

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»  
**МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОНИКИ И МАТЕМАТИКИ**

**РЕШЕНИЕ ЗАДАЧ РЕГРЕССИИ С ПОМОЩЬЮ НЕЙРОННЫХ СЕТЕЙ**

ПРОЕКТ СТУДЕНТА 1 КУРСА ОБРАЗОВАТЕЛЬНОЙ ПРОГРАММЫ БАКАЛАВРИАТА  
«ПРИКЛАДНАЯ МАТЕМАТИКА»  
ПО НАПРАВЛЕНИЮ ПОДГОТОВКИ 01.03.04 ПРИКЛАДНАЯ МАТЕМАТИКА

Студент  
Морозов Д.С.

Руководитель проекта  
Доктор физико-математических наук, профессор  
Виктор Юрьевич Попов

**Москва 2024г.**

# Содержание

<b>1</b>	<b>Аннотация</b>	<b>2</b>
<b>2</b>	<b>Введение</b>	<b>2</b>
<b>3</b>	<b>Датасет</b>	<b>2</b>
<b>4</b>	<b>Предобработка данных</b>	<b>5</b>
<b>5</b>	<b>Создаем и обучаем нейросети</b>	<b>11</b>
<b>6</b>	<b>AutoML</b>	<b>15</b>
<b>7</b>	<b>Вывод</b>	<b>18</b>
<b>8</b>	<b>Приложения</b>	<b>19</b>

# 1 Аннотация

В этом проекте был взят датасет и на его основе была решена задача регрессии с помощью нейронных сетей. Различные модели нейронных сетей были использованы и сравнены. Датасет содержит различные характеристики автомобилей и ее финальную стоимость, которую мы и хотим предсказать. Сначала мы предобработали данные, удалив некорректные из них и превратив все в числа. Также в проекте используется *AutoML* и производится также сравнение результатов, полученных и с помощью него.

## 2 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* [1] и *AutoKeras* [4].

**Цели:**

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

## 3 Датасет

Для решения задач регрессии с помощью нейронных сетей будем использовать датасет, представленный в таблице 1. Он был взят с сайта [kaggle.com](https://www.kaggle.com)

Таблица 1: Датасет

Nº	year	make	model	trim	body	transmission	vin
1	2015	Kia	Sorento	LX	SUV	automatic	5xy...
...	...	...	...	...	...	...	...
23	2014	BMW	5 series	528i	Sedan	automatic	wba...
...	...	...	...	...	...	...	...
Nº	state	condition	odometr	color	interior	seller	mmr
1	ca	5	16639	white	black	kia motors...	20500
...	...	...	...	...	...	...	...
23	ca	29	25969	black	black	financial services...	34200
...	...	...	...	...	...	...	...
Nº	selling price	selling date					
1	21500	Tue Dec 16...					
...	...	...					
23	3000	Tue Feb 03...					
...	...	...					

Значение данных в каждом столбце:

"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, "odometr" - пробег автомобиля, "color" - цвет автомобиля, "interior" - цвет интерьера автомобиля, "seller" - продавец, "mmr" - ожидаемая рыночная цена автомобиля, "sellingprice" - цена продажи, "saledate" - дата продажи.

В таблице 1 полностью не уместились некоторые параметры. Например, "sellingdate" полностью выглядит так: Tue Dec 16 2014 12:30:00 GMT-0800 (PST), "vin" - 5xyktcab69fg566472, "seller" - kia motors america inc (для 1-го автомобиля). В нем даны различные характеристики машин, а мы будем предугадывать финальную цену машины, она находится в колонке "sellingprice".

## 4 Предобработка данных

Сначала подключим нужные нам библиотеки. Также подключим нашу базу данных и проверим вывод. Для работы с данными используем *Pandas* [5]. Для вывода графиков используем *MyPlotLib* [3], а также для работы с массивами, векторами и т. п. *NumPy* [2].

```
▶ !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_uxoyafzrhQnXQPAENWiBVc6Z8W7F/view?usp=share_link'  
url='https://drive.google.com/uc?id=' + url.split('/')[-2]  
df = pd.read_csv(url)  
df.head()
```

	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)

```
[ ] 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	0	0	# выбросим все строки содержащие нулевые элементы
make	10301	10301	df.dropna(inplace=True)
model	10399	10399	# как видим все они удалились
trim	10651	10651	df.isnull().sum()
body	13195	13195	
transmission	65352	65352	
vin	4	4	
state	0	0	
condition	11820	11820	
odometer	94	94	
color	749	749	
interior	749	749	
seller	0	0	
mmr	38	38	
sellingprice	12	12	
saledate	0	0	
			dtype: int64

В данном датасете оказались нулевые данные, так что уберем, те записи в которых, хотя бы один из параметров нулевой.

Теперь рассмотрим две колонки: "seller" и "saledate". В колонке "seller" содержится 11923 уникальных значения.

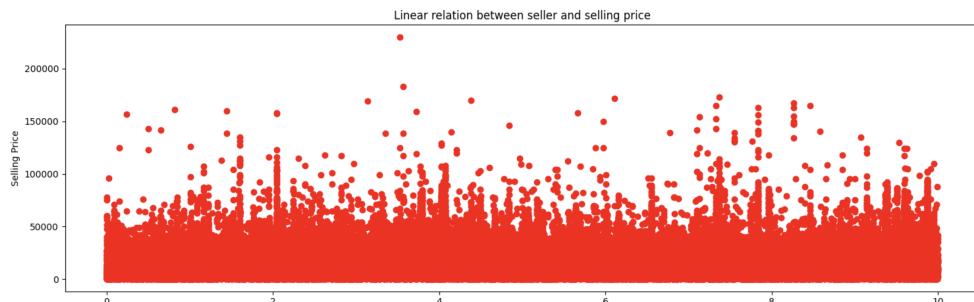
```
[ ] # посмотрим кол-во продавцов
df['seller'].nunique()

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
```

	plt.figure(figsize=(15,5))	plt.scatter(sellers, df['sellingprice'], color='red')	plt.xlabel("seller")	plt.ylabel('Selling Price')	plt.title("Linear relation between seller and selling price")	plt.tight_layout()	plt.show()



Тут чтобы перевести текстовые данные воспользуемся хэшированием. Используем функцию hash(). На графике видно, что цена не имеет явной зависимости от "seller", а данных их очень

много. Это плохо повлияет на обучение нашей нейронной сети, поэтому уберем данную колонку из нашего датасета.

Перейдем к "saledate". Тут преобразуем данные так: возьмем минимальный год и будем считать 0 днем 1 января этого года, а далее просто прибавляем дни. Если дата в нестандартном формате для данного датасета или в поле написано что-то другое, то тоже меняем на 0. По полученному графику, приведенному ниже понятно, что цена не зависит от даты. В нём есть провал, но в то время продавалось меньше машин и это не так значительно, поэтому исключим данный столбец по тем же причинам.

```
▶ # так теперь обработаем дату продажи, если она в неправильном формате, то сопоставим ноль, иначе количество дней от начала минимального года
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]))

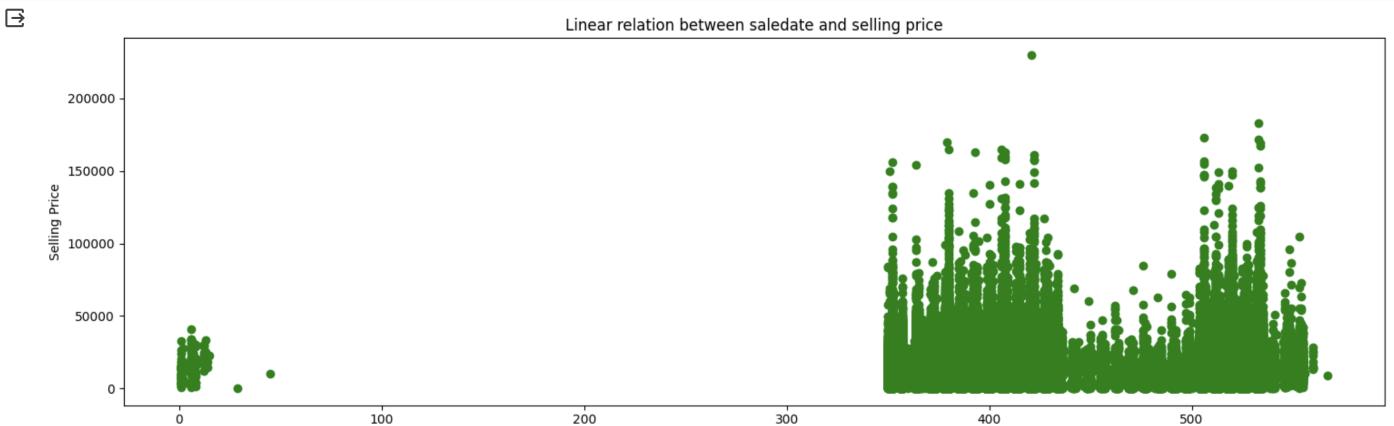
min_year = min(years)

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)
```

```

plt.figure(figsize=(15,5))
plt.scatter(_sale_date, df['sellingprice'], color='green')
plt.xlabel("saledate")
plt.ylabel('Selling Price')
plt.title("Linear relation between saledate and selling price")
plt.tight_layout()
plt.show()

```



Продолжим. Возьмем только 0.6 записей от всего датасета. Так как я уже говорил, выкинем две колонки, а также колонку "vin", ведь у каждой машины он уникален.

```

#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 вином, 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()

```

	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()}')

```

Посмотрим количество переменных в каждой категориальной колонке, и, если они встречается меньше 10 раз, то заменим эту категорию на Other. Тем самым сократим кол-во категорий.

```

[ ] printUniqueAmount(df)

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

[ ] #если какая-то категория встречается меньше 10 раз, то мы 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 наше кол-во уникальных категорий:
printUniqueAmount(df)

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

```

Далее, переведем все данные в числа. От года выпуска машины перейдем к возрасту. Для этого вычтем год выпуска из 2015 (максимальный год в базе). С помощью библиотеки *pandas* [5] и функции *get\_dummies* переведем "automatic" в 0, "manual" - 1 для колонки "transmission". С помощью библиотеки *category\_encoders* (*ce*) [6], переведем оставшиеся категориальные переменные "make", "body", "interior", "color", "state". Каждому элементу он сопоставит n-мерный вектор из нулей и единиц. Затем, используя *FeatureHasher* из *sklearn.feature\_extraction* [6] наши текстовые данные. И наконец-то добавим в конец нашей таблицы все получившиеся столбцы и удалим из нее "year", "model", "trim". Посмотрим, что получили на следующем рисунке.

```
[ ] #год выпуска машины меняем на возраст машины
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	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	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	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	0	1	0	0	0	0	0	0.0	0.0	-1.0	0.0	0.0	0.0	0.0

5 rows x 74 columns

Так как предсказывать будем цену, то распределим данные так, в  $y$  - "sellingprice", а в  $X$  всё остальное. Используя `train_test_split` из `sklearn.model_selection` [6] разделим на обучающую и тестовую выборки наши данные. Размер тестового набора 20% от  $X$ .

```
[ ] #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)
```

Используя `StandardScaler` из `sklearn.preprocessing` [6] нормализуем наши данные  $X$ . И посмотрим, что получилось.

```
➊ #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]]
```

Мы подготовили данные для обучения нейронной сети. Перейдем к следующему разделу.

## 5 Создаем и обучаем нейросети

Сначала напишем функцию для проверки результатов:

```
▶ #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 значения')
    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])
```

Теперь создадим модель нейронной сети и запустим обучение. Для этого используем библиотеку

*keras* [1].

```
▶ #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 1/100
```

Я попробовал разные модели, но это работала лучше всех. Нейронная сеть состоит из 5 слоев

Dense, активация каждого из них - "relu". Количество эпох 100. При запуске 150 эпох уже заметно,

что после 100 положительной тенденции не наблюдается. На сотой эпохе минимальная ошибка.

Внизу на рисунке результаты запуска на 150 эпох.

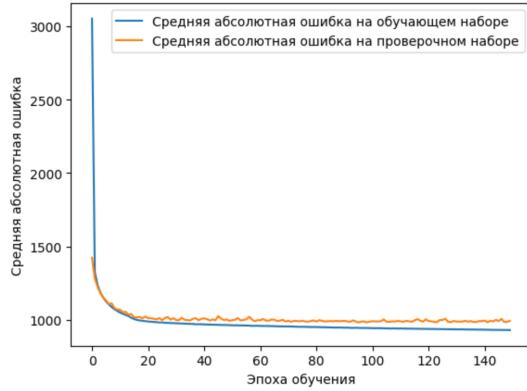


Рис. 1: График зависимости средней абсолютной ошибки от эпох обучения

По рис. 1 видно, что после 100 эпохи растет ошибка на проверочном наборе. Если в массиве из ошибок взять минимальный элемент, то получим, что такой элемент был сотым. Поэтому и выбрали 100 эпох обучения. Проверим модель в тестовом режиме.

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

#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]
```

На рисунках ниже (рис. 2 и 3) можно увидеть среднюю ошибку для обучающего набора - 949 ед. и 984 ед. - для тестового набора данных.

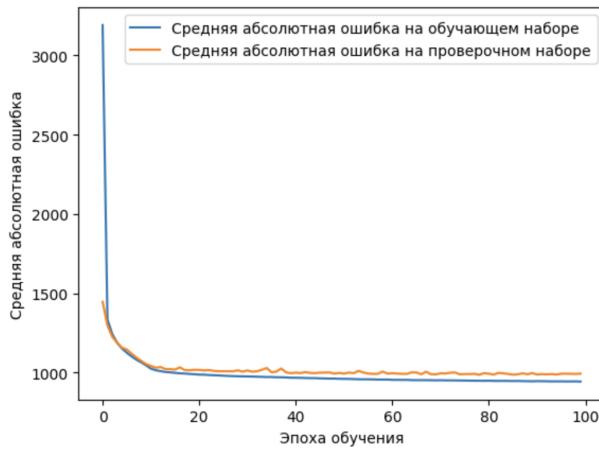


Рис. 2: Результаты 1

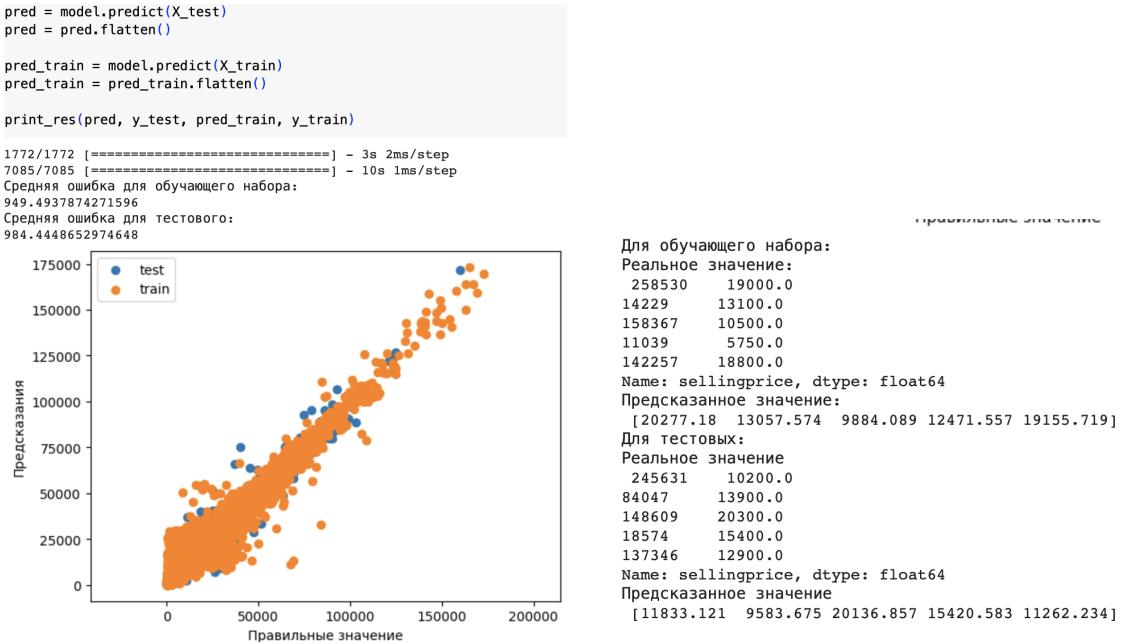


Рис. 3: Результаты 1

Теперь попробуем обучить нашу нейросеть на нормализованных данных  $y$ . Для этого, как и для  $X$ , используем *StandartScaler* [6].

```

#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 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]

```

После нормализации  $y\_test$  и  $y\_train$  приступим к обучению. Тут используем такую же нейросеть. Сначала попробуем optimizer Adam с learning rate  $10^{-4}$ .

```

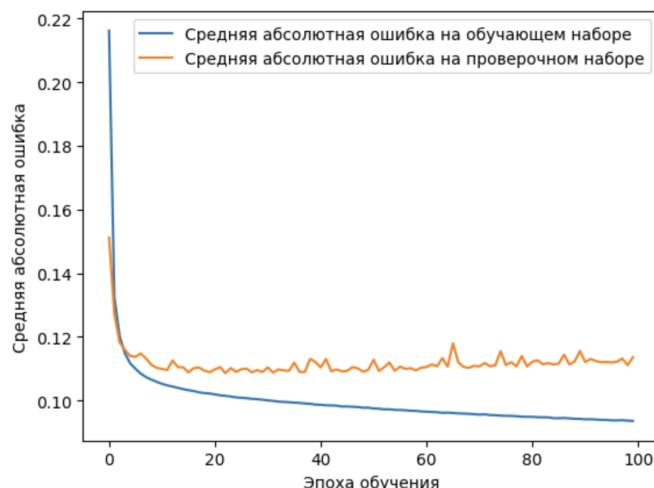
#создаем модель нейронки
models = keras.Sequential([
    keras.Input(shape=(73,)),
    keras.layers.Dense(120, activation='relu'),
    keras.layers.Dense(80, activation='relu'),
    keras.layers.Dense(60, activation='relu'),
    keras.layers.Dense(10, activation='relu'),
    keras.layers.Dense(1),
])

modelS.compile(optimizer=keras.optimizers.Nadam(learning_rate=1e-4), loss='mean_squared_error', metrics=['mae'])

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

#построим графики ошибки от эпох обучения
plt.plot(historyS.history['mae'],
          label='Средняя абсолютная ошибка на обучающем наборе')
plt.plot(historyS.history['val_mae'],
          label='Средняя абсолютная ошибка на проверочном наборе')
plt.xlabel('Эпоха обучения')
plt.ylabel('Средняя абсолютная ошибка')
plt.legend()
plt.show()

```



```

❷ #делаем предсказание и приводим его к начальному виду и находим среднее значение
pred = modelS.predict(X_test)
predUnscaled = yScaler.inverse_transform(pred).flatten()

pred_train = modelS.predict(X_train)
predUnscaled_train = yScaler.inverse_transform(pred_train).flatten()

print_res(predUnscaled, y_test, predUnscaled_train, y_train)

❸ 1772/1772 [=====] - 3s 1ms/step
7085/7085 [=====] - 11s 2ms/step
Средняя ошибка для обучающего набора:
934.884551975201
Средняя ошибка для тестового:
1087.1992151236054

```

Получаем, что для тестовых значений ошибка почти сразу начинает увеличиваться. Но если

использовать  $learning\_rate = 1$ , то получим такие результаты:

```

❶ #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)

```

```

#построим графики ошибки от эпох обучения
plt.plot(historyS.history['mae'],
         label='Средняя абсолютная ошибка на обучающем наборе')
plt.plot(historyS.history['val_mae'],
         label='Средняя абсолютная ошибка на проверочном наборе')
plt.xlabel('Эпоха обучения')
plt.ylabel('Средняя абсолютная ошибка')
plt.legend()
plt.show()

```

```

#делаем предсказание и приводим его к начальному виду и находим среднее значение ошибки
pred = models.predict(X_test)
predUnscaled = yScaler.inverse_transform(pred).flatten()

pred_train = models.predict(X_train)
predUnscaled_train = yScaler.inverse_transform(pred_train).flatten()

print_res(predUnscaled, y_test, predUnscaled_train, y_train)

```

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:  
[19869.287 13718.192 10545.157 12923.345 19695.512]  
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  
[10767.873 9596.717 19530.602 16111.357 11323.871]

Тут снова делаем предсказание и передаем его в нашу функцию, которая считает ошибки.

Получилось, что на ненормализованных  $y$  данных выдает результат лучше.

## 6 AutoML

Теперь попробуем использовать *AutoML* для решения нашей задачи. *AutoML* автоматизирует различные процессы машинного обучения. Для нашей задачи будем использовать библиотеку

*AutoKeras* [4], основанную на *Keras* [1]. В *AutoKeras* [4] все параметры подбираются автоматический, если не указано другого. По умолчанию модель данной библиотеки делает 1000 эпох и 100 попыток для модели нейросети, но остановиться она может и раньше. Например, если на протяжении 10 эпох ошибка только увеличивается, то данная попытка закончится. Мы поставим ограничение на максимальное количество эпох 150, попыток - 5.

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

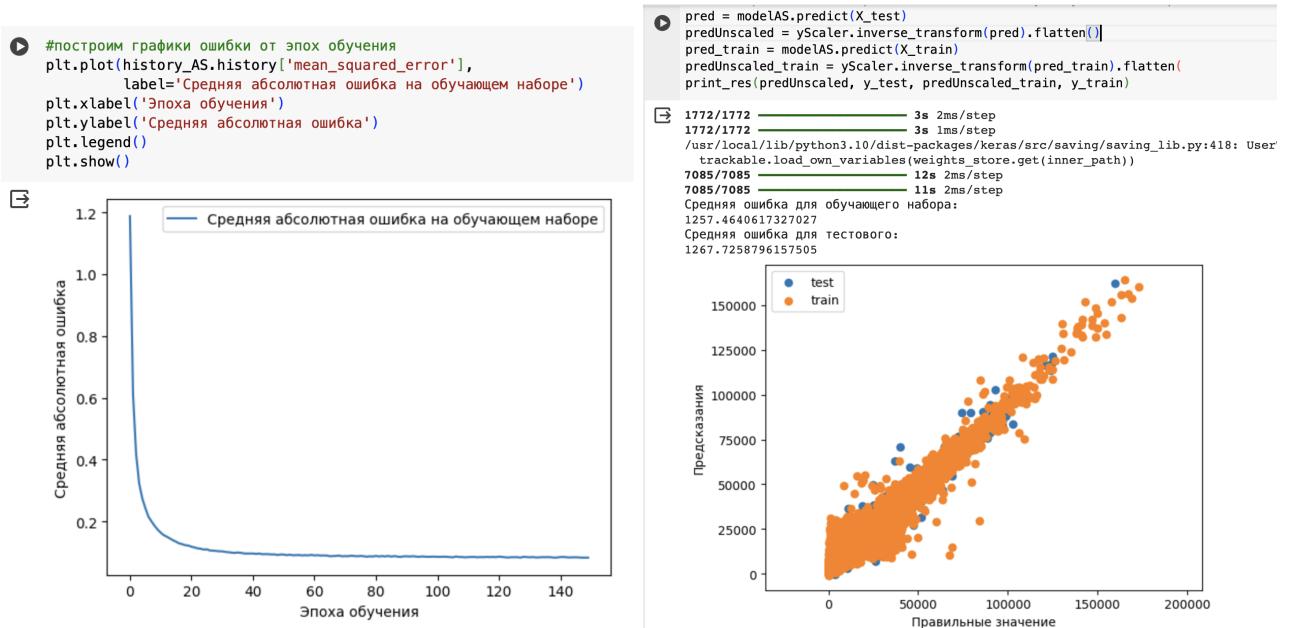
▶ #будем использовать AutoKeras
import autokeras

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
)
[ ] #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'
trackable.load_own_variables(weights_store.get(inner_path))
1772/1772 3s 1ms/step - loss: 0.0342 - mean_squared_error: 0.0342
[0.0348929725587368, 0.0348929725587368]
```

Тут установили *autokeras* [4], создали и обучили модель. Далее, построим график средней абсолютной ошибки на обучающем наборе, сделаем предсказание и найдем средние ошибки.



```

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:
 [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]

```

Тут я решил попробовать один вариант. Решил не ставить максимум на эпохи, то есть максимум автоматический 1000, но нейросеть может остановится раньше, если не будет улучшения в течении 10 эпох, но поставил три максимальных попытки и обучал на ненормализованном  $y$ . И если сравнивать, то получились самые лучшие результаты.

```

❶ #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)]
)

❷ Trial 3 Complete [00h 00m 50s]
val_loss: 2777095.25

Best val_loss So Far: 2313908.75
Total elapsed time: 01h 07m 16s
Epoch 1/342
7085/7085 ━━━━━━━━━━ 3s 384us/step - loss: 79319880.0000 - mean_squared_error: 79319880.0000
Epoch 342/342
7085/7085 ━━━━━━━━━━ 3s 372us/step - loss: 2035549.3750 - mean_squared_error: 2035549.3750

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

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]

```

```

#построим графики ошибки от эпох обучения
plt.plot(history_A.history['mean_squared_error'],
label='Средняя квадратичная ошибка на обучающем наборе')
plt.xlabel('Эпоха обучения')
plt.ylabel('Средняя квадратичная ошибка')
plt.legend()
plt.show()

[ ] #делаем предсказание и приводим его к начальному виду и находим среднее значение ошибки
pred = modelA.predict(X_test)

pred_train = modelA.predict(X_train)

1772/1772 - 0s 262us/step
1772/1772 - 0s 259us/step
7085/7085 - 2s 254us/step
7085/7085 - 2s 254us/step

print_res(pred.reshape(1, -1).squeeze(), np.array(y_test), pred_train.reshape(1, -1).squeeze(), np.array(y_train))

Средняя ошибка для обучающего набора:
923.1337389565
Средняя ошибка для тестового:
951.6978463716204

```

Средняя абсолютная ошибка на обучающем наборе

Эпохи обучения

Предсказания

Правильные значения

test

train

Для обучающего набора:

Реальное значение:  
[19000. 13100. 10500. 5750. 18800.]

Предсказанное значение:  
[19880.111 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 ]

## 7 Вывод

Лучшие значения получили с *AutoKeras* [4] на ненормализованных данных, но в нашей первой модели результаты почти такие же. С помощью нейросетей задачи регрессии решаются довольно

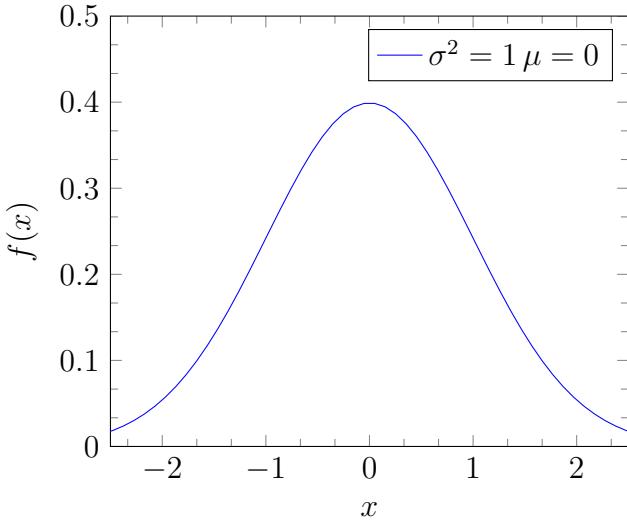
хорошо, но в различных ситуациях результат может сильно ухудшаться, когда в датасете очень много некорректных данных или он в принципе плохо предобработан. Более того, на результат будет влиять и нормализация данных. Причем, казалось бы, с помощью нормализованных данных нейросеть должна обучаться лучше, 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.

## 8 Приложения

Формула многомерного Гауссового интеграла (1) и график нормального распределения (8) для выполнения требований отчета.

$$\int_{-\infty}^{+\infty} dx_1 \dots \int_{-\infty}^{+\infty} dx_d e^{-\frac{1}{2} \sum_{i,j} x_i A_{ij} x_j + \sum_k b_k x_k} = \frac{(2\pi)^{\frac{d}{2}}}{\sqrt{\det A}} e^{\frac{1}{2} \sum_{ij} b_i A_{ij}^{-1} b_j} \quad (1)$$

нормальное распределение



## Список литературы

- [1] François Chollet et al. Keras. <https://keras.io>, 2015.
- [2] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.
- [3] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.
- [4] Haifeng Jin, FranÃ§ois Chollet, Qingquan Song, and Xia Hu. Autokeras: An automl library for deep learning. *Journal of Machine Learning Research*, 24(6):1–6, 2023.
- [5] The pandas development team. pandas-dev/pandas: Pandas, feb 2020.
- [6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.