

# Моё решение

## (Прянишников Александр)

### Целевая переменная

Первое, что меня смутило: задача состоит в прогнозировании **сдачи** теста, в то время как в программе в качестве целевой переменной используется тестовый балл. С одной стороны да, мы можем точнее оценить потенциал ученика. Но с другой стороны, даже в первых пяти записях видно, что под индексами 2 и 4 ученики абсолютно идентичны, но у одного `target = 1`, у другого - 0. Также видно, что человек с 79 баллами за тест не сдал, в то время как с 72 и 77 сдали. (И это только `head()`!)

Тут есть несколько вариантов. Первое - выборка была собрана с ошибками, тогда вообще теряется смысл что-то по ней предсказывать. Но я оптимист - считаю, что данные +- корректны, и дальше буду этого придерживаться. Поэтому нужно сказать, что в таком случае `posttest` - это не целевая переменная, так как от неё напрямую не следует то, что был факт сдачи экзамена, поэтому логичнее бы было делать целевой переменной `target`.

### Предобработка данных

Затем, сразу видно столбцы `"Unnamed: 0"` и `"Unnamed: 0.1"` которые уже своим названием говорят о том, что это лишние столбцы. Но по факту они просто дублируют индекс, поэтому от них стоило бы избавиться на этапе предобработки, а не в конце по показателям коэффициентов линейной регрессии. Кстати, в последних 10 строках `df` у нас есть пропуски в `target`, видимо, мы это и должны предсказать, либо это же пропуски, и их нужно

убрать. Так как реальное значение узнать нельзя, я бы просто удалил эти строки.

Следующий этап - работа с пропусками и выбросами. Я посмотрел на данные, пропуски есть только в столбце возраста студента. Почему-то в программе возраст меняли на среднее, которое является не целым, а дробным числом, хотя возраст, видимо, подразумевает как количество полных лет. Соответственно, ломается логика составления этого признака.

Также есть ещё один минус: у train и test разные средние, так как модель обучена на train, не стоит удивляться, что для test коэффициент не полностью корректен.

Но самое главное, что метрика среднего - вообще не слишком подходит под задачу. Я бы предложил выбирать значение из тех строк, которые больше всего похожи на ту, где меняем пропуск. Например, в строке с индексом 2 в df пропущено значение возраста, но видно, что рядом с ним ещё куча строк с абсолютно той же школой, типом, кабинетом, учебным методом. Можно найти те строки, расстояние по признакам с которыми минимальное, а затем взять оттуда чаще всего встречающееся значение. То есть по сути найти класс, с которым человек занимался, и оттуда брать значение. Конечно, придётся на время векторизовать, но это не сложно.

## Новая переменная

По поводу новой переменной woe\_agegender. Во-первых, преобразование было применено к df, а не x\_train и x\_test, соответственно, она нигде не используется. Во-вторых, я посмотрел по каждому возрасту и заметил, что пол практически не играет роли для этого признака. Более того, в нашей модели и так фигурируют пол и возраст. Так как в создании признака фигурирует  $\geq$  для возраста, это говорит о том, что показатель woe\_agegender будет монотонно возрастать вместе с возрастом, соответственно, мы дублируем уже имеющиеся признаки. Об этом же свидетельствует и высокая корреляция – 0.543714.

Но для проверки целесообразности я попробовал обучить модель уже с этим признаком, и на выходе получилось, что коэффициент перед этим признаком (посмотрел сам, с p-value не путал), равняется -2.47. Учитывая, что значения у признака от 0.46 до 0.61, разброс этого признака = 0.15 \*

$(-2.47) = 0.37$ . Предсказываем мы балл теста (хотя я не считаю это правильным, о чём говорил вначале), поэтому влияние этого признака маленькое.

## Encoding

Следующий момент - one-hot encoding. В коде вообще-то используется LabelEncoder. Я стараюсь его избегать, так как мы наводим порядок на множество. И если для `teaching_method` это может быть нормально (всего два варианта, `Standard` и `Experimental` это можно как-то понять), то порядок для самого названия школы вводить неправильно, особенно учитывая, что мы не знаем, насколько имя школы говорит о чём-то. У нас этот столбец вообще считается некорректно.

One-hot тут бы подошёл. У `school_setting`, `school_type`, `teaching_method`, `gender` и `lunch` всего несколько дискретных значений, и вот тут можно спокойно использовать one-hot. К `School` я бы попробовал бы сделать так: для каждой школы бы подсчитал среднее значение сдачи (именно `mean`, потому что `median` выдаст 0 или 1), а затем бы подставил подсчитанные коэффициенты вместо кода. Минус такого подхода в том, что если будут новые школы в будущих, то подсчитать коэффициент для него не получится, и придётся повторять процедуру заново.

Для `Classroom` тоже нужно применить One-Hot. Хотя мы и плодим столбцы, но потенциально это киллер-фича, потому что по сути она отражает класс учеников, при этом если пройтись по данным, то видно, что в классе многие +- одного уровня.

## Оценка модели

"Ошибка на тесте не сильно больше чем на трейне. Значит модель отличная это не так, потому что по сути у нас используется `train_test_split`. У датафрейме у нас есть много строк, которые ничем не отличаются, поэтому для них предсказанное значение будет абсолютно одинаковым. У нас в `train` и `test` после смешивания все равно будут одинаковые строки -> предсказанное значение будет одинаковым, поэтому что-то говорить о качестве модели нельзя. Вот если бы мы посмотрели на другие данные...

P-value, думаю, применять не корректно. Причина в том, что наши дан-

ные не нормализованы, поэтому для `classroom` это значение огромное, но дело не в важности признака, а в том, что после `LabelEncoder` там значения от 0 до 96, в то время как другие признаки кроме возраста меняются максимум от 0 до 2. Также для применения этого теста надо быть убеждённым, что признаки не коррелируют друг с другом.

## Мои действия

Резюмируя те действия, которые бы стоило сделать, чтобы улучшить модель:

1. Разобрался бы с тем, что мы действительно предсказываем (балл или сдал/не сдал).
2. Сразу бы убрал признаки, которые повторяют индекс строк.
3. Убрал бы строки, для которых `target` не известен.
4. Для каждой школы подсчитал бы среднее по нашему выбранному на пункте 1, заменил бы значение.
5. Заменил бы пропуски в возрасте на медианное значение возраста ближайших строк.
6. Векторизовал бы через `one-hot` все оставшиеся категориальные переменные.
7. Подсчитал бы признак `woe`, добавил бы в таблицу.
8. Применил бы нормализацию для возраста и `woe`.
9. Разбил бы на `X_train` и `X_test`.
10. Только здесь применил бы линейную регрессию.

Я применил эти шаги, в итоге получил значение метрики 2.470574 на тесте – гораздо лучше, чем в исходном. Прикрепляю листинг кода, и скрин метрики.

## Листинг 1: Листинг кода

```
import itertools
import numpy as np
import pandas as pd
import scipy
from sklearn.base import BaseEstimator
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import mean_absolute_error
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt

# Read Dataset
df = pd.read_csv('dataset.csv', sep=';')
print(df.shape)

# Drop columns and last strings with Nan
df.drop(["Unnamed: 0", "Unnamed: 0.1"], axis = 1, inplace = True)
N = df.shape[0]
df = df.iloc[: (N - 10), :]

# Change school_value
school_value = df[["school", "target"]].groupby('school').agg('mean')
df["school"] = df["school"].apply(lambda x: school_value.loc[x])

# One-Hot Encoding with Pandas
categorical_features = df.columns[df.dtypes == 'object']
df = pd.get_dummies(df, columns = categorical_features, drop_first = True)

def find_most_similar_strings(df, st):
    max = 0
    ages = []
    for st1 in df.index:
        sum = 0
        if (st1 != st):
            for col in df.columns:
                sum += int(df.loc[st, col] == df.loc[st1, col])
            if (sum > max):
                max = sum
                ages = []
                age = df.loc[st1, "n_student"]
                if not (np.isnan(age)):
                    ages.append(age)
            elif (sum == max):
                age = df.loc[st1, "n_student"]
                if not (np.isnan(age)):
```

```

        ages.append(age)
    return np.median(np.array(ages))

# Fill Nan with age = age in similar strings
for st in df.index:
    if (np.isnan(df.loc[st, 'n_student'])):
        df.loc[st, 'n_student'] = find_most_similar_strings(df, st)

# Create new feature
for i in df.gender_Male.unique():
    for j in df.n_student.unique():
        l=df.loc[(df.gender_Male==i) & (df.n_student>=j)]
        woe=l.target.sum()/(l.target.count()-df.target.mean()-1)
        df.loc[(df.gender_Male==i) & (df.n_student>=j), 'woe_agegender']=woe

# Min-Max scale for age and woe
scaler = MinMaxScaler()
df['n_student'] = scaler.fit_transform(pd.DataFrame(df['n_student']))
df['woe_agegender'] = scaler.fit_transform(pd.DataFrame(df['woe_agegender']))

# Drop strings (around 10) with nans (similar strings with nan too)
df = df.fillna(df.mean())

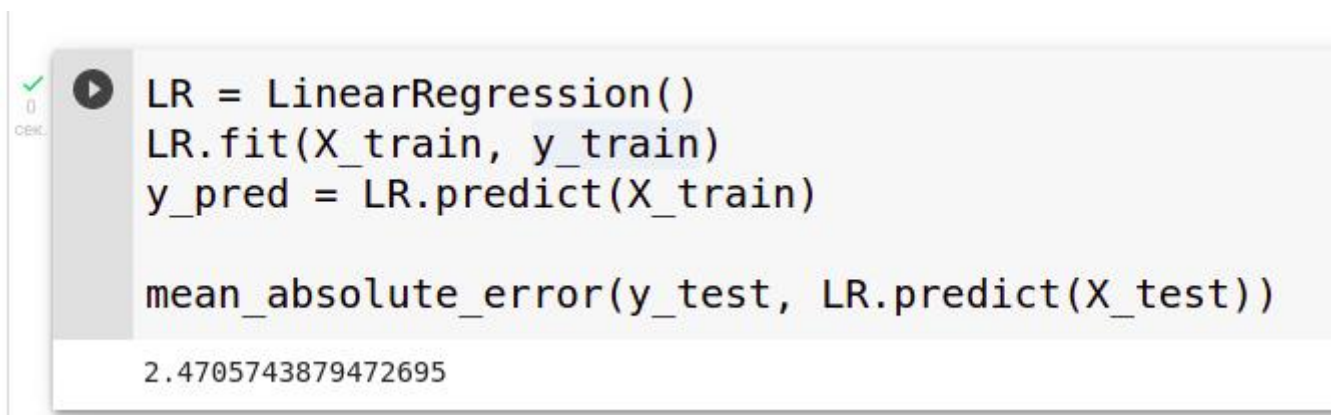
# Create X and Y
X = df.drop(['posttest'], axis=1)
X = X.drop(['target'], axis=1)
y = df['posttest']

# Train-Test split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1234)

# Learn model
LR = LinearRegression()
LR.fit(X_train, y_train)
y_pred = LR.predict(X_train)

# Result
print(mean_absolute_error(y_test, LR.predict(X_test)))

```



The image shows a Jupyter Notebook cell with a green checkmark icon and a play button icon on the left. The code in the cell is as follows:

```
LR = LinearRegression()  
LR.fit(X_train, y_train)  
y_pred = LR.predict(X_train)  
  
mean_absolute_error(y_test, LR.predict(X_test))
```

The output of the cell is the value 2.4705743879472695.

Рис. 1: Демонстрация моего результата