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 студент группы М80-101М-21
Фейзуллин К.М.

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

Оценка:

Москва, 2021

Оглавление

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:

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

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

$$Y = F(\alpha * X + \beta),$$

Где X – входной сигнал, Y – выходной сигнал, $F(*)$ – некая пороговая функций, α – масштабирующий коэффициент, а β – пороговое значение классификатора.

Последовательно варьируя пороговое значение β и находя в каждом

значении β значение TPR и FPR будет определена пороговая функция ROC вида, как на рисунке 1:

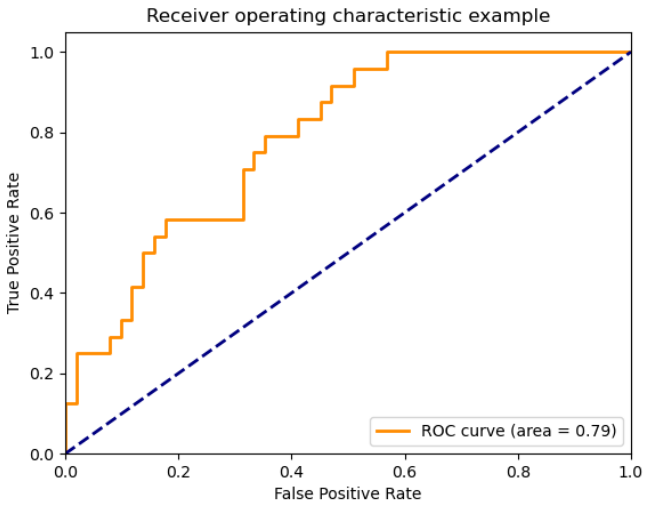

Рисунок 1. График пороговой функции ROC

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:

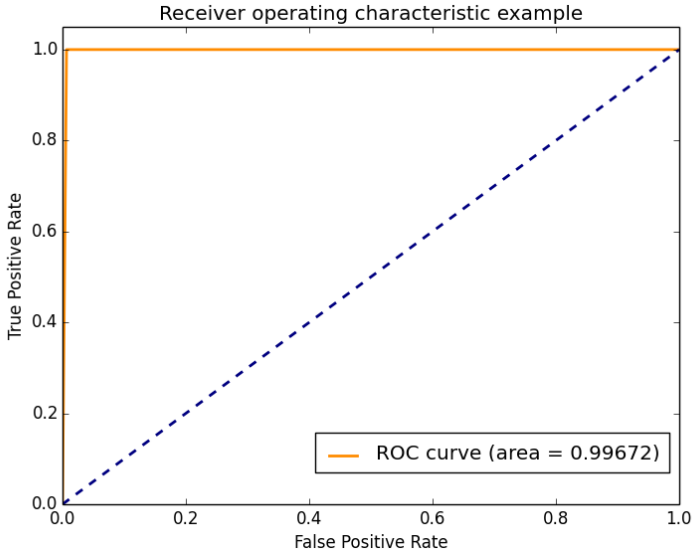

Рисунок 2. Предельный вид пороговой функции ROC

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.

	id	bin_0	bin_1	bin_2	bin_3	bin_4	nom_0	nom_1	nom_2	nom_3	nom_4	nom_5	nom_6	nom_7	nom_8	nom_9	ord_0	ord_1	ord_2	ord_3	ord_4	ord_5	day	month	target
0	0	0.0	0.0	0.0	F	N	Red	Trapezoid	Hamster	Russia	Bassoon	de4c57ee2	a64bc7ddf	598080a91	0256c7a4b	02e7c8990	3.0	Contributor	Hot	c	U	Pw	6.0	3.0	0
1	1	1.0	1.0	0.0	F	Y	Red	Star	Axolotl	NaN	Theremin	2bb3c3e5c	3a3a936e8	1dddb8473	52ead350c	f37df64af	3.0	Grandmaster	Warm	e	X	pE	7.0	7.0	0
2	2	0.0	1.0	0.0	F	N	Red	NaN	Hamster	Canada	Bassoon	b574c9841	708248125	5ddc9a726	745b909d1	NaN	3.0	NaN	Freezing	n	P	eN	5.0	9.0	0
3	3	NaN	0.0	0.0	F	N	Red	Circle	Hamster	Finland	Theremin	673bdf1f6	23edb8da3	3a33ef960	bdaa56dd1	f9d456e57	1.0	Novice	Lava Hot	a	C	NaN	3.0	3.0	0
4	4	0.0	NaN	0.0	T	N	Red	Triangle	Hamster	Costa Rica	NaN	777d1ac2c	3a7975e46	bc9cc2a94	NaN	c5361037c	3.0	Grandmaster	Cold	h	C	OZ	5.0	12.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
599995	599995	0.0	1.0	0.0	T	N	Red	Polygon	Axolotl	India	Theremin	014770cf0	da5014b01	a7059911d	158183c63	015c63324	3.0	Novice	Freezing	a	R	GZ	5.0	NaN	0
599996	599996	1.0	0.0	0.0	T	Y	Blue	Polygon	Dog	Costa Rica	Oboe	NaN	2023ed4ed	83bdea3a5	e9fde8fa8	a02ae6a63	2.0	Novice	Boiling Hot	n	N	sf	NaN	3.0	0
599997	599997	0.0	0.0	0.0	F	Y	Red	Circle	Axolotl	Russia	Theremin	c7dc5d460	5d7d341ac	114b1dbf3	cccbca824	40f9610c1	2.0	Contributor	Freezing	n	H	MV	7.0	5.0	0
599998	599998	1.0	1.0	0.0	F	Y	NaN	Polygon	Axolotl	NaN	Piano	4d7780407	209e1054e	fba315672	4164322bd	c1a8374a0	1.0	Master	Warm	m	X	Ey	1.0	5.0	0
599999	599999	0.0	0.0	0.0	T	N	Blue	Triangle	Dog	Russia	Theremin	17f06e644	f7706ca16	67a8d4ebb	NaN	e2aea7784	1.0	Contributor	Boiling Hot	b	O	ul	5.0	8.0	0

600000 rows × 25 columns

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.

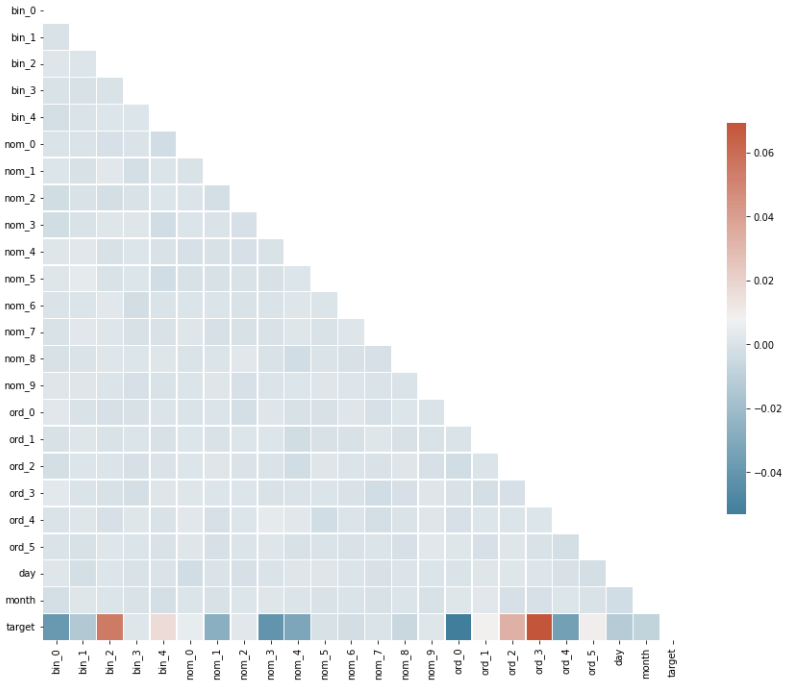

Рисунок 4. Корреляционная треугольная матрица.

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 ∓ 0.0013

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 совершатьнеподъемные 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.

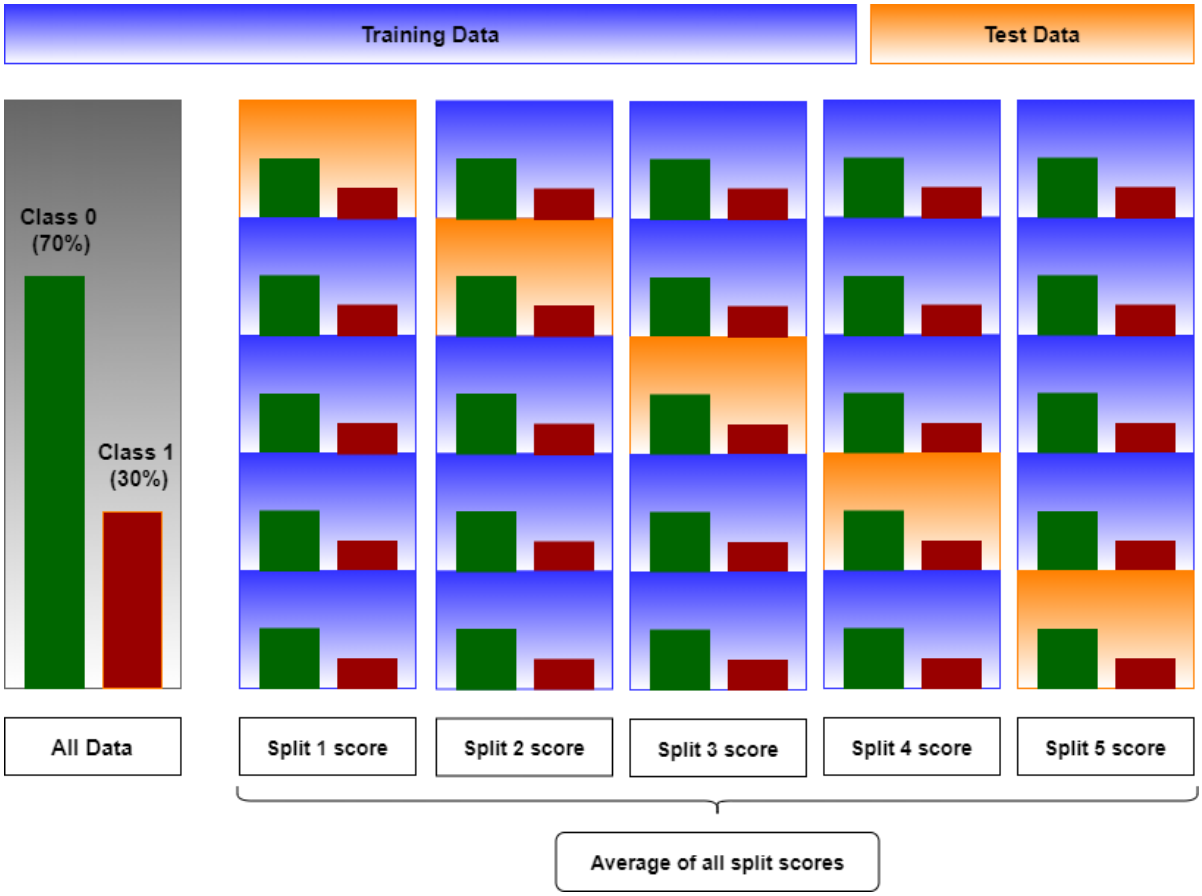

Рисунок 5.Схема стратифицированной кросс валидации

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.

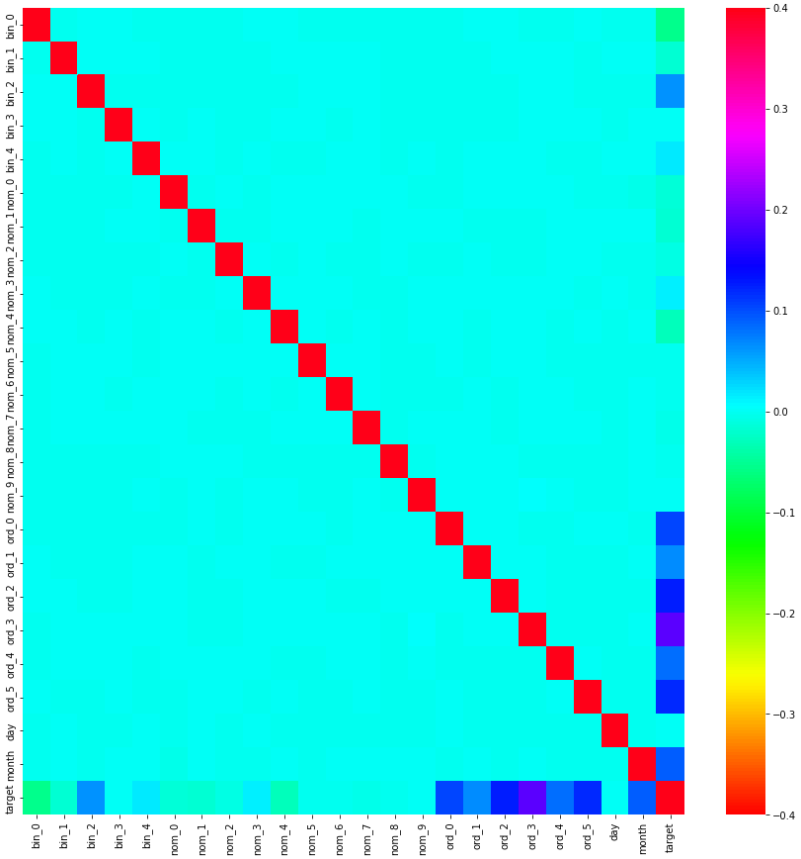

Рисунок 6.Корреляционная матрица признаков.

Результаты исследования

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

Номер	Метод	ROC AUC
1	CatBoostCLassifer + Факторизация	0,7701 ± 3*0,0013
2	NN + восстановление градации + OneHotEncoding	0,7506 ± 3*0,0013
3	XGBClassifier + Хэширование 400 признаков	0,7454 ± 3*0,0016
4	XGBClassifier + восстановление градации + OneHotEncoding	0,7453 ± 3*0,0015
5	XGBClassifier r + Факторизация	0,7449 ± 3*0,0017
6	LogReg r + Факторизация с применением полинома второй степени	0,7432 ± 3*0,0015
7	XGBClassifier + Хэширование 200 признаков	0,7427 ± 3*0,0015
8	XGBClassifier + Хэширование 100 признаков	0,7419 ± 3*0,0014
9	XGBClassifier + Хэширование 300 признаков	0,7419 ± 3*0,0014
10	XGBClassifier + Хэширование 50 признака	0,7343 ± 3*0,0016
11	XGBClassifier + OneHotEncoding	0,7279 ± 3*0,0013
12	XGBClassifier + Хэширование 24 признака	0,7163 ± 3*0,0016

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, Universit́ e de Montŕ eal, Montŕ eal, Qú 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()
```