Министерство образования и науки Российской Федерации

**Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана»**

Кафедра «Прикладная математика»

Курсовая работа

по дисциплине «Методы вычислений»

Прогнозирование магнитных параметров солнечного излучения с помощью методов машинного обучения

Выполнил студент группы ФН12-61 *Голубев А.А.*

Научный руководитель к.т.н. *Макаренко А.В.*

Москва — 2018

Оглавление

Введение

В наше время широкое развитие получили методы машинного обучения. Наиболее распространенным подходом в настоящее время можно считать нейронные сети — методы глубокого обучения. Предпосылками для развития этих алгоритмов в последнее десятилетие стали: повышение вычислительной мощности компьютеров, стремительное увеличение доступных данных для обучения и развитие алгоритмов. В данной работе рассматриваются методы машинного обучения для решения задачи прогнозирования магнитных параметров солнечного излучения.

Постановка задачи

В качестве исходных данных задачи использовались ежедневные измерения параметров излучения, начиная с 2000-го года, которые были получены из лаборатории рентгеновской астрономии Солнца ФИАН.

В измерениях для каждой даты представлены следующие солнечные параметры:

1. **R10** - радиоизлучение на длине волны 10.7 см;
2. **SN** - число пятен на Солнце в этот день;
3. **SA** - общая площадь пятен в этот день (в миллионных долях площади солнечного диска);
4. **C** - число вспышек C класса в этот день;
5. **M** - число вспышек M класса (примерно в 10 раз более сильные, чем C);
6. **X** - число вспышек X класса (примерно в 10 раз более сильные, чем M);

Также, для каждой даты имеется совокупность магнитных параметров:

1. **A** - индекс A за этот день;
2. **K0003** - измерение индекса Kp для интервала с 00:00 до 03:00;
3. **K0306** - измерение индекса Kp для интервала с 03:00 до 06:00;
4. **K0609** - измерение индекса Kp для интервала с 06:00 до 09:00;
5. **K0912** - измерение индекса Kp для интервала с 09:00 до 12:00;
6. **K1215** - измерение индекса Kp для интервала с 12:00 до 15:00;
7. **K1518** - измерение индекса Kp для интервала с 15:00 до 18:00;
8. **K1821** - измерение индекса Kp для интервала с 18:00 до 21:00;
9. **K2124** - измерение индекса Kp для интервала с 21:00 до 24:00;

В совокупности данные состоят из 15 временных рядов, в каждом — около 6500 измерений.

Методами нейронных сетей необходимо построить прогнозирующую модель, определяющую значение магнитных параметров в зависимости от солнечных на заданное число дней вперед. Магнитные параметры должны прогнозироваться независимо друг от друга, то есть, каждый отдельно взятый магнитный параметр должен прогнозироваться по совокупности солнечных. Фактически, нужно получить независимую модель для каждого магнитного параметра.

1. Предварительный анализ данных

Прежде чем переходить к непосредственному построению модели, необходимо было провести обработку данных. Во-первых, пропуски данных были заполнены средними значениями соответствующих временных рядов. Затем был проведен статистический анализ данных для выбора подходящей модели. Для каждого параметра средствами библиотеки **matplotlib** были построены визуализации временных рядов и их спектральные плотности (в том числе в логарифмических масштабах по каждой из осей), гистограммы. Для всех параметров были построены графики корреляции.

На примере двух магнитных параметров **A** и **K0003** приведем получившиеся результаты. С помощью функции **matplotlib.pyplot.plot** были построены визуализации временных рядов.

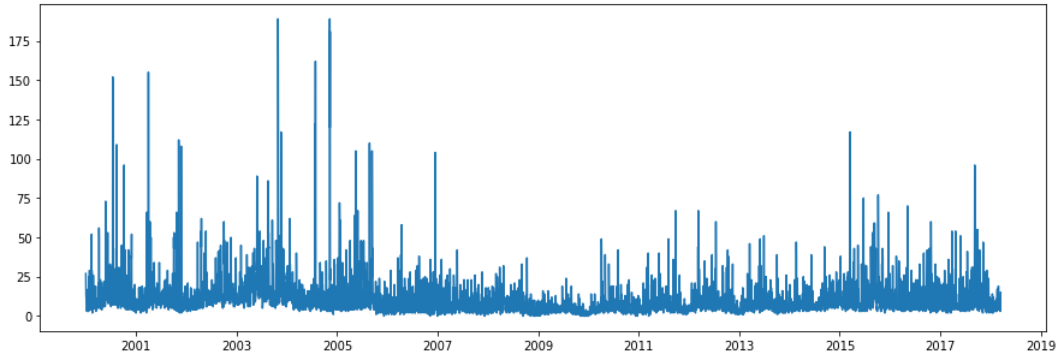

Рис. 1.1. Визуализация временного ряда параметра A

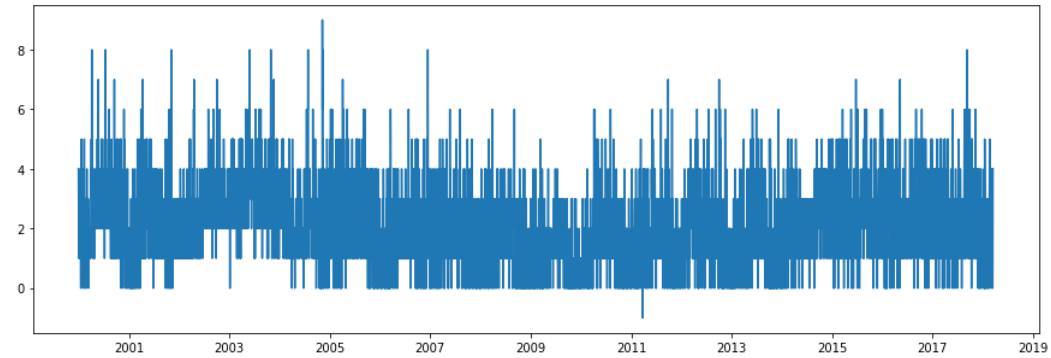

Рис. 1.2. Визуализация временного ряда параметра K0003

На этих графиках мы можем наблюдать высокую волатильность каждого из параметров. Наличие многочисленных выбросов на рисунке (1.1) свидетельствует о том, что прогнозирование статистическими методами будет обладать невысокой точностью.

С помощью функции **matplotlib.pyplot.psd** были построены графики спектральных плотностей, по которым удалось установить временную периодичность данных.

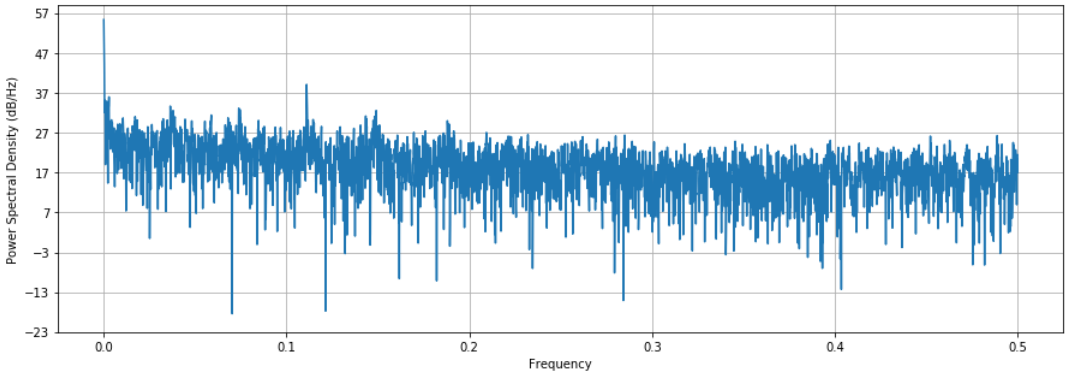

Рис. 1.3. Спектральная плотность параметра A

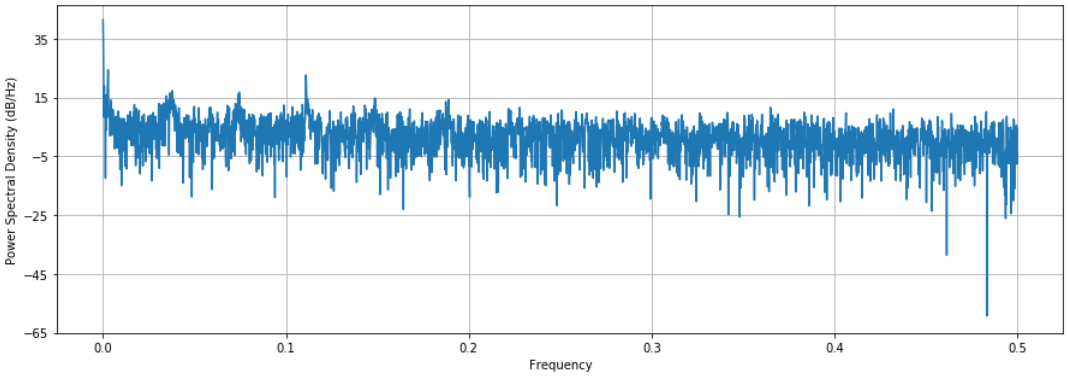

Рис. 1.4. Спектральная плотность параметра K0003

Из анализа максимумов спектральной плотности на рисунке (1.3) следует, что временной период составляет около 27 дней, что соответствует частоте $1/27 = 0.03703$. Это соответствует периоду вращения Солнца. Известно, что цикл солнечной активности составляет 11 лет. По причине малой глубины имеющихся данных обнаружить этот цикл не удалось. Информация о 27-дневном периоде в дальнейшем использовалась для корректного определения входных параметров модели.

Для изучения взаимной корреляции данных с помощью функции **matplotlib.pyplot.scatter** были построены графики попарных диаграмм рассеивания.

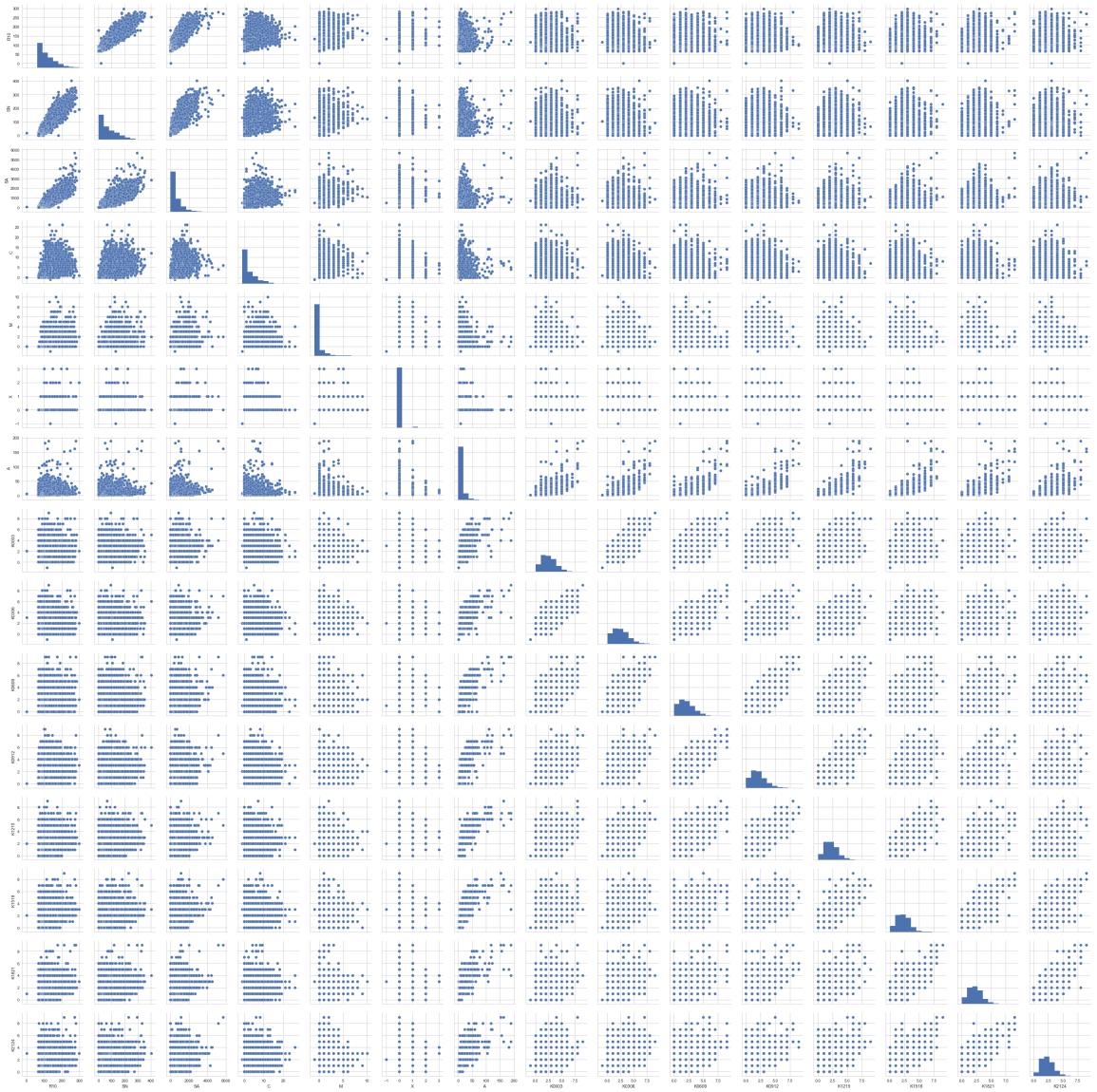

Рис. 1.5. Диаграмма рассеивания солнечных и магнитных параметров

В верхней правой четверти графика мы можем видеть рассеивание магнитных параметров в зависимости от солнечных. По внешнему виду диаграмм рассеивания, можно сделать вывод, что высокой степени корреляции не наблюдается. Это означает, что задача прогнозирования будет нетривиальна.

По результатам предварительного анализа данных можно сделать вывод, что статистические методы прогнозирования неприменимы к данным рядам. Этим объясняется наш выбор методов нейронных сетей для прогнозирования магнитных параметров солнечного излучения.

2. Машинное обучение. Задача регрессии

Машинное обучение — обширный подраздел искусственного интеллекта, изучающий методы построения алгоритмов, способных обучаться. Машинное обучение находится на стыке математической статистики, методов оптимизации и классических математических дисциплин, но имеет также и собственную специфику, связанную с проблемами вычислительной эффективности и переобучения. Многие методы машинного обучения разрабатывались как альтернатива классическим статистическим подходам.

Общая постановка задачи обучения по прецедентам. Дано конечное множество прецедентов, по каждому из которых собраны некоторые данные. Данные о прецеденте называют также его описанием. Совокупность всех имеющихся описаний прецедентов называется обучающей выборкой. Требуется по этим частным данным выявить общие зависимости, закономерности, взаимосвязи, присущие не только этой конкретной выборке, но вообще всем прецедентам, в том числе тем, которые ещё не наблюдались. Говорят также о восстановлении зависимостей по эмпирическим данным — этот термин был введён в работах Вапника и Червоненкиса.

Наиболее распространённым способом описания прецедентов является признаковое описание. Фиксируется совокупность n показателей, измеряемых у всех прецедентов. Если все n показателей числовые, то признаковые описания представляют собой числовые векторы размерности n. Возможны и более сложные случаи, когда прецеденты описываются временными рядами или сигналами, изображениями, видеорядами, текстами, попарными отношениями сходства или интенсивности взаимодействия, и т.д.

Для решения задачи обучения по прецедентам в первую очередь фиксируется модель восстанавливаемой зависимости. Затем вводится функционал качества, значение которого показывает, насколько хорошо модель описывает наблюдаемые данные. Алгоритм обучения ищет такой набор параметров модели, при котором функционал качества на заданной обучающей выборке принимает оптимальное значение. Процесс настройки модели по выборке данных в большинстве случаев сводится к применению численных методов оптимизации.

Задача прогнозирования магнитных параметров представляет из себя задачу регрессии машинного обучения. Ее суть состоит в нахождении заранее неизвестной функциональной зависимости величины магнитного параметра M от набора количественных признаков — солнечных параметров $\vec{S}$. Данная зависимость находится по множеству прецедентов $\vec{S}_i, M_i$. Мерой качества найденной оценки неизвестной функции является mae — среднее абсолютное значение ошибки оценки магнитного параметра M на обучающей выборке. Будем называть ее **функцией потерь**.

3. Описание прогнозирующей модели

Нейронная сеть представляет из себя нелинейное преобразование вектора признаков. Она изображена на рисунке (3.1), где $x_1..x_n$ — входные данные для слоя, $\omega_1..\omega_n$ — веса, b — смещение, y — выход слоя. Значком интеграла обозначена функция активации.

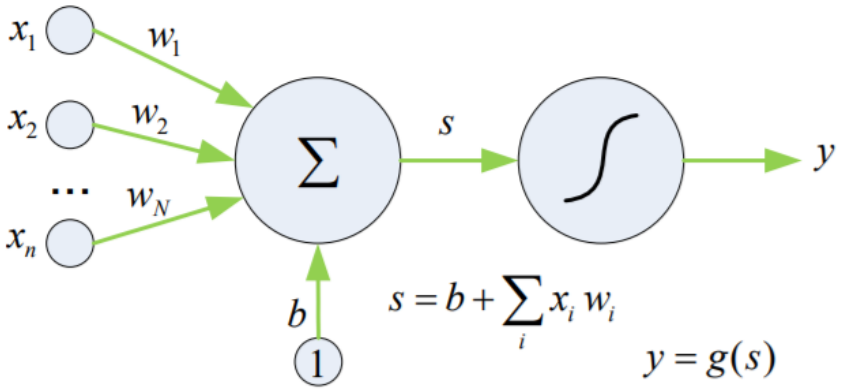

Рис. 3.1. Устройство слоя нейронной сети

Параметры ω и b отыскиваются на обучающей выборке из условия минимизации функции потерь. Модель обучения нейронной сети подразумевает следующие составные части:

1. слои, которые складываются в модель;
2. совокупность прецедентов M и S;
3. функция потерь, которая минимизируется в процессе обучения;
4. блок оптимизации для нахождения минимума функции потерь.

Процесс обучения нейронной сети можно представить в виде:

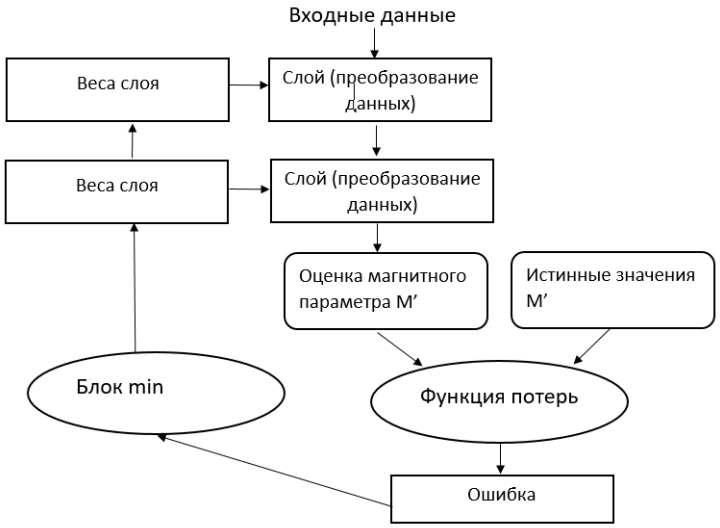

Рис. 3.2. Блок-схема процесса обучения нейронной сети

Если в нейронной сети присутствует более одного внутреннего слоя, она называется **глубокой**. В данной работе были рассмотрены три варианта архитектуры сети.

Полносвязная нейронная сеть (Fully Connected Neural Network, FCNN) формируется как последовательная комбинация элементарных нейронов. При расчетах использовалась трехслойный вариант сети. Во избежание проблемы переобучения были добавлены слои **keras.layers.Dropout**. Используемая функция активации — **sigmoid**. Для увеличения точности прогнозирования входные данные были разделены на 27-дневные части, что соответствует определенному выше периоду.

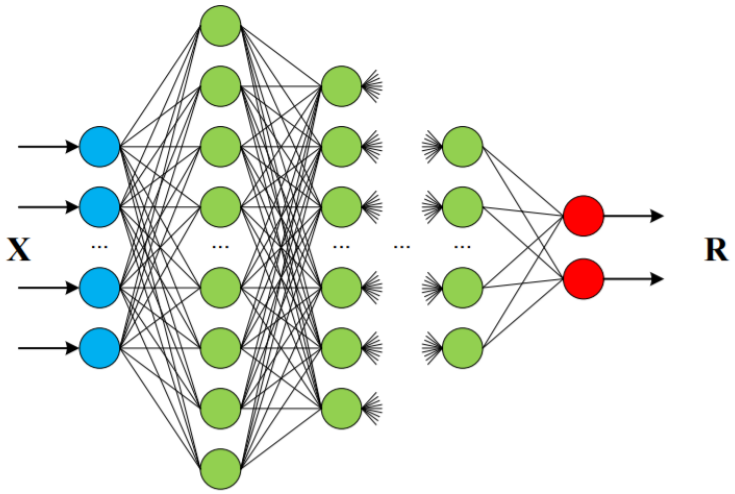

Рис. 3.3. Схема полносвязной нейронной сети

Рекуррентная нейронная сеть (Recurrent Neural Network, RNN) формируется как последовательная комбинация рекуррентных слоёв — слоёв охваченных обратными связями по состоянию нейрона и/или через его вход/выход. Эта сеть подходит для обучения на временных рядах.

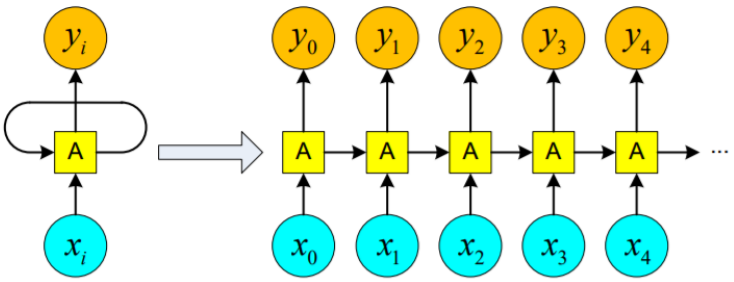

Рис. 3.4. Схема рекуррентной нейронной сети

В работе использовались два типа рекуррентных сетей: **GRU** и **LSTM**.

4. Анализ результатов

Среди рассмотренных сетей для решения данной задачи лучше всего проявила себя полносвязная нейронная сеть. Максимальная точность прогнозирования, характеризуемая коэффициентом детерминации, составила приблизительно 0.71. В качестве результатов приведем два магнитных параметра, точность прогнозирования которых оказалась наилучшей и наихудшей среди всех.

Наилучший результат. Параметр K0306.

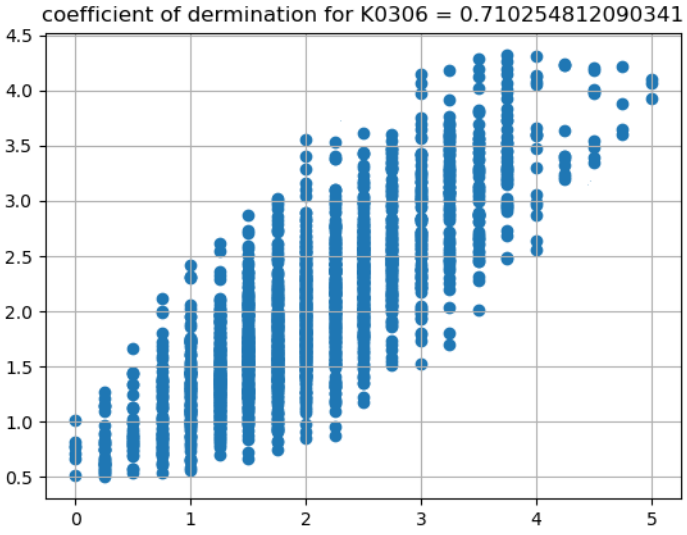

Рис. 4.1. Коэффициент детерминации

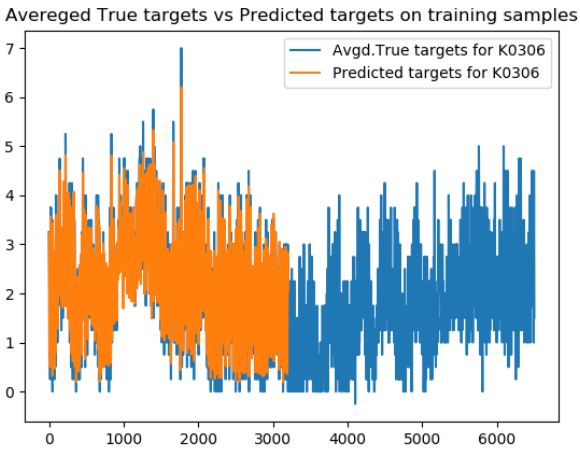

Рис. 4.2. Тренировочная выборка

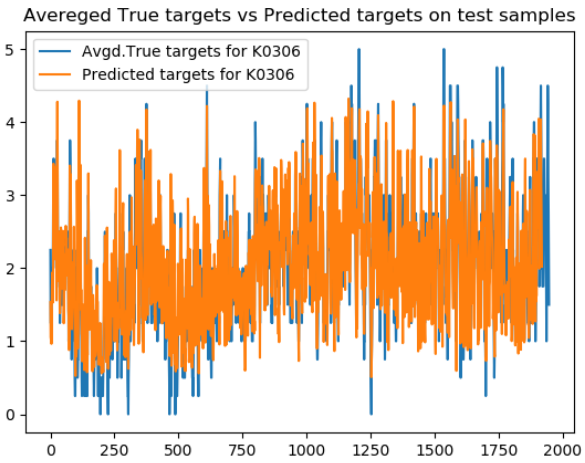

Рис. 4.3. Тестовая выборка

Наихудший результат. Параметр K1518.

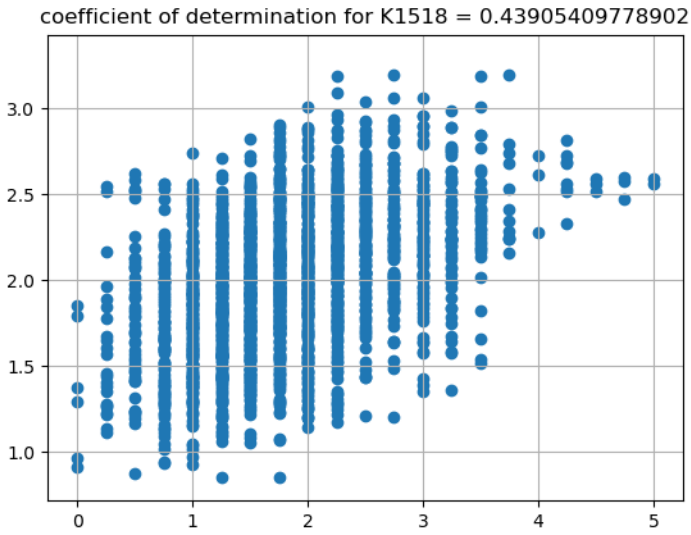

Рис. 4.4. Коэффициент детерминации

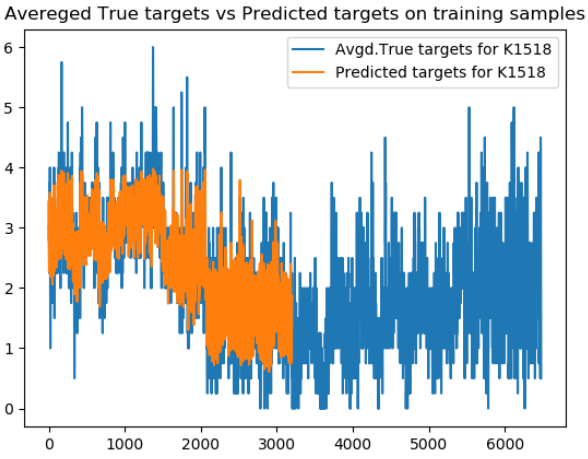

Рис. 4.5. Тренировочная выборка

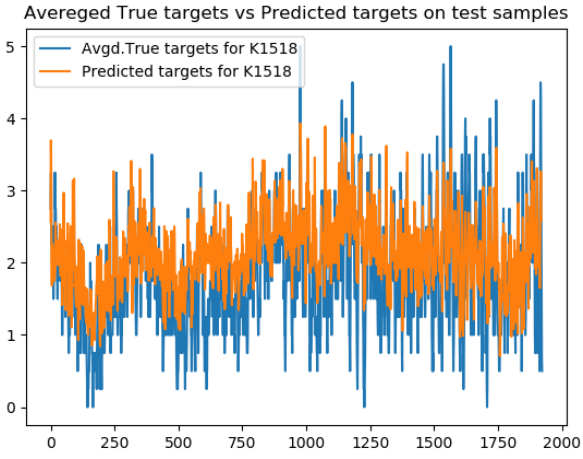

Рис. 4.6. Тестовая выборка

По результатам работы получили удовлетворительную точность прогнозирования некоторых параметров. Для увеличения точности необходимо применять модели с более сложной архитектурой, что потребует больших вычислительных затрат.

Заключение

В ходе выполнения курсовой работы были изучены основы машинного обучения и языка программирования Python, получено представление о работе с данными средствами библиотек **numpy** и **pandas**. Были получены навыки работы с нейронными сетями типа FCNN, GRU и LSTM, а также построена прогнозирующая модель с адекватным значением точности.

Список литературы

1. Себастьян Рашка, Python Machine Learning. — Отдельное издание., испр. — ДМК Пресс, 2017. — 418 с. — ISBN 978-5-97060-409-0
2. Дэвид Бизли, Python: Essential Reference. — Отдельное издание., испр. — Символ-Плюс, 2010. — 864 с. — ISBN 978-5-93286-157-8
3. Francois Chollet, Deep Learning with Python, Manning Publications Co. — 2018. — 386 с. — ISBN 978-1-61729-443-3
4. Ивченко Г. И., Медведев Ю. Н., Математическая статистика: Учебник. — М.: Книжный дом "ЛИБРОКОМ 2014. — 352 с. — ISBN 978–5–397–04141–6
5. Макаренко А. В., Методы глубокого обучения: сегодняшние возможности и ближайшие перспективы

Приложение

Листинг программы на языке Python

```
from __future__ import print_function
import numpy as np
import sys
import os
import argparse

input_dir = None
output_dir = None
log_dir = None
val_steps=None
test_steps =None

import numpy as np
import keras
keras.__version__
from keras import backend as K

from keras.models import Model
from keras.models import Sequential
from keras.layers import Input
from keras.layers import Lambda
from keras.layers import Layer
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Activation
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.optimizers import SGD
from keras.optimizers import RMSprop
from sklearn.metrics import r2_score
from keras import regularizers

def generator_train(data, lookback, delay, min_index, max_index, numtargets,
numfeatures, shuffle=False, batch_size=128, step=6):
    if max_index is None:
        max_index = len(data) - delay - 1
    i = min_index + lookback
    js=0
    while 1:
        if shuffle:
            rows = np.random.randint(
                min_index + lookback, max_index, size=batch_size)
```

```
        else:
            if i + batch_size >= max_index:
                i = min_index + lookback
            rows = np.arange(i, min(i + batch_size, max_index))

        samples = np.zeros((len(rows),
                            lookback // step,
                            #data.shape[-1]
                            numfeatures.shape[0]
                            ))
        targets = np.zeros((len(rows), numtargets.shape[0]))
        for j, row in enumerate(rows):
            indices = range(rows[j] - lookback, rows[j], step)
            samples[j] = data[indices][:,numfeatures]
            targets[j] = data[rows[j] + delay][numtargets]
        i += len(rows)
        js+=1
        yield samples, targets

def generator_val(data, lookback, delay, min_index, max_index, numtargets,
 numfeatures, shuffle=False, batch_size=128, step=6):
    if max_index is None:
        max_index = len(data) - delay - 1
    i = min_index + lookback
    js=0
    while 1:
        if shuffle:
            rows = np.random.randint(
                min_index + lookback, max_index, size=batch_size)
        else:
            if i + batch_size >= max_index:
                i = min_index + lookback
            rows = np.arange(i, min(i + batch_size, max_index))

        samples = np.zeros((len(rows),
                            lookback // step,
                             #data.shape[-1]
                            numfeatures.shape[0]
                            ))
        targets = np.zeros((len(rows), numtargets.shape[0]))
        for j, row in enumerate(rows):
            indices = range(rows[j] - lookback, rows[j], step)
            samples[j] = data[indices][:,numfeatures]
            targets[j] = data[rows[j] + delay][numtargets]
        i += len(rows)
        js+=1
        yield samples, targets
```

```
def generator_test(data, lookback, delay, min_index, max_index, numtargets,
numfeatures, shuffle=False, batch_size=128, step=6):
    if max_index is None:
        max_index = len(data) - delay - 1
    i = min_index + lookback
    js=0
    while 1:
        if shuffle:
            rows = np.random.randint(
                min_index + lookback, max_index, size=batch_size)
        else:
            if i + batch_size >= max_index:
                i = min_index + lookback
            rows = np.arange(i, min(i + batch_size, max_index))
            i += len(rows)
            js+=1
        samples = np.zeros((len(rows),
                            lookback // step,
                            #data.shape[-1]
                            numfeatures.shape[0]
                            ))
        targets = np.zeros((len(rows), numtargets.shape[0]))
        for j, row in enumerate(rows):
            indices = range(rows[j] - lookback, rows[j], step)
            samples[j] = data[indices][:,numfeatures]
            targets[j] = data[rows[j] + delay][numtargets]
        yield samples, targets

def main():

    f = open('data_whole.csv')
    data = f.read()
    f.close()

    lines = data.split('\n')
    header = lines[0].split(';')
    header=header[1:]
    header_array=np.array(header,dtype='<U7')

    lines = lines[1:]

    print(header)
    print(len(lines))
    #import numpy as np

    dataset_len = len(lines)
```

```
train_index = int(dataset_len*0.5)
val_index = int(dataset_len*0.7)
test_index = dataset_len-1
lookback = 27
step = 1
batch_size = 128
min_index=0
max_index=0
shuffle=False
delay=1
steps_per_epoch=dataset_len//batch_size

float_data = np.zeros((len(lines), len(header)))
for i, line in enumerate(lines):
    values = [float(x) for x in line.split(';')[1:]]
    float_data[i, :] = values

from matplotlib import pyplot as plt

list_num_targets = [14]
list_num_features = [0,1,2,3,4,5,6,7,8,9,10,11,12,13]

num_targets = np.array(list_num_targets,dtype='int32')
num_features = np.array(list_num_features,dtype='int32')

print(header_array[list_num_targets])
print(header_array[list_num_features])

temp =np.array( float_data[:, num_targets])

MA=5
tempMA=np.zeros(temp.shape)

for i in range(temp.shape[0]):
        tempMA[i]=temp[max(0,i-MA//2):min(i+MA//2,
        temp.shape[0]-1)].mean(axis=0)

mean = float_data[:].mean(axis=0)
float_data -= mean
std = float_data[:].std(axis=0)
float_data /= std


train_gen =generator_train(float_data,lookback=lookback, delay=delay,
min_index=0, max_index=train_index,numtargets=num_targets,
numfeatures=num_features, shuffle=False,  batch_size=batch_size,step=step)
```

```
val_gen = generator_val(float_data,lookback=lookback,delay=delay,
min_index=train_index +1, max_index=val_index, numtargets=num_targets,
numfeatures=num_features,shuffle=False, batch_size=batch_size,step=step)

test_gen = generator_test(float_data,lookback=lookback,delay=delay,
min_index=val_index+1,max_index=None,numtargets=num_targets,
numfeatures=num_features, shuffle=False,batch_size=batch_size,step=step)

train_steps=(train_index-lookback+1)// batch_size
val_steps = (val_index  - train_index - lookback) // batch_size

test_steps = (len(float_data) -delay - val_index-1 ) // batch_size

model=Sequential()

model.add(keras.layers.Flatten(input_shape=(lookback//step,
num_features.shape[0])))

model.add(keras.layers.Dense(128, activation='sigmoid'))
model.add(keras.layers.Dropout(0.6))
model.add(keras.layers.Dense(128, activation='sigmoid'))
model.add(keras.layers.Dropout(0.6))
#model.add(keras.layers.Dense(256, activation='sigmoid'))
#model.add(keras.layers.Dropout(0.4))
model.add(keras.layers.Dense(num_targets.shape[0]))
model.compile(optimizer=RMSprop(lr=0.0007), loss='mae')
#sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
#model.compile(optimizer=sgd, loss='mae')
history = model.fit_generator(train_gen,
                              steps_per_epoch=train_steps,
                              epochs=36,
                              validation_data=val_gen,
                              validation_steps= val_steps)

loss=history.history['loss']
val_loss=history.history['val_loss']
epochs=range(len(loss))
plt.figure()

plt.plot(epochs, loss,'bo',label='Training loss')
plt.plot(epochs, val_loss,'b',label='Validating loss')
plt.title('Training and Validating loss')
plt.legend()
#plt.xlabel('Epochs')
#plt.ylabel('Validation MAE')
plt.show()
```

```
#reset train generator
train_gen =generator_train(float_data,lookback=lookback,
delay=delay, min_index=0, max_index=train_index, numtargets = num_targets,
numfeatures = num_features, shuffle=False, batch_size=batch_size, step=step)
trainPredict =model.predict_generator(train_gen,steps = train_steps)

#reset test generator
test_gen = generator_test(float_data,lookback=lookback,delay=delay,
min_index=val_index+1, max_index=None, numtargets = num_targets,
numfeatures = num_features, shuffle=False,batch_size=batch_size,step=step)

testPredict =model.predict_generator(test_gen,steps = test_steps)

trainPredict *=std[num_targets]
trainPredict +=mean[num_targets]
testPredict *=std[num_targets]
testPredict +=mean[num_targets]

for i,num in enumerate(num_targets):
    plt.figure()
    plt.plot(tempMA[lookback+delay:,i],
    label='Avgd.True targets for '+header[num])

    plt.plot(trainPredict[:,i],label='Predicted targets for '+header[num])
    plt.title('Avereged True targets vs
    Predicted targets on training samples')

    plt.legend()
    plt.show()

    plt.figure()
    plt.plot(tempMA[1+val_index +lookback+delay:,i],label
    ='Avgd.True targets for '+header[num])

    plt.plot(testPredict[:,i],label=
    'Predicted targets for '+header[num])

    plt.title('Avereged True targets vs
    Predicted targets on test samples')
    plt.legend()
    plt.show()

    r2= coefficient_of_determination = r2_score(tempMA[1+val_index +
    1+delay:1+val_index+1+delay+testPredict.shape[0],i],
    testPredict[:,i])

    print("coefficient of determination for " + header[num] +" = ", r2)
```

```
plt.figure()
plt.scatter(tempMA[1+val_index +lookback+delay:1+val_index +
lookback+delay+testPredict.shape[0],i],testPredict[:,i])

plt.title("coefficient of determination
for " + header[num] + " = " +str(r2))

plt.grid()
plt.show()
```