисунок 16 – Гиперпараметры

На рисунках 17-20 представлен код программы.

```
# Определение целевой переменной и признаков
X = df.drop('is_canceled', axis=1)
y = df['is_canceled']

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

# Разделение данных на обучающий и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Замена отсутсвующих и бесконечных значений на ноль
X_train = X_train.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)
X_test = X_test.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)

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

Рисунок 17 – Разделение данных и нормализация

```
# Обучение моделей
logistic_model_1.fit(X_train_scaled, y_train)
logistic_model_2.fit(X_train_scaled, y_train)
```

Рисунок 18 – обучение модели

```python
# Получение предсказаний для тестового набора данных
logistic_predictions_1 = logistic_model_1.predict(X_test_scaled)
logistic_predictions_2 = logistic_model_2.predict(X_test_scaled)

# Оценка производительности моделей на тестовом наборе данных
accuracy_1 = accuracy_score(y_test, logistic_predictions_1)
accuracy_2 = accuracy_score(y_test, logistic_predictions_2)

print(f"Точность (Логистическая регрессия - Начальные параметры) : {accuracy_1}")
print(f"Accuracy (Логистическая регрессия - Кастомные параметры) : {accuracy_2}")
```

Рисунок 19– Проверка модели и вывод результатов

В ходе обучения модели были выявлены лучшие гиперпараметры. На рисунке 20 представлены лучшие гиперпараметры, точность модели на реальных данных и отчёт по классификации.

```
Точность (Логистическая регрессия - Начальные параметры) : 0.9909562638991846
Accuracy (Логистическая регрессия - Кастомные параметры) : 0.9909562638991846
Classification Report (Логистическая регрессия - Начальные параметры):
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      4486
           1       1.00      0.97      0.99      2259

    accuracy                           0.99      6745
   macro avg       0.99      0.99      0.99      6745
weighted avg       0.99      0.99      0.99      6745

Classification Report (Логистическая регрессия - Кастомные параметры):
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      4486
           1       1.00      0.97      0.99      2259

    accuracy                           0.99      6745
   macro avg       0.99      0.99      0.99      6745
weighted avg       0.99      0.99      0.99      6745
```

Рисунок 20 – Результаты

4.3 Вывод

Модель логистической регрессии, обученная на датасете отеля для предсказания параметра is_canceled, демонстрирует высокую точность. Как в случае использования начальных параметров, так и при применении кастомных параметров, точность модели составляет 99.1%. Это указывает на то, что модель хорошо справляется с классификацией и предсказанием отмены бронирования.

Анализ classification report также подтверждает высокие показатели модели. Метрики precision, recall и f1-score для обоих классов (0 и 1) близки к 1, что свидетельствует о хорошей способности модели правильно классифицировать оба значения.

Таким образом, модель логистической регрессии демонстрирует отличные показатели как по accuracy, так и по метрикам precision, recall и f1-score, что говорит о ее высокой предсказательной способности в контексте данного датасета.

Из плюсов: быстрая обучаемость модели.

Из минусов: может быть неэффективной при низкой линейной разделимости классов: В случае, если классы плохо разделимы линейно, модель может демонстрировать низкую точность.

5 Модель наивного Байеса

5.1 Теория

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

Основные концепции:

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

Формула теоремы Байеса:

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

где:

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

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

P(C) - априорная вероятность класса C.

P(X) - вероятность признаков X.

Наивное предположение о независимости признаков:

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

Типы наивных Байесовских моделей:

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

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

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

Применение модели:

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

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

5.2 Задачи

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

Ход работы:

Модель наивного Байеса:

В модели нет гиперпараметров.

На рисунках 21-23 представлен остальной код программы.

```
# Определение целевой переменной и признаков
X = df.drop('is_canceled', axis=1)
y = df['is_canceled']

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

# Разделение данных на обучающий и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Замена отсутсвующих и бесконечных значений на ноль
X_train = X_train.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)
X_test = X_test.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)

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

Рисунок 21 – Разделение данных и нормализация

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

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

Рисунок 22 – Обучение модели и проверка модели

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

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

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

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

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

Рисунок 23 – Вывод результатов

```
Точность: 0.9825055596738325
              precision    recall  f1-score   support

           0       1.00      0.97      0.99      4486
           1       0.95      1.00      0.97      2259

    accuracy                           0.98      6745
   macro avg       0.98      0.99      0.98      6745
weighted avg       0.98      0.98      0.98      6745
```

Рисунок 24 – Результаты

5.3 Вывод

Исходя из представленного отчета по классификации и точности модели наивного Байеса, можно сделать следующие выводы:
Модель, обученная методом наивного Байеса для предсказания параметра is_canceled на датасете отеля, показывает хорошие результаты. Точность модели составляет 98.25%, что свидетельствует о ее способности правильно классифицировать данные.

Анализ classification report также подтверждает хорошие показатели модели. Метрики precision, recall и f1-score для обоих классов (0 и 1) также демонстрируют высокие значения, особенно для класса 0, где precision достигает 100% и recall составляет 97%. Для класса 1 precision равен 95%, а recall - 100%. Это указывает на то, что модель хорошо справляется как с предсказанием отмены бронирования (класс 1), так и с предсказанием успешного бронирования (класс 0).

В целом, модель наивного Байеса демонстрирует высокую точность и хорошие метрики precision, recall и f1-score, что подтверждает ее способность предсказывать параметр is_canceled на основе предоставленных данных эффективно и точно.

6 Модель решающего дерева

6.1 Теория

Модель решающего дерева:

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

Основные концепции:

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

Узлы и листья:

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

Принцип работы:

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

Гиперпараметры:

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

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

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

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

6.2 Задачи

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

Ход работы:

В модели используем следующие гиперпараметры:

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

Рисунок 25 – Гиперпараметры

На рисунках 26-28 представлен код программы.

```
# Выборка признаков и целевой переменной
features = ['hotel', 'lead_time', 'arrival_date_year', 'arrival_date_month', 'arrival_date_week_number', 'arrival_date_day_
target = 'is_canceled'

# Определение целевой переменной и признаков
X = df[features]
y = df[target]

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

# Разделение данных на обучающий и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Замена отсутсвующих и бесконечных значений на ноль
X_train = X_train.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)
X_test = X_test.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)

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

Рисунок 26 – Разделение данных и нормализация

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

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

Рисунок 27 – Обучение модели

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

Рисунок 28 – Проверка модели и вывод результатов

В ходе обучения модели были выявлены лучшие гиперпараметры. На рисунке 29 представлены лучшие гиперпараметры, точность модели на реальных данных и отчёт по классификации.

```
Лучшие параметры для решающего дерева: {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 10}
Точность настроенных параметров решающего дерева: 0.8443291326908822
Classification Report of Tuned Decision Tree:
              precision    recall  f1-score   support

           0       0.85      0.93      0.89      4486
           1       0.84      0.66      0.74      2259

    accuracy                           0.84      6745
   macro avg       0.84      0.80      0.81      6745
weighted avg       0.84      0.84      0.84      6745
```

Рисунок 29 – Результаты

6.3 Вывод

Модель решающего дерева, обученная для предсказания параметра is_canceled на датасете отеля, показывает приемлемые результаты. Лучшие параметры для данной модели были определены как max_depth: 10, min_samples_leaf: 2, min_samples_split: 10, что привело к достижению точности 84.43%.

Анализ classification report также позволяет проанализировать эффективность модели. Метрики precision, recall и f1-score для обоих классов (0 и 1) говорят о том, что модель достаточно хорошо предсказывает оба класса, но с некоторыми ограничениями. Для класса 0 значение precision составляет 85%, а recall - 93%, а для класса 1 - 84% и 66% соответственно.

Таким образом, модель решающего дерева достигла точности на уровне 84.43% и демонстрирует хорошие метрики для класса 0, но менее убедительные результаты для класса 1. Это может указывать на то, что модель более успешно предсказывает успешные бронирования (класс 0), чем отмененные бронирования (класс 1).

7 Модель случайного леса

7.1 Теория

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

Основные концепции:

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

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

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

Гиперпараметры модели случайного леса:

n_estimators: Количество деревьев в случайном лесу.

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

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

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

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

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

7.2 Задачи

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

Ход работы:

В модели используем следующие гиперпараметры:

```
params_rf = {
    'n_estimators': [50, 100],
    'max_depth': [None, 5],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1]
}
```

Рисунок 30 – Гиперпараметры

На рисунках 31-33 представлен код программы.

```
# Выборка признаков и целевой переменной
features = ['hotel', 'lead_time', 'arrival_date_year', 'arrival_date_month', 'arrival_date_week_number', 'arrival_date_day_c
target = 'is_canceled'

# Определение целевой переменной и признаков
X = df[features]
y = df[target]

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

# Разделение данных на обучающий и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Замена отсутсвующих и бесконечных значений на ноль
X_train = X_train.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)
X_test = X_test.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)

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

Рисунок 31– Разделение данных и нормализация

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

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

Рисунок 32 – Обучение модели

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

Рисунок 33 – Проверка модели и вывод результатов

В ходе обучения модели были выявлены лучшие гиперпараметры. На рисунке 34 представлены лучшие гиперпараметры, точность модели на реальных данных и отчёт по классификации.

```
Лушчие параметры для случайного леса: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 100}
Точность настроенных параметров случайного леса: 0.8704225352112676
Classification Report of Tuned Random Forest:
              precision    recall  f1-score   support

           0       0.89      0.92      0.90      4486
           1       0.83      0.77      0.80      2259

    accuracy                           0.87      6745
   macro avg       0.86      0.85      0.85      6745
weighted avg       0.87      0.87      0.87      6745
```

Рисунок 34 – Результаты

7.3 Вывод

Модель случайного леса, обученная для предсказания параметра is_canceled на датасете отеля, показывает очень хорошие результаты. Лучшие параметры для данной модели были определены как 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 100, что привело к достижению точности 87.04%.

Анализ classification report также позволяет проанализировать эффективность модели. Метрики precision, recall и f1-score для обоих классов (0 и 1) говорят о том, что модель успешно предсказывает оба класса. Для класса 0 значение precision составляет 89%, а recall - 92%, что является очень хорошим показателем. Для класса 1 метрики немного ниже, precision - 83% и recall - 77%, но все равно находятся на уровне, обозначающем хорошие прогнозы для данного класса.

Таким образом, модель случайного леса достигла высокой точности на уровне 87.04%. Метрики precision, recall и f1-score подтверждают, что модель успешно предсказывает оба класса, однако она может быть более успешной в предсказании успешных бронирований (класс 0) по сравнению с отмененными бронированиями (класс 1).

8 Модель CatBoost

8.1 Теория

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

Основные концепции:

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

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

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

Гиперпараметры:

depth: Максимальная глубина деревьев.

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

l2_leaf_reg: Коэффициент регуляризации L2 для весов листьев деревьев.

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

loss_function: Функция потерь для оптимизации.

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

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

8.2 Задачи

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

Ход работы:

В модели используем следующие гиперпараметры:

```
# Определение параметров для поиска
param_grid = {
    'iterations': [100, 200],
    'learning_rate': [0.01, 0.1],
    'depth': [3, 6],
    'loss_function': ['MultiClass']
}
```

Рисунок 35 – Гиперпараметры

На рисунках 36-38 представлен код программы.

```
# Определение целевой переменной и признаков
X = df.drop('is_canceled', axis=1)
y = df['is_canceled']
X = pd.get_dummies(X)

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

# Разделение данных на обучающий и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Замена отсутсвующих и бесконечных значений на ноль
X_train = X_train.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)
X_test = X_test.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)

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

Рисунок 36 – Разделение данных и нормализация

```
# Создание объекта GridSearchCV для модели
grid_search = GridSearchCV(
    estimator=CatBoostClassifier(verbose=False),  # Установка verbose=False,чтобы не было видно как обучается модель
    param_grid=param_grid,
    cv=5,
    n_jobs=-1
)


# Подгонка модели к данным
grid_search.fit(X_train_scaled, y_train)
```

Рисунок 37 – Обучение модели

Процесс обучения модели представлен на рисунке 51.

```
# Получение лучших параметров и оценщика с лучшими параметрами
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

print(f'Лучшие параметры: {best_params}')

# Получение прогнозов с использованием лучшей модели
y_pred = best_model.predict(X_test_scaled)

# Вычисление и вывод точности модели
accuracy = accuracy_score(y_test, y_pred)
print(f'Точность модели: {accuracy}')

# Оценка производительности модели
report = classification_report(y_test, y_pred, zero_division=1)
print(f'Отчет о классификации:\n{report}')

# Определение отсутствующих классов в прогнозах
classes = np.unique(y_test)
predicted_classes = np.unique(y_pred)
```

Рисунок 38 – Проверка модели и вывод результатов

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

```
Лучшие параметры: {'depth': 3, 'iterations': 100, 'learning_rate': 0.01, 'loss_function': 'MultiClass'}
Точность модели: 1.0
Отчет о классификации:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      4486
           1       1.00      1.00      1.00      2259

    accuracy                           1.00      6745
   macro avg       1.00      1.00      1.00      6745
weighted avg       1.00      1.00      1.00      6745
```

Рисунок 39 – Результаты

8.3 Вывод

Точность модели CatBoost для предсказания параметра is_canceled на датасете отеля составляет 100%, что является исключительно высоким показателем. Метрики precision, recall и f1-score для обоих классов (0 и 1) также составляют 100%, что указывает на то, что модель успешно предсказывает оба класса. Эти результаты наводят на мысль, что модель работает очень хорошо, и ее прогнозы могут быть достаточно надежными.

Такие высокие метрики могут быть вызваны переобучением модели, особенно если используются параметры, которые слишком хорошо подходят к обучающим данным. Поэтому при использовании модели CatBoost с лучшими параметрами {'depth': 3, 'iterations': 100, 'learning_rate': 0.01, 'loss_function': 'MultiClass'} рекомендуется провести дополнительный анализ и проверить модель на новых данных, чтобы удостовериться в ее эффективности.

Тем не менее, в текущем контексте исходя из предоставленных данных, модель CatBoost показывает идеальную точность предсказаний и может использоваться в приложениях, где требуется высокая надежность предсказаний.

Заключение

Результаты анализа различных моделей машинного обучения выявили сильные и слабые стороны каждого подхода. Модель k-ближайших соседей показывает хорошие результаты в обработке классов с несбалансированным распределением, но имеет ограничения в предсказании негативного класса. Для улучшения модели можно попробовать балансировку классов или использование других методов классификации

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

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

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

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

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

CatBoost в моём наборе данных показал самую высокую точность, однако, стоит отметить, что результаты 100% точности могут быть признаком переобучения модели на обучающих данных, поэтому рекомендуется провести дополнительный анализ

Список используемых источников

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

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

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

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

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

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

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

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

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

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