МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

ОТЧЕТ ПО КУРСОВОЙ РАБОТЕ

по дисциплине «Статистические методы обработки данных»

Тема: «Методы кодирования категориальных признаков в задаче бинарной классификации»

Выполнил cтудент группы М80-101М-21

         Фейзуллин К.М.

Руководитель: канд. физ.-мат. наук, доцент кафедры 804 Платонов Е.Н.

Оценка:

Оглавление

[**Введение** 3](#_Toc91430868)

[**Постановка задачи** 4](#_Toc91430869)

[**Описание эксперимента** 5](#_Toc91430870)

[**Функционал качества** 5](#_Toc91430871)

[**Описание исходных данных** 8](#_Toc91430872)

[**Базовая модель** 10](#_Toc91430873)

[**Экспериментальная установка** 11](#_Toc91430874)

[**Методы кодирования** 13](#_Toc91430875)

[**Результаты исследования** 15](#_Toc91430876)

[**Заключение** 16](#_Toc91430877)

[**Список литературы** 17](#_Toc91430878)

[**Приложение** 18](#_Toc91430879)

# **Введение**

Целью работы является исследование влияния методов кодирования категориальных признаков в задаче бинарной классификации на функционал качества работы прогнозирующей математической модели.

В эру больших данных и расцвета автоматизированных алгоритмов обработки информации и прогнозирования важную роль занимают сами методы обработки. Помимо целочисленных и вещественных признаков нередко приходиться сталкиваться и с категориальными данными, которые требуют особый подход для дальнейшей работы с ними.

Актуальность данного исследования состоит в том, что до сих пор точно неизвестно, для какого рода задач и для каких сущностей переменных то или иное кодирование покажет наилучший результат.

Причем, данная задача бинарной классификации нередко используется для решения разного типа проблем, будь то прогноз подорожания или роста акции, решение о предоставления кредита или ипотеки, определение вредоносности программного обеспечения по показателям системы. Как можно понять, данная задача имеет множество приложений и в том числе выгодных с экономической точки зрения.

Но очень важно учитывать, что качество работы математической модели имеет зависимость с качеством данных, которые обрабатываются алгоритмом прогноза.

Поэтому нахождение оптимальных методов кодирования напрямую скажется на качестве работы прогнозирующей модели.

# **Постановка задачи**

Целью задачи является исследование методов кодирования категориальных признаков в задаче бинарной классификации. Функционалом качества работы алгоритма бинарной классификации является метрика ROC AUC (Receiver Operating Characteristic Area Under Curve), которая представляет из себя интегральную сумму от пороговой функции отношения доли верно определенных положительных меток и доли ложно определенных положительно меток.

Необходимо используя различные способы кодирования категориальных признаков и через создание новых признаков добиться оптимального результата.

Все варианты кодирования (различные наборы признаков) и соответствующую точность решения задачи будут занесены в сводную таблицу.

# **Описание эксперимента**

## **Функционал качества**

Как уже было сказано выше, положительное влияние тех или иных методов кодирования будет определяться с помощью функционала качества ROC AUC[1]. Чтобы определить ее введем карту обозначений, приведенную в таблице 1.

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Прогноз | |
| 1 | 0 |
| Правильный ответ | 1 | TP (True – positives) | FN (False – negatives) |
| 0 | FP (False – positives) | TN (True – negatives) |

Табл.1 – карта обозначений.

Определим обозначения, в нашем случае, как TP (True – positives) – число раз, когда объект положительно определен верно, FP (False – positives) – когда объект положительно определен ложно, FN (False – negatives) – когда объект отрицательно определен ложно и TN (True – negatives) – когда объект отрицательно определен верно.

Из этих характеристик определяются TPR (True positive rate) и FPR (False positive rate), доля истинно положительных ответов и доля ложно положительных ответов соответственно:

Далее вносит свой вклад природа бинарного классификатора. Простейший линейный классификатор имеет вид:

),

Где X – входной сигнал, Y – выходной сигнал, – некая пороговая функций, – масштабирующий коэффициент, а – пороговое значение классификатора.

Последовательно варьируя пороговое значение и находя в каждом значении значение TPR и FPR будет определена пороговая функция ROC вида, как на рисунке 1:

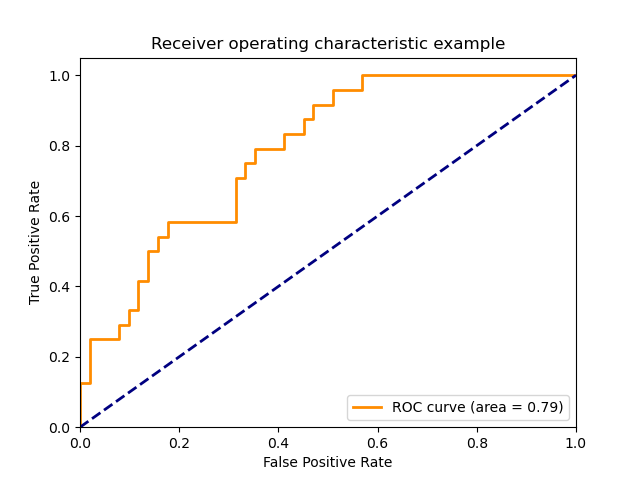


Рисунок 1. График пороговой функции ROC

Как видно, данная функция будет стремиться к левому верхнему углу и ROC AUC (площадь под данной кривой) будет стремиться к единице. То есть, чем больше площадь под кривой AUC, тем больше свободы в выборе порогового значения для наилучшей работы алгоритма классификации, ведь нам желательно максимизировать TPR и минимизировать FPR.

Предельный вид кривой ROC на рисунке 2:

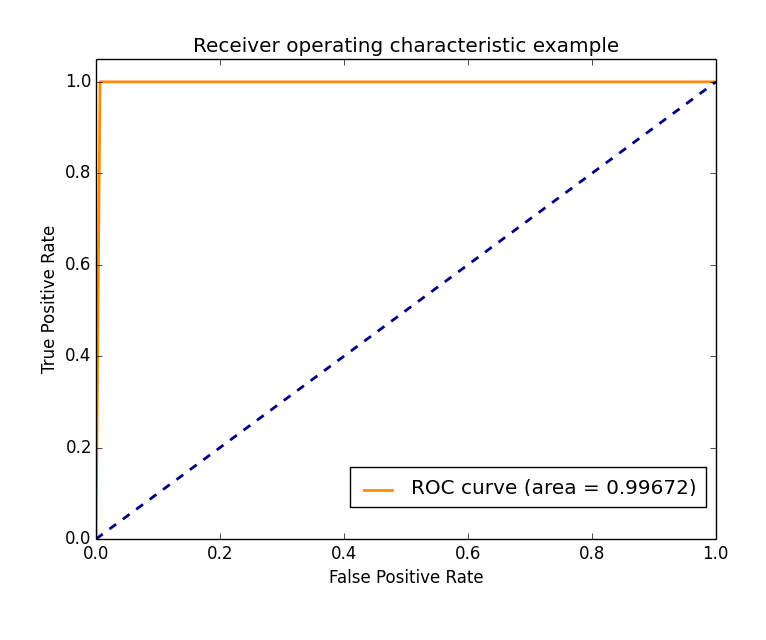


Рисунок 2. Предельный вид пороговой функции ROC

В случае классификатора, имеющий функцию ROC как на рисунке 2, можно выбрать порог классификатора так, чтобы TPR был равен единице, что значит все положительные метки определены верно, а показатель FPR стремится к нулю, что практически значит отсутствие ложно положительных прогнозов.

Добавим, что в сравнительном исследовании будет приниматься значение метрики, полученной на тестовой выборке.

## **Описание исходных данных**

Для исследования были взяты анонимизированые признаки. Набор данных содержит только категориальные функции и включает:

* бинарные признаки
* номинальные признаки с низкой и высокой частотой категорий
* порядковые признаки с низкой и высокой частотой категорий
* (потенциально) циклические признаки

Исходные данные составляют не только численные данные, но и строковые. Пример исходных данных на рисунке 3.

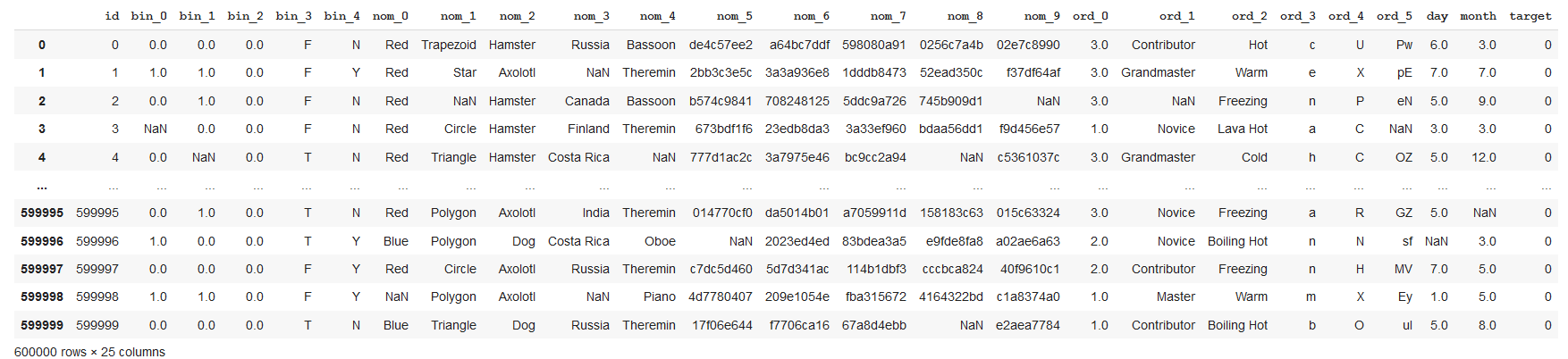


Рисунок 3. Исходные данные

Помимо признаковых переменных данные содержат и ответ (target). В каждой строке определенны значения переменной и соответствующий им ответ.

Стоит добавить, что каждая переменная имеет потери данных 3-4% и при исключении всех строк с потерями, количество данных сокращается вдвое.

Так же важным фактором является то, что присутствует сильное преобладание отрицательных ответов над положительными, диспропорция составлять 5:1. Но так как в данной работе функционалом качества определена площадь под кривой AUC, которая не чувствительна к такому роду проблем, никаких действий можно не предпринимать.

Если слепо факторизовать каждую переменную и построить корреляционную матрицу, то получим следующее на рисунке 4.

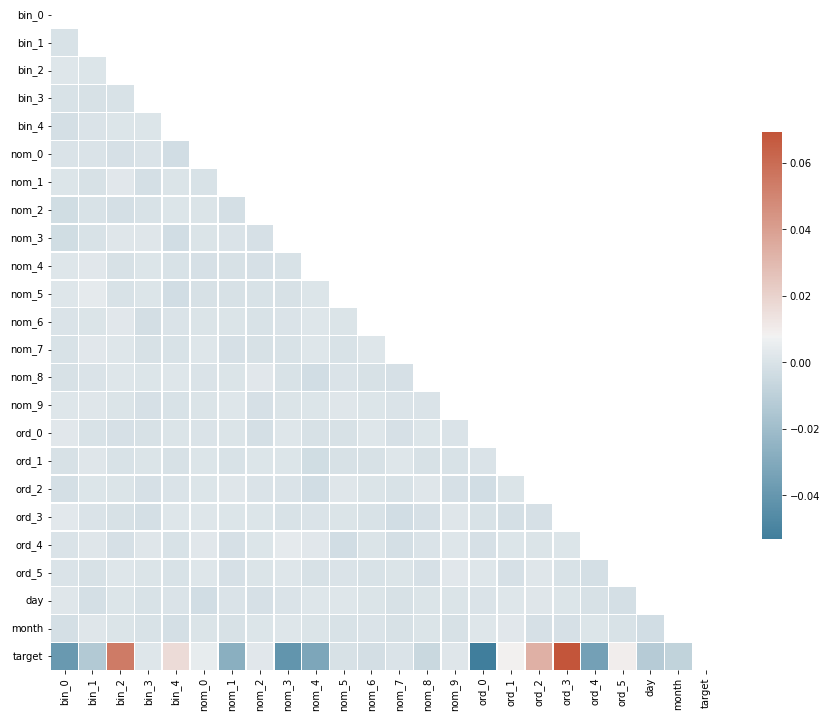


Рисунок 4. Корреляционная треугольная матрица.

Как видно, некоторые переменные выделяются на фоне остальных (bin\_2, ord\_0, ord\_3), имея наибольшие значения корреляции с target переменной, но данная сила стохастической связи все еще очень мала.

## **Базовая модель**

Перед проведением экспериментов следует определить базовую модель, от функционала качества которой нужно будет отталкиваться. Так как в нашем случае признаковое пространство исключительно категориальное, то для базовой модели будет взят вариант реализации градиентного бустинга от компаниия Яндекс – модель CatBoost.

Инициализация модели происходила с параметры по – умолчанию. Для оценки функционала качества модели была использована кросс валидация со стратификацией и был получен результат для ROC AUC в 0.7701

## **Экспериментальная установка**

Исследование методов категорийного кодирования реализовано на высокоуровневом языке программирования Python, с использованием библиотек scikit-learn, keras, tensorflow, XGBoost, CatBoost для кодирования признаков и реализации алгоритма бинарной классификации. Запуск программы совершен на бесплатной облачной платформе Google Colab, которая позволяет совершать неподъемные для домашней машины вычисления.

Для сравнения методов кодирования используется модель градиентного бустинга, реализованный в библиотеке XGBoost.

Чтобы избежать ложных выводов по результатам работы модели на тестовом множестве, в исследовании используется кросс валидация со стратификацией, при разбиении на 5 долей. По итогу кросс валидации будет браться средняя по функционалу качества, на основе которой и будет сравнение.



Рисунок 5.Схема стратифицированной кросс валидации

Также в исследовании участвует алгоритм логистической регрессии с параметрами по умолчанию.

Также для сравнения была использована нейронная сеть прямого распространения, состоящей из 5 слоев по 512 нейронов в каждом, с использованием нормализации и методов регуляризации[1][2][3]. На выходном слое функция активации – сигмоида. Обучение проходило с использованием бинарной перекрестной энтропии и методом оптимизации RMSProp.

Заполнение пропусков будет одинаковым для любого кодирования. Так как все данные категориальные и пропуски немногочисленные – около 3% на каждый признак, то для заполнения будет браться случайная целочисленная равномерная величина, соответствующая индексу уникального значения в признаке.

Все случайные генераторы инициализировались с начальным числом 42.

## **Методы кодирования**

В данной работе рассмотрены наиболее частые и стандартные методы категорийного кодирования[4]:

* Факторизация (кодирование метки);
* Горячее кодирование (One Hot Encoding);
* Хэширование.

Далее рассмотрим каждый метод подробнее.

Факторизация – каждому значению переменной будет присвоен свой порядковый номер. Данный метод наиболее распространен для перехода от строковых значений к численным, для дальнейшей обработки алгоритмом классификации. В данном методе пропускам в данных будет отведена отдельная категория.

Горячее кодирование – каждая переменная преобразуется в признаковый вектор размерности, равной количеству уникальных признаков. Тогда каждое значение будет кодироваться как вектор, состоящей из единицы в соответствующему значению столбце и нулей.

Хэширование – данный метод преобразует категориальные переменные в пространство целых чисел высших измерений, где расстояние между двумя векторами категориальных переменных приблизительно поддерживается в преобразованном числовом пространстве. При использовании хеширования число измерений будет намного меньше количества измерений с кодировкой, например One Hot Encoding. Однако данный метод позволяет и расширить пространство признаков.

Помимо методов кодирования также были разобраны методы обработки признаков:

* Скрещивание признаков;
* Восстановление семантической значимости.

Также рассмотрим каждый подробнее.

Скрещивание признаков – отличие от прошлого метода в том, что один признак может быть скрещен со многими другими одновременно. Данный метод будет реализован с помощью нейронной сети, реализующая множество скрещиваний благодаря своей природе.

Восстановление семантической значимости – слепое кодирование категориальных строковых признаков не всегда позволит достичь наивысшего показателя функционала качества. Если количество уникальных значений категориального признака не велико, то можно постараться закодировать его самостоятельно. Это поможет с такими признаками, градация уровней мастерства, градация температуры: холодно, тепло, горячо; иначе говоря, при кодирование сохраняется порядок значений. После преобразования данных корреляционная матрица на рисунке значительно изменилась.

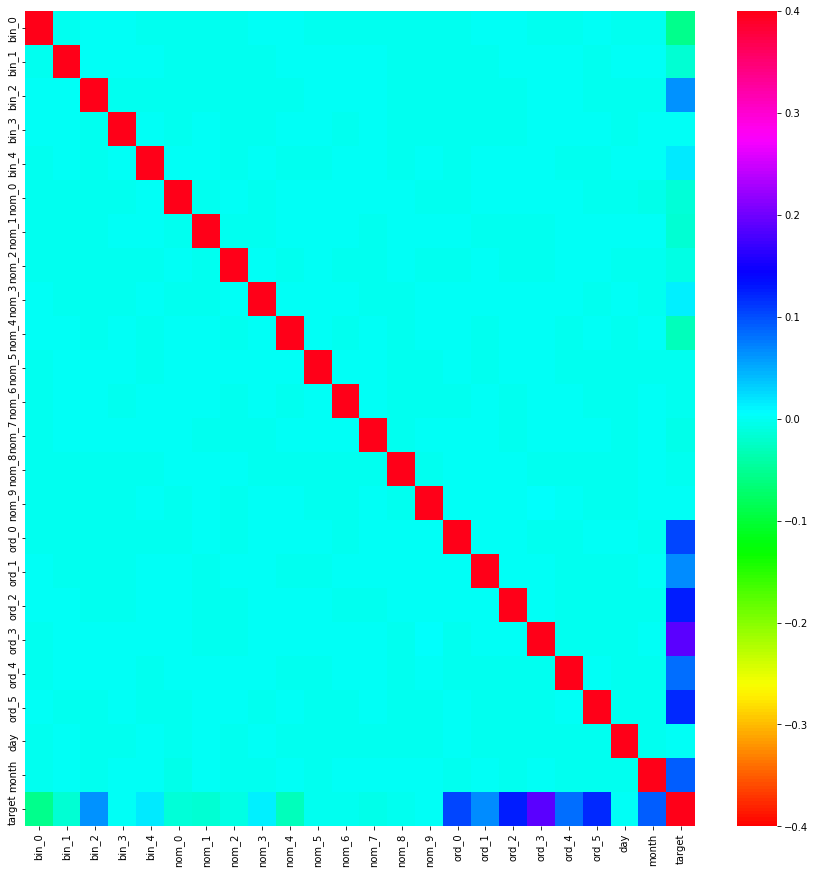


Рисунок 6.Корреляционная матрица признаков.

# **Результаты исследования**

С учетом всего вышесказанного были получены следующие результаты, описанные в таблице 2.

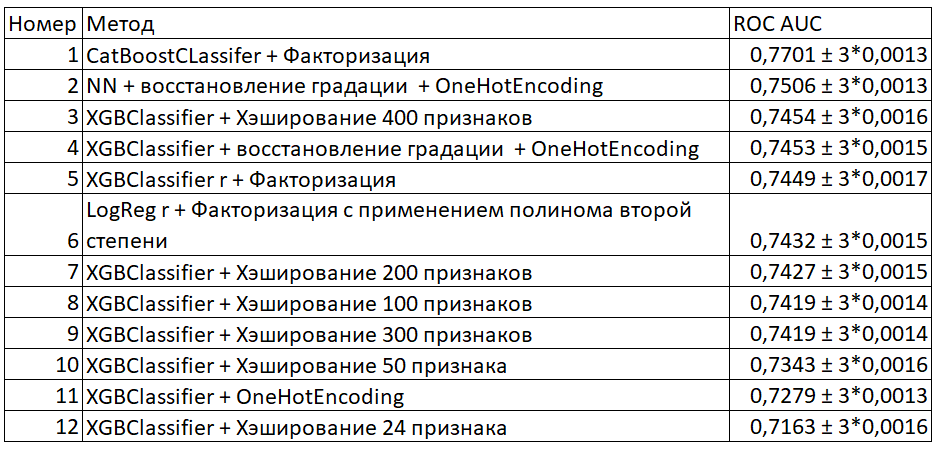


Таблица 2. Результаты экспериментов.

По результатам исследования лучший результат дает модель градиентного бустинга с реализацией Яндекс. Ее функционал качества достигает 0,7701 с СКО в 0,0013. К данному результату не приблизилась ни одна рассматриваемая модель с кодированием или без.

На втором месте оказалась нейронная сеть прямого распространения из пяти слоев по 512 нейронов и с учетом количества входных параметров модель состоит из 1 млн обучаемых коэффициентов модели. Данная модель достигла ROC AUC равному 0,7506 с СКО в 0,0013.

Занимательно, что градиентный бустинг с обычной факторизацией достигает значения ROC AUC в 0,7449 с СКО в 0,0017. Когда как при применении кодирование в превалирующем большинстве случаев значение функционала качества меньше данного.

# **Заключение**

В данной работе были рассмотрены методы кодирования категориальных признаков в задаче бинарной классификации, были рассмотрены такие методы, как факторизация (Label encoding), горячее кодирование (One Hot Encoding) и Хэширование (Hashing).

Также были рассмотрены методы обработки признаков, такие как скрещивание и восстановление цифрового порядка значений.

Все методы были исследованы в одинаковых условиях с заданным стартовым значением генератора случайных чисел и были получены результаты, отраженные в таблице.

На основании результатов исследования, можно сделать вывод, что градиентный бустинг позволяет достичь удовлетворительных значений функционала качества без особых усилий.

Но если исключить из рассмотрения разработку модели градиентного бустинга от Яндекс, то лучший результат приносит нейронная сеть прямого распространения с количеством обучаемых параметров в 1 млн., что не слишком много для современных мощностей ЭВМ.

**Список литературы**

1. Глубокое обучение / Ян Гудфеллоу, Иошуа Бенджио, Аарон Курвилль // ДМК Пресс, 2018г., второе цветное издание, исправленное.
2. Understanding the difficulty of training deep feedforward neural networks. Xavier GlorotYoshua BengioDIRO, Universit ́e de Montr ́eal, Montr ́eal, Qu ́ebec, Canada
3. Глубокое обучение. / Николенко С., Кадурин А., Архангельская Е. // СПб: Питер, 2018. — 480 с.: ил. — (Серия «Библиотека программиста»).
4. Методы кодирования / https://innovation.alteryx.com/encode-smarter/

# **Приложение**

"""CrossValSelectiveFactorizeCurse.ipynb

Automatically generated by Colaboratory.

"""

import tensorflow as tf

tf.random.set\_seed(42)

!pip install swifter

import pandas as pd

import numpy as np

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import roc\_auc\_score, accuracy\_score, roc\_curve

from keras.layers import Dense, Dropout, Input, BatchNormalization

from keras.models import Sequential, Model

from keras.regularizers import l2 # L2-regularisation

import gc

from sklearn.preprocessing import OneHotEncoder

from sklearn.model\_selection import train\_test\_split

from keras import callbacks

import swifter

import h5py

from tensorflow.keras.utils import Sequence

from sklearn.ensemble import ExtraTreesClassifier

import matplotlib.pyplot as plt

from sklearn.feature\_extraction import FeatureHasher

from sklearn.model\_selection import StratifiedKFold

from google.colab import drive

drive.mount('/content/drive')

df = pd.read\_csv('/content/drive/MyDrive/ML/DA/train.csv.zip', dtype = {'id':'str'})

df

df.dtypes

colls = df.iloc[:, 1:-1].columns.tolist()

for coll in colls:

print(df[coll].value\_counts())

print('\n\n')

"""#Исследование строковых признаков"""

colls = df.columns.tolist()

collMinMax = {}

for col in colls:

collMinMax[col] = (df[col].dropna().min(), df[col].dropna().max())

collMinMax

"""Есть подозрения, что признаки с nom\_5 по nom\_9 являются числами в 16-ой системе исчисления, давайте преобразуем их в десятичные числа"""

def get\_decimal(hex):

return (int(hex, 16) if(not pd.isna(hex)) else hex)

# if(pd.isna(hex)):

# return hex

# return l.index(hex)

colls = ['nom\_5',

'nom\_6',

'nom\_7',

'nom\_8',

'nom\_9']

for col in colls:

df[col] = df[col].swifter.apply(lambda x: get\_decimal(x))

df.describe()

"""Проверим данные на пропуски"""

colls = df.columns.tolist()

isNA = {}

for col in colls:

isNA[col] = any(df[col].isna())

isNA

"""Пропуски присутстсуют у каждого признака, кроме идентификатора и таргета"""

print('Количество строк без имзенений ', df.shape[0])

print('Количество строк после удаления всех пропусков ', df.dropna().shape[0])

"""Почти половина данных теряется. Рассмотрим целостноть каждого признака в частности"""

naColls = {k:v for k,v in isNA.items() if (v == True)}

naColls

colls = df.columns.tolist()

dropNaCount = {}

for col in naColls:

dropNaCount[col] = round(df[col].dropna().shape[0] / df.shape[0] \* 100, 2)

dropNaCount

"""Потери для каждого признака в частности незначительные, но видимо, относительно всей выборки, распределение пропусков построчно близко к равномерному и 3-4% пропусков каждого признака делает половину данных зашумленными.

Для бинарных признаков и порядковых признаков месяца и дня введем заполнение пропусков через равномерное распределение.

Для остальных категорий для заполнения пропусков отведем отдельную категорию

Для образование новой категории в каждом признаке возьмем его максимальное значение и увеличим на еденицу

"""

def get\_filled\_na(df):

bincolls = ['bin\_0', 'bin\_1', 'bin\_2', 'day', 'month', 'ord\_0']

for coll in bincolls:

min\_ = df[coll].min()

max\_ = df[coll].max()

df[coll] = df[coll].swifter.apply(lambda x: round(np.random.uniform(min\_, max\_)) if(np.isnan(x)) else x)

serialcolls = ['nom\_5', 'nom\_6', 'nom\_7', 'nom\_8', 'nom\_9']

for coll in serialcolls:

max\_ = df[coll].max()

df[coll] = df[coll].swifter.apply(lambda x: (max\_ + 1) if(np.isnan(x)) else x)

df[coll] = df[coll].factorize(sort = True)[0]

textserialColls = ['bin\_3', 'bin\_4', 'ord\_3', 'ord\_4', 'ord\_5', 'nom\_0', 'nom\_1', 'nom\_2', 'nom\_3', 'nom\_4']

for coll in textserialColls:

uniq = df[coll].dropna().unique().tolist()

n = len(uniq)

df[coll] = df[coll].swifter.apply(lambda x: uniq[round(np.random.uniform(0, n-1))] if(pd.isna(x)) else x)

df[coll] = df[coll].factorize(sort = True)[0]

textserialColls = ['ord\_1', 'ord\_2']

for coll in textserialColls:

uniq = df[coll].dropna().unique().tolist()

n = len(uniq)

df[coll] = df[coll].swifter.apply(lambda x: uniq[round(np.random.uniform(0, n-1))] if(pd.isna(x)) else x)

cat = {'Novice':0, 'Contributor':1, 'Expert':2, 'Master':3, 'Grandmaster':4}

df['ord\_1'] = df['ord\_1'].swifter.apply(lambda x: cat[x])

cat = {'Freezing':0, 'Cold':1, 'Warm':2, 'Hot':3, 'Boiling Hot':4, 'Lava Hot':5}

df['ord\_2'] = df['ord\_2'].swifter.apply(lambda x: cat[x])

return df

df = get\_filled\_na(df)

display(df.describe())

print(df.shape)

print(df.columns.tolist())

colls = df.columns.tolist()

dropNaCount = {}

for col in colls:

dropNaCount[col] = round(df[col].dropna().shape[0] / df.shape[0] \* 100, 2)

dropNaCount

"""Все пропуски восполнены

# BaseLine

Так как распределение target меток сильно смещено в сторону значения 0, просто сделаем все предсказанные ответы равные нулю

"""

y\_pred\_baseline = [0] \* df.shape[0]

print('Наш baseline ROC AUC:', round(roc\_auc\_score(df.target.values.tolist(), y\_pred\_baseline), 2))

"""# Экспериментальная установка

Так как мы не можем оценить линейность или нелинейность признаков из-за большой размерности, возьмем за экспериментальную установку 5 сверточных слоев с классификатором на выходном слое и методами регуляризации после каждой свертки. Этого должно более чем хватить для исслоедования категориального кодирования.

"""

def get\_model(num\_vars, num\_out):

units = 512

l2\_lambda = 0.0001

inputs = Input(shape=(num\_vars,))

#1

x = BatchNormalization()(inputs)

x = Dense(units, activation = 'relu', kernel\_regularizer=l2(l2\_lambda),

bias\_regularizer= l2(l2\_lambda),

activity\_regularizer = l2(l2\_lambda))(x)

x = Dropout(0.25)(x)

#2

x = Dense(units, activation = 'relu', kernel\_regularizer=l2(l2\_lambda),

bias\_regularizer= l2(l2\_lambda),

activity\_regularizer = l2(l2\_lambda))(x)

x = Dropout(0.25)(x)

#3

x = Dense(units, activation = 'relu', kernel\_regularizer=l2(l2\_lambda),

bias\_regularizer= l2(l2\_lambda),

activity\_regularizer = l2(l2\_lambda))(x)

x = Dropout(0.25)(x)

#4

x = Dense(units, activation = 'relu', kernel\_regularizer=l2(l2\_lambda),

bias\_regularizer= l2(l2\_lambda),

activity\_regularizer = l2(l2\_lambda))(x)

x = Dropout(0.25)(x)

#5

x = Dense(units, activation = 'relu', kernel\_regularizer=l2(l2\_lambda),

bias\_regularizer= l2(l2\_lambda),

activity\_regularizer = l2(l2\_lambda))(x)

x = Dropout(0.25)(x)

x = Dense(num\_out, activation = 'sigmoid', kernel\_regularizer=l2(l2\_lambda),

bias\_regularizer= l2(l2\_lambda),

activity\_regularizer = l2(l2\_lambda))(x)

return Model(inputs=inputs, outputs=x)

def get\_X\_Y(df):

X = df.iloc[:, 1:-1]

gc.collect()

Y = df.target

gc.collect()

return X, Y

from sklearn.ensemble import ExtraTreesClassifier

def hashing\_features(X, Y, n):

h = FeatureHasher(n\_features= n)

X\_fi = h.fit\_transform(X.drop(['nom\_5' ,'nom\_6' ,'nom\_7' ,'nom\_8', 'nom\_9', 'ord\_5', 'ord\_3', 'ord\_4'], axis = 1).to\_dict(orient = 'records'))

X\_fi = pd.DataFrame(X\_fi.toarray()).join(X[['nom\_5' ,'nom\_6' ,'nom\_7' ,'nom\_8', 'nom\_9', 'ord\_5', 'ord\_3', 'ord\_4']])

return X\_fi,Y

def OneHotEncoding(X, Y):

X\_fi = X.drop(['nom\_5' ,'nom\_6' ,'nom\_7' ,'nom\_8', 'nom\_9', 'ord\_5', 'ord\_3', 'ord\_4'], axis = 1)

X\_ = pd.DataFrame()

colls = X\_fi.columns.tolist()

for coll in colls:

gc.collect()

X\_ = pd.concat([X\_, pd.get\_dummies(X\_fi[coll])], axis = 1)

gc.collect()

X\_fi = pd.concat([X\_, X[['nom\_5' ,'nom\_6' ,'nom\_7' ,'nom\_8', 'nom\_9', 'ord\_5', 'ord\_3', 'ord\_4']]], axis = 1)

return X\_fi,Y

def get\_factorize\_X\_Y(df):

X = get\_factorize\_df(df.iloc[:, 1:-1])

Y = df.target

return X, Y

def get\_dummies\_X\_Y(df):

X = pd.get\_dummies(df.iloc[:, 1:-1])

Y = df.target

return X, Y

gc.collect()

X, Y = get\_X\_Y(df)

gc.collect()

X.head()

import seaborn as sns

fig, ax = plt.subplots(figsize = (15, 15))

d = X.join(df.target).corr()

ax = sns.heatmap(d, vmin = -0.4, vmax = 0.4, cmap = 'hsv')

display(X.head())

print(X.shape)

X\_train, X\_val, Y\_train, Y\_val = train\_test\_split(X, Y, test\_size = 0.25, random\_state=42)

gc.collect()

print('Распределение тестовых меток\n', dict(Y\_val.value\_counts()))

print('Распределение тренировочных меток\n',dict(Y\_train.value\_counts()))

display(X\_train.describe)

display(Y\_train.describe)

print(X\_val.columns.tolist())

display(X\_val.describe)

display(Y\_val.describe)

gc.collect()

"""Классы распределены более менее равномерно"""

X = (X - X.mean()) / X.std()

# количество эпох\итераций для обучения

model = get\_model(X\_train.shape[1], 1)

model.compile(loss='categorical\_crossentropy',

optimizer='rmsprop',

metrics= [tf.keras.metrics.AUC(), 'accuracy']

)

print(model.summary())

gc.collect()

class DummiesSequenceTrain(Sequence):

def \_\_init\_\_(self, batch\_size):

self.batch\_size = batch\_size

def \_\_len\_\_(self):

return int(np.floor(X\_train.shape[0] / self.batch\_size))

def \_\_getitem\_\_(self, idx):

batch\_x = X\_train.iloc[idx \* self.batch\_size:(idx + 1) \*

self.batch\_size].values

batch\_y = Y\_train.iloc[idx \* self.batch\_size:(idx + 1) \*

self.batch\_size].values

return batch\_x, batch\_y

class DummiesSequenceValid(Sequence):

def \_\_init\_\_(self, batch\_size):

self.batch\_size = batch\_size

def \_\_len\_\_(self):

return int(np.floor(X\_val.shape[0] / self.batch\_size))

def \_\_getitem\_\_(self, idx):

batch\_x = X\_val.iloc[idx \* self.batch\_size:(idx + 1) \*

self.batch\_size].values

batch\_y = Y\_val.iloc[idx \* self.batch\_size:(idx + 1) \*

self.batch\_size].values

return batch\_x, batch\_y

batch = 8192

train\_gen = DummiesSequenceTrain( int(batch))

val\_gen = DummiesSequenceValid( int(batch))

class GarbageCollectorCallback(tf.keras.callbacks.Callback):

def on\_epoch\_end(self, epoch, logs=None):

gc.collect()

def on\_epoch\_start(self, epoch, logs=None):

gc.collect()

Commented out IPython magic to ensure Python compatibility.

%%time

path = '/content/drive/MyDrive/ML/DA/WeightsDA/checkpoint\_NN'

callbacks = [

tf.keras.callbacks.EarlyStopping(patience=5, monitor='loss'), GarbageCollectorCallback(),

tf.keras.callbacks.EarlyStopping(patience=5, monitor='val\_loss'), GarbageCollectorCallback(),

tf.keras.callbacks.ModelCheckpoint(filepath = path

, monitor='val\_loss'

, verbose=1

, save\_best\_only= True

, save\_weights\_only= True

, mode = 'min')

]

epochs = 50

# gc.collect()

# kfold = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)

# model = XGBClassifier(verbosity = 1)

# results = cross\_val\_score(model, X.values, Y.values, cv=kfold, scoring = 'roc\_auc', verbose = 1)

# print("ROC AUC: %.2f%% (%.2f%%)" % (results.mean()\*100, results.std()\*100))

kfold = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)

cvscores = []

for train, val in kfold.split(X, df.target):

class DummiesSequenceTrain(Sequence):

def \_\_init\_\_(self, batch\_size):

self.batch\_size = batch\_size

def \_\_len\_\_(self):

return int(np.floor(len(X.values[train]) / self.batch\_size))

def \_\_getitem\_\_(self, idx):

batch\_x = (X.values[train])[idx \* self.batch\_size:(idx + 1) \*

self.batch\_size]

batch\_y = (Y.values[train])[idx \* self.batch\_size:(idx + 1) \*

self.batch\_size]

return batch\_x, batch\_y

class DummiesSequenceValid(Sequence):

def \_\_init\_\_(self, batch\_size):

self.batch\_size = batch\_size

def \_\_len\_\_(self):

return int(np.floor(len(X.values[val])/ self.batch\_size))

def \_\_getitem\_\_(self, idx):

batch\_x = (X.values[val])[idx \* self.batch\_size:(idx + 1) \*

self.batch\_size]

batch\_y = (Y.values[val])[idx \* self.batch\_size:(idx + 1) \*

self.batch\_size]

return batch\_x, batch\_y

train\_gen = DummiesSequenceTrain(int(batch))

val\_gen = DummiesSequenceValid(int(batch))

model = get\_model(X\_train.shape[1], 1)

model.compile(loss='binary\_crossentropy',

optimizer='rmsprop',

metrics= [tf.keras.metrics.AUC(), 'accuracy']

)

history = model.fit(x = train\_gen

, epochs=epochs

, callbacks=callbacks

, validation\_data = val\_gen )

# evaluate the model

scores = model.evaluate(x = val\_gen, verbose=0)

print(model.metrics\_names,'\n', scores)

cvscores.append(scores[1])

print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores)))

print("%.6f (+/- %.6f)" % (np.mean(cvscores), np.std(cvscores)))

history = history.history

import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 2, figsize = (30, 10))

ax[0].plot(history['loss'], label = 'Loss')

ax[1].plot(history['auc'], label = 'ROC AUC')

ax[0].plot(history['val\_loss'], label = 'Validation Loss')

ax[0].grid('both')

ax[0].legend()

ax[0].set\_xlabel('Эпохи')

ax[0].set\_ylabel('Функция потерь')

ax[1].plot(history['val\_auc'], label = 'Validation ROC AUC')

ax[1].legend()

ax[1].grid('both')

ax[1].set\_xlabel('Эпохи')

ax[1].set\_ylabel('ROC AUC')

import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 2, figsize = (30, 10))

ax[0].plot(history['loss'], label = 'Loss')

ax[1].plot(history['auc\_7'], label = 'ROC AUC')

ax[0].plot(history['val\_loss'], label = 'Validation Loss')

ax[0].grid('both')

ax[0].legend()

ax[0].set\_xlabel('Эпохи')

ax[0].set\_ylabel('Функция потерь')

ax[1].plot(history['val\_auc\_7'], label = 'Validation ROC AUC')

ax[1].legend()

ax[1].grid('both')

ax[1].set\_xlabel('Эпохи')

ax[1].set\_ylabel('ROC AUC')

plt.savefig('/content/drive/MyDrive/ML/DA/CrossValSelective\_Encoding\_OHE\_X.png')

"""# Self Sumbission"""

model.load\_weights(path)

results = model.predict(X)

#results = np.argmax(results,axis = 1)

results\_ = np.round(results)

print('ROC AUC', roc\_auc\_score(pd.get\_dummies(df.target), results\_, average=None))

print('Accuracy', accuracy\_score(pd.get\_dummies(df.target), results\_))

#results\_ = np.argmax(results,axis = 1)

# print('ROC AUC', roc\_auc\_score(df.target, results\_))

# print('Accuracy', accuracy\_score(df.target, results\_))

results

"""# Submission"""

model.load\_weights(path)

df\_test = pd.read\_csv('/content/drive/MyDrive/ML/DA/test.zip', dtype = {'id':'str'})

display(df\_test.head())

print(df\_test.shape)

def get\_decimal(hex):

return (int(hex, 16) if(not pd.isna(hex)) else hex)

# if(pd.isna(hex)):

# return hex

# return l.index(hex)

colls = ['nom\_5',

'nom\_6',

'nom\_7',

'nom\_8',

'nom\_9']

for col in colls:

df\_test[col] = df\_test[col].swifter.apply(lambda x: get\_decimal(x))

df\_test = get\_filled\_na(df\_test)

df\_test.describe().shape

gc.collect()

X\_test, Y = OneHotEncoding(df\_test.iloc[:, 1:], pd.DataFrame())

Y\_pred = model.predict(X\_test)

Y\_test = list(map(np.argmax, Y\_pred))

Y\_test

Y\_test = pd.DataFrame({'id':df\_test.id,

'target':Y\_test})

display(Y\_test.describe())

print(Y\_test.shape)

Y\_test.head()

Y\_test.to\_csv('/content/drive/MyDrive/ML/DA/OHE\_submission.csv', index = False)

Y\_test.target.value\_counts()