

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №3
по дисциплине «Искусственные нейронные сети»
Тема: Регрессионная модель изменения цен на дома в Бостоне

Студентка гр. 7382

Еременко А.

Преподаватель

Жукова Н.А.

Санкт-Петербург

2020

Цель работы.

Реализовать предсказание медианной цены на дома в пригороде Бостона в середине 1970-х по таким данным, как уровень преступности, ставка местного имущественного налога и т. д.

Порядок выполнения работы.

1. Ознакомиться с задачей регрессии
2. Изучить отличие задачи регрессии от задачи классификации
3. Создать модель
4. Настроить параметры обучения
5. Обучить и оценить модели
6. Ознакомиться с перекрестной проверкой

Требования к выполнению задания.

1. Объяснить различия задач классификации и регрессии
2. Изучить влияние кол-ва эпох на результат обучения модели
3. Выявить точку переобучения
4. Применить перекрестную проверку по K блокам при различных K
5. Построить графики ошибки и точности во время обучения для моделей, а также усредненные графики по всем моделям

Основные теоретические положения.

1. *Классификационное моделирование* - это задача приближения функции отображения f от входных переменных (X) к дискретным выходным переменным (Y).

- Задача классификации требует, разделения объектов в один или два класса.
- Классификация может иметь действительные или дискретные входные переменные.

- Проблема с двумя классами часто называется проблемой двухклассной или двоичной классификации.
- Проблема с более чем двумя классами часто называется проблемой классификации нескольких классов.
- Проблема, когда для примера назначается несколько классов, называется проблемой классификации по нескольким меткам.

2. *Регрессионное моделирование* - это задача приближения функции отображения f от входных переменных (X) к непрерывной выходной переменной (Y).

- Задача регрессии требует предсказания количества.
- Регрессия может иметь действительные или дискретные входные переменные.
- Проблема с несколькими входными переменными часто называется проблемой многомерной регрессии.
- Проблема регрессии, когда входные переменные упорядочены по времени, называется проблемой прогнозирования временных рядов.

Ход работы.

Была создана и обучена модель искусственной нейронной сети. Код предоставлен в приложении А.

Для выполнения поставленной задачи были опробованы разнообразные архитектуры сети, обучение проводилось при различных параметрах, было изменено количество блоков 4,5,6.

Рассмотрим модель с 6-ю блоками. Точность будем оценивать с помощью средней абсолютной ошибки. Графики ошибок и точности предоставлены на рис. 1-12.

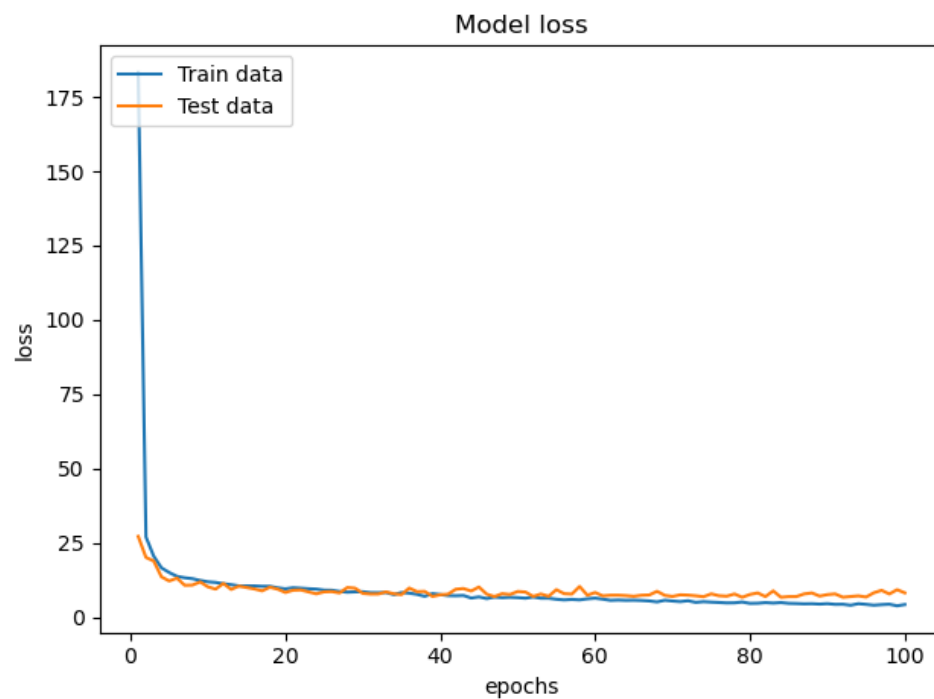


Рисунок 1 – График ошибки $k=1$

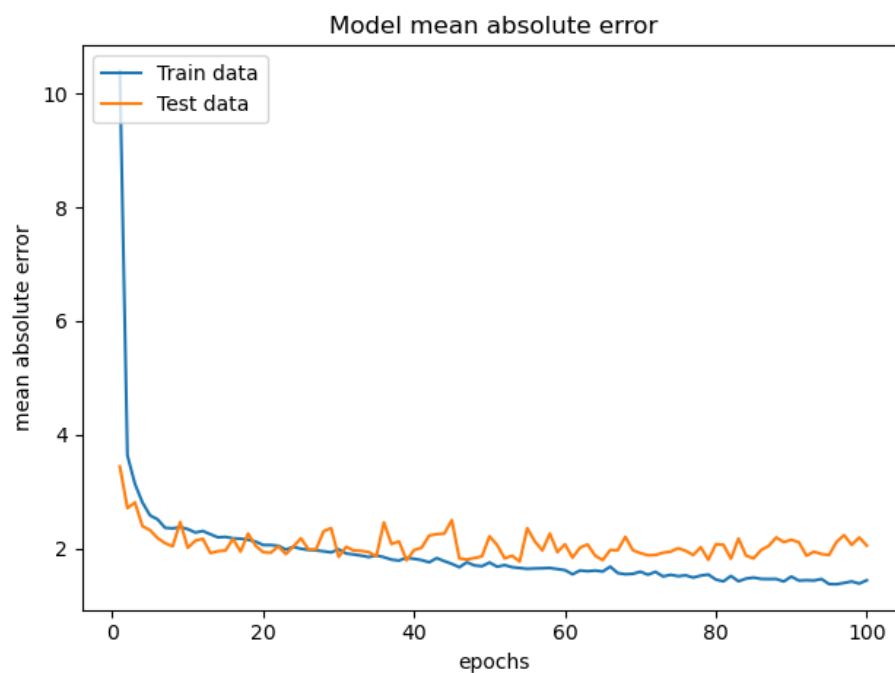


Рисунок 2 – График оценки mae $k=1$

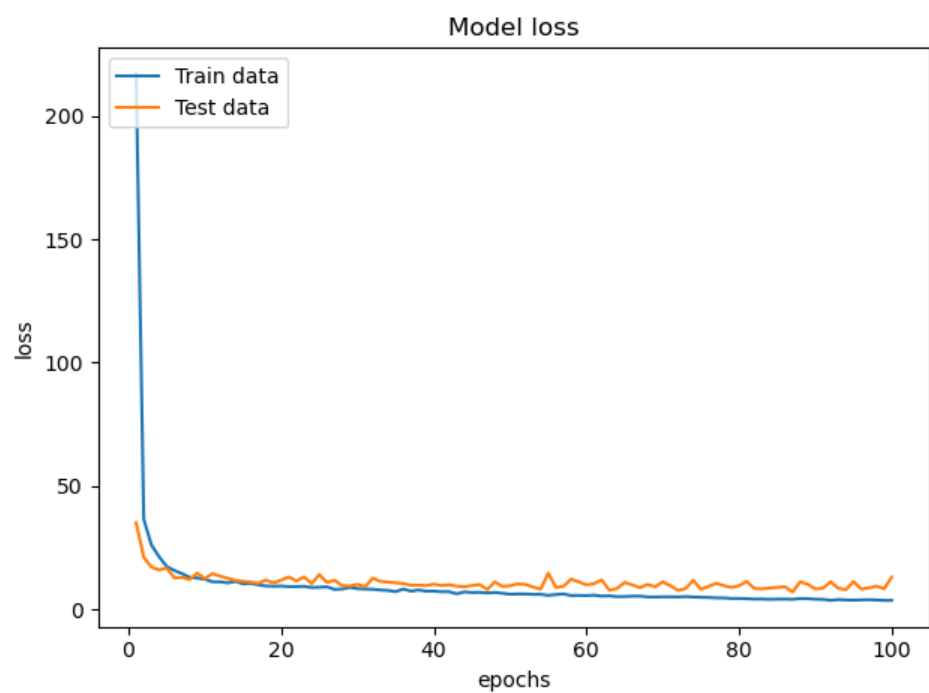


Рисунок 3 – График ошибки $k=2$



Рисунок 4 – График оценки mae $k=2$

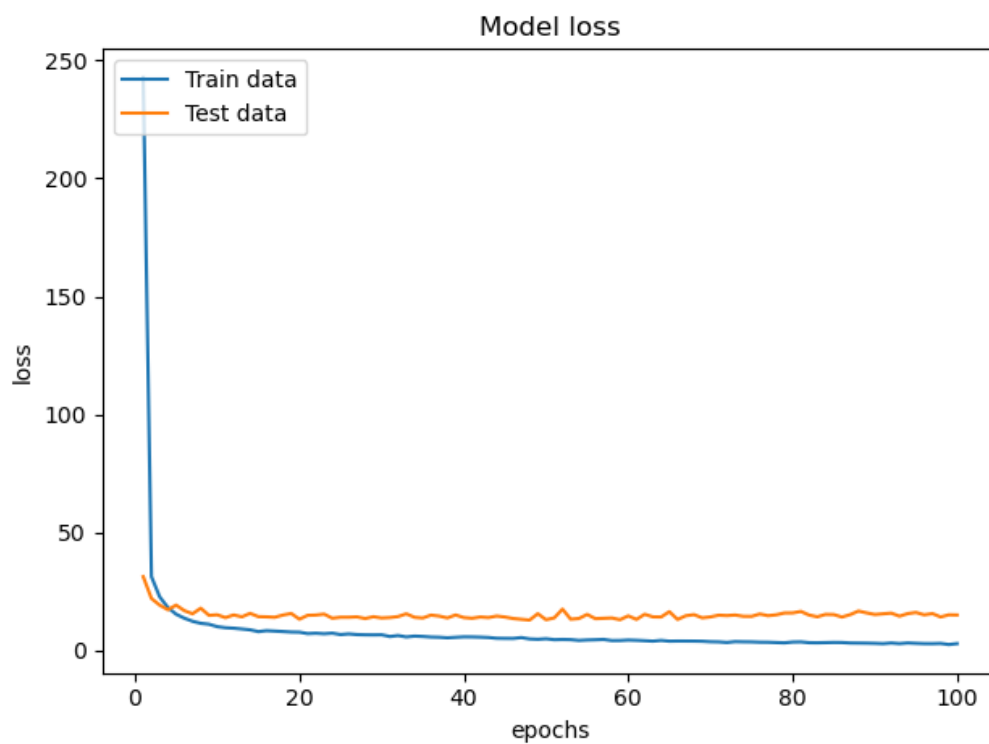


Рисунок 5 – График ошибки $k=3$

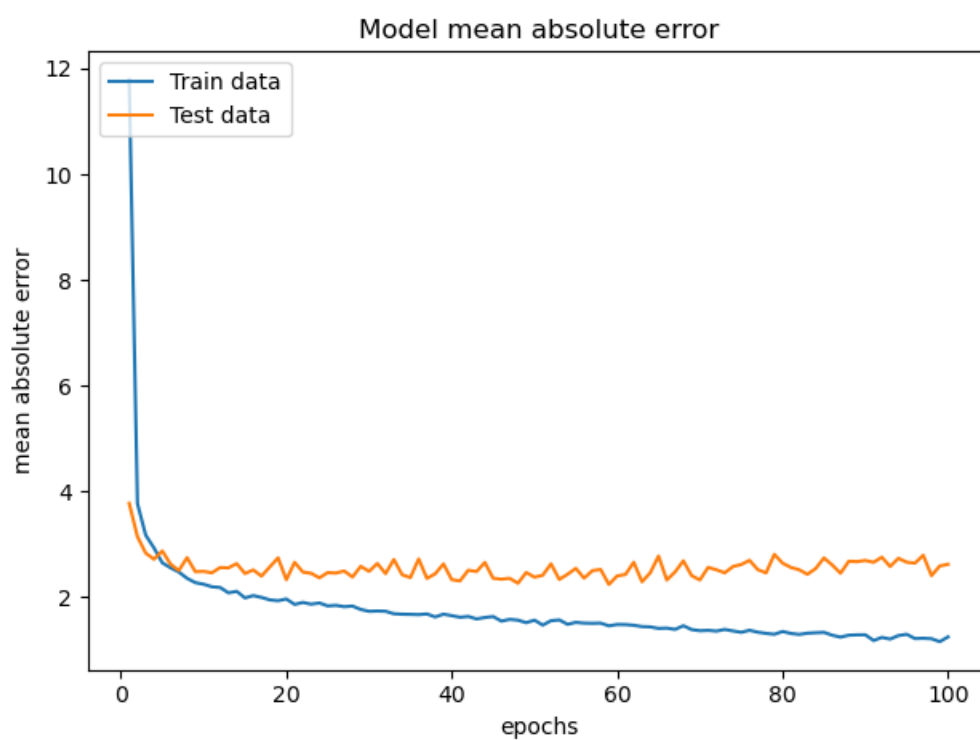


Рисунок 6 – График оценки mae $k=3$

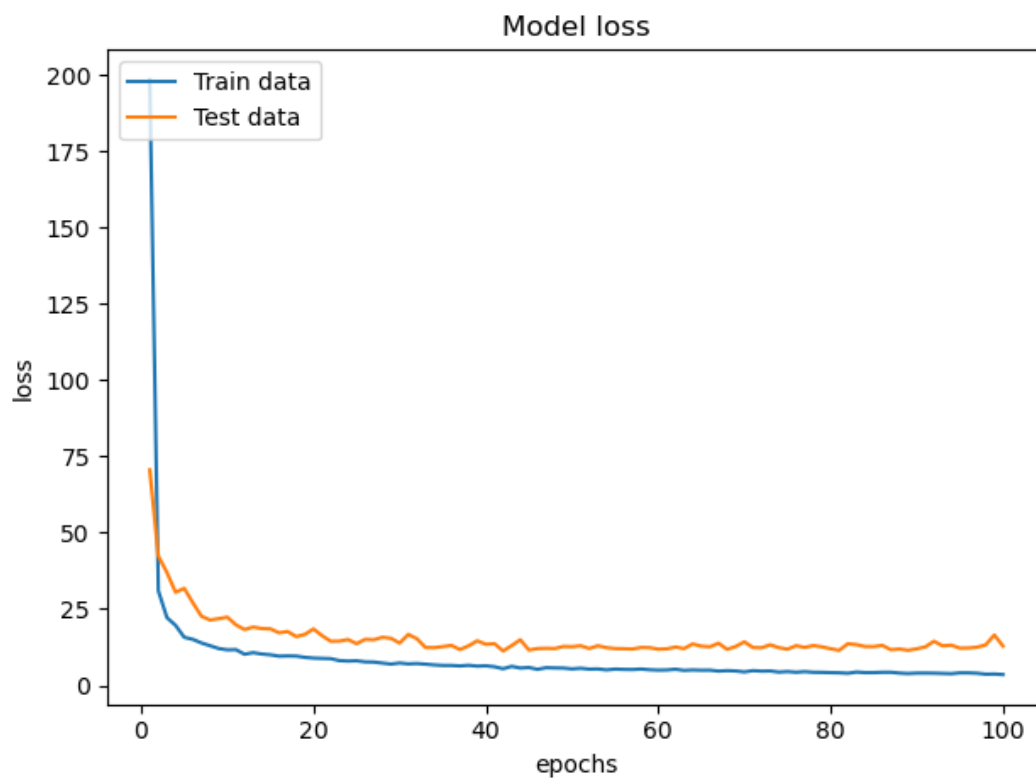


Рисунок 7 – График ошибки $k=4$

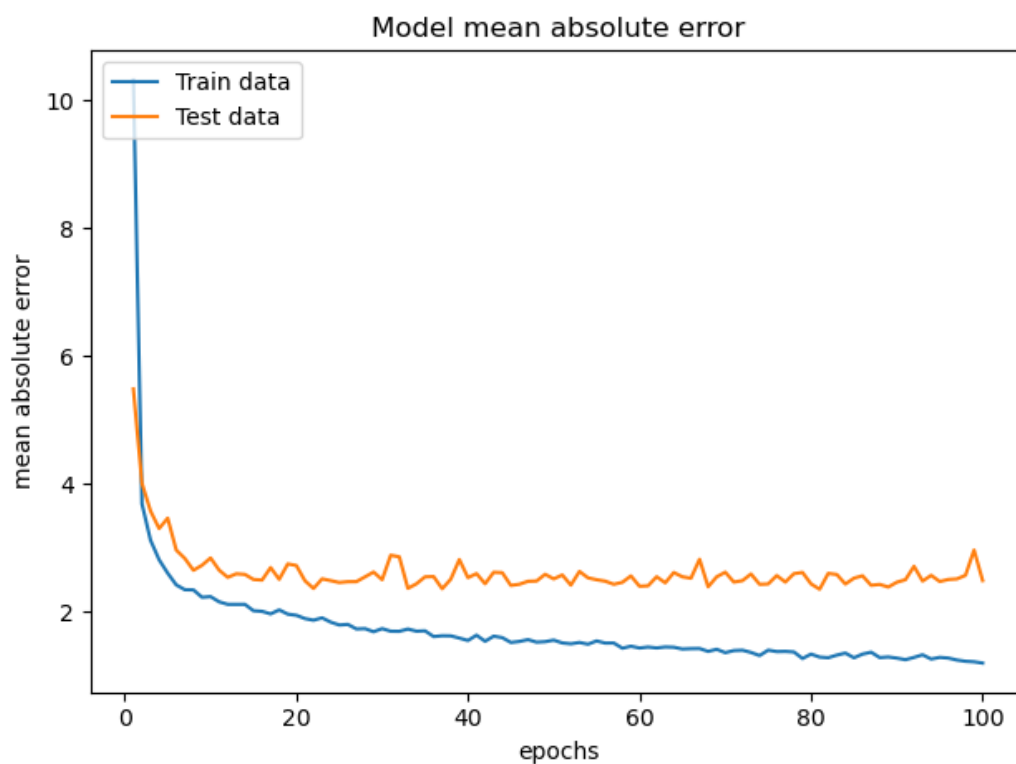


Рисунок 8 – График оценки mae $k=4$

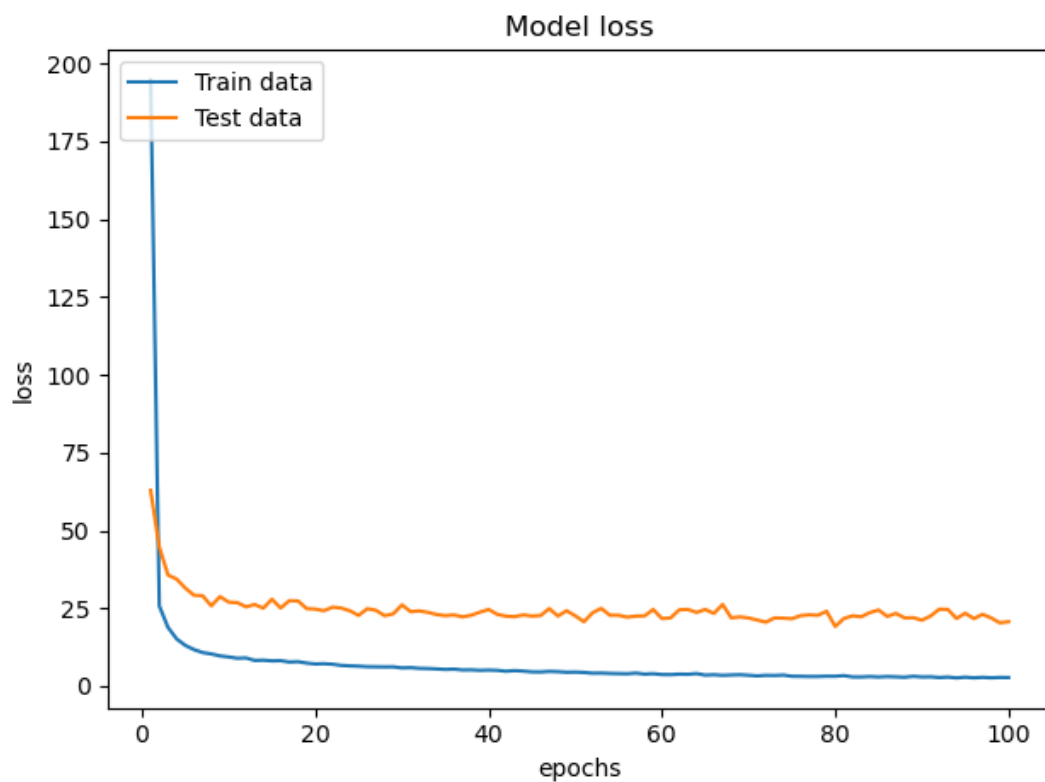


Рисунок 9 – График ошибки $k=5$

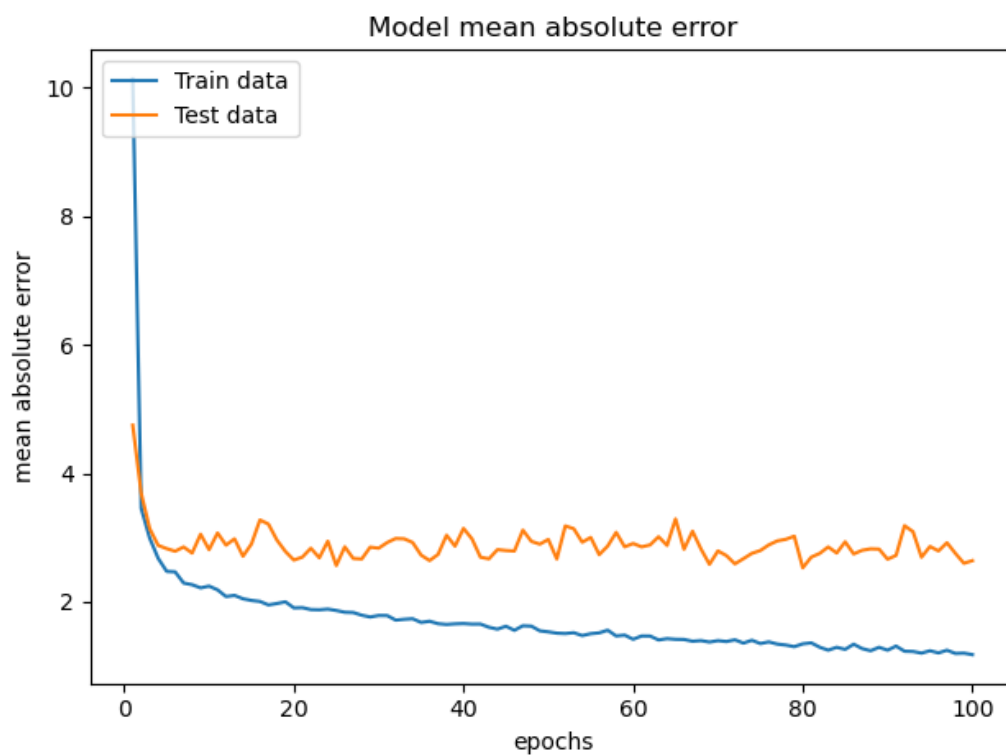


Рисунок 10 – График оценки mae $k=5$

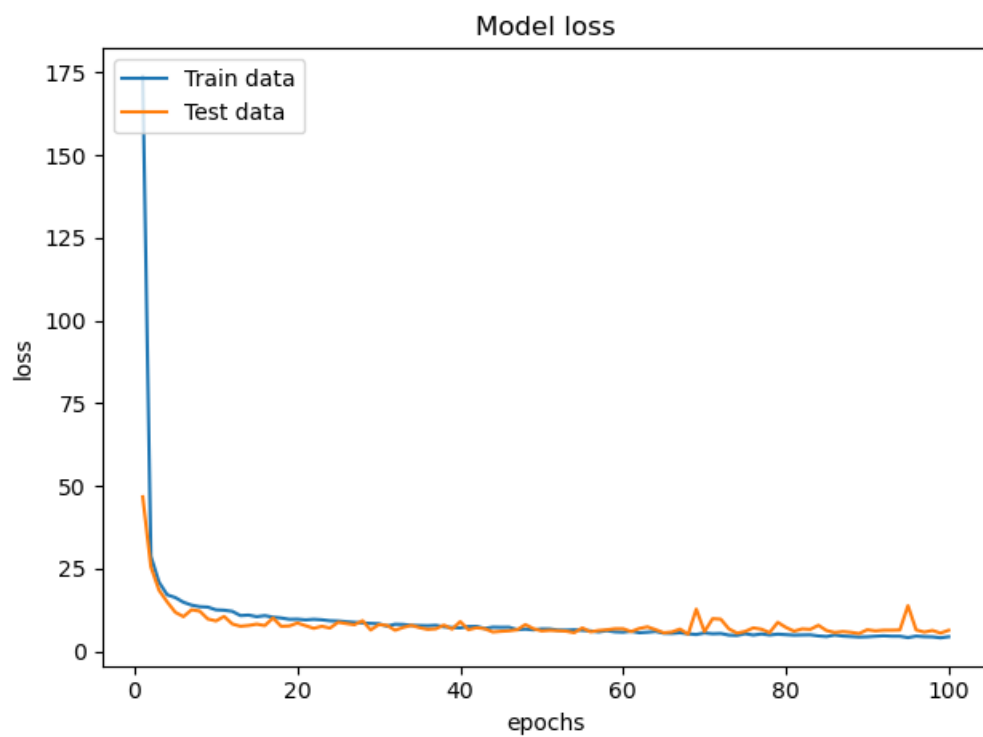


Рисунок 11 – График ошибки $k=6$

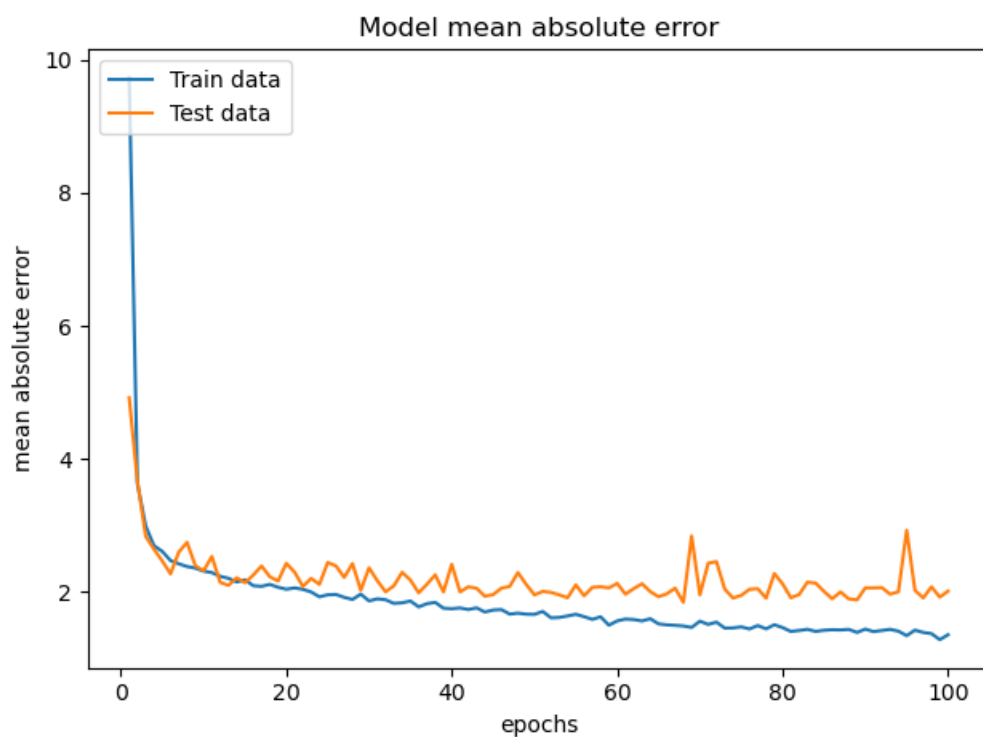


Рисунок 12 – График оценки mae $k=6$

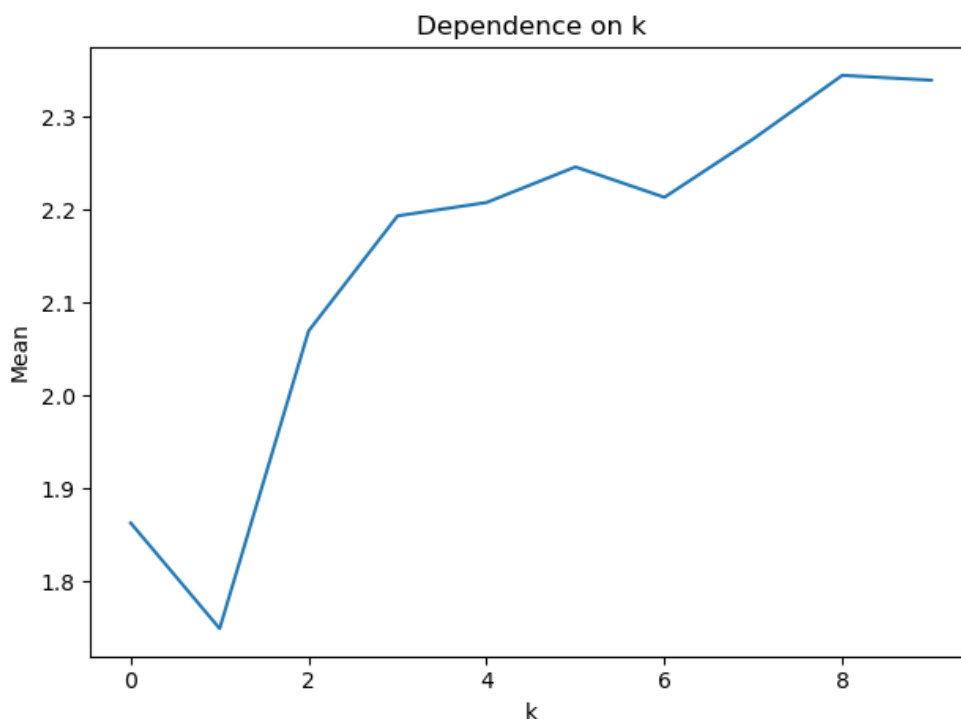


Рисунок 13 – Зависимость средней точности от числа блоков К при перекрестной проверке

Можно заметить, что с ростом числа блоков К точность увеличивается, но скорость увеличения точности уменьшается, поэтому при 6 блоках достигается оптимальное соотношение точности и временных затрат на перекрестную проверку.

Выводы.

В ходе работы было изучено влияние числа эпох на результат обучения в задаче регрессии, найдена точка переобучения, которое происходит на 30 эпохах. Оптимальным вариантом будет модель с 6-ю блоками и 30 эпохами.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
import tensorflow
import numpy as np
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import boston_housing
import matplotlib.pyplot as plt

class lab3:
    def __init__(self, epochs_count=100):
        (self.train_data, self.train_targets), (self.test_data,
self.test_targets) = boston_housing.load_data()
        self.mean = self.train_data.mean(axis=0)
        self.train_data -= self.mean
        self.std = self.train_data.std(axis=0)
        self.train_data /= self.std
        self.model = self.build_model()
        self.test_data -= self.mean
        self.test_data /= self.std
        self.k = 10
        self.num_val_samples = len(self.train_data) // self.k
        self.num_epochs = epochs_count
        self.all_scores = []

    def build_model(self):
        model = Sequential()
        model.add(Dense(64, activation='relu',
input_shape=(self.train_data.shape[1],)))
        model.add(Dense(64, activation='relu'))
        model.add(Dense(1))
        model.compile(optimizer='rmsprop', loss='mse',
metrics=['mae'])
        return model

    def find_overfit(self):
        for i in range(3, self.k):
            print('processing fold #', i)
            val_data = self.train_data[i * self.num_val_samples: (i +
1) * self.num_val_samples]
            val_targets = self.train_targets[i * self.num_val_samples:
(i + 1) * self.num_val_samples]
            partial_train_data = np.concatenate(
                [self.train_data[:i * self.num_val_samples],
self.train_data[(i + 1) * self.num_val_samples:]], axis=0)
            partial_train_targets = np.concatenate(
```

```

        [self.train_targets[:i * self.num_val_samples],
self.train_targets[(i + 1) * self.num_val_samples:]],
        axis=0)
        self.model = self.build_model()
        history = self.model.fit(partial_train_data,
partial_train_targets, epochs=self.num_epochs, batch_size=1,
                                verbose=0,
validation_data=(val_data, val_targets))
        loss = history.history['loss']
        mae = history.history['mean_absolute_error']
        v_loss = history.history['val_loss']
        v_mae = history.history['val_mean_absolute_error']
        x = range(1, self.num_epochs + 1)

        val_mse, val_mae = self.model.evaluate(val_data,
val_targets, verbose=0)
        self.all_scores.append(val_mae)
        plt.plot(x, loss)
        plt.plot(x, v_loss)
        plt.title('Model loss')
        plt.ylabel('loss')
        plt.xlabel('epochs')
        plt.legend(['Train data', 'Test data'], loc='upper left')
        plt.show()

        plt.plot(x, mae)
        plt.plot(x, v_mae)
        plt.title('Model mean absolute error')
        plt.ylabel('mean absolute error')
        plt.xlabel('epochs')
        plt.legend(['Train data', 'Test data'], loc='upper left')
        plt.show()

def fit_model(self):
    res = []
    for i in range(self.k):
        print('processing fold #', i)
        val_data = self.train_data[i * self.num_val_samples: (i +
1) * self.num_val_samples]
        val_targets = self.train_targets[i * self.num_val_samples:
(i + 1) * self.num_val_samples]
        partial_train_data = np.concatenate([self.train_data[:i *
self.num_val_samples], self.train_data[(i + 1) *
self.num_val_samples:]], axis=0)
        partial_train_targets = np.concatenate(
            [self.train_targets[:i * self.num_val_samples],
self.train_targets[(i + 1) * self.num_val_samples:]],
            axis=0)
        self.model = self.build_model()
        history = self.model.fit(partial_train_data,
partial_train_targets, epochs=self.num_epochs, batch_size=1,

```

```

                                verbose=0)
        val_mse,    val_mae    =    self.model.evaluate(val_data,
val_targets, verbose=0)
        self.all_scores.append(val_mae)
        res.append(np.mean(self.all_scores))
    plt.plot(range(self.k), res)
    plt.title('Dependence on k')
    plt.ylabel('Mean')
    plt.xlabel('k')
    plt.show()
    print(np.mean(self.all_scores))

```

```

lab = lab3(30)
lab.fit_model()

```