МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева (Самарский университет)»

Институт информатики, математики и электроники

Факультет информатики Кафедра программных систем

# ОТЧЁТ

к лабораторной работе №2 по дисциплине

«Нейронные сети глубокого обучения»

Студент В.А. Артамонов

Студент Д.А. Смирнов

Преподаватель А.Н. Жданова

Самара 2022

СОДЕРЖАНИЕ

[ОТЧЁТ 1](#_Toc73570772)

[1 Постановка задачи 3](#_Toc73570773)

[2 Библиотеки python 4](#_Toc73570774)

[3 Архитектура модели ARIMAX 6](#_Toc73570775)

[4 Вычислительные эксперименты 7](#_Toc73570776)

[5 Результаты работы 8](#_Toc73570777)

[6 Вывод 8](#_Toc73570778)

[Список использованных источников 9](#_Toc73570779)

[Приложение А Исходный код 10](#_Toc73570780)

1. Постановка задачи

Цель лабораторной работы: реализовать модель для прогнозирования временных рядов.

В статистике, обработке сигналов и многих других областях под временным рядом понимаются последовательно измеренные через некоторые (зачастую равные) промежутки времени данные. Анализ временных рядов объединяет методы изучения временных рядов, как пытающиеся понять природу точек данных (откуда они взялись? что их породило?), так и пытающиеся построить прогноз. Прогнозирование временных рядов заключается в построении модели для предсказания будущих событий основываясь на известных событиях прошлого, предсказания будущих данных до того, как они будут измерены.

Понятие анализ временных рядов используется для того, чтобы отделить эту задачу от в первую очередь от более простых задач анализа данных (когда нет естественного порядка поступления наблюдений) и, во-вторых, от анализа пространственных данных, в котором наблюдения зачастую связаны с географическим положением. Модель временного ряда в общем смысле отражает идею, что близкие во времени наблюдения будут теснее связаны, чем удалённые. Кроме того, модели временных рядов зачастую используют однонаправленный порядок по времени в том смысле, что значения в ряду выражаются в некотором виде через прошлые значения, а не через последующие.

Методы анализа временных рядов зачастую делят на два класса: анализ в частотной области и анализ во временной области. Первый основывается на спектральном анализе и с недавних пор вейвлетном анализе, и может рассматриваться в качестве не использующих модели методов анализа, хорошо подходящих для исследований на этапе разведки. Методы анализа во временной области также имеют безмодельное подмножество, состоящее из кросскорреляционного анализа и автокорреляционного анализа, но именно здесь появляются частично и полностью определённые модели временных рядов.

Существует большое число вариантов обозначения временных рядов. Одним из типичных является X= {X1, X2, …}, обозначающее ряд с натуральными индексами. Другое стандартное представление: Y= {Yt: t ϵ T}.

Общий вид авторегрессивной модели задаётся следующим образом:

Yt = α0+ α1Yt-1+ α2Yt-2+…+ αpYt-p+εt, где εt – источник случайность, белый шум. Белый шум имеет следующие свойства:

E[εt] = 0

E[] = σ2

E[εt εs] = 0, t ≠ s

В этих предположениях процесс определён вплоть до моментов второго порядка и, при определённых условиях на коэффициенты, может быть стационарным в широком смысле.

Если шумы имеют нормальное распределение, их называют нормальным белым шумом. В этом случае авторегрессионный процесс может быть строго стационарен, опять же, при выполнении некоторых условий на коэффициенты [1].

1. Библиотеки python

При реализации лабораторной работы использовались библиотеки: pandas и numpy.

* 1. Pandas

Pandas – это быстрый, мощный, гибкий и простой в использовании инструмент для анализа и обработки данных с открытым исходным кодом, созданный на языке программирования Python. На данный момент библиотека Pandas является ключевой в анализе данных (Data Mining).

В основе Pandas лежит DataFrame – структура данных табличного типа. Любое табличное представление данных, например, электронные таблицы или базы данных, можно использовать как DataFrame. Объект DataFrame составлен из объектов Series – одномерных массивов, объединенных под одним названием и типом данных.

* 1. NumPy

NumPy – это фундаментальный пакет для научных вычислений на Python. Это библиотека Python, которая предоставляет объект многомерного массива, различные производные объекты (такие как замаскированные массивы и матрицы) и набор процедур для быстрых операций с массивами, включая математические, логические, манипуляции с формами, сортировку, выбор, ввод-вывод, дискретные преобразования Фурье, базовая линейная алгебра, базовые статистические операции, случайное моделирование и многое другое.

В основе пакета NumPy лежит объект ndarray. Это инкапсулирует n -мерные массивы однородных типов данных, при этом многие операции выполняются в скомпилированном коде для повышения производительности. Между массивами NumPy и стандартными последовательностями Python есть несколько важных отличий:

Массивы NumPy имеют фиксированный размер при создании, в отличие от списков Python (которые могут расти динамически). Изменение размера ndarray создаст новый массив и удалит оригинал.

Все элементы в массиве NumPy должны иметь один и тот же тип данных и, следовательно, будут иметь одинаковый размер в памяти. Исключение: можно иметь массивы объектов (Python, включая NumPy), что позволяет создавать массивы элементов разного размера.

Массивы NumPy упрощают сложные математические и другие типы операций с большим количеством данных. Как правило, такие операции выполняются более эффективно и с меньшим количеством кода, чем это возможно при использовании встроенных последовательностей Python.

Растущее множество научных и математических пакетов на основе Python используют массивы NumPy; хотя они обычно поддерживают ввод последовательности Python, они преобразуют такой ввод в массивы NumPy перед обработкой и часто выводят массивы NumPy. Другими словами, чтобы эффективно использовать большую часть (возможно, даже большую часть) современного научного / математического программного обеспечения на основе Python, недостаточно просто знать, как использовать встроенные типы последовательностей Python – также необходимо знать, как использовать массивы NumPy [3].

1. Архитектура модели ARIMAX

ARIMAX (AutoRegressive Integrated Moving Average eXtended) — это математическая модель для анализа временных рядов, объединяющая в себе интегрированную авторегрессию, скользящее среднее и возможность учета дополнительных внешних факторов.

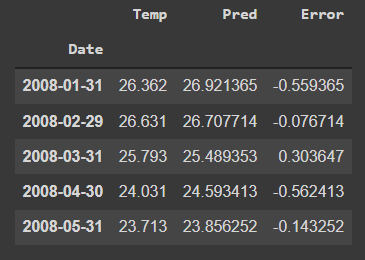
Модели ARIMA применяются для решения задач, в которых требуется построить прогноз на основе имеющихся данных, то есть вычислить последующие значения ряда на основе предыдущих. Временным рядом могут быть любые данные в разрезе времени, например, продажи товаров, количество заказов, поток клиентов и т.д [4].

1. Вычислительные эксперименты

Вычислительные эксперименты заключались обучении модели SARIMAX.

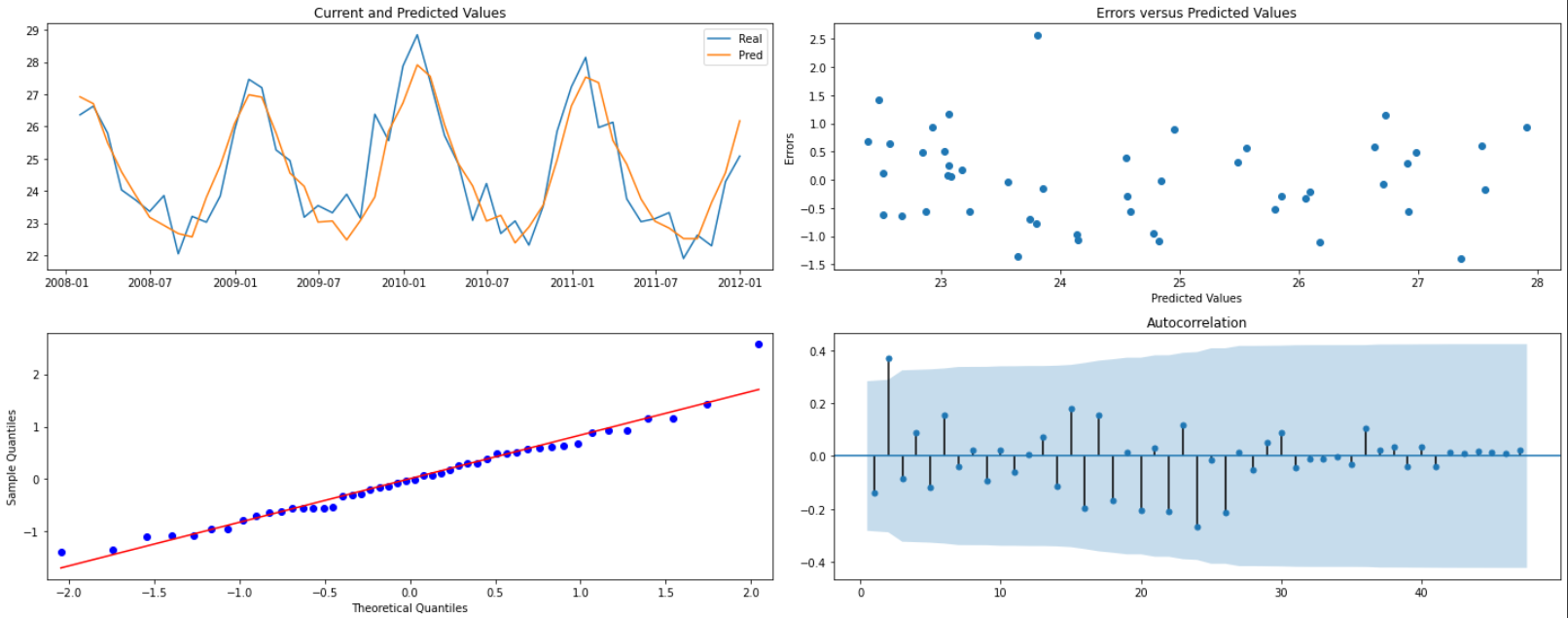
На рисунке 1 представлены значения фактической температуры, предсказанной и разница между ними в градусах (первые 5 позиций).

Рисунок 1 – История прогнозирования температуры (первые 5 позиций)



На рисунке 2 представлено создание нового объединения набора для обучения и проверки состоящий из: сравнения фактической и спрогнозированной температур, сравнение значения фактических и спрогнозированных, теоретических квантилей и автокорреляции.

Рисунок 2 – Графики фактической и спрогнозированной температуры



На рисунке 3 отображено сравнение значений в проверяющей модели, построенной на тех же параметрах подобранной модели для обучения и проверки.



Рисунок 3 – Графики текущих и экстраполированных значений

На рисунке 4 предоставлена полная статистика по обученной модели.

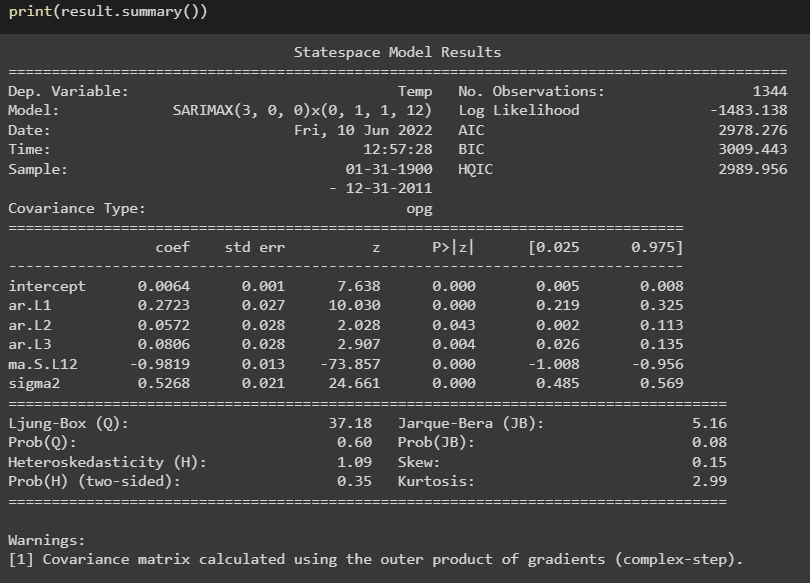


Рисунок 4 – Суммарная статистика по модели

На рисунке 5 продемонстрирован прогноз временного ряда обученной модели на последущие 3 года.

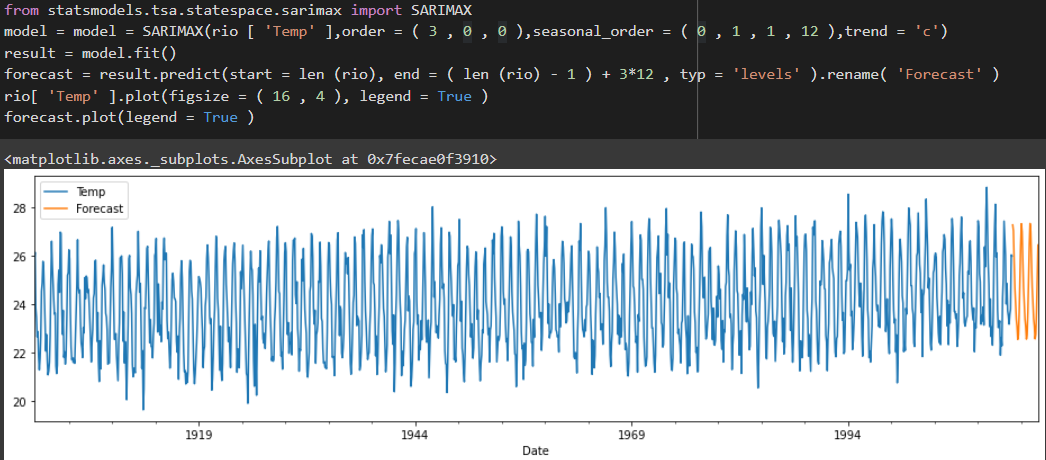


Рисунок 5 – Прогнозирование временного ряда на 3 последующих года

1. Результаты работы

Был произведен расчет среднеквадратического отклонения, которое составляет 0.3996200930145111.

Была построена и обучена модель для прогнозирования временных рядов и получен предикт на последующие 3 года на основе модели.

Исходный код лабораторной работы представлен в приложении А и доступен по ссылке – https://colab.research.google.com/drive/16rmaMWFuQ9XqqAPNggY-2kVxIR773FtQ?usp=sharing

1. Вывод

В лабораторной работе была реализована модель для прогнозирования временных рядов при помощи библиотек Pandas и NumPy для прогнозирования температуры в Рио де Жанейро. Также, были изучены: метод обучения, параметры обучения, параметры модели.

**Список использованных источников**

1. Временной ряд [Электронный ресурс]. URL: http://www.machinelearning.ru/wiki/index.php?title=Временной\_ряд (дата обращения: 19.05.2022).
2. Pandas [Электронный ресурс]. URL: https://python-school.ru/wiki/pandas/#:~:text=Pandas%20–%20это%20быстрый%2C%20мощный%2C,в%20анализе%20данных%20(Data%20Mining) (дата обращения: 19.05.2022).
3. NumPy [Электронный ресурс]. URL: https://ru.stackoverflow.com/questions/1172451/Что-такое-numpy-и-с-чем-его-едят#:~:text=NumPy%20-%20это%20фундаментальный%20пакет,случайное%20моделирование%20и%20многое%20другое (дата обращения: 19.05.2022).
4. ARIMAX [Электронный ресурс]. URL: https://help.loginom.ru/userguide/processors/datamining/arimax.html (дата обращения: 19.05.2022).

Приложение А

Исходный код

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import statsmodels.api as sm

from statsmodels.tsa.stattools import adfuller

from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf

from sklearn.metrics import mean\_squared\_error

from math import sqrt

import warnings

warnings.filterwarnings('ignore')

%matplotlib inline

from google.colab import drive

drive.mount('/content/drive', force\_remount=True)

!cp /content/drive/'MyDrive'/archive.zip .

!unzip /content/archive.zip

cities = pd.read\_csv('/content/GlobalLandTemperaturesByCity.csv')

rio = cities.loc[cities['City'] == 'Rio De Janeiro', ['dt','AverageTemperature']]

rio.columns = ['Date','Temp']

rio['Date'] = pd.to\_datetime(rio['Date'])

rio.reset\_index(drop=True, inplace=True)

rio.set\_index('Date', inplace=True)

rio = rio.loc['1900':'2013-01-01']

rio = rio.asfreq('M', method='bfill')

rio.head()

plt.figure(figsize=(22,6))

sns.lineplot(x=rio.index, y=rio['Temp'])

plt.title('Температура в Рио с 1900 по 2012')

plt.show()

# как меняется температура по месяцам

rio['month'] = rio.index.month

rio['year'] = rio.index.year

pivot = pd.pivot\_table(rio, values='Temp', index='month', columns='year', aggfunc='mean')

pivot.plot(figsize=(20,6))

plt.title('Температура в Рио по годам')

plt.xlabel('Месяца')

plt.ylabel('Температура')

plt.xticks([x for x in range(1,13)])

plt.legend().remove()

plt.show()

monthly\_seasonality = pivot.mean(axis=1)

monthly\_seasonality.plot(figsize=(20,6))

plt.title('Изменение температуры по месяцам за год')

plt.xlabel('Месяца')

plt.ylabel('Температура')

plt.xticks([x for x in range(1,13)])

plt.show()

# Более высокие температуры наблюдаются примерно в ноябре и феврале, а более низкие-в период с июля по сентябрь

year\_avg = pd.pivot\_table(rio, values='Temp', index='year', aggfunc='mean')

year\_avg['10 Years MA'] = year\_avg['Temp'].rolling(10).mean()

year\_avg[['Temp','10 Years MA']].plot(figsize=(20,6))

plt.title('Годовая средняя температура в Рио')

plt.xlabel('Месяца')

plt.ylabel('Температура')

plt.xticks([x for x in range(1900,2012,3)])

plt.show()

# разбиваем данные на тренировочный, тестовый и валидационный наборы

train = rio[:-60].copy()

val = rio[-60:-12].copy()

test = rio[-12:].copy()

# Убираем NaN значения и готовим данные. Готовим базовый прогноз, к которому будем стремиться

baseline = val['Temp'].shift()

baseline.dropna(inplace=True)

baseline.head()

def measure\_rmse(y\_true, y\_pred):

    return sqrt(mean\_squared\_error(y\_true,y\_pred))

rmse\_base = measure\_rmse(val.iloc[1:,0],baseline)

print(f'RMSE базового прогноза {round(rmse\_base,4)} градусов по цельсию')

def check\_stationarity(y, lags\_plots=48, figsize=(22,8)):

    # создаем графики

    y = pd.Series(y)

    fig = plt.figure()

    ax1 = plt.subplot2grid((3, 3), (0, 0), colspan=2)

    ax2 = plt.subplot2grid((3, 3), (1, 0))

    ax3 = plt.subplot2grid((3, 3), (1, 1))

    ax4 = plt.subplot2grid((3, 3), (2, 0), colspan=2)

    y.plot(ax=ax1, figsize=figsize)

    ax1.set\_title('Изменение температуры в Рио-де-Жанейро')

    plot\_acf(y, lags=lags\_plots, zero=False, ax=ax2);

    plot\_pacf(y, lags=lags\_plots, zero=False, ax=ax3);

    sns.distplot(y, bins=int(sqrt(len(y))), ax=ax4)

    ax4.set\_title('Диаграмма распределения')

    plt.tight\_layout()

    print('Результаты теста Дики-Фуллера:')

    adfinput = adfuller(y)

    adftest = pd.Series(adfinput[0:4], index=['Test Statistic','p-value','Lags Used','Number of Observations Used'])

    adftest = round(adftest,4)

    for key, value in adfinput[4].items():

        adftest["Critical Value (%s)"%key] = value.round(4)

    print(adftest)

    if adftest[0].round(2) < adftest[5].round(2):

        print('\nThe Test Statistics is lower than the Critical Value of 5%.\nThe serie seems to be stationary')

    else:

        print("\nThe Test Statistics is higher than the Critical Value of 5%.\nThe serie isn't stationary")

# Первый подход заключается в проверке ряда без каких-либо преобразований.

check\_stationarity(train['Temp'])

check\_stationarity(train['Temp'].diff(12).dropna())

def walk\_forward(training\_set, validation\_set, params):

    '''

    Params: it's a tuple where you put together the following SARIMA parameters: ((pdq), (PDQS), trend)

    '''

    history = [x for x in training\_set.values]

    prediction = list()

    # Использование параметров SARIMA и подгонка данных

    pdq, PDQS, trend = params

    # Прогнозирование на один период вперед в наборе проверки

    for week in range(len(validation\_set)):

        model = sm.tsa.statespace.SARIMAX(history, order=pdq, seasonal\_order=PDQS, trend=trend)

        result = model.fit(disp=False)

        yhat = result.predict(start=len(history), end=len(history))

        prediction.append(yhat[0])

        history.append(validation\_set[week])

    return prediction

 # Давайте проверим это в наборе проверки

val['Pred'] = walk\_forward(train['Temp'], val['Temp'], ((3,0,0),(0,1,1,12),'c'))

# Measuring the error of the prediction

rmse\_pred = measure\_rmse(val['Temp'], val['Pred'])

print(f"The RMSE of the SARIMA(3,0,0),(0,1,1,12),'c' model was {round(rmse\_pred,4)} celsius degrees")

print(f"It's a decrease of {round((rmse\_pred/rmse\_base-1)\*100,2)}% in the RMSE")

# Creating the error column

val['Error'] = val['Temp'] - val['Pred']

def plot\_error(data, figsize=(20,8)):

    '''

    There must have 3 columns following this order: Temperature, Prediction, Error

    '''

    plt.figure(figsize=figsize)

    ax1 = plt.subplot2grid((2,2), (0,0))

    ax2 = plt.subplot2grid((2,2), (0,1))

    ax3 = plt.subplot2grid((2,2), (1,0))

    ax4 = plt.subplot2grid((2,2), (1,1))

    # График текущих и прогнозируемых значений

    ax1.plot(data.iloc[:,0:2])

    ax1.legend(['Real','Pred'])

    ax1.set\_title('Current and Predicted Values')

    # Остаточные и прогнозируемые значения

    ax2.scatter(data.iloc[:,1], data.iloc[:,2])

    ax2.set\_xlabel('Predicted Values')

    ax2.set\_ylabel('Errors')

    ax2.set\_title('Errors versus Predicted Values')

    ## QQ График остатка

    sm.graphics.qqplot(data.iloc[:,2], line='r', ax=ax3)

    # График автокорреляции невязки

    plot\_acf(data.iloc[:,2], lags=(len(data.iloc[:,2])-1),zero=False, ax=ax4)

    plt.tight\_layout()

    plt.show()

# Нам нужно удалить некоторые столбцы, чтобы построить диаграммы

val.drop(['month','year'], axis=1, inplace=True)

val.head()

plot\_error(val)

# Создание нового объединения набора для обучения и проверки:

future = pd.concat([train['Temp'], val['Temp']])

future.head()

# Используя те же параметры подобранной модели

model = sm.tsa.statespace.SARIMAX(future, order=(3,0,0), seasonal\_order=(0,1,1,12), trend='c')

result = model.fit(disp=False)

test['Pred'] = result.predict(start=(len(future)), end=(len(future)+13))

test[['Temp', 'Pred']].plot(figsize=(22,6))

plt.title('Текущие значения по сравнению с экстраполированными')

plt.show()

test\_baseline = test['Temp'].shift()

test\_baseline[0] = test['Temp'][0]

rmse\_test\_base = measure\_rmse(test['Temp'],test\_baseline)

rmse\_test\_extrap = measure\_rmse(test['Temp'], test['Pred'])

print(f'The baseline RMSE for the test baseline was {round(rmse\_test\_base,2)} celsius degrees')

print(f'The baseline RMSE for the test extrapolation was {round(rmse\_test\_extrap,2)} celsius degrees')

print(f'That is an improvement of {-round((rmse\_test\_extrap/rmse\_test\_base-1)\*100,2)}%')

print(result.summary())

from sklearn.metrics import mean\_squared\_error

print('SARIMAX model MSE:{}'.format(mean\_squared\_error(test['Temp'],test['Pred'])))