УДК 004.89

Исследование эффективности различных методов машинного обучения

Пронин Д.А., Клепиков П.Н.
Алтайский государственный университет, г. Барнаул
zayach-02@mail.ru, klepikov.math@gmail.com

Аннотация

Целью данной исследовательской работы является исследование эффективности различных методов машинного обучения. В данной работе будут проанализированы и будет проведено сравнение нескольких широко используемых методов, включая линейную регрессию, PolynomialFeatures, метод градиентного бустинга, метод случайного леса.

Ключевые слова: Машинное обучение, метод случайного леса, метод градиентного бустинга, линейная регрессия, PolynomialFeatures.

1. Введение

В последние десятилетия машинное обучение стало одной из ключевых областей исследований в области компьютерных наук. Расширение доступности вычислительных ресурсов и возможности обработки больших объемов данных привели к резкому развитию методов машинного обучения и их успешному применению в различных сферах, включая медицину, финансы, транспорт, рекламу и другие.

Целью машинного обучения является разработка алгоритмов и моделей, которые способны обучаться на основе имеющихся данных и прогнозировать, классифицировать или принимать решения без явного программирования. Однако существует множество различных методов машинного обучения, каждый из которых имеет свои преимущества и недостатки, а также области применения, в которых он может быть наиболее эффективным.

Целью данной исследовательской работы является исследование эффективности различных методов машинного обучения. В данной работе будут проанализированы и будет проведено сравнение нескольких широко используемых методов, включая линейную регрессию, Polynomial Features, метод градиентного бустинга, метод случайного леса.

Для достижения данной цели будет использован набор данных о квартирах в регионах России и проводить эксперименты с использованием этих методов машинного обучения. Будут проведены оценка и сравнение результатов в терминах точности предсказаний, времени обучения и других соответствующих метрик. Это позволит выявить преимущества и ограничения каждого метода и сделать выводы о его эффективности в различных сценариях.

В итоге, данная работа будет способствовать более глубокому пониманию методов машинного обучения и их сравнительной эффективности. Результаты и выводы исследования могут быть полезными для принятия обоснованных решений при выборе подходящего метода машинного обучения в конкретных прикладных задачах.

2. Ход работы

Обработка набора данных и разработка модели машинного обучения проводились на языке программирования Python в интерактивной онлайн-среде GoogleColab. Для предобработки и визуализации данных были использованы библиотеки Pandas, Matplotlib и NumPy, а для построения модели линейной регрессии библиотека scikit-learn. Ниже приведён фрагмент кода, в котором происходит подключение этих пакетов (первые 7 строк) и загрузка набора данных в формате CSV (последняя строка):

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import re
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
df = pd.read_csv('/content/drive/MyDrive/curseovaya/curseach.csv',sep=';')
```

Так как линейная регрессия [1] очень восприимчива к выбросам, пропущенным значениям, дубликатам и прочим неровностям в наборе данных, то сначала проведём обработку и очистку данных. Для начала следует разделить признак date на 3 разных показателя: day, month, year. Это необходимо для нормального распознавания такого параметра как date методом машинного обучения

```
df['year'] = pd.to_datetime(df['date']).dt.year
df['month'] = pd.to_datetime(df['date']).dt.month
df['day'] = pd.to_datetime(df['date']).dt.day
df = df.drop(columns='date')
```

Убираем пропущенные значения:

```
Missing_Value = df.isnull().sum().sort_values(ascending=False)
Missing_Percent = (df.isnull().sum() /
df.isnull().count() * 100).sort_values(ascending=False)
Missing_Data = pd.concat([Missing_Value, Missing_Percent], axis=1,
keys=['Пропущенные значения', 'Процент'])
Missing_Data.reset_index()
```

Проводим оцифровку некоторых параметров:

```
df['geo_lat'] = df['geo_lat'].apply(lambda x: float(x.replace(',', '.')))
df['geo_lon'] = df['geo_lon'].apply(lambda x: float(x.replace(',', '.')))
df['area'] = df['area'].str.replace(',','.').astype(float)
df['kitchen_area'] =
df['kitchen_area'].apply(lambda x: float(x.replace(',', '.')))
```

Уберём дубликаты:

```
df = df.drop_duplicates()
```

Уберём ошибочные значения:

```
df = df[df['price'] > 0]
df_pred_nullprice = df[df['price'] == 0.0].index
df = df.drop(df_pred_nullprice, axis=0)
df = df[df['rooms']>0]
```

Ограничим максимальную цену квартиры:

```
df=df[df['price']<20000000]
```

Избавимся от выбросов в некоторых признаках методом квантилей:

```
q_low = df['area'].quantile(0.01)
q_hi  = df['area'].quantile(0.99)
df = df[(df['area'] <q_hi) & (df['area'] >q_low)]
q_low = df["price"].quantile(0.01)
q_hi  = df["price"].quantile(0.99)
df = df[(df["price"] <q_hi) & (df["price"] >q_low)]
q_low = df["kitchen_area"].quantile(0.01)
q_hi  = df["kitchen_area"].quantile(0.99)
df = df[(df["kitchen_area"] <q_hi) & (df["kitchen_area"] >q_low)]
```

После небольшой обработки данных следует провести оценку важности параметров. Чтобы узнать зависимость цены от каждого признака воспользуемся следующим методом:

```
correlation_table = df.corr()['price']
print(correlation_table)
```

Таблица 1

Коэффициенты корреляции признаков относительно признака price

Характеристика	Корреляция
price	1,000000
geo_lat	0,163618
geo_lon	-0,371137
Region	-0,417262
building_type	0,030618
level	0,256392
levels	0,378396
rooms	0,326898
area	0,485185
kitchen_area	0,403799
object_type	-0,004045
year	-0,009143
month	0,019470
day	0,062183

При помощи оценки важности было выяснено, что наилучшую ценность представляют следующие параметры: region, level, levels, area, kitchen_area. Остальные параметры либо имеют слишком низкую оценку рамках метода, либо конфликтуют с параметрами, которые имеют большее влияние на цену и друг на друга. Следующим шагом является разделение данных на две выборки тестовую и обучающую:

```
x = df[['region','level','levels','area','kitchen_area']]
```

```
y = df['price']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
```

И теперь приступаем собственно к самому обучению :

```
model=LinearRegression()
model.fit(x, y)
pred = model.predict(x_test)
print(f"Точность обучающей выборки: {model.score(x_train, y_train)}")
print(f"Точность тестовой выборки: {model.score(x_test, y_test)}")
Точность обучающей выборки: 0.48741417415392396
Точность тестовой выборки: 0.48517573919049684
```

После обучения получаем точность ниже приемлемой, чтобы повысить точность попробуем воспользоваться полиномиальной регрессией [2, 3]:

```
poly_features = PolynomialFeatures(degree=2, include_bias=False)
x_poly = poly_features.fit_transform(x)
x_train, x_test, y_train, y_test = train_test_split(x_poly, y, test_size=0.2)
model = LinearRegression()
model.fit(x_train, y_train)
print(f"Точность обучающей выборки: {model.score(x_train, y_train)}")
print(f"Точность тестовой выборки: {model.score(x_test, y_test)}")
Точность обучающей выборки: 0.5519781885540176
Точность тестовой выборки: 0.5487837851751622
```

В ходе анализа результатов экспериментов было обнаружено, что линейная регрессия может иметь недостаточную точность в некоторых сценариях. Это связано с тем, что линейная регрессия предполагает линейную зависимость между признаками и зависимой переменной. В случаях, когда связь является нелинейной или имеет сложную структуру, линейная регрессия может не справиться с предсказанием точных значений. Для того, чтобы предсказывать с лучшей точностью следует воспользоваться другими методами, такими как градиентный бустинг или метод случайного леса.

3. Реализация более продвинутых методов машинного обучения

```
rf = RandomForestRegressor()
rf_params = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10]
}
```

В этом участке кода создается экземпляр модели случайного леса rf с использованием класса RandomForestRegressor(). Затем определяются параметры модели в переменной rf_params:

- "n_estimators" отвечает за количество деревьев, которые будут построены в случайном лесу. В данном случае, три значения [100, 200, 300] указывают на то, что модель будет обучена с использованием 100, 200 и 300 деревьев.
- "max_depth" определяет максимальную глубину каждого дерева в случайном лесу. Значения [None, 5, 10] указывают на то, что модель будет обучена с тремя разными ограничениями глубины деревьев: без ограничения глубины (None), с максимальной глубиной 5 и с максимальной глубиной 10.

```
gb = GradientBoostingRegressor()
gb_params = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.1, 0.01, 0.001]
}
```

В этом участке кода создается экземпляр модели градиентного бустинга gb с использованием класса GradientBoostingRegressor(). Затем определяются параметры модели в переменной gb_params.

- "n_estimators" отвечает за количество деревьев, которые будут построены в градиентном бустинге. В данном случае, три значения [100, 200, 300] указывают на то, что модель будет обучена с использованием 100, 200 и 300 деревьев.
- "learning_rate" определяет величину, на которую каждое новое дерево будет влиять на окончательное предсказание модели. Значения [0.1, 0.01, 0.001] указывают на три разных скорости обучения: высокую (0.1), среднюю (0.01) и низкую (0.001).

Теперь создаjм список моделей:

```
models = [
    ('Полином + линейная регрессия', lr_pipe, lr_params),
    ('Случайный лес', rf, rf_params),
    ('Градиентный бустинг', gb, gb_params)
]
```

Следующим шагом приступаем к обучению и выводим результаты:

```
for name, model, params in models:
    grid_search = GridSearchCV(model, params, n_jobs=-2)
    grid_search.fit(x_train, y_train)
    print(f'{name}:')
    print(f'Обучающая точность: {grid_search.score(x_train, y_train):.3f}')
    print(f'Тестовая точность: {grid_search.score(x_test, y_test):.3f}')
    print('Лучшие значения параметров:')
    for param, val in grid_search.best_params_.items():
        match = re.search(r'\w+?__(\w+)', param)
        if match is not None:
            param = match.group(1)
        print(f'{param} = {val}')
    print('----------------------------------------------')
```

В этом участке кода происходит цикл for, который проходит по каждому элементу в списке моделей models. В каждой итерации цикла выполняются следующие действия:

Создается экземпляр GridSearchCV с текущей моделью model и параметрами params. GridSearchCV — это инструмент для перебора комбинаций гиперпараметров модели и выбора наилучшей комбинации на основе кросс-валидации.

Выполняется обучение модели grid_search на обучающих данных x_train и y_train с помощью метода fit(). Это приводит к перебору различных комбинаций параметров модели и выбору лучшей комбинации, которая демонстрирует наилуч- шую производительность на обучающем наборе данных.

Выводится название модели name.

Выводится точность модели на обучающем наборе данных и тестовом наборе данных с помощью метода score(). Значение grid_search.score(x_train, y_train) представляет обучающую точность, а grid_search.score(x_test, y_test) тестовую точность.

Выводятся лучшие значения параметров модели, найденные в процессе перебора с помощью GridSearchCV. Эти значения доступны через атрибут best_params_ объекта grid_search. В цикле for проходятся все элементы словаря grid_search.best_params_, и если параметр содержит двойное подчеркивание (например model__parameter), то извлекается только само имя параметра (например, parameter).

Выводится разделительная линия для отделения результатов разных моделей. После выполнения цикла получаем представленные ниже данные:

```
Полином + линейная регрессия:
Обучающая точность: 0.520
Тестовая точность: 0.515
Лучшие значения параметров:
degree = 6
include_bias = True
interaction_only = True
--------------------------------------------
Случайный лес:
Обучающая точность: 0.886
Тестовая точность: 0.773
Лучшие значения параметров:
max_depth = 10
n_estimators = 300
--------------------------------------------
Градиентный бустинг:
Обучающая точность: 0.813
Тестовая точность: 0.777
Лучшие значения параметров:
learning_rate = 0.1
n_estimators = 200
--------------------------------------------
```

Лучше всех себя показал метод градиентного бустинга. Тестовая точность которого составила 0.777. Такие результаты являются приемлемыми.

4. Заключение

В ходе проведения данной исследовательской работы были исследованы различные методы машинного обучения, включая линейную регрессию, Polynomial Features. Целью работы было оценить эффективность этих методов на наборе данных о квартирах в различных регионах России.

В ходе анализа результатов экспериментов было обнаружено, что линейная регрессия может иметь недостаточную точность в некоторых сценариях. Это связано с тем, что линейная регрессия предполагает линейную зависимость между признаками и зависимой переменной. В случаях, когда связь является нелинейной или имеет сложную структуру, линейная регрессия может не справиться с предсказанием точных значений.

Для того, чтобы предсказывать с лучшей точностью, было принято решение, использовать другие модели машинного обучения, такие как метод градиентного бустинга и метод случайного леса.

Используя эти методы удалось получить большие тестовую и обучающую точность.

Список литературы

1. Мюллер А., Гвидо С. Введение в машинное обучение с помощью Python. — М. : Диалектика, 2017. — 472 с.

2. Bishop C.M. Pattern Recognition and Machine Learning. — NY : Springer, 2006.

3. Жерон О. Прикладное машинное обучение с помощью Scikit-Learn, Keras и TensorFlow. Концепции, инструменты. : Пер. с англ. — М. : Диалектика, 2020. — 1040 с.