A Project Report

On

WAITING TIME PREDICTION IN QUEUING SYSTEMS USING MACHINE LEARNING

BY

GROUP 1

Team Members:

1) 1)Shubhan Mital (2019B4A30900H) 2) Nahata Muskan (2017B4A41747H)

3) Pranjal Jasani (2019B4A70831H) 4) Akshat Deshmukh (2019B4AA0904H)

5) Harrshit Limbodia (2019B4A40815H) 6) Pranavi Gunda (2019B4A71068H)

7)Ritik Pachauri (2019B4AA0890H) 8) Akshat Singh (2019B4AA0842H)

9) Raghav Gupta (2019B4PA30927H)

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS OF

MATH F242: OPERATIONS RESEARCH

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

HYDERABAD CAMPUS

(APRIL 2021)

CONTENTS

1) Introduction 3
2) Machine Learning and Queuing Theory 3
3) Dataset 4
4) Data Preprocessing and Feature Engineering 4
5) Experimental Setup 5
6) Artificial Neural Network (ANN) & ITS WORKING 5-6
7) Code 7-17
8) Results and Conclusions 18
9) References 19

INTRODUCTION

Queueing problems exist when multiple people need access to a resource, and the service cannot match the level of demand. On the one hand, queueing is a necessary evil when accessing valuable resources like health-related ones, as the idle time of those resources is expensive . On the other hand, queues may get unnecessarily big even for this purpose and may lead to long idle times for the customers.

There is also a link between long waiting times and customer dissatisfaction, and thus, industries should strive for better resource allocation that will optimize waiting queues. Queueing optimization techniques are used in several industries to improve customer service.

MACHINE LEARNING AND QUEUING THEORY

Machine learning techniques and simulation models can also be of use to deal with queueing problems. An improvement to the prediction of the overall waiting time for daily radiation treatment appointments, when compared to the rough estimates.

We will try predicting solution for the scenario of clients waiting to be served in a bank. We will then propose our generic queueing system that can be used in various industries and can exploit queue specific parameters when predicting the estimated waiting time of the clients.

DATASET

Dataset is a collection of instances. It corresponds to the contents of a single database table, or a single statistical data matrix, where every column of the table represents a particular variable, and each row corresponds to a given member of the data set in question.

There are two types of dataset.Training Dataset is the dataset that we feed into our machine learning algorithm to train our model. Testing Dataset is the dataset that we use to validate the accuracy of our model but is not used to train the model. It may be called the validation dataset.

In our code, we split the dataset into training and testing datasets into 80% and 20% respectively. The most supported file type for a tabular dataset is "Comma Separated File," or CSV. In our code, we are analyzing a dataset with 3 banks for 5 days in a week. Unfortunately, the dataset does not specify the number of servers.

DATA PREPROCESSING AND FEATURE ENGINEERING

We used the following features as the input for modeling: The people waiting in the queue,the day of the week,the hour,the waiting time variable forms the output of the model.

For each client, we have calculated the number of people waiting in the queue at the time the client joined the queue. To do so, we calculated the queue departure time for each client by adding the waiting time to the arrival time, and then we counted the number of people that were yet to depart from the queue at the time a new client joined the queue.

We used mean encoding,also known as target encoding, to encode new features from those 3 existing categorical features and the target variable.The idea of mean encoding for a regression task is simple.

EXPERIMENTAL SETUP

Machine learning primarily involves two main parts- ‘developing’ or ‘building’ a model using certain machine learning algorithms and then ‘deploying’ that model to ,for instance, create predictive applications. Our experimental setup involves only the first part, i.e., building the model. We have used Python 3 to build the model. We have used Jupyter notebooks. It integrates code and its output into a single document that combines visualizations, narrative text, mathematical equations, and other rich media. We decided to use an artificial neural network (ANN) over other machine learning models.

ARTIFICIAL NEURAL NETWORK

ANN (Artificial Neural Network) is a multi-layer network of neurons that we use to classify things, make predictions, etc. that are inspired by, but not identical to biological neural networks that constitute animal brains. Like the human brain consisting of many brain cells, ANN also consists of a collection of neurons that are interconnected. The ANN model consists of three layers mainly input layer, output layer and hidden layer(s). Each layer is connected by neurons with interconnection from input to output layers. The number of neuron in each layer depends on the nature of the problem, for example number of neurons in the input layer is fixed by number on inputs required while the number of output layer depends on our requirement of what we would like to calculate.The best suited number of neurons in the hidden layer for the particular problem with given input and output is selected to predict the unknown data. ANN (Artificial Neural Network) models gain more attention owing to their application in various fields like mathematics, transportation, weather and market trend forecasting etc. We have decided to use a neural network over other machine learning models because of the continuous training capabilities of a neural network. When a new batch of training data is gathered, an existing neural network can be trained solely on those, and there is no need to train on all available training data regularly. This is ideal for a queue management system, as new data are consistently. Traditional Machine Learning algorithms tend to perform at the same level when the data size increases but ANN outperforms traditional Machine Learning algorithms when the data size is huge.

WORKING OF ARTIFICIAL NEURAL NETWORK

Weights play an important role in ANN. Every neuron has some weights. Neural Networks learn through the weights, by adjusting weights the neural networks decide whether certain features are important or not. One epoch is when the entire dataset is passed forward and backward through the neural network only once. In our code, we have used 500 epochs. Since one epoch is too big to feed to the computer at once we divide it in several smaller batches. The total number of training examples present in a single batch is called the batch size. In our code, we have taken the batch size as 256.

The working of ANN can be broken down into two phases: Forward Propagation and Back Propagation. Forward propagation involves multiplying weights, adding bias, and applying activation function to the inputs and propagating it forward. The purpose of an activation function is to introduce non-linearity to the data. Introducing non-linearity helps to identify the underlying patterns which are complex. It is also used to scale the value to a particular interval. Examples of activation functions are sigma function, tanh function, ReLU function, linear function etc. In our code, we have used ReLU function for input and hidden layers and linear (identity) function for output layer. A function that is used to calculate error is called loss Function .Error is a function of internal parameters of the model i.e. weights and biases. Different loss functions are used to deal with different types of tasks such as regression and classification. There are several regression loss functions - MSE(Mean Squared Error, MAE(Mean Absolute Error), Huber's loss etc. In our code, we use MAE as a loss function. It is measured as the average of the sum of absolute differences between predictions and actual observations. For accurate prediction, one needs to minimize the error calculated using loss function. This is done during back propagation using optimization function or optimizer.The optimizers or optimization functions used in ANN varies, such as Adam (Adaptive Moment Estimation), RMSProp, Gradient Descent, etc. In our code, we have used the Adam function as an optimizer .

CODE

The coding for the project was done in python language on jupyter notebooks. It integrates code,the input and its output into a single document which we then converted to PDF format.

OR_Project

April 2, 2021

Importing Libraries

[1]:
```
import os
import csv
import pandas as pd
import datetime as dt
import numpy as np
```

1 Data Preprocessing

Loading and Reading the Dataset

[2]:
```
def num_of_rows(file_path):
    file = open(file_path)
    reader = csv.reader(file)
    i= len(list(reader))
    return i
```

[3]:
```
def read_file(file_path, numberOfRows):
    indexCounter = 0
    indexCounter = 0
    with open(file_path,'r') as file:
        nRows = numberOfRows
        nColumns = 4
        dataset = np.zeros(shape=(nRows, nColumns))
        arrivalTimes = []
        for line in file:
            try:
                dataInstance = line.split(',')
                dataInstance[1] = dataInstance[1].replace('"','')
                dataInstance[2] = dataInstance[2].replace('"','')
                dataInstance[3] = dataInstance[3].replace('"','')
                arrivalTime = dataInstance[1] #splits the line at the comma and␣
↪takes the first bit
                arrivalTime = arrivalTime[:arrivalTime.index(':')+3]
                arrivalTime = dt.datetime.strptime(arrivalTime, '%H:%M')
                arrivalHour = arrivalTime.hour
                arrivalMinute = arrivalTime.minute
                waitingMinutes = dataInstance[2]
```

```
                serviceMinutes = dataInstance[3]

                arrivalTimes.append(arrivalTime)
                dataset[indexCounter] = [arrivalHour, arrivalMinute,␣
↪waitingMinutes, serviceMinutes]
                indexCounter = indexCounter + 1
            except:
                #print('index' + str(indexCounter) + 'error')
                pass
    return dataset, arrivalTimes
```

[4]:

```
filenames = []
rootFilePath = './BankDataCsv/'
fullDataset = pd.DataFrame()

for bankCounter in range(3):
    for dayCounter in range(5):
            filename = 'Bank' + str(bankCounter + 1) + 'Day' + str(dayCounter +␣
↪1)
            fullPath = rootFilePath + filename + '.csv'
            filenames.append(fullPath)

            numberOfRows = num_of_rows(fullPath) - 1
            print('Reading ' + str(filename) + ' that contains ' +␣
↪str(numberOfRows) + ' entries')
            tempFeatures, tempArrivalTimes = read_file(rootFilePath + filename␣
↪+ '.csv', numberOfRows)
            dfTempFeatures = pd.DataFrame(np.array(tempFeatures),␣
↪columns=['hour', 'minutes', 'waitingTime', 'serviceTime'])
            dfTempArrivalTimes = pd.DataFrame(np.array(tempArrivalTimes),␣
↪columns=['arrivalTime'])

            timeLeavingTheQueue = []
            for arrivalTimeCounter in range(numberOfRows):
                timeLeavingTheQueue.append(dfTempArrivalTimes.
↪at[arrivalTimeCounter, 'arrivalTime'] + pd.Timedelta(minutes =␣
↪dfTempFeatures.at[arrivalTimeCounter, 'waitingTime']))
            dftimeLeavingTheQueue = pd.DataFrame(np.array(timeLeavingTheQueue),␣
↪columns=['timeLeavingTheQueue'])

            waitingPeople = np.zeros(numberOfRows)
            for i in range(numberOfRows):
                for j in range(i):
                    if (dfTempArrivalTimes.at[i, 'arrivalTime'] <␣
↪dftimeLeavingTheQueue.at[j, 'timeLeavingTheQueue']):
                        waitingPeople[i] += 1
```

```
            dfWaitingPeople = pd.DataFrame(np.array(waitingPeople),␣
↪columns=['waitingPeople'])

            dfWaitingPeople['waitingPeople'] = dfWaitingPeople['waitingPeople'].
↪astype(int)
            dfTempFeatures['hour'] = dfTempFeatures['hour'].astype(int)
            dfTempFeatures['minutes'] = dfTempFeatures['minutes'].astype(int)

            tempDataset = pd.concat([dfTempFeatures, dfWaitingPeople], axis=1)

            fullDataset = pd.concat([fullDataset, tempDataset], axis=0)

fullDataset = fullDataset.reset_index(drop = True)
```

```
Reading Bank1Day1 that contains 857 entries
Reading Bank1Day2 that contains 981 entries
Reading Bank1Day3 that contains 1057 entries
Reading Bank1Day4 that contains 899 entries
Reading Bank1Day5 that contains 996 entries
Reading Bank2Day1 that contains 1034 entries
Reading Bank2Day2 that contains 1009 entries
Reading Bank2Day3 that contains 891 entries
Reading Bank2Day4 that contains 948 entries
Reading Bank2Day5 that contains 890 entries
Reading Bank3Day1 that contains 988 entries
Reading Bank3Day2 that contains 784 entries
Reading Bank3Day3 that contains 648 entries
Reading Bank3Day4 that contains 891 entries
Reading Bank3Day5 that contains 752 entries
```

Printing the dataset

[5]:
```
fullDataset
```

[5]:

	hour	minutes	waitingTime	serviceTime	waitingPeople
0	8	0	8.538438	10.111179	0
1	8	0	6.101840	10.831172	1
2	8	0	6.725150	7.261361	2
3	8	0	7.388745	8.409101	3
4	8	0	8.374004	9.022523	4
…	…	…	…	…	…
13620	14	57	13.000000	14.000000	23
13621	14	58	9.000000	12.000000	22
13622	14	59	8.000000	14.000000	22
13623	14	59	11.000000	16.000000	23
13624	14	59	9.000000	16.000000	24

[13625 rows x 5 columns]

2 Exploratory Data Analysis

Importing Libraries

[6]:
```
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
```

Description of the Data

[7]:
```
fullDataset.describe()
```

[7]:

	hour	minutes	waitingTime	serviceTime	waitingPeople
count	13625.000000	13625.000000	13625.000000	13625.000000	13625.000000
mean	8.497028	28.272881	11.081423	12.793214	154.245872
std	3.690302	17.348661	3.612460	5.711441	272.493315
min	1.000000	0.000000	3.005860	4.006226	0.000000
25%	8.000000	13.000000	8.254524	8.266549	18.000000
50%	9.000000	27.000000	11.000000	11.000000	27.000000
75%	11.000000	43.000000	13.391827	16.788842	48.000000
max	14.000000	59.000000	19.989746	28.998169	824.000000

Information of the Dataset

[8]:
```
fullDataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13625 entries, 0 to 13624
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   hour           13625 non-null  int32
 1   minutes        13625 non-null  int32
 2   waitingTime    13625 non-null  float64
 3   serviceTime    13625 non-null  float64
 4   waitingPeople  13625 non-null  int32
dtypes: float64(2), int32(3)
memory usage: 372.7 KB
```

Checking for null values

[9]:
```
fullDataset.isnull().sum()
```

[9]:
```
hour             0
minutes          0
waitingTime      0
serviceTime      0
waitingPeople    0
dtype: int64
```

3 Feature Engineering

[10]:
```
workingCopyDataset = fullDataset
workingCopyDataset.drop(['serviceTime'], axis=1);

# mean encoding for regression output
def mean_encoder_regression(input_vector, output_vector):
    assert len(input_vector) == len(output_vector)
    numberOfRows = len(input_vector)

    temp = pd.concat([input_vector, output_vector], axis=1)
    # Compute target mean
    averages = temp.groupby(by=input_vector.name)[output_vector.name].
 ↪agg(["mean", "count"])

    print(averages)
    return_vector = pd.DataFrame(0, index=np.arange(numberOfRows),␣
 ↪columns={'feature'})


    for i in range(numberOfRows):
        return_vector.iloc[i] = averages['mean'][input_vector.iloc[i]]

    return return_vector

encoded_input_vector_hour = mean_encoder_regression(workingCopyDataset['hour'],␣
 ↪workingCopyDataset['waitingTime'])
encoded_input_vector_hour.columns = ['hour']
encoded_input_vector_minutes =␣
 ↪mean_encoder_regression(workingCopyDataset['minutes'],␣
 ↪workingCopyDataset['waitingTime'])
encoded_input_vector_minutes.columns = ['minutes']
```

```
           mean  count
hour
1     10.375956   1163
2     10.385665   1319
8     10.990211   3132
9     11.120934   2153
10    11.036177   1493
11    11.151889   1548
12    11.069769   1700
13    12.716157    458
14    12.854325    659
              mean  count
minutes
0        11.228229    267
```

1	11.240516	291
2	10.739989	197
3	11.156258	260
4	11.061478	211
5	11.305955	320
6	10.954633	168
7	11.148384	287
8	10.899530	247
9	11.168025	243
10	11.068515	194
11	10.843569	325
12	11.017874	166
13	11.223202	251
14	10.989250	195
15	11.159026	289
16	10.913197	213
17	11.267012	345
18	11.211631	146
19	11.146790	365
20	10.958319	281
21	11.220374	231
22	10.606201	136
23	11.019538	358
24	11.057395	171
25	11.293645	296
26	10.665262	181
27	10.897144	290
28	11.072144	162
29	11.442482	275
30	11.409386	145
31	10.959884	235
32	11.005181	216
33	11.038094	212
34	10.927980	155
35	11.195711	324
36	11.102606	172
37	10.945550	264
38	10.855043	179
39	11.198368	220
40	10.953246	181
41	11.470296	218
42	11.300919	128
43	11.179408	278
44	11.017274	164
45	11.375910	247
46	10.900147	114
47	11.006391	298
48	11.125049	165

```
49        11.080418    242
50        10.972189    223
51        10.935155    242
52        10.729395    168
53        11.056941    254
54        11.115319    120
55        11.003035    273
56        10.830563    190
57        11.325616    197
58        11.032547    166
59        11.114004    274
```

[11]:
```python
X = pd.concat([encoded_input_vector_hour['hour'],␣
↪encoded_input_vector_minutes['minutes'], pd.
↪DataFrame(workingCopyDataset['waitingPeople'])], axis=1)
y = workingCopyDataset['waitingTime']
X.describe()
```

[11]:

	hour	minutes	waitingPeople
count	13625.000000	13625.000000	13625.000000
mean	11.081423	11.081423	154.245872
std	0.578091	0.175970	272.493315
min	10.375956	10.606201	0.000000
25%	10.990211	10.958319	18.000000
50%	11.036177	11.068515	27.000000
75%	11.120934	11.198368	48.000000
max	12.854325	11.470296	824.000000

4 Splitting the Dataset

Importing the libraries

[12]:
```python
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from tensorflow.python import keras
```

[13]:
```python
trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
print(trainX.shape, trainy.shape)
print(testX.shape, testy.shape)
```

```
(10900, 3) (10900,)
(2725, 3) (2725,)
```

[14]:
```python
def scale_input(X, means, stds):
    return (X - means) / stds
def descale_input(X, means, stds):
    return (X * stds) + means
```

```
meansX = trainX.mean(axis=0)
stdsX = trainX.std(axis=0) + 1e-10

trainX_scaled = scale_input(trainX, meansX, stdsX)
testX_scaled = scale_input(testX, meansX, stdsX)
meansX = trainX.mean(axis=0)
stdsX = trainX.std(axis=0) + 1e-10
```

5 Neural Network

[15]:
```
#Create the model
inputVariables = 3
model = keras.models.Sequential()
model.add(keras.layers.Dense(12, input_dim=inputVariables,␣
 ↪kernel_initializer='normal', activation='relu'))
model.add(keras.layers.Dense(8, activation='relu'))
model.add(keras.layers.Dense(1, activation='linear'))
model.summary()

model.compile(loss='mae', optimizer='adam')
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 12)                48
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 104
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 9
=================================================================
Total params: 161
Trainable params: 161
Non-trainable params: 0
_________________________________________________________________
```

[16]:
```
#Train the model
numberOfEpochs = 500
batchSize = 256
history = model.fit(trainX_scaled, trainy, epochs=numberOfEpochs,␣
 ↪batch_size=batchSize, verbose=1, validation_split=0.2)
```

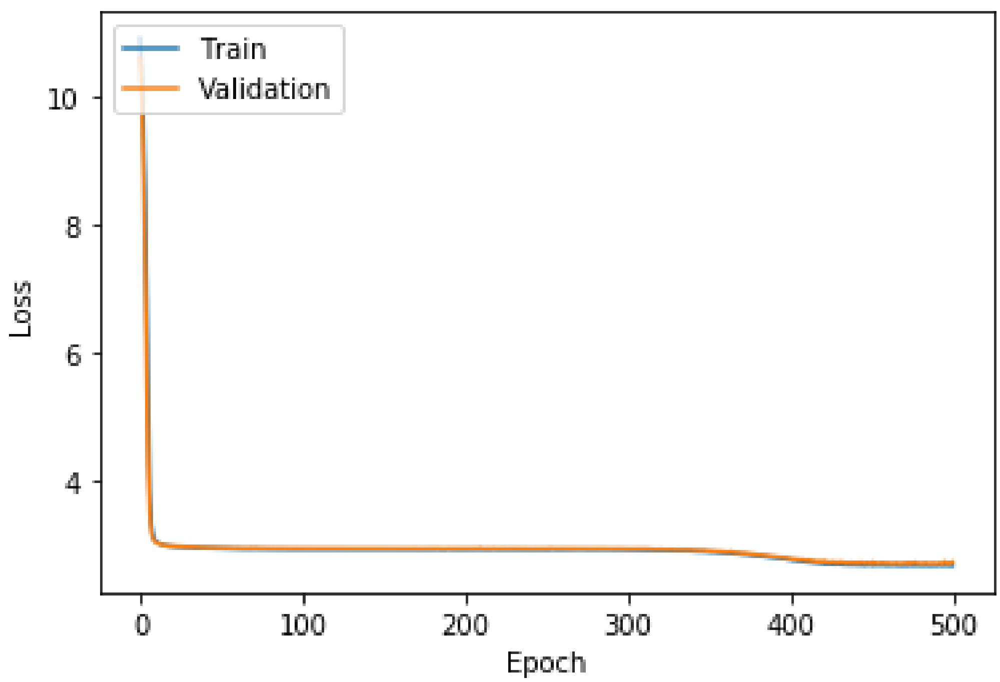
```
Epoch 1/500
35/35 [==============================] - 1s 16ms/step - loss: 10.8939 -
val_loss: 10.5925
Epoch 2/500
35/35 [==============================] - 0s 6ms/step - loss: 10.4473 - val_loss:
10.0005
```

```
Epoch 499/500
35/35 [==============================] - 0s 4ms/step - loss: 2.6640 - val_loss:
2.7132
Epoch 500/500
35/35 [==============================] - 0s 4ms/step - loss: 2.6620 - val_loss:
2.7188
```

[17]:
```
# list all data in history
print(history.history.keys())
```

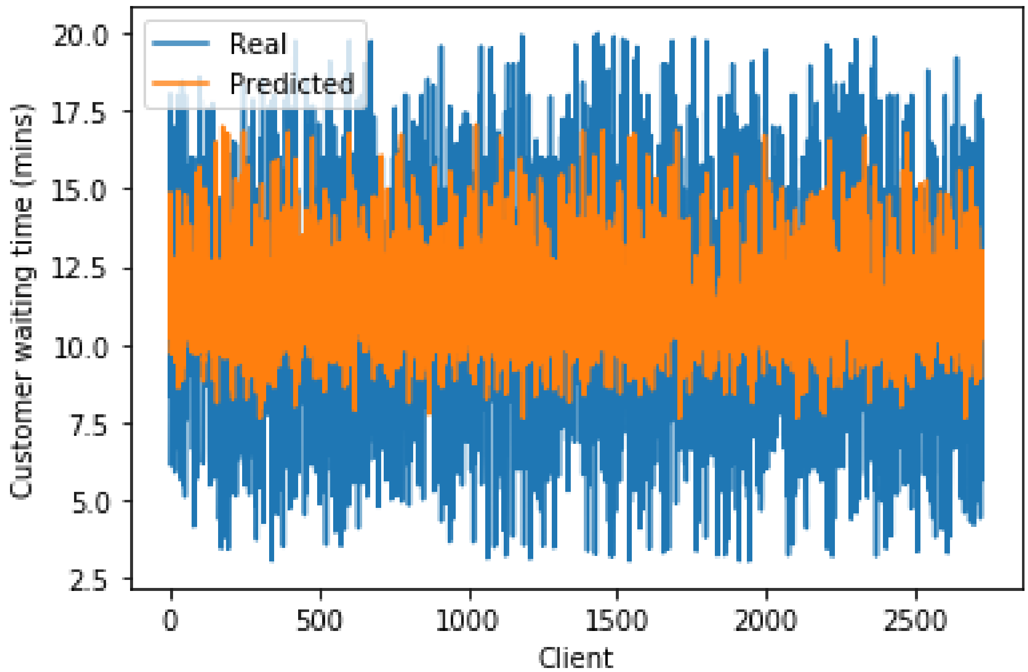
```
dict_keys(['loss', 'val_loss'])
```

[18]:
```
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
#plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
#plt.show()
plt.savefig('./loss.pdf')
```



[19]:
```
testy_pred = model.predict(testX_scaled)
myLength = len(testy_pred)
plt.plot(range(myLength), testy)
```

```
plt.plot(range(myLength), testy_pred)
plt.ylabel('Customer waiting time (mins)')
plt.xlabel('Client')
plt.legend(['Real', 'Predicted'], loc='upper left')
plt.savefig('./realVsPredictedWaitingTimes.pdf')
```



[20]:
```
myMae = mean_absolute_error(testy, testy_pred)
print(f'The mean absolute error I get with the neural network is {myMae}␣
 ↪minutes.')
```

```
The mean absolute error I get with the neural network is 2.6828083057243663
minutes.
```

[21]:
```
myLength = len(testy_pred)
myFMean = np.mean(trainy)
myFMedian = np.median(trainy)
testyMean = testy_pred.copy()
testyMedian = testy_pred.copy()
for i in range(myLength):
    testyMean[i] = myFMean
    testyMedian[i] = myFMedian
```

[22]:
```
myMaeNaiveMean = mean_absolute_error(testy, testyMean)
print(f'The mean absolute error I get with the naive mean model is␣
 ↪{myMaeNaiveMean} minutes.')
```

```
The mean absolute error I get with the naive mean model is 2.943643774768819
minutes.
```

[23]:
```
myMaeNaiveMedian = mean_absolute_error(testy, testyMedian)
print(f'The mean absolute error I get with the naive median model is␣
 ↪{myMaeNaiveMedian} minutes.')
```

```
The mean absolute error I get with the naive median model is 2.9410409835583633
minutes.
```

RESULTS AND CONCLUSIONS

After 500 epochs of training, a mean absolute error of 2.68 minutes was achieved on the test set. To have an estimation of the predictive capability of the trained model, we are comparing against the naive mean and the naive median model; these are the models that always predict the mean and the median waiting time of the training set. The naive mean model had a mean absolute error of 2.94 minutes on the test set, and the naive median an error of 2.94 minutes. Our model has, therefore, achieved an improvement of 8.8% over the naive mean model and 8.8% over the naive median one.

We have explored how machine learning will be used for predicting the waiting time of individuals queueing in lines. We started by employing a publically obtainable dataset of queues in banks, and by coaching a neural network, we achieved a mean absolute error of 2.68 minutes, improving over the performances of naive models. Sadly, we could not directly compare against queueing theory as a result of the dataset lacking information concerning the deployed servers. After presenting a particular case on how machine learning will be used to predict waiting times in queueing situations, we are generalizing in a lot of industries. Using a simulator on a web application, we are able to verify the prognostic capabilities of the queue specific neural networks. As future work, we'll embody the ability to feature queue specific parameters at the queue creation phase with predefined responses. This extra info will be exploited by the underlying waiting time-predicting neural network of every queue, and totally different distributions of each parameter response may be simulated exploitation of the machine.

REFERENCES

The entire project structure is based on these resources:

- ❑ Kyritsis A.I. and M. Deriaz, M. (2019). "A Machine Learning Approach to Waiting Time Prediction in Queueing Scenarios" , Second International Conference on Artificial Intelligence for Industries (AI4I), 17-21, doi:10.1109/AI4I46381.2019.00013.
- ❑ Data has been collected from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5997939

The theory for ANN has been based on:

- ❑ Senniappan, P. (2015). ANN Simulation of single server infinite capacity queuing system.
- ❑ https://towardsdatascience.com/an-illustrated-guide-to-artificial-neural-networks-f149a549ba74
- ❑ https://towardsdatascience.com/a-beginners-guide-to-neural-networks-d5cf7e369a13
- ❑ https://towardsdatascience.com/understanding-neural-networks-19020b758230
- ❑ https://towardsdatascience.com/introduction-to-artificial-neural-networks-ac338f4154e5
- ❑ https://www.analyticsvidhya.com/blog/2020/11/popular-classification-models-for-machine-learning
- ❑ https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression-23e0ef3e14d3
- ❑ https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23
- ❑ https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9
- ❑ https://medium.com/@vivek.atwal01/loss-function-back-propagation-optimization-function-f8e3cbd85923