Проект 3: Множественная линейная регрессия в машинном обучении

В этом проекте я буду рассматривать **множественную линейную регрессию**. В отличие от простой линейной регрессии, где есть одна независимая переменная и одна зависимая переменная, в множественной линейной регрессии используется несколько независимых переменных, которые могут влиять на зависимую переменную.

Набор данных

Он содержит информацию о 50 стартапах и включает 5 столбцов:

- R&D Spend — затраты на исследования и разработки.

- Administration — административные расходы.

- Marketing Spend — затраты на маркетинг.

- State — штат, в котором находится стартап.

- Profit — прибыль стартапа.

Цель проекта

Наша цель — построить модель, которая сможет предсказать прибыль (**"Profit"**) на основе независимых переменных, описанных выше. Таким образом, прибыль — это зависимая переменная, а остальные четыре столбца — независимые переменные.

Шаг 1: Загрузка набора данных

Ниже приведен фрагмент кода для загрузки набора данных. Мы будем использовать библиотеку pandas.

- X содержит все независимые переменные: "R&D Spend", "Administration", "Marketing Spend" и "State".

- y — зависимая переменная, которая представляет собой "Profit".

Для X мы используем dataset.iloc[:, :-1].values, что означает "взять все строки и все столбцы, кроме последнего".

Для y мы используем dataset.iloc[:, 4].values, что означает "взять все строки и только столбец с индексом 4" (в Python индексация начинается с 0, поэтому индекс 4 соответствует пятому столбцу, который является "Profit").

```python
# Шаг 1 - Загрузка данных
```

```
import pandas as pd

dataset = pd.read_csv("50_Startups.csv")

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, 4].values
```

Шаг 2: Преобразование текстовой переменной в числа

В нашем наборе данных есть категориальная переменная "State", которую нужно закодировать. Мы используем класс LabelEncoder для преобразования текста в числа.

```python
# Шаг 2 - Преобразование текстовой переменной "State" в числа

from sklearn.preprocessing import LabelEncoder

labelEncoder_X = LabelEncoder()

X[:, 3] = labelEncoder_X.fit_transform(X[:, 3])
```

После выполнения этого кода все штаты будут преобразованы в числа. Например:

- New York → 2

- California → 0

- Florida → 1

Шаг 3: Использование OneHotEncoder для создания фиктивных переменных

Фиктивные переменные (Dummy Variables)

Фиктивные переменные используются для представления категориальных данных (например, штаты, цвета, типы продуктов) в числовом формате, чтобы их можно было использовать в моделях машинного обучения, таких как линейная регрессия. Категориальные данные не могут быть напрямую использованы в моделях, так как они не имеют числового значения.

Пример:

Предположим, у нас есть категориальная переменная **"State"** с тремя значениями: **California**, **Florida**, **New York**. Мы не можем просто заменить их числами (например, 0, 1, 2), так как модель может интерпретировать эти числа как ранги

(например, New York > California). Чтобы избежать этого, мы создаем фиктивные переменные.

Создание фиктивных переменных:

1. Каждое уникальное значение категориальной переменной преобразуется в отдельный бинарный столбец (0 или 1).
2. Например, для переменной **"State"**:
    - **California** → [1, 0, 0]
    - **Florida** → [0, 1, 0]
    - **New York** → [0, 0, 1]

Если оставить данные в текущем состоянии, модель может ошибочно интерпретировать числовые значения как ранги (например, New York = 2 > California = 0). Чтобы избежать этого, мы используем OneHotEncoder для создания фиктивных переменных.

```python
# Шаг 3 - Использование OneHotEncoder для создания фиктивных переменных
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
import numpy as np
ct = ColumnTransformer(transformers=[('encoder',OneHotEncoder(),[3])], remainder='passthrough')
X = np.array(ct.fit_transform(X))
```

После выполнения этого кода в набор данных будут добавлены три фиктивные переменные, соответствующие трем штатам.

transformers=[('encoder', OneHotEncoder(), [3])]

Применяет OneHotEncoder к столбцу с индексом 3 (4-й столбец, так как индексация с 0).

Результат будет содержать несколько бинарных столбцов (по числу уникальных значений в столбце 3).

remainder='passthrough'

Остальные столбцы (0, 1, 2, 4, ...) остаются без изменений.

Шаг 4: Устранение ловушки фиктивных переменных

Мы должны удалить одну из фиктивных переменных, чтобы избежать ловушки фиктивных переменных. В следующем фрагменте кода мы удаляем первый столбец.

Если мы включим все фиктивные переменные в модель, это может привести к мультиколлинеарности (высокой корреляции между независимыми переменными), что ухудшит качество модели. Чтобы избежать этого, мы удаляем одну из фиктивных переменных. Например, если у нас три штата, мы оставляем две фиктивные переменные.

```python
# Шаг 4 - Устранение ловушки фиктивных переменных
X = X[:,1:]
```

Шаг 5: Разделение набора данных на обучающую и тестовую выборки

Мы разделим набор данных на обучающую и тестовую выборки. Для этого используем метод train_test_split из библиотеки model_selection.

- test_size=0.2 означает, что тестовая выборка будет содержать 10 наблюдений, а обучающая — 40.

- random_state=0 используется для воспроизводимости результатов.

```python
# Шаг 5 - Разделение данных
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Шаг 6: Обучение модели линейной регрессии

Мы используем класс LinearRegression из библиотеки sklearn.linear_model для обучения модели на обучающей выборке.

```python
# Шаг 6 - Обучение модели
from sklearn.linear_model import LinearRegression
```

```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Шаг 7: Предсказание на тестовой выборке

Используя обученную модель, мы предскажем значения для тестовой выборки и сравним их с фактическими значениями.

```python
# Шаг 7 - Предсказание
y_pred = regressor.predict(X_test)
```

Шаг 8: Обратное исключение (Backward Elimination)

Мы используем метод обратного исключения, чтобы определить, какие независимые переменные наиболее значимы для модели. Этот процесс помогает улучшить точность модели, удаляя незначительные переменные.

```python
# Шаг 8 - Обратное исключение
import statsmodels.api as sm
X = np.append(arr=np.ones((50, 1)).astype(int), values=X, axis=1)
X_opt = X[:, [0, 1, 2, 3, 4, 5]]
regressor_OLS = sm.OLS(endog=y, exog=X_opt).fit()
regressor_OLS.summary()
```

Мы повторяем процесс, удаляя переменные с наибольшим p-значением, пока не останутся только значимые переменные.

Обратное исключение (Backward Elimination)

Обратное исключение — это метод выбора признаков, который используется для определения наиболее значимых независимых переменных в модели. Цель этого метода

— удалить незначительные переменные, которые не оказывают существенного влияния на зависимую переменную, чтобы улучшить точность модели.

Шаги обратного исключения:

1. **Начальная модель:** Включаем все независимые переменные в модель.
2. **Выбор уровня значимости (SL):** Обычно выбирается уровень значимости 0.05 (5%).
3. **Оценка p-значений:** Для каждой независимой переменной вычисляется p-значение, которое показывает вероятность того, что переменная не влияет на зависимую переменную.
4. **Удаление переменной:** Если p-значение переменной больше уровня значимости (SL), эта переменная удаляется из модели.
5. **Повторение процесса:** Модель пересчитывается без удаленной переменной, и процесс повторяется до тех пор, пока все оставшиеся переменные не станут значимыми.

Пример:

Предположим, у нас есть следующие независимые переменные: **R&D Spend**, **Administration**, **Marketing Spend**, **State** (закодированные как фиктивные переменные).

1. **Шаг 1:** Включаем все переменные в модель.
2. **Шаг 2:** Вычисляем p-значения для каждой переменной.
    - Если p-значение для **Administration** равно 0.99 (99%), что больше уровня значимости 0.05, мы удаляем эту переменную.
3. **Шаг 3:** Пересчитываем модель без **Administration**.
4. **Шаг 4:** Повторяем процесс для оставшихся переменных.

Реализация в Python:

```
python

import statsmodels.api as sm

# Добавляем столбец единиц для учета свободного члена (intercept)

X = np.append(arr=np.ones((50, 1)).astype(int), values=X, axis=1)

# Начальная модель со всеми переменными

X_opt = X[:, [0, 1, 2, 3, 4, 5]]

regressor_OLS = sm.OLS(endog=y, exog=X_opt).fit()

regressor_OLS.summary()
```

```
# Удаляем переменную с наибольшим p-значением (например, Administration)
X_opt = X[:, [0, 1, 3, 4, 5]]
regressor_OLS = sm.OLS(endog=y, exog=X_opt).fit()
regressor_OLS.summary()
# Повторяем процесс, пока все переменные не станут значимыми
```

Итог:

- **Фиктивные переменные** позволяют использовать категориальные данные в моделях машинного обучения, но важно избегать ловушки фиктивных переменных, удаляя одну из них.
- **Обратное исключение** помогает улучшить модель, удаляя незначительные переменные и оставляя только те, которые действительно влияют на зависимую переменную.

Эти методы являются важными инструментами для повышения точности и интерпретируемости моделей машинного обучения.

После выполнения всех шагов мы получаем модель, которая предсказывает прибыль стартапа на основе наиболее значимых переменных. Этот процесс помогает улучшить точность модели и интерпретируемость результатов.