**Регрессионная модель предсказания расхода топлива**

В задачах *регрессии* ставится цель дать прогноз какого-либо значения, например цены или вероятности. Это отличает ее от задачи *классификации*, где нужно предсказать конкретный класс или категорию (например, цифру или предмет по картинке).

Для иллюстрации решения задачи регрессии используется классический набор данных [Auto MPG](https://archive.ics.uci.edu/ml/datasets/auto+mpg) (данные конца 70-х и начала 80-х), по данным которого строится модель, которая будет предсказывать эффективность расхода топлива автомобилей. Набор данных содержит описания множества различных автомобилей того времени. Эти описания содержат такие параметры как количество цилиндров, лошадиных сил, объем двигателя и вес.

При решении задачи регрессии используется API tf.keras (подробное описание API можно найти по [ссылке](https://www.tensorflow.org/guide/keras)). Установка библиотек описана в приложении к лабораторной работе 3.

**1. Загрузка необходимых библиотек**

import pathlib  
  
import matplotlib.pyplot as plt  
import pandas as pd  
# Импорт библиотеки seaborn для построения парных графиков  
import seaborn as sns  
  
import tensorflow as tf  
from tensorflow import keras  
from tensorflow.keras import layers  
  
print(tf.\_\_version\_\_)

2.0.0

**2. Загрузка набора данных Auto MPG**

Набор данных доступен в [репозитарии машинного обучения UCI](https://archive.ics.uci.edu/ml/datasets/auto+mpg).

Загружаем данные из репозитория в локальную папку компьютера.

dataset\_path = keras.utils.get\_file("auto-mpg.data", "http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data")  
dataset\_path

Downloading data from http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data

32768/30286 [================================] - 0s 0us/step

'/root/.keras/datasets/auto-mpg.data' # Каталог может быть другой

Импортируем его при помощи библиотеки Pandas:

column\_names = ['Расход топлива','Кол-во цилиндров','Объем двигателя','Л.с.','Вес',  
                'Разгон до 100 км/ч', 'Год выпуска', 'Страна выпуска']  
raw\_dataset = pd.read\_csv(dataset\_path, names=column\_names,  
                      na\_values = "?", comment='\t',  
                      sep=" ", skipinitialspace=True)  
  
dataset = raw\_dataset.copy()

# Вывод последних 5 строк  
dataset.tail()

|  | Расход топлива | Кол-во цилиндров | Объем двигателя | Л.с. | Вес | Разгон до 100 км/ч | Год выпуска | Страна выпуска |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 393 | 27.0 | 4 | 140.0 | 86.0 | 2790.0 | 15.6 | 82 | 1 |
| 394 | 44.0 | 4 | 97.0 | 52.0 | 2130.0 | 24.6 | 82 | 2 |
| 395 | 32.0 | 4 | 135.0 | 84.0 | 2295.0 | 11.6 | 82 | 1 |
| 396 | 28.0 | 4 | 120.0 | 79.0 | 2625.0 | 18.6 | 82 | 1 |
| 397 | 31.0 | 4 | 119.0 | 82.0 | 2720.0 | 19.4 | 82 | 1 |

**3. Подготовка данных**

Необходимо почистить данные, так как набор данных содержит несколько неизвестных (незаполненных) значений. Подсчет количества таких значений выполняется следующим образом:

dataset.isna().sum()

Расход топлива 0

Кол-во цилиндров 0

Объем двигателя 0

**Л.с. 6**

Вес 0

Разгон до 100 км/ч 0

Год выпуска 0

Страна выпуска 0

dtype: int64

Чтобы было проще, можно удалить параметры, где отсутствуют значения (присутствуют значения NA):

dataset = dataset.dropna()

"Страна выпуска" - это колонка с указанием категории, а не значений. Ее необходимо в задаче регрессии перевести в двоичный код:

# Извлечение столбца с удалением его из набора

origin = dataset.pop('Страна выпуска')

# Добавление новых столбцов

dataset['США'] = (origin == 1)\*1.0  
dataset['Европа'] = (origin == 2)\*1.0  
dataset['Япония'] = (origin == 3)\*1.0

# Вывод последних 5 строк  
dataset.tail()

Набор данных после преобразования

|  | Расход топлива | Кол-во цилиндров | Объем двигателя | Л.с. | Вес | Разгон до 100 км/ч | Год выпуска | США | Европа | Япония |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 393 | 27.0 | 4 | 140.0 | 86.0 | 2790.0 | 15.6 | 82 | 1.0 | 0.0 | 0.0 |
| 394 | 44.0 | 4 | 97.0 | 52.0 | 2130.0 | 24.6 | 82 | 0.0 | 1.0 | 0.0 |
| 395 | 32.0 | 4 | 135.0 | 84.0 | 2295.0 | 11.6 | 82 | 1.0 | 0.0 | 0.0 |
| 396 | 28.0 | 4 | 120.0 | 79.0 | 2625.0 | 18.6 | 82 | 1.0 | 0.0 | 0.0 |
| 397 | 31.0 | 4 | 119.0 | 82.0 | 2720.0 | 19.4 | 82 | 1.0 | 0.0 | 0.0 |

**3.1 Разделение данных на тренировочные и проверочные наборы**

На следующем шаге необходимо разделить данные на две части: для обучения и для проверки.

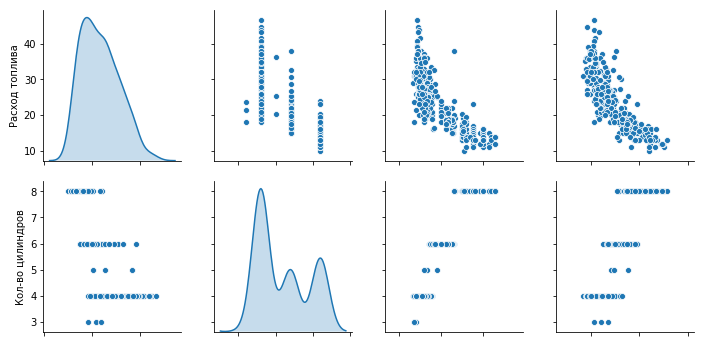
Точность получившейся модели, построенной на первом наборе, можно проверить на втором наборе данных.

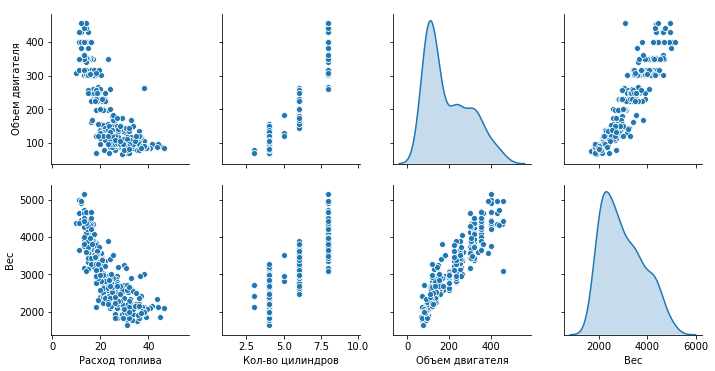
train\_dataset = dataset.sample(frac=0.8,random\_state=0)  
test\_dataset = dataset.drop(train\_dataset.index)

**3.2 Проверка данных**

На графике можно вывести совместное распределение нескольких пар колонок из тренировочного набора данных:

sns.pairplot(train\_dataset[["Расход топлива", "Кол-во цилиндров", "Объем двигателя", "Вес"]], diag\_kind="kde")





Вывод общей статистики:

train\_stats = train\_dataset.describe()  
train\_stats.pop("Расход топлива")  
train\_stats = train\_stats.transpose()  
train\_stats

|  | count | mean | std | min | 25% | 50% | 75% | max |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Кол-во цилиндров | 314.0 | 5.477707 | 1.699788 | 3.0 | 4.00 | 4.0 | 8.00 | 8.0 |
| Объем двигателя | 314.0 | 195.318471 | 104.331589 | 68.0 | 105.50 | 151.0 | 265.75 | 455.0 |
| Л.с. | 314.0 | 104.869427 | 38.096214 | 46.0 | 76.25 | 94.5 | 128.00 | 225.0 |
| Вес | 314.0 | 2990.251592 | 843.898596 | 1649.0 | 2256.50 | 2822.5 | 3608.00 | 5140.0 |
| Разгон до 100 км/ч | 314.0 | 15.559236 | 2.789230 | 8.0 | 13.80 | 15.5 | 17.20 | 24.8 |
| Год выпуска | 314.0 | 75.898089 | 3.675642 | 70.0 | 73.00 | 76.0 | 79.00 | 82.0 |
| США | 314.0 | 0.624204 | 0.485101 | 0.0 | 0.00 | 1.0 | 1.00 | 1.0 |
| Европа | 314.0 | 0.178344 | 0.383413 | 0.0 | 0.00 | 0.0 | 0.00 | 1.0 |
| Япония | 314.0 | 0.197452 | 0.398712 | 0.0 | 0.00 | 0.0 | 0.00 | 1.0 |

**3.3 Выделение целевых значений**

Необходимо отделить целевые значения от характеристик и использовать эти значения для обучения модели регрессии:

# Извлечение целевого столбца с удалением из набора

train\_labels = train\_dataset.pop('Расход топлива')  
test\_labels = test\_dataset.pop('Расход топлива')

**3.4 Нормализация данных**

Если взглянуть на блок train\_stats, то можно обратить внимания, что значения каждой характеристики измеряются в разных единицах и изменяются в различных диапазонах (масштабах).

Нормализация параметров данных, в которых используются разные единицы измерений - один из первых шагов подготовки данных к обучению. Несмотря на то, что модель *может* стремиться к пределу без нормализации, но это излишне усложняет обучение, а также делает получившуюся модель зависимой от выбора единиц измерения входных данных.

Для обучения модели будут использоваться обновленные, нормализованные данные. Также необходимо нормализовать проверочные данные.

# Функция нормализации

def norm(x):  
  return (x - train\_stats['mean']) / train\_stats['std']

# Нормализация данных  
normed\_train\_data = norm(train\_dataset)  
normed\_test\_data = norm(test\_dataset)

**4. Построение модели**

В качестве модели используется (последовательную) модель Sequential с двумя полносвязными слоями Dense, выходной слой будет возвращать одно непрерывно изменяющееся значение. Структура модели описывается в функции build\_model. Эта функция будет использована позже для создания еще одной модели.

from tensorflow.keras import models

def build\_model():  
 model = keras.Sequential([  
 layers.Dense(64, activation=tf.nn.relu, input\_shape=[len(train\_dataset.keys())]),  
 layers.Dense(64, activation=tf.nn.relu),  
 layers.Dense(1)  
 ])  
# optimizer = tf.train.RMSPropOptimizer(0.001)

opt = tf.keras.optimizers.RMSprop(learning\_rate=0.001)# mse - mean\_squared\_error (средняя квадратическая ошибка)

# mae - mean\_absolute\_error (средняя абсолютная ошибка)

model.compile(loss='mse',  
                optimizer=opt,  
                metrics=['mae', 'mse'])  
 return model

model = build\_model()

Воспользуемся методом .summary, чтобы посмотреть как выглядит построенная модель:

model.summary()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

dense (Dense) (None, 64) 640

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 64) 4160

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 1) 65

=================================================================

Total params: 4,865

Trainable params: 4,865

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**5. Проверка модели**

Проверим построенную модель. Для этого используем небольшой пакет из 10 примеров данных из тренировочного набора и попробуем предсказать результат.

example\_batch = normed\_train\_data[:10]  
example\_result = model.predict(example\_batch)  
example\_result

array([[-0.07893069],

[-0.17232396],

[-0.3558741 ],

[ 0.05699661],

[-0.29774278],

[-0.08043784],

[-0.29601038],

[-0.8701949 ],

[-0.15903525],

[-0.06046572]], dtype=float32)

Модель показывает результат ожидаемой формы и класса.

**6. Обучение модели**

Обучение модели проводится в течение 1000 эпох. В объекте history фиксируется точность модели на тренировочных и проверочных данных.

# Выведем прогресс обучения в виде точек после каждой завершенной эпохи  
class PrintDot(keras.callbacks.Callback):  
  def on\_epoch\_end(self, epoch, logs):

# Перевод строки после 100 эпох  
    if epoch % 100 == 0: print('')  
    print('.', end='')  
  
EPOCHS = 1000  
# validation\_split = 0.2 - 20% данных отводится на проверку  
history = model.fit(  
  normed\_train\_data, train\_labels,  
  epochs=EPOCHS, validation\_split = 0.2, verbose=0,  
  callbacks=[PrintDot()])

..........................................................................

.........................................................................

..........................................................................

Выполним визуализацию процесса обучения при помощи статистики из объекта history:

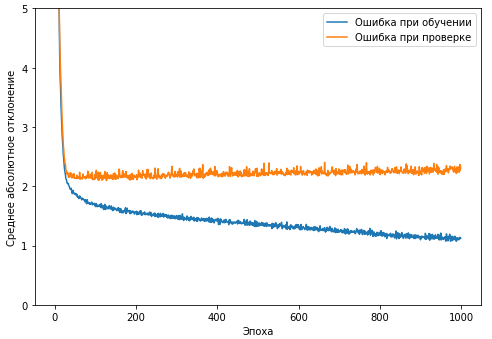
hist = pd.DataFrame(history.history)  
hist['epoch'] = history.epoch

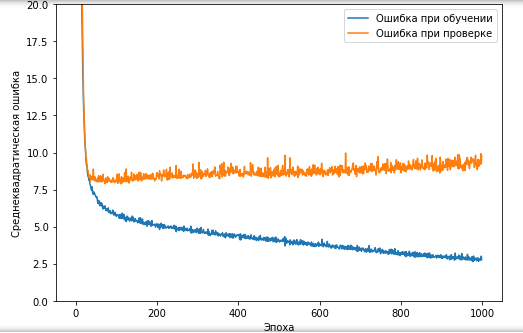
# Вывод последних 5 строк  
hist.tail()

|  | loss | mae | mse | val\_loss | val\_mae | val\_mse | epoch |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 995 | 2.935207 | 1.103273 | 2.935207 | 8.995369 | 2.340968 | 8.995369 | 995 |
| 996 | 2.931519 | 1.084985 | 2.931519 | 8.977335 | 2.278823 | 8.977335 | 996 |
| 997 | 2.632797 | 1.047357 | 2.632797 | 8.893933 | 2.321596 | 8.893933 | 997 |
| 998 | 3.048073 | 1.054709 | 3.048073 | 8.839587 | 2.274392 | 8.839587 | 998 |
| 999 | 2.657460 | 1.044908 | 2.657460 | 8.850338 | 2.321004 | 8.850338 | 999 |

# Функция визуализации процесса обучения

def plot\_history(history):  
  hist = pd.DataFrame(history.history)  
  hist['epoch'] = history.epoch  
  
  plt.figure(figsize=(8,12))  
  
  plt.subplot(2,1,1)  
  plt.xlabel('Эпоха')  
  plt.ylabel('Среднее абсолютное отклонение')  
  plt.plot(hist['epoch'], hist['mae'],  
           label='Ошибка при обучении')  
  plt.plot(hist['epoch'], hist['val\_mae'],  
           label = 'Ошибка при проверке')  
  plt.ylim([0,5])  
  plt.legend()  
  
  plt.subplot(2,1,2)  
  plt.xlabel('Эпоха')  
  plt.ylabel('Среднеквадратическая ошибка')  
  plt.plot(hist['epoch'], hist['mse'],  
           label='Ошибка при обучении')  
  plt.plot(hist['epoch'], hist['val\_mse'],  
           label = 'Ошибка при проверке')  
  plt.ylim([0,20])  
  plt.legend()  
  plt.show()  
  
plot\_history(history)





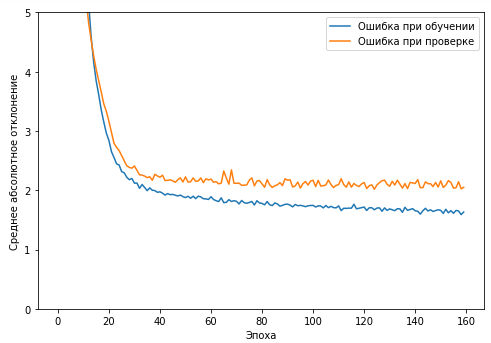
Полученные графики показывают, что после нескольких сотен эпох модель улучшается совсем незначительно в процессе обучения (а после 50 эпох наступает переобучение).

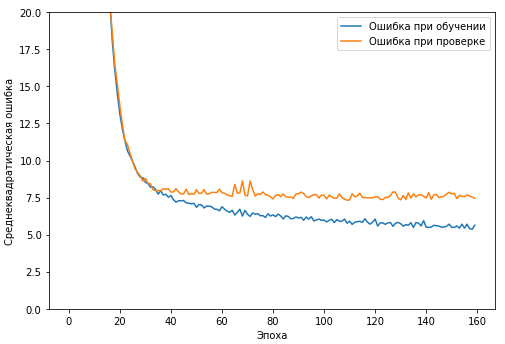
Обновим метод model.fit, чтобы автоматически прекращать обучение как только показатель проверки *Val loss* не улучшается. Для этого используется функция обратного вызова *callback*, которая проверяет показатели обучения после каждой эпохи. Если после определенного количество эпох нет никаких улучшений, то функция автоматически остановит его.

Подробная информация о функции *callback* приводится [здесь](https://www.tensorflow.org/versions/master/api_docs/python/tf/keras/callbacks/EarlyStopping).

model = build\_model()  
  
# Параметр patience определяет количество эпох, которые можно пропустить без улучшений  
early\_stop = keras.callbacks.EarlyStopping(monitor='val\_loss', patience=50)  
  
history = model.fit(normed\_train\_data, train\_labels, epochs=EPOCHS,  
                    validation\_split = 0.2, verbose=0, callbacks=[early\_stop, PrintDot()])  
  
plot\_history(history)

....................................................................................................................................................





Графики показывают что среднее значение ошибки на проверочных данных - около 2 галлонов на милю. Хорошо это или плохо - решать исследователю.

На следующем шаге обученная модель проверяется на наборе данных для проверки:

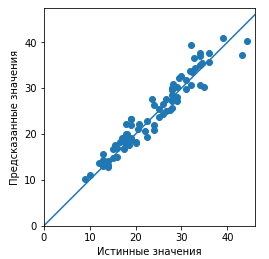
loss, mae, mse = model.evaluate(normed\_test\_data, test\_labels, verbose=0)  
  
print("Среднее абсолютное отклонение на проверочных данных: {:5.2f} галлон на милю".format(mae))

Среднее абсолютное отклонение на проверочных данных: 1.82 галлон на милю

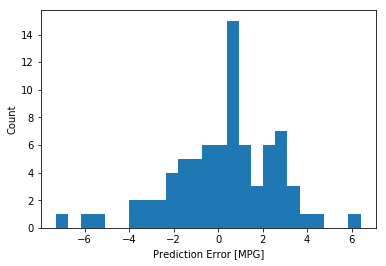
**7. Выполнение предсказания**

Выполним предсказания показателей расхода топлива, используя набор наши проверочные данные:

test\_predictions = model.predict(normed\_test\_data).flatten()  
  
plt.scatter(test\_labels, test\_predictions)  
plt.xlabel('Истинные значения')  
plt.ylabel('Предсказанные значения')  
plt.axis('equal')  
plt.axis('square')  
plt.xlim([0,plt.xlim()[1]])  
plt.ylim([0,plt.ylim()[1]])  
\_ = plt.plot([-100, 100], [-100, 100])



error = test\_predictions - test\_labels  
plt.hist(error, bins = 25)  
plt.xlabel("Prediction Error [MPG]")  
\_ = plt.ylabel("Count")



**Задание.** Постройте модель заданной архитектуры, обучите и проверьте ее точность. Сохраните модель. Восстановите модель и сравните точность обеих моделей.

**Дополнительное задание**: Измените параметры архитектуры модели сети и выберите оптимальные значения

1. Оцените качество модели методом перекрестной проверки по K блокам.
2. Используйте метод ранней остановки для избежания переобучения