



Московский институт электроники и
математики им. А.Н. Тихонова

Департамент
Прикладной математики

Москва
2024

Решение задач регрессии с помощью нейронных сетей

Морозов Денис студент «Прикладная математика»



Введение

В современном мире задачи регрессии играют важную роль в различных областях. Точные прогнозы и анализ данных становятся ключевыми элементами для принятия обоснованных решений. Один из способов решения задач регрессии является использование нейросетей. Есть много способов создания нейронных сетей. В этом проекте будет использоваться язык программирования Python. Мы рассмотрим данный способ на примере библиотек Keras и AutoKeras.

Цели:

1. Выбрать и предобработать датасет.
2. Сравнить различные модели нейросетей.
3. Использовать AutoML и сравнить результаты с нашей нейросетью.



Подключаем библиотеки

```
▶ !pip install category_encoders
```

```
▶ import numpy as np
import pandas as pd
import category_encoders as ce
import matplotlib.pyplot as plt

from tensorflow import keras

import statsmodels.api as sm

from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction import FeatureHasher
```

```
▶ from google.colab import drive
drive.mount('/content/drive')
```

```
⇒ Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[ ] url='https://drive.google.com/file/d/1rBP_uxoyafzrhQnXOPAENWiBVc6Z8W7F/view?usp=share_link'
url='https://drive.google.com/uc?id=' + url.split('/')[-2]
df = pd.read_csv(url)
df.head()
```

Тут подключаем, те библиотеки, которые нам нужны

А тут загружаем наши данные и смотрим содержимое



Содержимое датасета

"year" - год выпуска автомобиля,
"make" - марка машины, "model" -
модель машины, "trim" -
дополнительное обозначение для
модели автомобиля, "body" - тип
кузова автомобиля, "transmission" -
коробка передач, "vin" - уникальный
идентификатор, "state" - штат
регистрации автомобиля, "condition"
- состояние машины по шкале от 1
до 49, "odometer" - пробег
автомобиля, "color" - цвет
автомобиля, "interior" - цвет
интерьера автомобиля, "seller" -
продавец, "mmr" - ожидаемая
рыночная цена автомобиля,
"sellingprice" - цена продажи,
"saledate" - дата продажи.

	year	make	model	trim	body	transmission	vin	state	condition	odometer	color	interior	seller	mmr	sellingprice	saledate
0	2015	Kia	Sorento	LX	SUV	automatic	5xyktca69fg566472	ca	5.0	16639.0	white	black	kia motors america inc	20500.0	21500.0	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
1	2015	Kia	Sorento	LX	SUV	automatic	5xyktca69fg561319	ca	5.0	9393.0	white	beige	kia motors america inc	20800.0	21500.0	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
2	2014	BMW	3 Series	328i SULEV	Sedan	automatic	wba3c1c51ek116351	ca	45.0	1331.0	gray	black	financial services remarketing (lease)	31900.0	30000.0	Thu Jan 15 2015 04:30:00 GMT-0800 (PST)
3	2015	Volvo	S60	T5	Sedan	automatic	yv1612tb4f1310987	ca	41.0	14282.0	white	black	volvo na rep/world omni	27500.0	27750.0	Thu Jan 29 2015 04:30:00 GMT-0800 (PST)
4	2014	BMW	6 Series Gran Coupe	650i	Sedan	automatic	wba6b2c57ed129731	ca	43.0	2641.0	gray	black	financial services remarketing (lease)	66000.0	67000.0	Thu Dec 18 2014 12:30:00 GMT-0800 (PST)



Предобработка данных

```
[ ] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 558837 entries, 0 to 558836
Data columns (total 16 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   year        558837 non-null   int64  
 1   make         548536 non-null   object  
 2   model        548438 non-null   object  
 3   trim         548186 non-null   object  
 4   body          545642 non-null   object  
 5   transmission 493485 non-null   object  
 6   vin           558833 non-null   object  
 7   state         558837 non-null   object  
 8   condition     547017 non-null   float64 
 9   odometer      558743 non-null   float64 
 10  color          558088 non-null   object  
 11  interior       558088 non-null   object  
 12  seller         558837 non-null   object  
 13  mmr            558799 non-null   float64 
 14  sellingprice   558825 non-null   float64 
 15  saledate       558825 non-null   object  
dtypes: float64(4), int64(1), object(11)
memory usage: 68.2+ MB
```

```
df.isnull().sum()
```

	year	make	model	trim	body	transmission	vin	state	condition	odometer	color	interior	seller	mmr	sellingprice	saledate
	0	10301	10399	10651	13195	65352	4	0	11820	94	749	749	0	38	12	12

```
[ ] # выбросим все строки содержащие нулевые элементы
df.dropna(inplace=True)
# как видим все они удалились
df.isnull().sum()
```

	year	make	model	trim	body	transmission	vin	state	condition	odometer	color	interior	seller	mmr	sellingprice	saledate
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Тут удалим все строки содержащие хоть в одной ячейке нулевые значения

Посмотрим количество ненулевых данных и какие типы данных содержат колонки

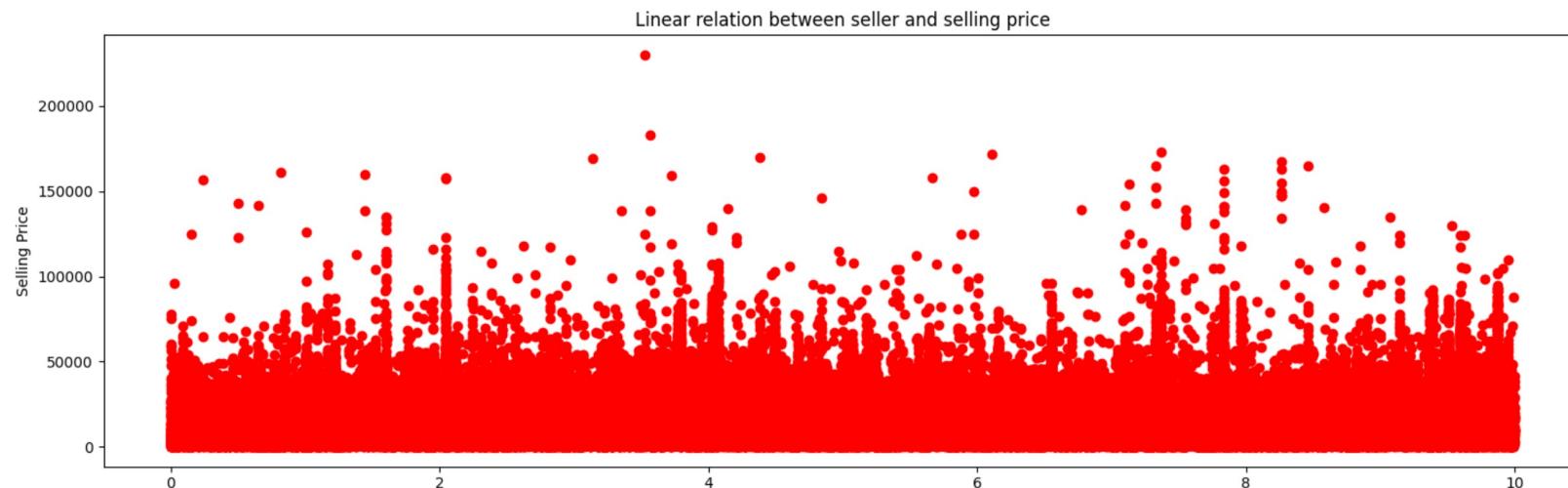
```
[ ] # посмотрим кол-во продавцов
df['seller'].nunique()

11923

▶ # тут захешируем наших продавцов
sellers = []
seller = list(df['seller'])
for el in seller:
    sellers.append((abs(hash(el)) % (10**8)) / 10000000)

[ ] pd.DataFrame(sellers).nunique() # уникальных значений осталось столько же

0    11921
dtype: int64
```



Рассмотрим более подробно колонку с продавцом. Как видим, тут у нас много данных относительно всех. И если посмотреть график зависимости цены автомобиля от продавца, то видно, что особо нет зависимости. Поэтому для нейросети будет лучше выкинуть данный столбец, так ее обучение пройдет лучше.



```
# так теперь обработаем дату продажи, если
def class_month(a):
    if a == 'Jan':
        return 0
    if a == 'Feb':
        return 31
    if a == 'Mar':
        return 59
    if a == 'Apr':
        return 90
    if a == 'May':
        return 120
    if a == 'Jun':
        return 151
    if a == 'Jul':
        return 181
    if a == 'Aug':
        return 212
    if a == 'Sep':
        return 243
    if a == 'Oct':
        return 273
    if a == 'Nov':
        return 304
    if a == 'Dec':
        return 334

sale_date = df['saledate']
n_sale_date = []
for el in sale_date:
    n_sale_date.append(el.split())

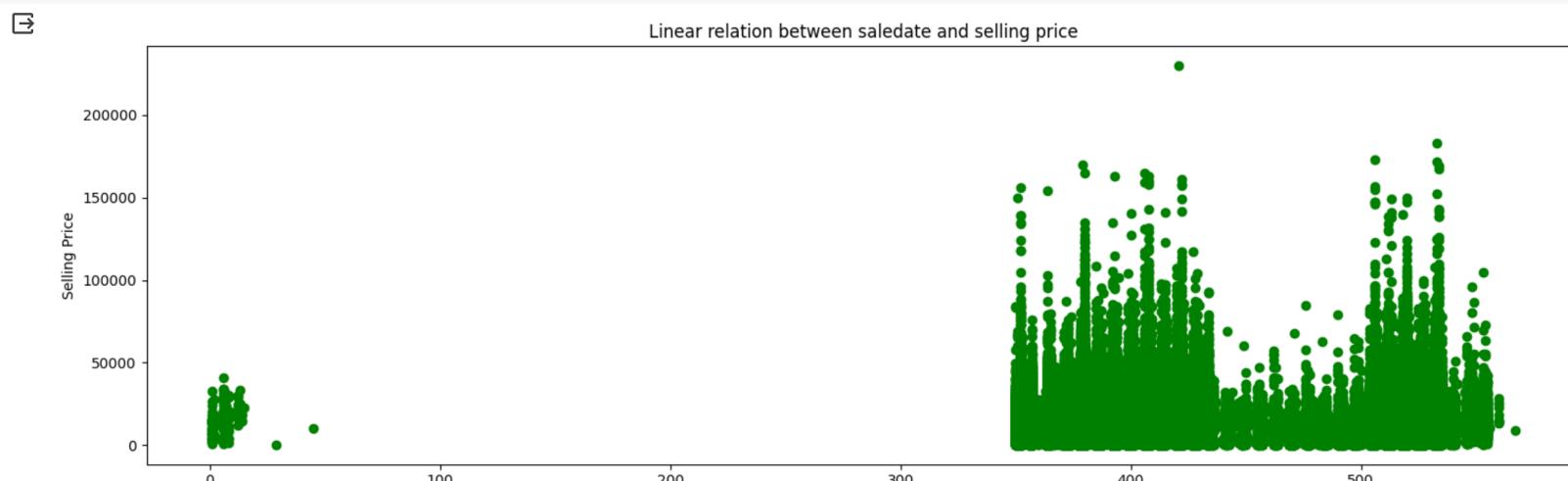
[ ] # пример одной правильно заполненной ячейки
n_sale_date[1]

['Tue', 'Dec', '16', '2014', '12:30:00', 'GMT-0800', '(PST)']

[ ] years = []
for el in n_sale_date:
    years.append(int(el[3]))

min_year = min(years)

f_sale_date = []
for el in n_sale_date:
    if len(el) == 7:
        f_sale_date.append(int(el[2]) + class_month(el[1]) + ((int(el[3]) - min_year) * 365))
    else:
        f_sale_date.append(0)
```



Теперь рассмотрим колонку, содержащую информацию о дате продажи автомобиля. Преобразовав дату в количество дней с минимального года продажи (также, если дата в неверном формате, то положим ее равной 0 дней), посмотрим на зависимость цены от даты продажи. Видим, что какой-то четкой нет, что и логично ведь данные с продажи где-то за 2 года. Есть просадка в середине, но там в принципе меньше данных о машинах. Поэтому удалим и эту колонку тоже.



```
#так как база данных слишком большая возьмем только ее часть
df = df.sample(frac=0.6, random_state=1, ignore_index=True)
#так же выкинем столбцы с вин, так как он у всех машин разный,
#продавцом и датой продажи, так как они не влияют
df.drop(['seller', 'vin', 'saledate'], axis=1, inplace=True)
print(df.shape)
df.head()
```

```
(283395, 13)
   year  make  model  trim  body  transmission  state  condition  odometer  color  interior  mmr  sellingprice
0  2014  Ford  Mustang  V6  Coupe  automatic    mo     39.0  25416.0  white  black  18300.0  17800.0
1  2012  Nissan  Altima  2.5 S  Sedan  automatic    mo     35.0  44298.0  burgundy  tan  11400.0  12800.0
2  2005 Chrysler      300      C  Sedan  automatic    il     19.0  80882.0  green  gray  7625.0  5900.0
3  2003  Nissan  Altima  2.5 SL  Sedan  automatic    pa     27.0  126454.0  blue  gray  2775.0  2600.0
4  2011  Nissan  Maxima  3.5 SV  Sedan  automatic    fl     47.0  45138.0  black  tan  16200.0  16200.0
```

```
[ ] #определим функцию которая будет нам выводить количество уникальных данных для каждой категории нашего датасета
categ = ['make', 'model', 'trim', 'body', 'state', 'color', 'interior', 'transmission']
def printUniqueAmount(df):
    for col in categ:
        print(f'{col}: {df[col].nunique()}')
```

Возьмем только 60% от всей базы данных, так как она очень большая. Так же еще удалим колонку "VIN", так как это уникально для каждой машины. И определим функцию, которая будет выводить количество уникальных данных для каждой категориальной колонки.



▶ printUniqueAmount(df)

```
make: 53
model: 749
trim: 1434
body: 85
state: 34
color: 20
interior: 17
transmission: 2
```

```
[ ] #тут мы смотрим если какая-то категория встречается меньше 10 раз, то мы такие заменяем на 'Other'
for col in categ:
    amount = df[col].value_counts()
    rare_categories = amount[amount <= 10].index.tolist()
    df[col] = df[col].apply(lambda x: 'Other' if x in rare_categories else x)
```

```
[ ] #теперь кол-во уникальных категорий:
printUniqueAmount(df)
```

```
make: 49
model: 585
trim: 797
body: 63
state: 34
color: 20
interior: 17
transmission: 2
```

Посмотрим сколько у нас уникальных значений в каждой категориальной колонке. Если какое-то значение в категориальной колонки встречается меньше 10 раз, то заменим его на 'Other'. Посмотрим сколько стало уникальных значений после.



```
#год выпуска машины меняем на сколько лет машине
df['cars_age'] = df['year'].apply(lambda x : 2015 - x)
#transmission переводим automatic - 0, manual - 1
df = pd.get_dummies(data=df, columns=['transmission'], drop_first=True, dtype=int)
#так же переводим наши категориальные данные с помощью BinaryEncoder
encoder = ce.binary.BinaryEncoder(cols=['make','body','interior','color','state'],drop_invariant=True).fit(df)
df = encoder.transform(df)
#так же преобразуем с помощью hasher текстовые данные
hasher = FeatureHasher(n_features=40, input_type='string')
hashed_features = hasher.transform(df[['model', 'trim']].astype(str).to_numpy())
hashed_features_df = pd.DataFrame(hashed_features.toarray())
hashed_features_df.columns = ['feature_' + str(i) for i in range(hashed_features_df.shape[1])]
#убираем year, model, trim, потому что мы их заменили другими
df.drop(['year', 'model', 'trim'], axis=1, inplace=True)
#склеим все получившиеся данные
df = pd.concat([df, hashed_features_df], axis=1)

df.head()
```

→

	make_0	make_1	make_2	make_3	make_4	make_5	body_0	body_1	body_2	body_3	...	feature_30	feature_31	feature_32	feature_33	feature_34	feature_35	fe
0	0	0	0	0	0	1	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0	0	0	0	1	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0	0	0	0	1	1	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0	0	0	0	1	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0	0	0	0	1	0	0	0	0	0	...	0.0	0.0	-1.0	0.0	0.0	0.0	0.0

5 rows × 74 columns

Тут переведем все данные в числа. Будем использовать хэширование и для категориальных колонок формат данных в виде вектора из 0 и 1. Более подробно на комментариях в коде.



Разделим на обучающую и тестовую выборки. В тестовой 20% данных. И нормализуем данные X.

```
[ ] #предсказывать будем стоимость машины
#разделяем на обучающий и тестовый наборы
X = df.drop(['sellingprice'], axis=1)
y = df['sellingprice']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

▶ #нормализуем наши данные для лучших результатов обучения
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

```
[ ] #посмотрим на наши получившиеся значения после нормализации
print("Набор для обучения X:\n", X_train)
print("Тестовый набор X:\n", X_test)
```

Набор для обучения X:
[[-0.15789894 1.78220326 -0.70851778 ... 0.00707289 0.11345535
-0.06124557]
[-0.15789894 -0.56110323 1.41139719 ... 0.00707289 0.11345535
6.18216708]
[-0.15789894 -0.56110323 1.41139719 ... 0.00707289 0.11345535
-0.06124557]
...
[-0.15789894 -0.56110323 -0.70851778 ... 0.00707289 0.11345535
-0.06124557]
[-0.15789894 -0.56110323 -0.70851778 ... 0.00707289 0.11345535
-0.06124557]
[-0.15789894 -0.56110323 1.41139719 ... 0.00707289 0.11345535
-0.06124557]]

Тестовый набор X:
[[-0.15789894 1.78220326 1.41139719 ... 0.00707289 0.11345535
-0.06124557]
[-0.15789894 1.78220326 1.41139719 ... 0.00707289 0.11345535
-0.06124557]
[-0.15789894 1.78220326 1.41139719 ... 0.00707289 -6.86677369
-0.06124557]]



Функция для оценки результатов работы нейронной сети

```
▶ #функция для оценки результатов
def print_res(pred, y_test, pred_train, y_train):
    delta = pred_train - y_train
    absDelta = abs(delta)
    print("Средняя ошибка для обучающего набора: ")
    print(sum(absDelta) / len(absDelta))

    delta = pred - y_test
    absDelta = abs(delta)
    print("Средняя ошибка для тестового: ")
    print(sum(absDelta) / len(absDelta))

    plt.scatter(y_test, pred, label='test')
    plt.scatter(y_train, pred_train, label='train')
    plt.xlabel('Правильные значение')
    plt.ylabel('Предсказания')
    plt.legend()
    plt.axis('equal')
    plt.xlim(plt.xlim())
    plt.ylim(plt.ylim())
    plt.show()

    print("Для обучающего набора:")
    print('Реальное значение:\n', y_train[:5], '\nПредсказанное значение:\n', pred_train[:5])
    print("Для тестовых:")
    print('Реальное значение\n', y_test[:5], '\nПредсказанное значение\n', pred[:5])
```

Напишем функцию которая будем нам выводить предсказания нейросети и реальные значения для тестового и тренировочного наборов. Также функция будет выводить средние значение ошибки для тренировочного и тестового набора.



Создаем и обучаем нейросеть

```
▶ #создаем модель нейронки
model = keras.Sequential([
    keras.Input(shape=(73,)),
    keras.layers.Dense(120, activation='relu'),
    keras.layers.Dense(80, activation='relu'),
    keras.layers.Dense(60, activation='relu'),
    keras.layers.Dense(10, activation='relu'),
    keras.layers.Dense(1),
])
```

```
model.compile(optimizer=keras.optimizers.Adam(learning_rate=2e-4), loss='mean_squared_error', metrics=['mae'])

history = model.fit(X_train, y_train, epochs=100, validation_split=0.2)
```

```
Epoch 100/100
5668/5668 [=====] - 16s 3ms/step - loss: 2275703.7500 - mae: 942.4656 - val_loss: 2502090.2500 - val_mae: 992.6893
```

Тут ошибки на последней эпохе

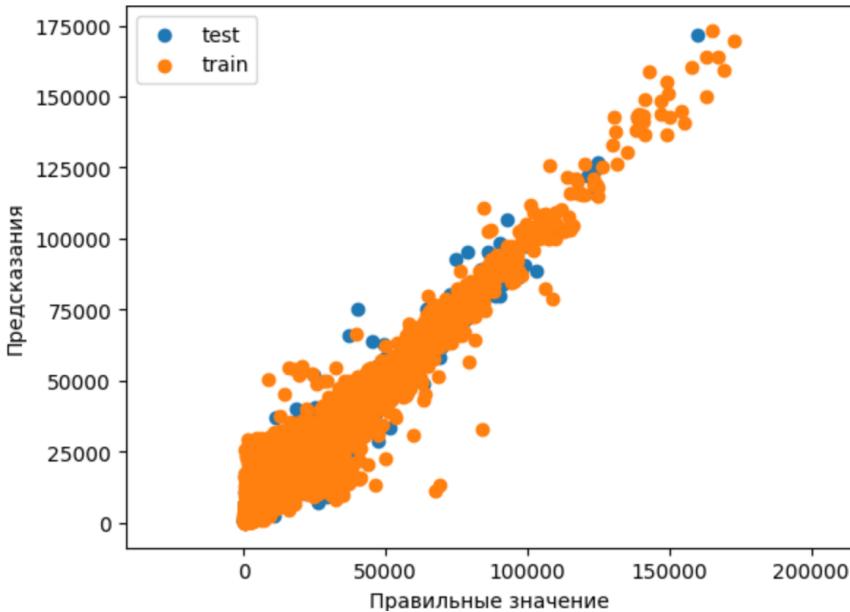
```
▶ #теперь посмотрим нашу модель на тестовых данных и ошибки которые она дает
test_loss = model.evaluate(X_test, y_test)
test_loss
```

```
→ 1772/1772 [=====] - 3s 1ms/step - loss: 2339773.2500 - mae: 984.4457
[2339773.25, 984.4456787109375]
```

Теперь посмотрим ошибки на тестовых данных



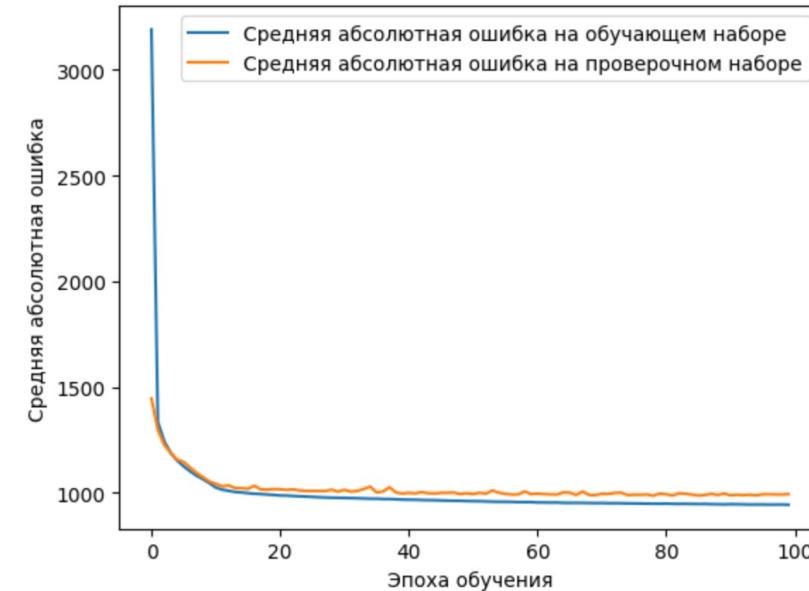
Результаты



```
1772/1772 [=====] - 3s 2ms/step
7085/7085 [=====] - 10s 1ms/step
Средняя ошибка для обучающего набора:
949.4937874271596
Средняя ошибка для тестового:
984.4448652974648
```

Результаты получили
с помощью написанной
нами функции

Для обучающего набора:
Реальное значение:
258530 19000.0
14229 13100.0
158367 10500.0
11039 5750.0
142257 18800.0
Name: sellingprice, dtype: float64
Предсказанное значение:
[20277.18 13057.574 9884.089 12471.557 19155.719]
Name: sellingprice, dtype: float64
Предсказанное значение
[11833.121 9583.675 20136.857 15420.583 11262.234]





Нормализуем данные y

```
▶ #нормализуем данные y_train
yScaler = StandardScaler()

yScaler.fit(np.array(y_train).reshape(-1, 1))

#нормализуем по нормальному распределению
yTrainScaled = yScaler.transform(np.array(y_train).reshape(-1, 1))

print(yTrainScaled.shape)
print(y_train[1])
print(yTrainScaled[1])
```

```
☒ (226716, 1)
12800.0
[-0.05949141]
```

Нормализуем колонку, содержащую цену автомобиля, таким же методом, как и данные X .

```
[ ] #нормализуем данные y_test
yScalerTest = StandardScaler()

yScalerTest.fit(np.array(y_test).reshape(-1, 1))

#нормализуем поциальному распределению
yTestScaled = yScalerTest.transform(np.array(y_test).reshape(-1, 1))

print(yTestScaled.shape)
print(y_train[1])
print(yTestScaled[1])

(56679, 1)
12800.0
[0.01457666]
```



Создаем и обучаем нейросеть на нормализованных y

▶ #создаем модель нейронки

```
models = keras.Sequential([
    keras.Input(shape=(73,)),
    keras.layers.Dense(120, activation='relu'),
    keras.layers.Dense(80, activation='relu'),
    keras.layers.Dense(60, activation='relu'),
    keras.layers.Dense(10, activation='relu'),
    keras.layers.Dense(1),
])

models.compile(optimizer=keras.optimizers.Adam(learning_rate=1e-5), loss='mean_squared_error', metrics=['mae'])

historyS = models.fit(X_train, yTrainScaled, epochs=100, validation_split=0.2)
```

Используем такую же модель нейросети.

Нейросеть из 5 слоев Dense, все с активацией "relu".

Лучшие результаты получаем при optimizer "Adam" с learning rate $1*10^{-5}$. Берем 100 эпох: дальше ошибка растет.
"Validation split" = 0.2.

```
-----  
Epoch 100/100  
5668/5668 [=====] - 16s 3ms/step - loss: 0.0254 - mae: 0.1024 - val_loss: 0.0292 - val_mae: 0.1092
```

Тут ошибки на последней эпохе

▶ #теперь посмотрим нашу модель на тестовых данных и ошибки которые она дает

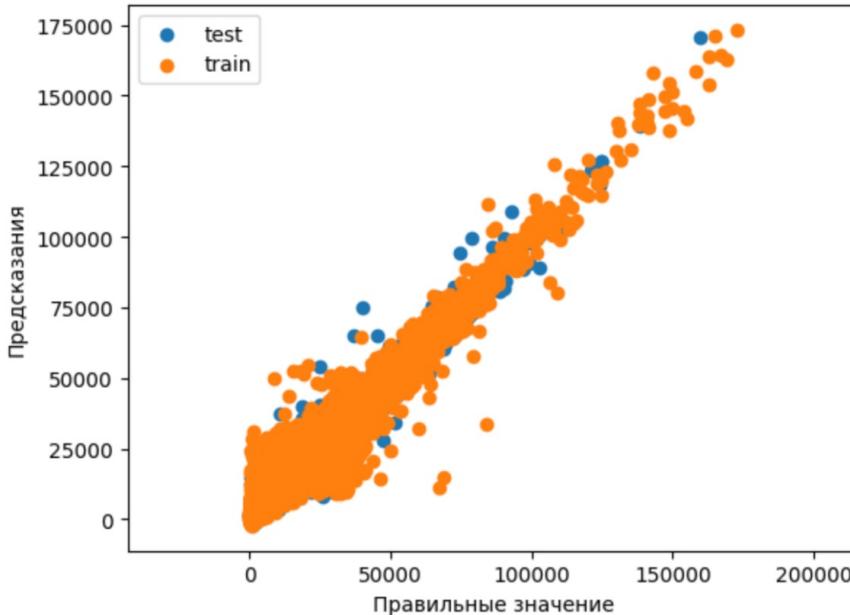
```
test_loss_S = models.evaluate(X_test, yTestScaled)  
test_loss_S
```

Теперь посмотрим ошибки на тестовых данных

⇒ 1772/1772 [=====] - 3s 2ms/step - loss: 0.0278 - mae: 0.1091
[0.02775326371192932, 0.10906072705984116]



Результаты



1772/1772 [=====] - 4s 2ms/step

7085/7085 [=====] - 11s 2ms/step

Средняя ошибка для обучающего набора:

995.6243831840126

Средняя ошибка для тестового:

1041.6329079816458

Для обучающего набора:

Реальное значение:

258530 19000.0

14229 13100.0

158367 10500.0

11039 5750.0

142257 18800.0

Name: sellingprice, dtype: float64

Предсказанное значение:

[10767.873 9596.717 19530.602 16111.357 11323.871]

Результаты получили
с помощью написанной
нами функции

Для тестовых:

Реальное значение

245631 10200.0

84047 13900.0

148609 20300.0

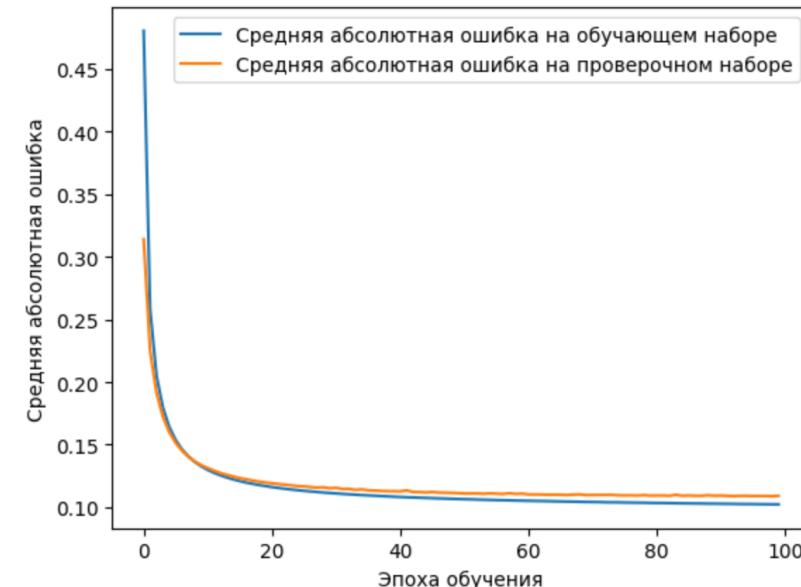
18574 15400.0

137346 12900.0

Name: sellingprice, dtype: float64

Предсказанное значение

[19869.287 13718.192 10545.157 12923.345 19695.512]





AutoML

!pip install autokeras
!!!!тут надо будет согласится с перезапуском, импортировать autokeras, а затем запустить все клетки, кроме секции (Создаем и обучаем нейросеть, нейросеть на нормализованных данных) !!!

#будем использовать AutoKeras
import autokeras as ak

Обучим на нормализованных данных

```
[ ] #тут создадим модель AutoKeras
modelAS = ak.AutoModel(
    inputs=[ak.Input()],
    outputs=[ak.RegressionHead()],
    max_trials=5
)

history_AS = modelAS.fit(
    [X_train],
    [yTrainScaled],
    epochs=150
)
```

```
100/100 [00:00, 20s/step - loss: 0.0844 - mean_squared_error: 0.0844
Epoch 150/150
7085/7085 [00:00, 21s 2ms/step - loss: 0.0840 - mean_squared_error: 0.0840
```

```
[ ] #теперь посмотрим нашу модель на тестовых данных и ошибки которые она дает
test_loss = modelAS.evaluate(np.array(X_test), np.array(yTestScaled))
test_loss
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/saving/saving_lib.py:418: UserWarning: Skipping variable loading for optimizer 'adam', because it has 2 va
    trackable.load_own_variables(weights_store.get(inner_path))
1772/1772 [00:00, 3s 1ms/step - loss: 0.0342 - mean_squared_error: 0.0342
[0.0348929725587368, 0.0348929725587368]
```

AutoKeras сам подбирает модель, которая работает лучше всего.

По умолчанию он делает до 100 попыток и до 1000 эпох. Он в какой-то момент может сам остановить на n-ной эпохе, если в течении 1- эпох ошибка ухудшается. Или может остановить на n-ной попытке подбора модели нейросети.

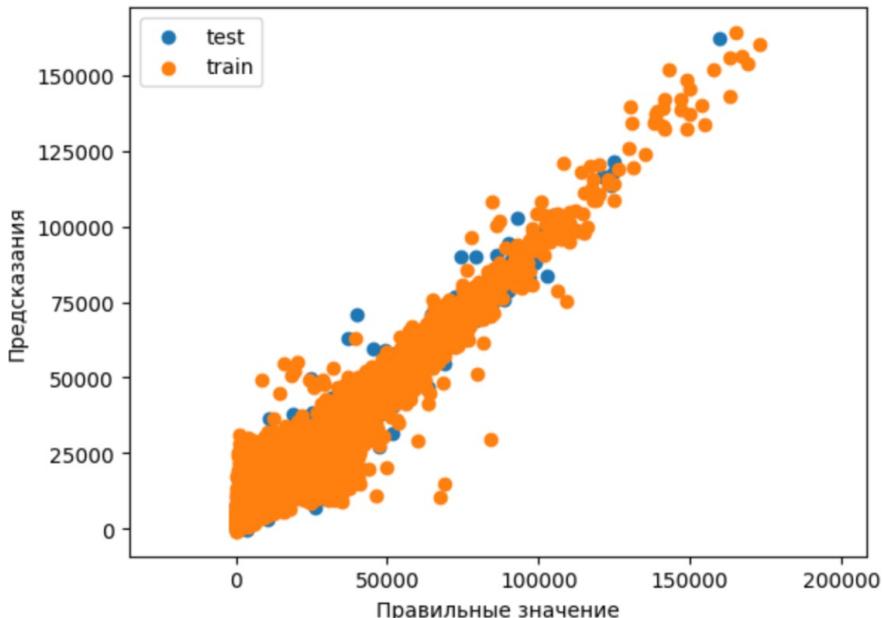
Я тут ограничил данные параметры: максимум 5 попыток и 150 эпох.

Все остальное автоматический. Тут обучаем на нормализованных y и X.

Тут ошибки на последней эпохе наилучшей модели

Теперь посмотрим ошибки на тестовых данных

Результаты



Для обучающего набора:

Реальное значение:

258530 19000.0

14229 13100.0

158367 10500.0

11039 5750.0

142257 18800.0

Name: sellingprice, dtype: float64

Предсказанное значение:

[10607.898 9699.511 19867.607 15923.122 11330.891]

Предсказанное значение:

[19036.193 13869.311 10423.389 12240.072 18821.832]

Для тестовых:

Реальное значение

245631 10200.0

84047 13900.0

148609 20300.0

18574 15400.0

137346 12900.0

Name: sellingprice, dtype: float64

Предсказанное значение

[10607.898 9699.511 19867.607 15923.122 11330.891]

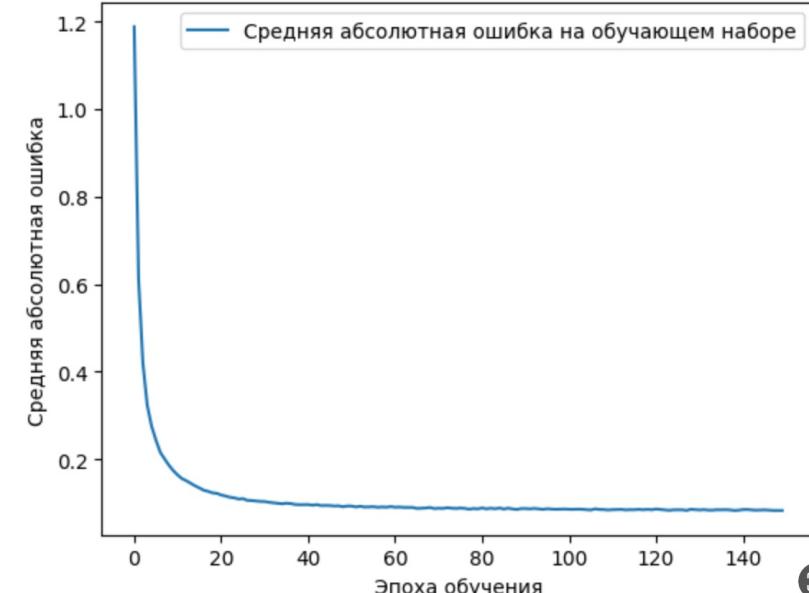
Результаты получили
с помощью написанной
нами функции

Средняя ошибка для обучающего набора:

1257.4640617327027

Средняя ошибка для тестового:

1267.7258796157505





AutoML

```
[ ] #тут создадим модель AutoKeras
modelA = ak.AutoModel(
    inputs=[ak.Input()],
    outputs=[ak.RegressionHead()],
    max_trials=3
)
```

```
history_A = modelA.fit(
    [X_train],
    [np.array(y_train)]
)
```

```
Epoch 342/342
7085/7085 ━━━━━━━━━━━━ 3s 372us/step - loss: 2035549.3750 - mean_squared_error: 2035549.3750
```

Тут я уже обучал на не нормализованных данных и не ограничивал количество эпох (т.е. максимум 1000 автоматический) и поставил максимум 3 попытки.

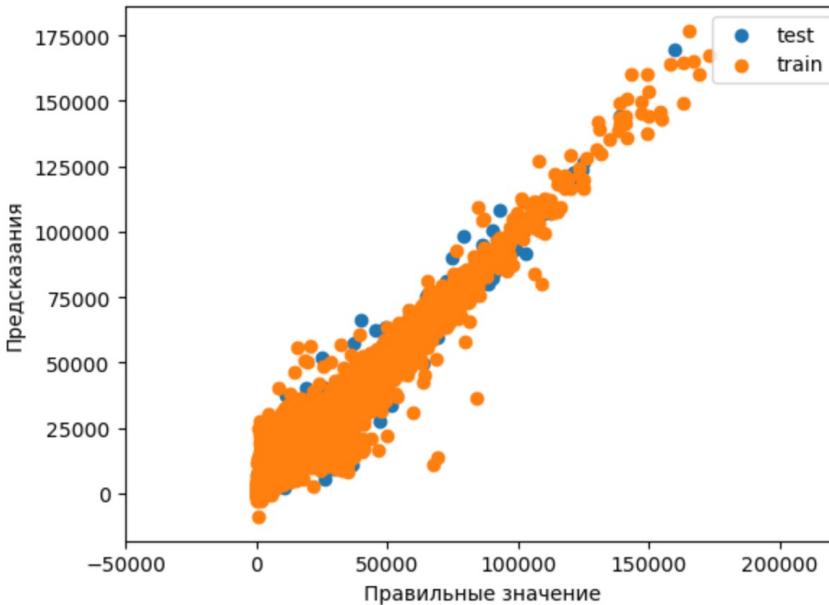
Тут ошибки на последней эпохе наилучшей модели

```
[ ] #теперь посмотрим нашу модель на тестовых данных и ошибки которые она дает
test_loss = modelA.evaluate(np.array(X_test), np.array(y_test))
test_loss
```

Теперь посмотрим ошибки на тестовых данных

```
194/1772 ━━━━━━━━━━ 0s 259us/step - loss: 2144696.7500 - mean_squared_error: 2144696.7500 /opt/anaconda3/lib/python3.11/site-packages/keras
saveable.load_own_variables(weights_store.get(inner_path))
1772/1772 ━━━━━━━━━━ 1s 250us/step - loss: 2119707.2500 - mean_squared_error: 2119707.2500
[2158839.25, 2158839.25]
```

Результаты



Для обучающего набора:

Реальное значение:

[19000. 13100. 10500. 5750. 18800.]

Предсказанное значение:

[19980.111 13757.781 10626.924 9889.385 19191.916]

Для тестовых:

Реальное значение

[10200. 13900. 20300. 15400. 12900.]

Предсказанное значение

[11669.423 9433.026 20259.572 16174.885 11604.96]

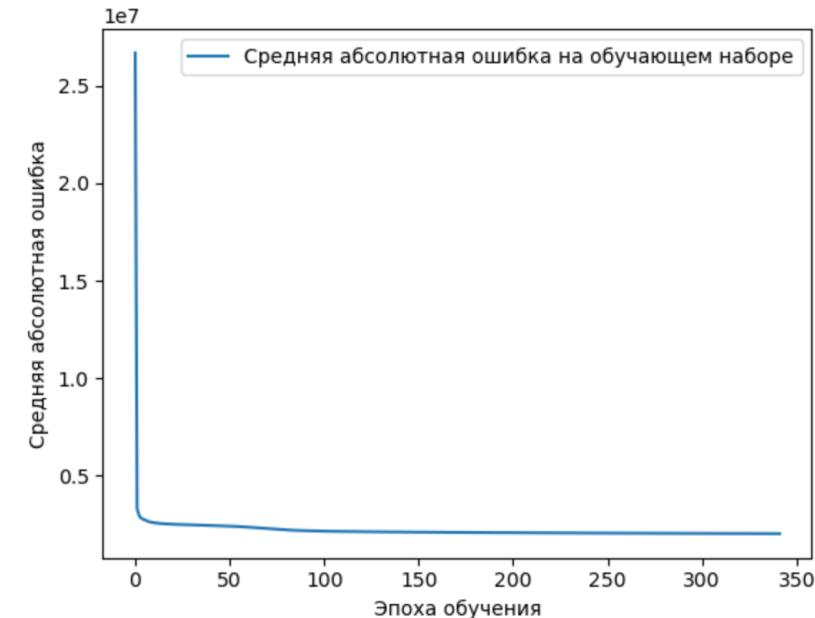
Результаты получили
с помощью написанной
нами функции

Средняя ошибка для обучающего набора:

923.13373899565

Средняя ошибка для тестового:

951.6978463716204





Вывод

Лучшие значения получили AutoKeras ненормализованных данных, но в нашей первой модели результаты почти такие же. С помощью нейросетей задачи регрессии решаются довольно хорошо, но в различных ситуациях результат может сильно ухудшаться, когда в датасете очень много некорректных данных или он в принципе плохо предобработан. Более того, на результат будет влиять и нормализация данных. Причем, казалось бы, с помощью нормализованных данных нейросеть должна обучаться лучше, но в нашем случае, когда мы нормализовали еще и данные, которые являлись результатом (цена автомобиля), получилось наоборот. Все полученные модели нейронных сетей, описанные здесь, выдают примерно одинаковые результаты. При решении задач регрессии с помощью нейронных сетей стоит рассматривать несколько вариантов, так как с первого раза могут получиться плохие результаты.



Московский институт электроники и
математики им. А.Н. Тихонова

Департамент
Прикладной математики

Москва
2024

Решение задач регрессии с помощью нейронных сетей

Спасибо за внимание