Assignment #3 - Neural Networks

DUE: Oct 29 (Tuesday) 11:00 pm

Jeet Jivrajani

I. Overview

The assignment is about to design a **Neural Network** by selecting its best parameters by performing **k-fold cross validation.** Cross-validation(CV) is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data[1]. This will help us to design the best network based on the dataset. The dataset used is **Online Shoppers Purchasing Intention Dataset Data Set** for non-linear logistic regression and **Beijing PM2.5 Data Data Set** for non-linear regression. By designing a neural network of best parameters the task is to implement **Non-linear regression and Non-linear classification methods.** At the end, task is to discuss about the cross validation results. Based on the CV results it will discuss about the parameter and network structure.

II. Data

Data-description
This hourly data set contains the PM2.5 data of US Embassy in Beijing. PM means **Particulate Matter** that have diameter about 2.5 micrometers. PM2.5 is the air quality data which needs to be moitored. The data is obtained from the environmental companies. The dataset has multivariate and time-series characteristics. it contains 43824 instances of data with 13 attributes. It also contains missing values. The data was collected in between january 1, 2010 to December 31, 2014[2].

No: row number
year: year of data in this row
month: month of data in this row
day: day of data in this row
hour: hour of data in this row
pm2.5: PM2.5 concentration (ug/m^3)
DEWP: Dew Point (â„ƒ)
TEMP: Temperature (â„ƒ)
PRES: Pressure (hPa)
cbwd: Combined wind direction
Iws: Cumulated wind speed (m/s)
Is: Cumulated hours of snow
Ir: Cumulated hours of rain

Source

1. Song Xi Chen, csx '@' gsm.pku.edu.cn, Guanghua School of Management, Center for Statistical Science, Peking University.

In [130]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn.linear_model import Ridge
from nn import NeuralNet
from util import Standardizer
from sklearn.metrics import mean_squared_error
from sklearn import model_selection
from sklearn.metrics import r2_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import roc_curve, auc
from grad import scg, steepest
from copy import copy
```

In [65]:
```
df_regression= pd.read_csv('PRSA_data_2010.1.1-2014.12.31.csv')
df_regression.head()
```

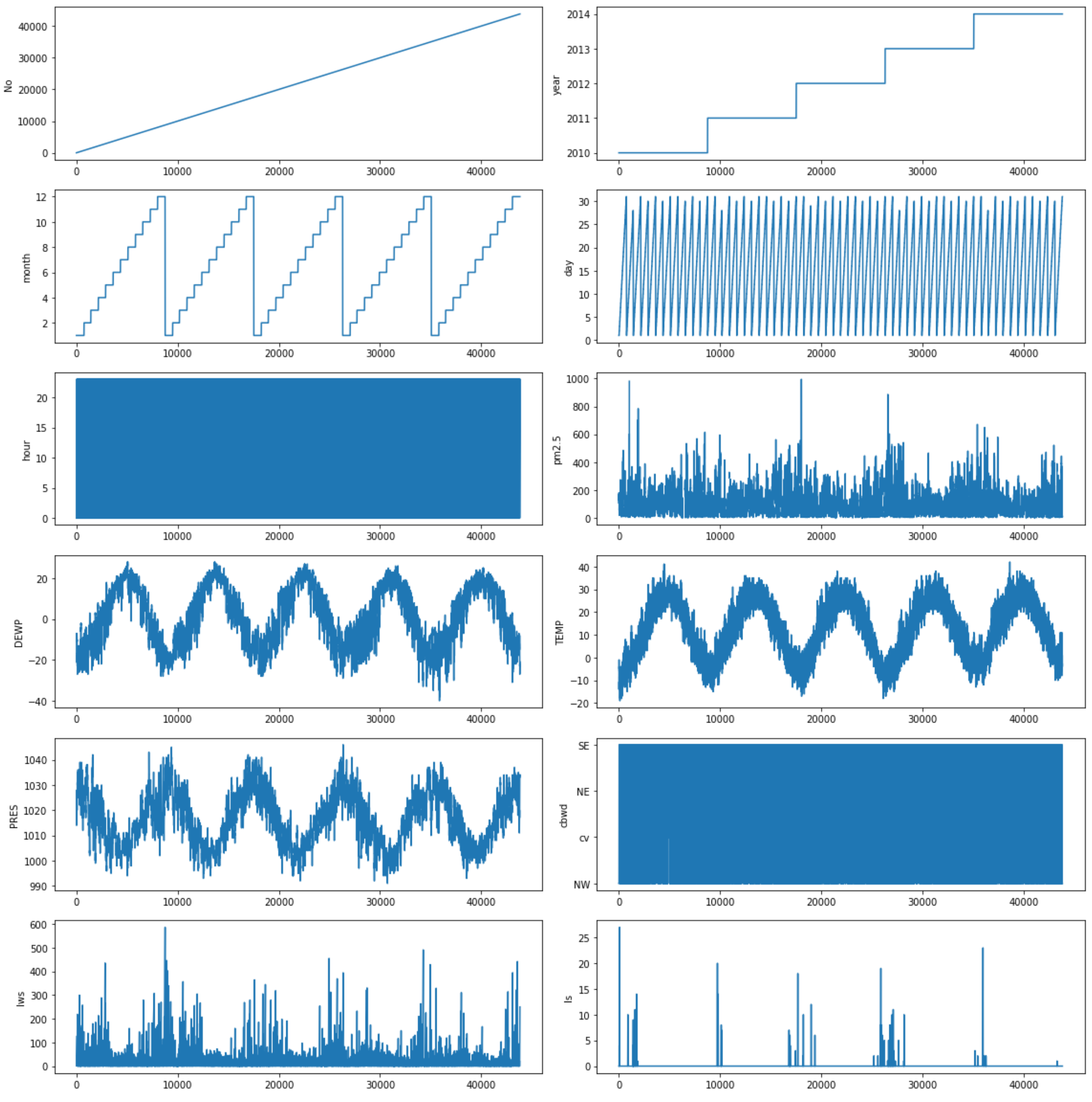
Out[65]:

	No	year	month	day	hour	pm2.5	DEWP	TEMP	PRES	cbwd	Iws	Is	Ir
0	1	2010	1	1	0	NaN	-21	-11.0	1021.0	NW	1.79	0	0
1	2	2010	1	1	1	NaN	-21	-12.0	1020.0	NW	4.92	0	0
2	3	2010	1	1	2	NaN	-21	-11.0	1019.0	NW	6.71	0	0
3	4	2010	1	1	3	NaN	-21	-14.0	1019.0	NW	9.84	0	0
4	5	2010	1	1	4	NaN	-20	-12.0	1018.0	NW	12.97	0	0

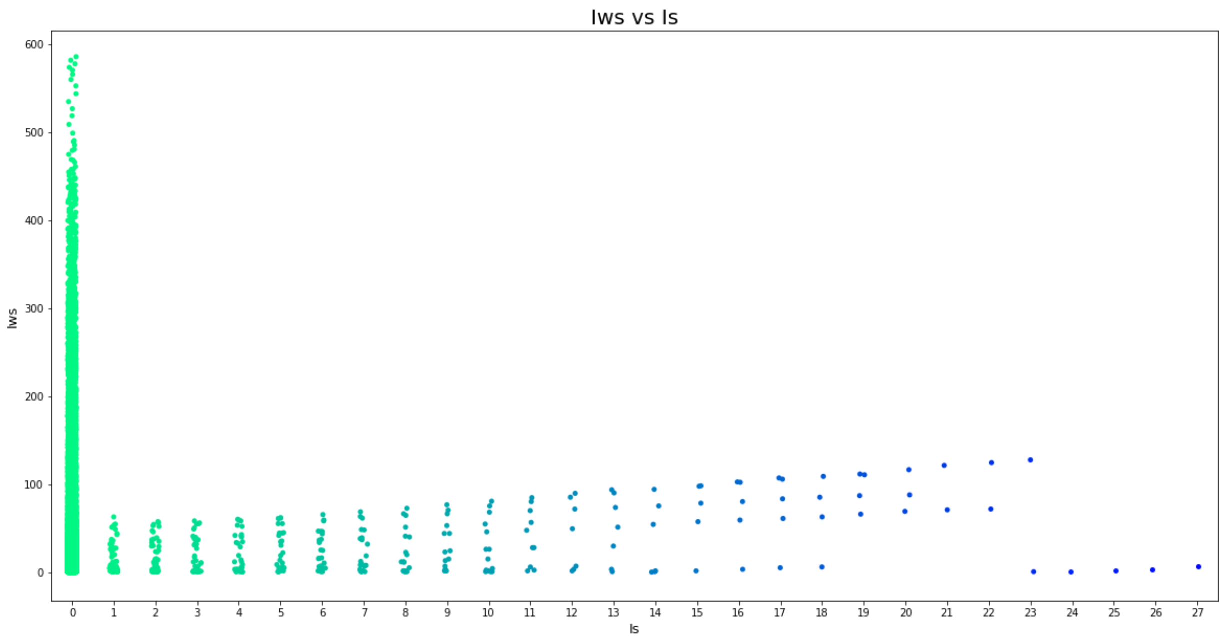
Plots to Visualize the data

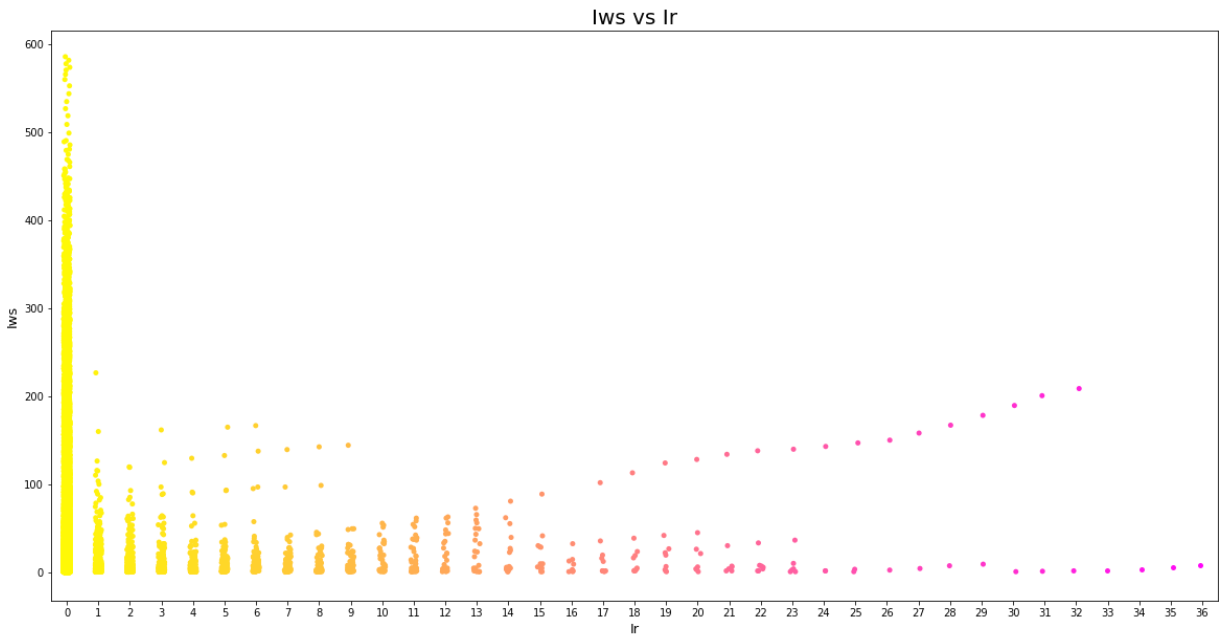
There are different kind of visualization techniques for visualizing the data like scatter plot, heatmap, bar plot, histogram plot and many more.
Initially, I have plotted the graph of all the features with the number of instances. Then generated the graphs of the all the features with all the other features by using the pairplot to check which features are more corelated and which features are uncorelated. Further generated the heatmap to quantify the corelation of the pairplots.

In [66]:
```
X = df_regression.iloc[:, :]
fig = plt.figure(figsize=(16,16))
plt.title("Plots of all features vs samples")
plt.clf()
for i in range(12):
    plt.subplot(6, 2, i+1)
    plt.plot(X.iloc[:, i])
    plt.ylabel(X.columns.values[i])
fig.tight_layout()
```



In [67]:
```
sns.stripplot(df_regression['Is'], df_regression['Iws'], palette = 'winter_r')
plt.title('Iws vs Is', fontsize = 20)
plt.xlabel('Is', fontsize = 13)
plt.ylabel('Iws', fontsize = 13)

plt.show()
sns.stripplot(df_regression['Ir'], df_regression['Iws'], palette = 'spring_r')
plt.title('Iws vs Ir', fontsize = 20)
plt.xlabel('Ir', fontsize = 13)
plt.ylabel('Iws', fontsize = 13)

plt.show()
```





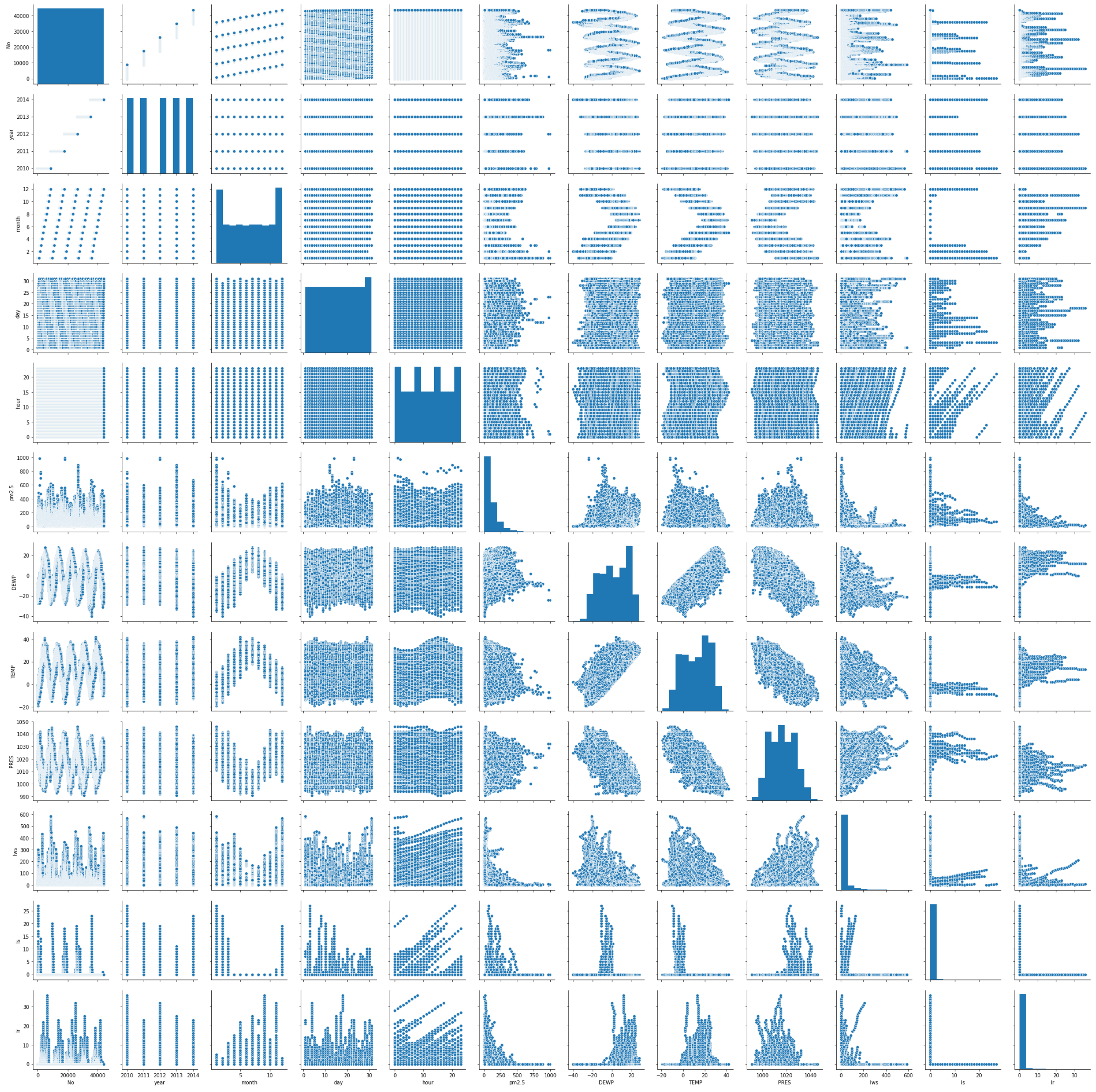
Pairplot creates a grid of axes in which every variable in data in y-axis

is mapped with the every other variable in x axis. The diagonal plots are generated in which both x and y axis share the same variable[4].

In [68]:
```
sns.pairplot(df_regression, palette='coolwarm')
```

```
C:\Users\jeetj\Miniconda3\envs\senses\lib\site-packages\numpy\lib\histograms.p
y:829: RuntimeWarning: invalid value encountered in greater_equal
  keep = (tmp_a >= first_edge)
C:\Users\jeetj\Miniconda3\envs\senses\lib\site-packages\numpy\lib\histograms.p
y:830: RuntimeWarning: invalid value encountered in less_equal
  keep &= (tmp_a <= last_edge)
```

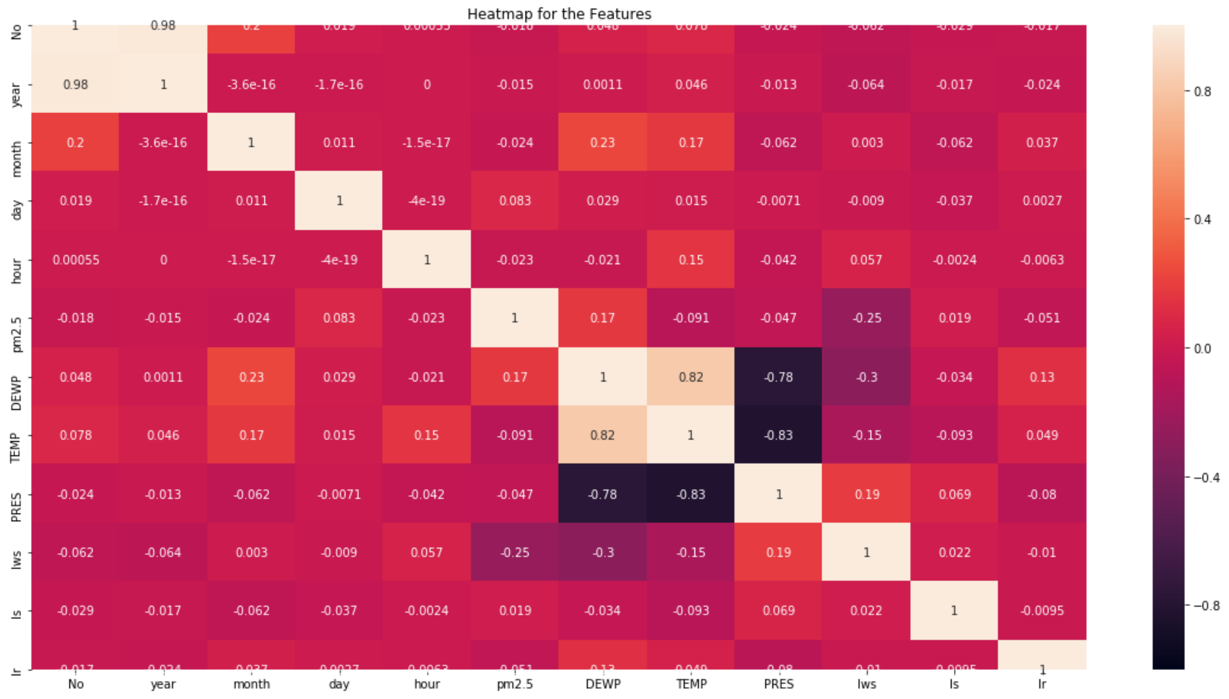
Out[68]:
```
<seaborn.axisgrid.PairGrid at 0x22ef63cd7f0>
```



Heatmap helps data which depends on two independent variables as a color coded image plot[3].

In [70]:
```python
plt.rcParams['figure.figsize'] = (20, 10)
sns.heatmap(df_regression.corr(),vmin=-1,vmax=1,annot=True)
plt.title("Heatmap for the Features")
```

Out[70]:
```
Text(0.5, 1, 'Heatmap for the Features')
```



Analyzing and observation of the plots

1. Dew point and Temperature are highly correlated data.
2. Dew Point and Pressure are less correlated data. So we can say that when both the attributes are high then the chances of occuring the event is less.
3. In beginning due to snow fall there is more amount of wind blowing where as in the end when there is no snow cumulative wind speed also decreases.
4. In beginning due to heavy rain there is more amount of wind blowing where as there is a little different trend in the end. As seen in above point, there is no snow so wind is less but with respect to rain there some wind blown though there is no rain.
5. Here from the first graph of subplots we can find that the feature such as **"No, year, month, day, hour and cbwd"** has no influence on the prediction of the **pm2.5** values so we can erradicate that features in the preprocessing part.

Data - Preprocessing of the data

1. Count the data and display the missing values

Since the data contains the missing values it needs to be handled just by replacing the missing values with Mean or value by applying some mathematical operations or just remove the entire data of the row.

2. Drop NaN values and obtain the remaining values. Initially data consists of 43824 values, after NaN values the size of data is 41757 instances and 13 features

In [71]:
```
df_regression.isnull().sum()
df_regression=df_regression.dropna()
df_regression.describe()
```

Out[71]:

	No	year	month	day	hour	pm2.5	
count	41757.000000	41757.000000	41757.000000	41757.000000	41757.000000	41757.000000	41757
mean	22279.380104	2012.042771	6.513758	15.685514	11.502311	98.613215	1
std	12658.168415	1.415311	3.454199	8.785539	6.924848	92.050387	14
min	25.000000	2010.000000	1.000000	1.000000	0.000000	0.000000	-40
25%	11464.000000	2011.000000	4.000000	8.000000	5.000000	29.000000	-10
50%	22435.000000	2012.000000	7.000000	16.000000	12.000000	72.000000	2
75%	33262.000000	2013.000000	10.000000	23.000000	18.000000	137.000000	15
max	43824.000000	2014.000000	12.000000	31.000000	23.000000	994.000000	28

Here the feature **"No"** is just the row number, **"year, month, day, and hour"** are time information that are not used for analysis. **"cbwd"** is just the feature of the wind direction which is not involved in any kind of predictions. By removing this fetures makes the visualizing task easy to find the trend and relation between the other attributes

In [72]:
```
df_regression = df_regression.drop('No',axis=1)
df_regression = df_regression.drop('year',axis=1)
df_regression = df_regression.drop('month',axis=1)
df_regression = df_regression.drop('day',axis=1)
df_regression = df_regression.drop('hour',axis=1)
df_regression = df_regression.drop('cbwd',axis=1)
df_regression1 = df_regression.iloc[:-2,1:]
df_regression = df_regression1
df_target = df_regression1.iloc[:,0:1]
# df_target = df_target.iloc[:-2,0]
```

In [73]:
```
print(df_regression.shape,df_target.shape)
```

```
(41755, 6) (41755, 1)
```

In [74]:
```
from nn import NeuralNet
from util import Standardizer
from grad import scg
```

In [75]:
```
df_regression1
```

Out[75]:

	DEWP	TEMP	PRES	Iws	Is	Ir
24	-16	-4.0	1020.0	1.79	0	0
25	-15	-4.0	1020.0	2.68	0	0
26	-11	-5.0	1021.0	3.57	0	0
27	-7	-5.0	1022.0	5.36	1	0
28	-7	-5.0	1022.0	6.25	2	0
...	...	...	...	...	...	...
43817	-22	-1.0	1033.0	221.24	0	0
43818	-22	-2.0	1033.0	226.16	0	0
43819	-23	-2.0	1034.0	231.97	0	0
43820	-22	-3.0	1034.0	237.78	0	0
43821	-22	-3.0	1034.0	242.70	0	0

41755 rows × 6 columns

In [76]:
```
df_target
```

Out[76]:

	DEWP
24	-16
25	-15
26	-11
27	-7
28	-7
...	...
43817	-22
43818	-22
43819	-23
43820	-22
43821	-22

41755 rows × 1 columns

III. Methods

III.A 5-fold Cross Validation

Summary of Cross validation and Correctness of Implementation

- In K Fold cross validation, the data is divided into k subsets. Such that each time, one of the k subsets is used as the test set/ validation set and the other k-1 subsets are put together to form a training set. The error estimation is averaged over all k trials to get total effectiveness of our model. As can be seen, every data point gets to be in a validation set exactly once, and gets to be in a training set k-1 times. This significantly reduces bias as we are using most of the data for fitting, and also significantly reduces variance as most of the data is also being used in validation set. Interchanging the training and test sets also adds to the effectiveness of this method. As a general rule and empirical evidence, K = 5 or 10 is generally preferred[1][5].

In [83]:
```
#Creating the list of list of ([[],[],[]]) our own data[5]
def k_fold(data):
    k_val = len(data) / 5.0
    partioned_data = []
    count = 0.0
    while count < len(data):
        partioned_data.append(data[int(count):int(count + k_val)])
        count = count + k_val
    return partioned_data
```

In [84]:
```
best_paramlist=[]
accuracies=[]
def cross_validate(X, T, parameters):
    feature_partition = k_fold(X)
    target_partition = k_fold(T)

    for i in range(5):
        print("For partition: ",i)
        X_test = feature_partition[i]
        T_test = target_partition[i]

        history_rmse=[]
        params=[]

        for k in range(5):
            print("k fold: ",k)
            if i == k:
                continue

            X_val = feature_partition[k]
            T_val = target_partition[k]

            X_train = feature_partition[not i and not k]
            T_train = target_partition[not i and not k]

            paramlist=[]
            rmselist=[]
            for param in parameters:
                model = NeuralNet(param)
                model.train(X_train, T_train)
                pred = model.use(X_val)

                valid_err = np.sqrt(mean_squared_error(T_val, pred))
                rmselist.append(valid_err)
                paramlist.append(param)

            print(paramlist)
            print(rmselist)
            print(paramlist[np.argmin(rmselist)])
            history_rmse.append(min(rmselist))
            params.append(paramlist[np.argmin(rmselist)])

        print(paramlist[np.argmin(history_rmse)])
        best_param=paramlist[np.argmin(history_rmse)]
        best_model = NeuralNet(best_param)

        X_train = feature_partition[not i]
        T_train = target_partition[not i]
        best_model.train(X_train, T_train)

        final_pred = best_model.use(X_test)
        final_err = np.sqrt(mean_squared_error(T_test, final_pred))
        accuracies.append(final_err)
        best_paramlist.append(best_param)

    return accuracies,best_paramlist
```

Correctness of Implementation

K-fold CV Procedure[6]

Choose K for K-fold cross validation.

Set nfold = 0. Initialize the lists res = [], testErrs = []. Split data X and label T into K number of partitions. For each TEST partition

- For each validation partition among the rest partitions
- Choose the rest partitions into a training set.
    - For each set P of parameters to test
    - Using P, train the model
    - Use the trained model on the validation data
    - Evaluate the validation result and store the restuls to res
- From the res result, pick the one with the best result.
- Retrain the model with the best parameter p in P.
- Apply the model with the best parameter p* to the test dataset.
- Evaluate the result err and store the tuple (p*, err) to testErrs.

Return testErrs.

Steps followed as per above method

1. Initially to create the K fold cross validation function, create the lists of list of your data.
2. Divide the data in 5 parts equally such that in every iteration 4 parts are trained and 1 part is test data. Append all 5 parts of the data as a list in one global list.
3. Here the partition of the data is performed and stored it in variable **feature_partition and target_partition.**
4. As we are implementing it for 5 fold cross validation we need to iterate the whole data 5 times such that in iteration 1 all the combination of the test and train of the data is performed. Based on the first iteration the **rmse error and parameters** are added to the list based on this value the neural network model is trained. Once the training of the model is done it check for the best parameter from the lists of the parameter. The parameter which has best value and the minimum error is appended to other list.
5. Step number 4 is repeated for 5 times. Such that in all 5 times all the data is trained and tested on the various paramter values.
6. After successful implementaion of the step 5, it returns the best parameters from all 5 iteration of the loop. Based on this observation we select the parameters which are repeated more number of times.

Presentation of CV result for NonLinear regression

The inference and the observation of the result are mentioned in the last section.

In [86]:
```
models = [[6,2,1],[6,3,1],[6,4,1], [6,5,1], [6,6,1]]
best_accuracy,best_parameters = cross_validate(np.array(df_regression), np.array
```

```
ing: arrays to stack must be passed as a "sequence" type such as list or tupl
e. Support for non-sequence iterables such as generators is deprecated as of
NumPy 1.16 and will raise an error in the future.
  return np.hstack(map(np.ravel, w))

[[6, 2, 1], [6, 3, 1], [6, 4, 1], [6, 5, 1], [6, 6, 1]]
[0.02798059482042428, 0.11387171408740628, 0.059445927066852845, 0.0885005567
7861034, 0.07986521569662286]
[6, 2, 1]
k fold:  4
[6, 5, 1]

C:\Users\jeetj\3D Objects\Machine Learning\Assignment 3\nn.py:113: FutureWarn
ing: arrays to stack must be passed as a "sequence" type such as list or tupl
e. Support for non-sequence iterables such as generators is deprecated as of
NumPy 1.16 and will raise an error in the future.
  return np.hstack(map(np.ravel, w))
```

In [87]:
```
print(best_accuracy,best_parameters)
```

```
[0.08674884276973308, 0.10105005684533504, 0.0969356844578647, 0.01666132197084
5744, 0.10588595220928144] [[6, 4, 1], [6, 2, 1], [6, 3, 1], [6, 4, 1], [6, 5,
1]]
```

III.B Nonlinear Regression and its Plots

Summary of the Neural network

A neural network is exactly what it says in the name. It is a network of neurons that are used to process information. To create these, scientists looked at the most advanced data processing machine at the time — the brain. Our brains process information using networks of neurons. They receive an input, process it, and accordingly output electric signals to the neurons it is connected to.

Summary of the nonlinear regression model

Non Linear Regression is any relationship between an independent variable X and a dependent variable y which results in a non-linear function modelled data. Essentially any relationship that is not linear, can be termed as non-linear and is usually represented by the polynomial of k degrees (maximum power of X).

NeuralNet class

NeuralNet class is imported in the first cell of the notebook for processing the data by passing different parameters. In Standardizer class it normalizes all the data. then in Neural Network class there are various functions and variables which has there own individual functionality as follows:

1. The nunits takes the input, output and the hidden units.

2. Forward pass in the neural network is as follows:

$$E = \frac{1}{N}\frac{1}{K}\sum_{n=1}^{N}\sum_{k=1}^{K}(t_{nk} - y_{nk})^2$$
$$Y = \Phi W$$

$$Y_{nk} = \Phi_n^\top W_k$$

Now, let $\phi(x) = h(x)$ where h is the *activation function*.

$$Z = h(Xl \cdot \ V)$$

$$Y = Zl \cdot W$$

3. Backward pass in the neural network is as follows:

$$V \leftarrow V + \rho_h \frac{1}{N}\frac{1}{K} Xl^\top \Big((T - Y) W^\top \odot (1 - Z^2) \Big),$$
$$W \leftarrow W + \rho_o \frac{1}{N}\frac{1}{K} Zlm^\top \Big(T - Y \Big)$$

where ρ_h and ρ_o are the learning rate for hidden and output layer weights.

4. Different activation function is used to make the model non linear to avoid the problem of the vanishing the gradients.
5. The optimtarget function tries to reduce the error by updating the weights.

Explanation of codes of Cross Validation Results and Non Linear Regression

For performing non linear regression on our dataset. We need to define the number of parameters for neural network based on our requirement of the dataset. As after preprocessing of the dataset the number of features of the dataset are 6, so I performed the 5 fold cross validation on 5 different parameter values such as [[6,2,1],[6,3,1],[6,4,1], [6,5,1], [6,6,1]] where first parameter of the list indicated the number of features i.e. number of input nodes. Second parameter is the number of layers for neural network and third parameter is the number of the output nodes.

Based on this experiment it howed that the network with 6 unput nodes, 3 layers and one output node performed exceptional out of all the values. So [6,3,1] was chosen for the non linear rregression task.

Once the parameter of the network is selected. The task was to divide the data in different train and test set and pass it to the model for training and testing. For validating the result RMSE error was calculated and the R2 score of the model was calculated. As per observation the model performed quite well as the the RMSE error was too low as 1% and the R2 score was observed to be almost 1.

Plots of results of Non Linear Regression

Testing accuracy is calculated and plotted its accuracy in the graph. The model performed well as the accuracy observed was 99.37. The graph of the actual values corresponding to the predicted value is also plotted to observe the prediction pattern. Additional to this, residual graph is plotted that shows the residuals of the vertical axis and the independent variable on the horizontal axis.

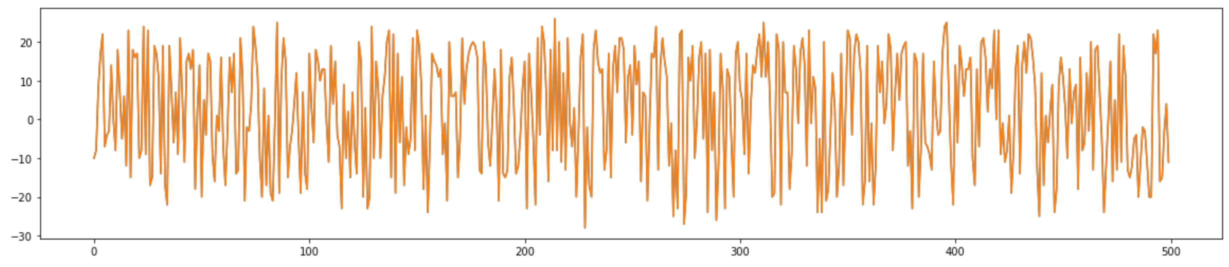
In [88]:
```
est,y_train,y_test = model_selection.train_test_split(np.array(df_regression),np.
```

In [17]:
```
model=NeuralNet([6,3,1])
model.train(X_train,y_train,niter=1000)
y_pred = model.use(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("RMSE Error: ",rmse)
score = r2_score(y_test,y_pred)
print("R2 score: ",score)
```

```
C:\Users\jeetj\3D Objects\Machine Learning\Assignment 3\nn.py:113: FutureWarnin
g: arrays to stack must be passed as a "sequence" type such as list or tuple. S
upport for non-sequence iterables such as generators is deprecated as of NumPy
1.16 and will raise an error in the future.
  return np.hstack(map(np.ravel, w))

RMSE Error:  0.011249913571529482
R2 score:  0.9999994029404657
```

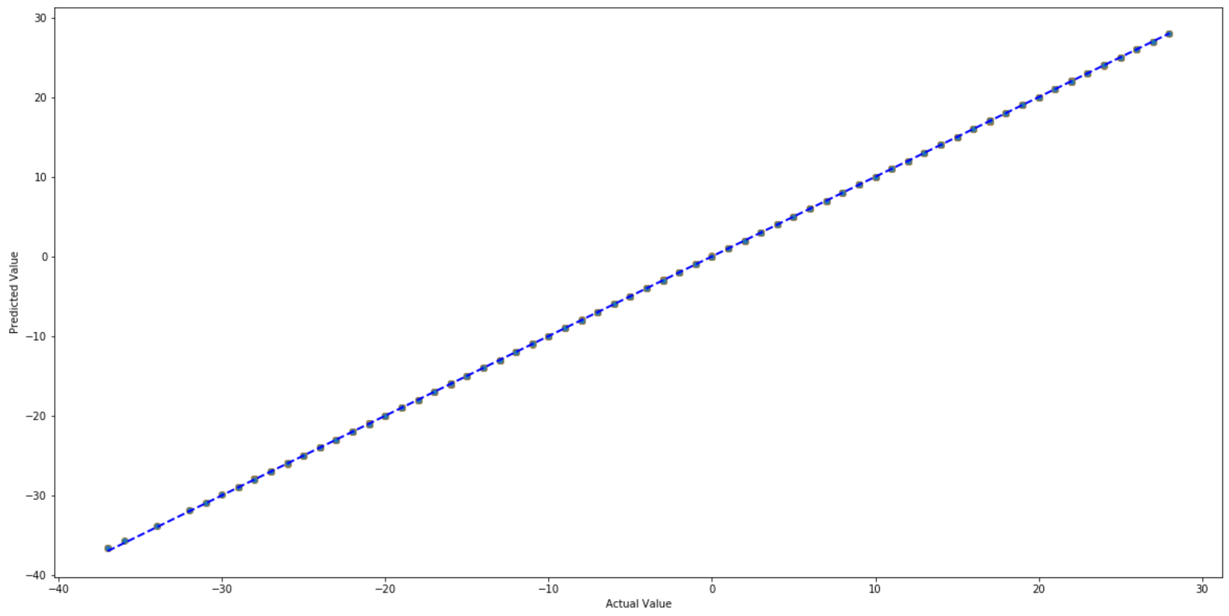
In [18]:
```
plt.figure(figsize = (20,4))
plt.plot(y_test[:500])
plt.plot(y_pred[:500])
print("Testing Accuracy: ", 100 - np.mean(np.abs(y_test - y_pred)) * 100, "%")
```

```
Testing Accuracy:  99.37616673896677 %
```

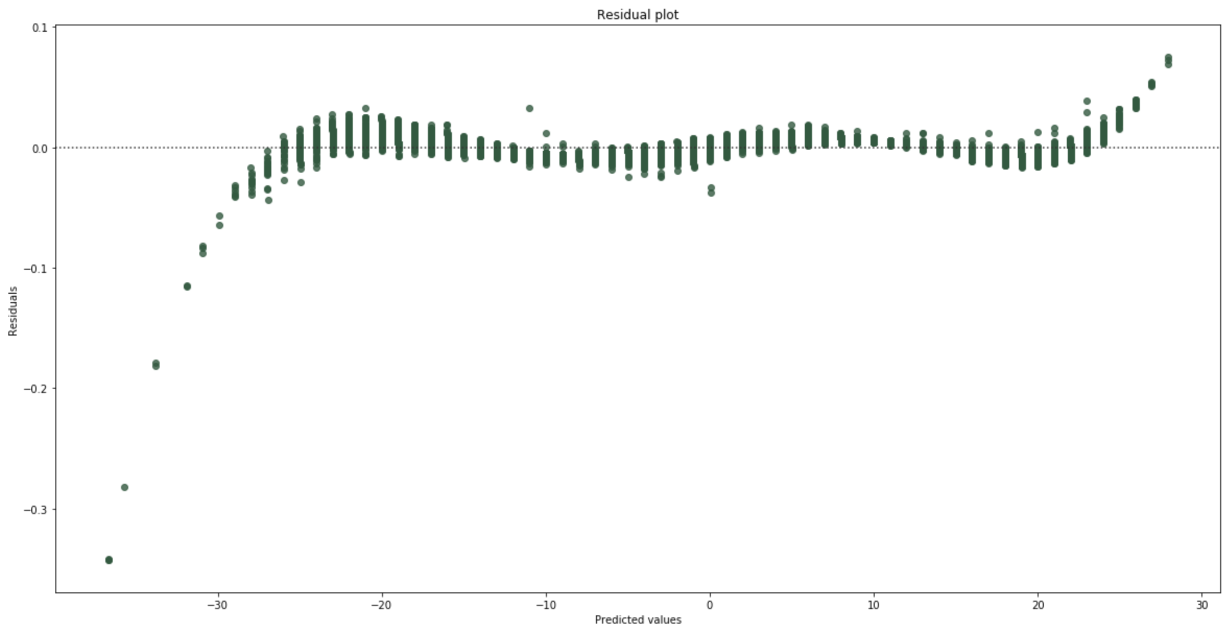


In [19]:
```
plt.figure(figsize = (15,4))
fig, ax = plt.subplots()
ax.scatter(y_test, y_pred, edgecolors=(0.6,0.5,0.3))
ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],'b--', lw=2)
ax.set_ylabel('Predicted Value')
ax.set_xlabel('Actual Value')
plt.show()
```

```
<Figure size 1080x288 with 0 Axes>
```



In [20]:
```
sns.residplot(y_pred.flatten(), y_test.flatten(), color=(0.2, 0.35, 0.25))
plt.title('Residual plot')
plt.xlabel('Predicted values')
plt.ylabel('Residuals');
```



III.C Nonlinear Logistic Regression for classification data

- Importing the classification data.
- Its visualization plots
- Preprocessing of data

Introduction of data for classification[7]
Dataset consists of feature vectors belonging to 12,330 sessions. It contains 10,422 negative class samples and 1908 positive class samples. It contains 18 attributes. The attributes has the information of the pages and time interval the user has visited that particular page. It also has the user system information, browser, region, and traffic information. It also keeps track of the users who enters the webpage and leave the page without triggering any requests. The dataset was formed so that each session would belong to a different user in a 1-year period to avoid any tendency to a specific campaign, special day, user profile, or period.

Source

1. C. Okan Sakar Department of Computer Engineering, Faculty of Engineering and Natural Sciences, Bahcesehir University, 34349 Besiktas, Istanbul, Turkey
2. Yomi Kastro Inveon Information Technologies Consultancy and Trade, 34335 Istanbul, Turkey

In [21]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import roc_curve, auc
import seaborn as sns
import math
```

In [22]:
```
df_classification = pd.read_csv('online_shoppers_intention.csv')
df_classification.head()
```

Out[22]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	Pr
0	0	0.0	0	0.0	1	
1	0	0.0	0	0.0	2	
2	0	0.0	0	0.0	1	
3	0	0.0	0	0.0	2	
4	0	0.0	0	0.0	10	

Preprocessing and Visualization of the Data

1. Count and display the missing values. Since there are no missing values, we do not need to remove or replace any data unless required.
2. Mapping string data type to inetger using dictionary in python. Columns such as Month, Revenue, Weekend and Visitor Type or perform One hot encoding by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction.
3. Perform label_encoding on target class which will bring value between 0 and n_classes-1.
Preprocessing of data helps to visualize the data and infer the patterns from the data.

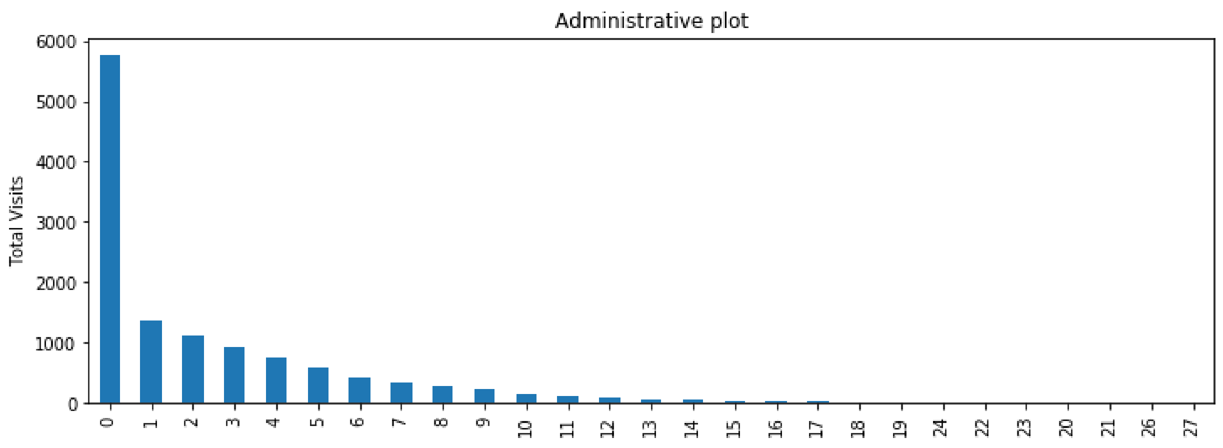
In [23]:
```
df_classification.isnull().sum()
```

Out[23]:
```
Administrative             0
Administrative_Duration    0
Informational              0
Informational_Duration     0
ProductRelated             0
ProductRelated_Duration    0
BounceRates                0
ExitRates                  0
PageValues                 0
SpecialDay                 0
Month                      0
OperatingSystems           0
Browser                    0
Region                     0
TrafficType                0
VisitorType                0
Weekend                    0
Revenue                    0
dtype: int64
```

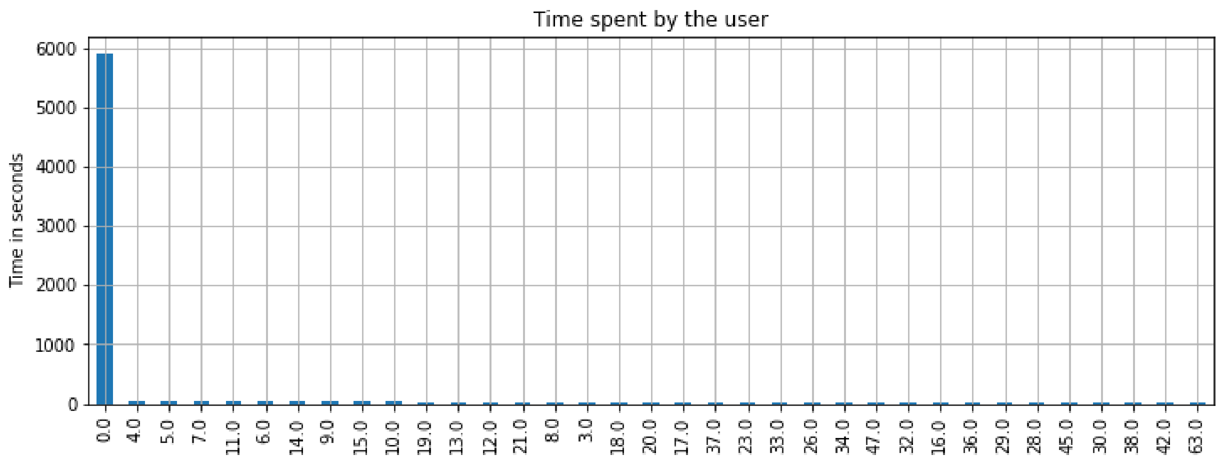
In [24]:
```
df_classification.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
Administrative             12330 non-null int64
Administrative_Duration    12330 non-null float64
Informational              12330 non-null int64
Informational_Duration     12330 non-null float64
ProductRelated             12330 non-null int64
ProductRelated_Duration    12330 non-null float64
BounceRates                12330 non-null float64
ExitRates                  12330 non-null float64
PageValues                 12330 non-null float64
SpecialDay                 12330 non-null float64
Month                      12330 non-null object
OperatingSystems           12330 non-null int64
Browser                    12330 non-null int64
Region                     12330 non-null int64
TrafficType                12330 non-null int64
VisitorType                12330 non-null object
Weekend                    12330 non-null bool
Revenue                    12330 non-null bool
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

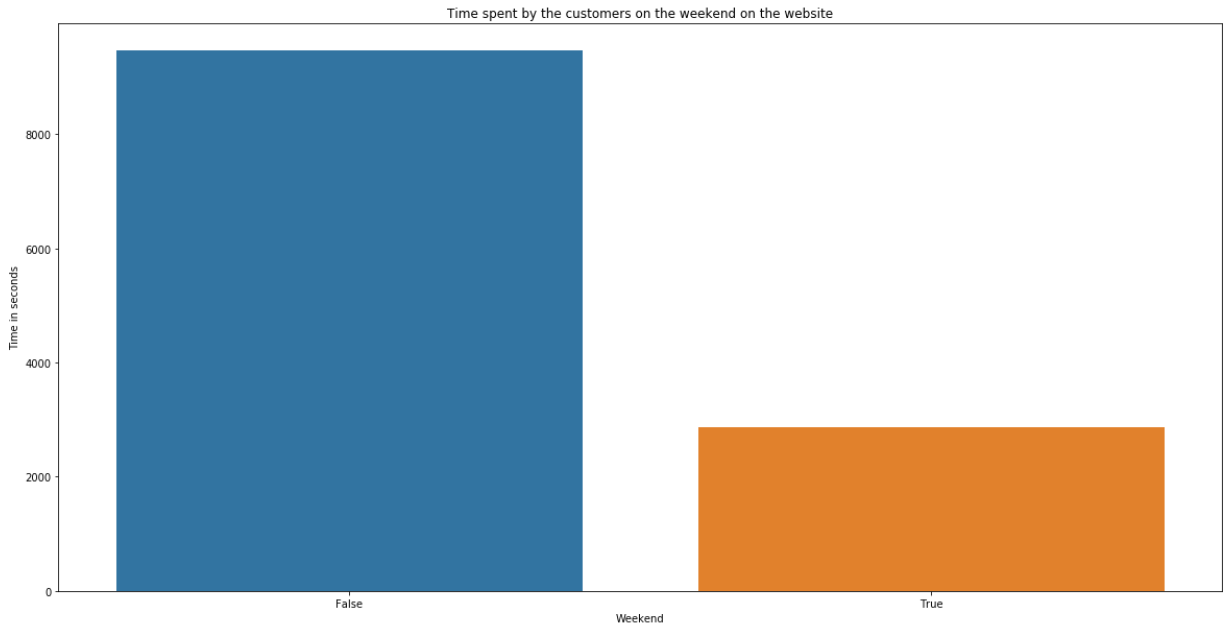
In [25]:
```
df_classification["Administrative"].value_counts().plot.bar(figsize=(12,4))
plt.title("Administrative plot")
plt.ylabel("Total Visits")
plt.show()
```



In [26]:
```
df_classification["Administrative_Duration"].value_counts().head(35).plot.bar(fi
plt.grid()
plt.title("Time spent by the user")
plt.ylabel("Time in seconds")
plt.show()
```

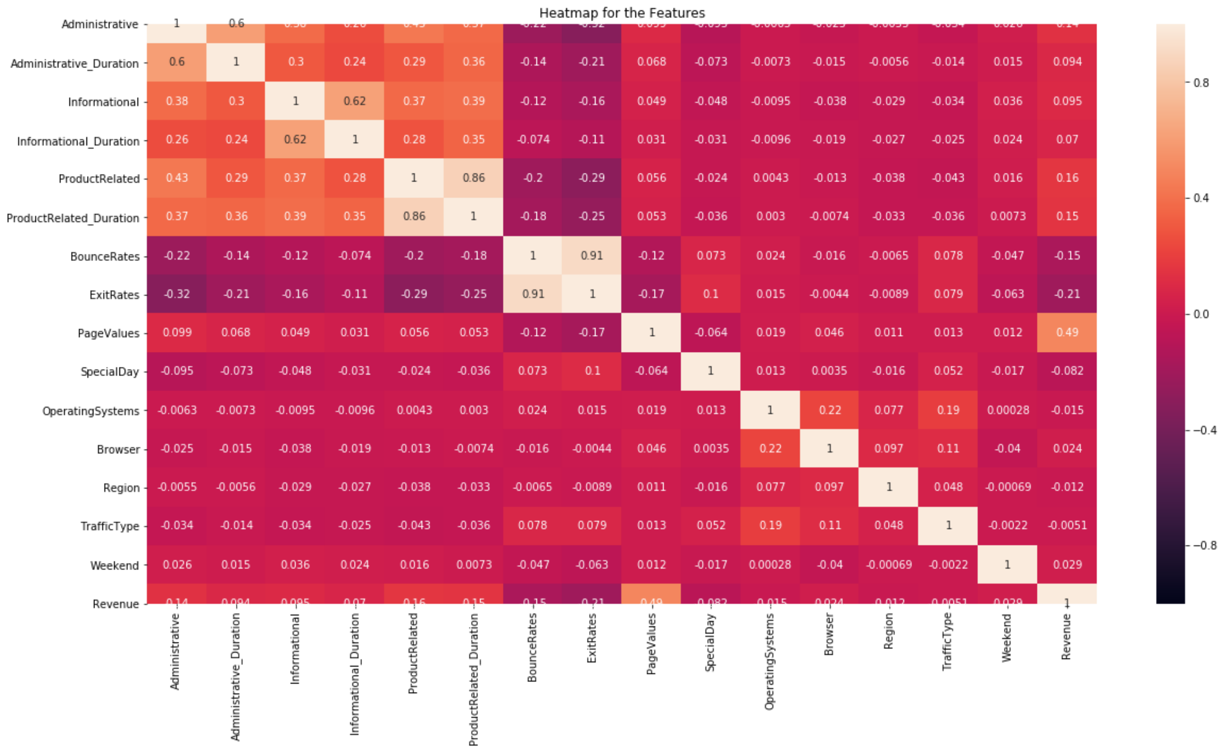


In [27]:
```
sns.countplot(x="Weekend",data=df_classification)
plt.title("Time spent by the customers on the weekend on the website")
plt.ylabel("Time in seconds")
plt.show()
```

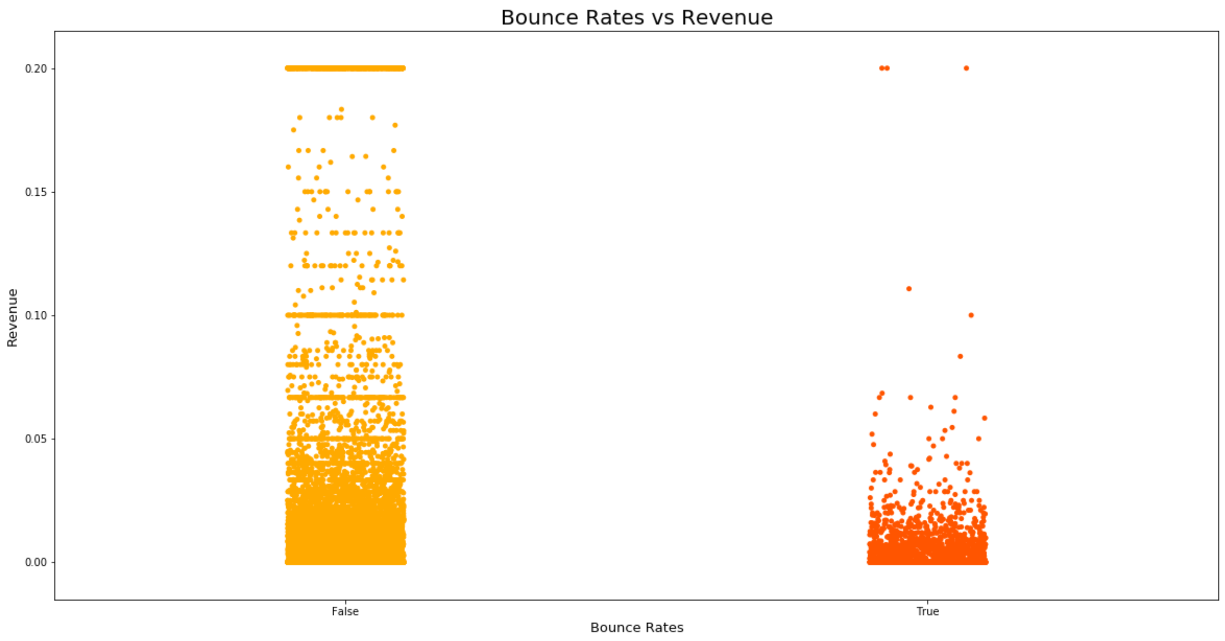


In [28]:
```
plt.rcParams['figure.figsize'] = (20, 10)
sns.heatmap(df_classification.corr(),vmin=-1,vmax=1,annot=True)
plt.title("Heatmap for the Features")
```

Out[28]:
```
Text(0.5, 1, 'Heatmap for the Features')
```



In [29]:
```
sns.stripplot(df_classification['Revenue'], df_classification['BounceRates'], pa
plt.title('Bounce Rates vs Revenue', fontsize = 20)
plt.xlabel('Bounce Rates', fontsize = 13)
plt.ylabel('Revenue', fontsize = 13)

plt.show()
```



In [30]:
```
drop = ['Administrative','Administrative_Duration','Informational','Informationa
df_classification=df_classification.drop(columns=drop)
```

In [31]:
```
df_classification_new = pd.get_dummies(df_classification)
```

In [32]:
```
le = LabelEncoder()
df_classification['Revenue'] = le.fit_transform(df_classification['Revenue'])
print(df_classification['Revenue'].value_counts())
revenue = {False:0, True: 1}
df_classification_new['Revenue'] = df_classification_new['Revenue'].apply(lambda
df_classification['Revenue'] = df_classification['Revenue'].apply(lambda x: rever
df_classification.head()
```

```
0    10422
1     1908
Name: Revenue, dtype: int64
```

Out[32]:

	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	Region	TrafficType	Revenue
0	1	0.000000	0.20	0.20	1	1	0
1	2	64.000000	0.00	0.10	1	2	0
2	1	0.000000	0.20	0.20	9	3	0
3	2	2.666667	0.05	0.14	2	4	0
4	10	627.500000	0.02	0.05	1	4	0

In [33]:
```
X = df_classification_new.iloc[:,0:6]
T = df_classification.iloc[:,6:7]

print("Shape of X:", X.shape)
print("Shape of T:", T.shape)
X.head()
```

```
Shape of X: (12330, 6)
Shape of T: (12330, 1)
```

Out[33]:

	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	Region	TrafficType
0	1	0.000000	0.20	0.20	1	1
1	2	64.000000	0.00	0.10	1	2
2	1	0.000000	0.20	0.20	9	3
3	2	2.666667	0.05	0.14	2	4
4	10	627.500000	0.02	0.05	1	4

In [34]:
```
T.shape
```

Out[34]:
```
(12330, 1)
```

Visualization of Data

There are different kind of visualization techniques for visualizing the data like scatter plot, heat map, bar plot, histogram plot and many more.

1. Plotted the graph of the Administrative vs the number of different kinds of webpages visited in the session to infer the relation between them.
2. Then plotted the graph of inital 35 data of time spent in first session in one page, to check how frequently the user changes the webpage.
3. Further plotted the graph to observe the traffic on the webpage and time spent by the user on the weekend.
4. Generated the graphs of the all the features with all the other features by using the pairplot[4] to check which features are more corelated and which features are uncorelated.
5. Further generated the heatmap[3] to quantify the corelation of the pairplots.

III.C Nonlinear Logistic Regression and its Plots

- Summarize the nonlinear logistic regression model.

The linear logistic regression uses the softmax layer for classification along with a linear model. In nonlinear logistic regression extra layers are added called as hidden layers. With the inclusion of hidden layer the softmax function is used in the last layer. Here, the weight updates are performed and based on the gradient descent the error is calculated as follows: From the error function $E(w)$, we can derive the gradient to update the weights for each layer.

$$v_{dg} \leftarrow v_{dg} - \alpha_h \frac{\partial E(W,V)}{\partial v_{dg}}$$

$$w_{gk} \leftarrow w_{gk} - \alpha_o \frac{\partial E(W,V)}{\partial w_{gk}},$$

where α_h and α_o are the learning rate for hidden and output layer respectively. Here, we denote the output of the neural network as κ.

Summary (Regression vs Classification)

	Regression	Class
Forward Pass	$Z = h(Xl \cdot V)$ $Y = Zl \cdot W$	$Z = h(Xl \cdot V)$ $Y = Zl \cdot W$ $G = softmax(Y)$
Backward Pass	$V \leftarrow V + \alpha_h \frac{1}{N}\frac{1}{K} Xl^\top \Big((T-Y)W^\top \odot (1-Z^2)\Big)$ $W \leftarrow W + \alpha_o \frac{1}{N}\frac{1}{K} Zl^\top \Big(T-Y\Big)$	$V \leftarrow V + \alpha_h Xl^\top \Big((T-G)W^\top \odot (1 \cdot$ $W \leftarrow W + \alpha_o Zl^\top \Big(T-G\Big)$

Note: Here T is a matrix with indicator

and G is the output matrix after the softm

Explaination of the codes

In a NeuralNetLogReg class there are many functions and variables as NeuralNetwork class which has their own meaning as follows:

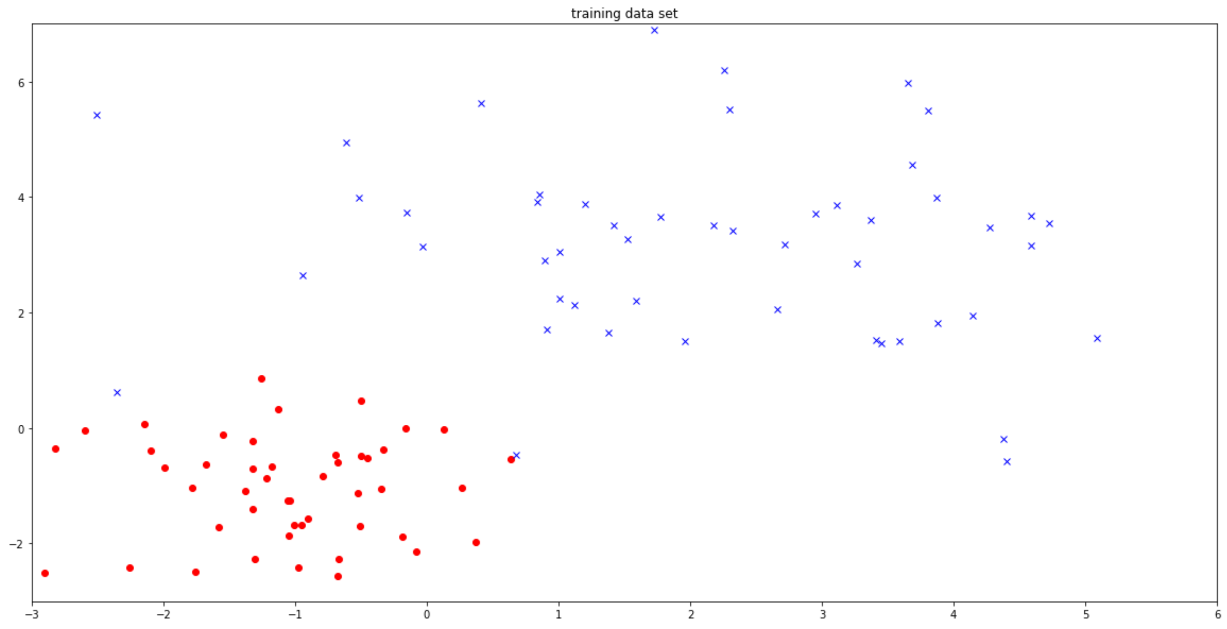
1. Initially the init function is a constructor of the class which calls the NeuralNet class for assigning the values of the parameters like value of the layers which is passed to the network.
2. rho is the learning rate of the class
3. The train function takes the features and the target as an input then it calls the other function where the weights are unpack and based on that it proceeds with the forward pass by calling forward function. Following to this, back propogation is called and based on the error parameter the update is performed and propogates the weights. In forward pass there is only one difference in last layer an additional softmax function is added which gives the output to the output layer in the form of the probability.Coverting vector in matrix form for the hidden weight update,
$$V \leftarrow V + \alpha_h X l^\top \Big((T - g(X)) W^\top \odot (1 - Z^2) \Big).$$
4. Here stdX and stdT are the data and target variable of the class.
5. Activation functions are used to make the network non-linear which solves the problem of the vanishing gradients.

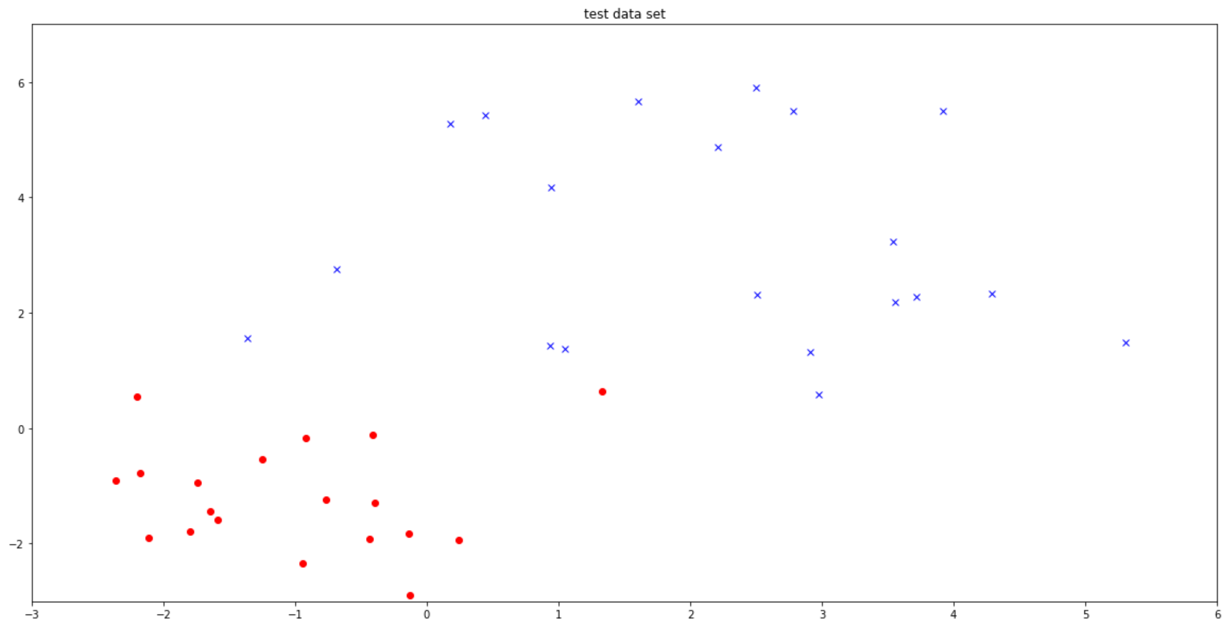
In [35]:
```
class NeuralNetLogReg(NeuralNet):
    """ Nonlinear Logistic Regression
    """

    # if you think, you need additional items to initialize here,
    # add your code for it here
    def __init__(self, nunits):
        NeuralNet.__init__(self, nunits)

    def softmax(self, z):
        if not isinstance(z, np.ndarray):
            z = np.asarray(z)
        expz = np.exp(z)
        return expz / (np.sum(expz, axis=1, keepdims=True) if len(z.shape) == 2 

    # Looking at the final summary or comparison table in lecture note,
    # add your codes for forward pass for logistic regression
    def forward(self, X):
        t = X
        Z = []

        for i in range(self._nLayers):
            Z.append(t)
            if i == self._nLayers - 1:
                t = np.dot(self.add_ones(t), self._W[i])
                t = self.softmax(t)
            else:
                t = np.tanh(np.dot(self.add_ones(t), self._W[i]))
                #t = self.RBF(np.dot(np.hstack((np.ones((t.shape[0],1)),t)),self
        return (t, Z)

    # This is the error function that we want to minimize
    # what was it? take a look at the lecture note to fill in
    def _objectf(self, T, Y, wpenalty):
        return -(np.sum(T*(np.log(Y)))+wpenalty)

    # you must reuse the NeuralNet train since you already modified
    # the objective or error function (maybe both),
    # you do not have many to change here.
    # MAKE SURE convert a vector label T to indicator matrix and
    # feed that for training
    def train(self, X, T, **params):
        verbose = params.pop('verbose', False)
        # training parameters
        _lambda = params.pop('Lambda', 0.)

        #parameters for scg
        niter = params.pop('niter', 1000)
        wprecision = params.pop('wprecision', 1e-10)
        fprecision = params.pop('fprecision', 1e-10)
        wtracep = params.pop('wtracep', False)
        ftracep = params.pop('ftracep', False)

        # optimization
        optim = params.pop('optim', 'scg')
        if self.stdX == None:
            explore = params.pop('explore', False)
            self.stdX = Standardizer(X, explore)
        Xs = self.stdX.standardize(X)
        if self.stdT == None and self.stdTarget:
            self.stdT = Standardizer(T)
            T = self.stdT.standardize(T)

        def gradientf(weights):
            self.unpack(weights)
            Y,Z = self.forward(Xs)
            error = self._errorf(T, Y)
            return self.backward(error, Z, T, _lambda)

        def optimtargetf(weights):
            """ optimization target function : MSE
            """
            self.unpack(weights)
            #self._weights[:] = weights[:]  # unpack
            Y,_ = self.forward(Xs)
            Wnb=np.array([])
            for i in range(self._nLayers):
                if len(Wnb)==0: Wnb=self._W[i][1:,].reshape(self._W[i].size-self
                else: Wnb = np.vstack((Wnb,self._W[i][1:,].reshape(self._W[i].si
            wpenalty = _lambda * np.dot(Wnb.flat ,Wnb.flat)
            return self._objectf(T, Y, wpenalty)

        if optim == 'scg':
            result = scg(self.cp_weight(), gradientf, optimtargetf,
                                        wPrecision=wprecision, fPrecision=fpreci
                                        nIterations=niter,
                                        wtracep=wtracep, ftracep=ftracep,
                                        verbose=False)
            self.unpack(result['w'][:])
            self.f = result['f']
        elif optim == 'steepest':
            result = steepest(self.cp_weight(), gradientf, optimtargetf,
                                nIterations=niter,
                                xPrecision=wprecision, fPrecision=fprecision,
                                xtracep=wtracep, ftracep=ftracep )
            self.unpack(result['w'][:])
        if ftracep:
            self.ftrace = result['ftrace']
        if 'reason' in result.keys() and verbose:
            print(result['reason'])

        return result


    # going through forward pass, you will have the probabilities for each label
    # now, you can use argmax to find class labels
    # return both label and probabilities
    def use(self, X):
        Y, Z = self.forward(X)
        index = np.argmax(Y,1)
        return Y,index
```

Testing on Toy dataset and its plots

In [36]:
```
# Data for testing
N1 = 50
N2 = 50
N = N1 + N2
D = 2
K = 2

mu1 = [-1, -1]
cov1 = np.eye(2)

mu2 = [2,3]
cov2 = np.eye(2) * 3

#
#  Train Data
#
C1 = np.random.multivariate_normal(mu1, cov1, N1)
C2 = np.random.multivariate_normal(mu2, cov2, N2)

plt.plot(C1[:, 0], C1[:, 1], 'or')
plt.plot(C2[:, 0], C2[:, 1], 'xb')

plt.xlim([-3, 6])
plt.ylim([-3, 7])
plt.title("training data set")

Xtrain = np.vstack((C1, C2))
Ttrain = np.zeros((N, 1))
Ttrain[50:, :] = 1  # labels are zero or one

means, stds = np.mean(Xtrain, 0), np.std(Xtrain, 0)
# normalize inputs
Xtrains = (Xtrain - means) / stds

#
#  Test Data
#
Ct1 = np.random.multivariate_normal(mu1, cov1, 20)
Ct2 = np.random.multivariate_normal(mu2, cov2, 20)

Xtest = np.vstack((Ct1, Ct2))
Ttest = np.zeros((40, 1))
Ttest[20:, :] = 1

# normalize inputs
Xtests = (Xtrain - means) / stds


plt.figure()
plt.plot(Ct1[:, 0], Ct1[:, 1], 'or')
plt.plot(Ct2[:, 0], Ct2[:, 1], 'xb')

plt.xlim([-3, 6])
plt.ylim([-3, 7])
plt.title("test data set")
```

Out[36]:
```
Text(0.5, 1.0, 'test data set')
```





In [37]:
```
clsf = NeuralNetLogReg([2, 4, 2])
clsf.train(Xtrain, Ttrain)
classes, Y = clsf.use(Xtest)
```

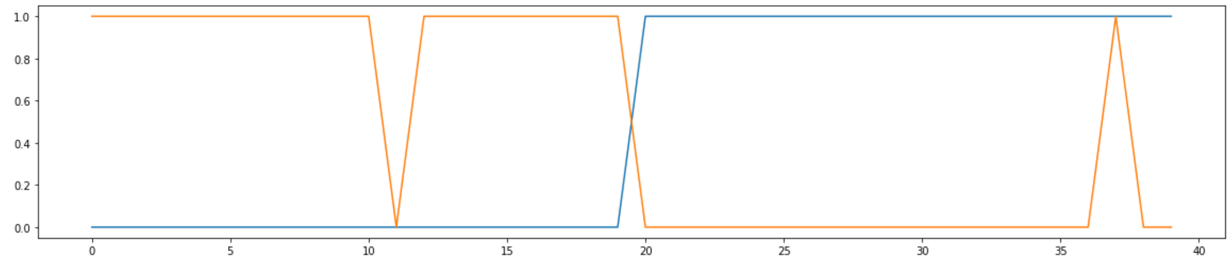
```
C:\Users\jeetj\3D Objects\Machine Learning\Assignment 3\nn.py:113: FutureWarnin
g: arrays to stack must be passed as a "sequence" type such as list or tuple. S
upport for non-sequence iterables such as generators is deprecated as of NumPy
1.16 and will raise an error in the future.
  return np.hstack(map(np.ravel, w))
C:\Users\jeetj\Miniconda3\envs\senses\lib\site-packages\ipykernel_launcher.py:1
4: RuntimeWarning: invalid value encountered in true_divide
```

In [38]:
```
classes,Y
```

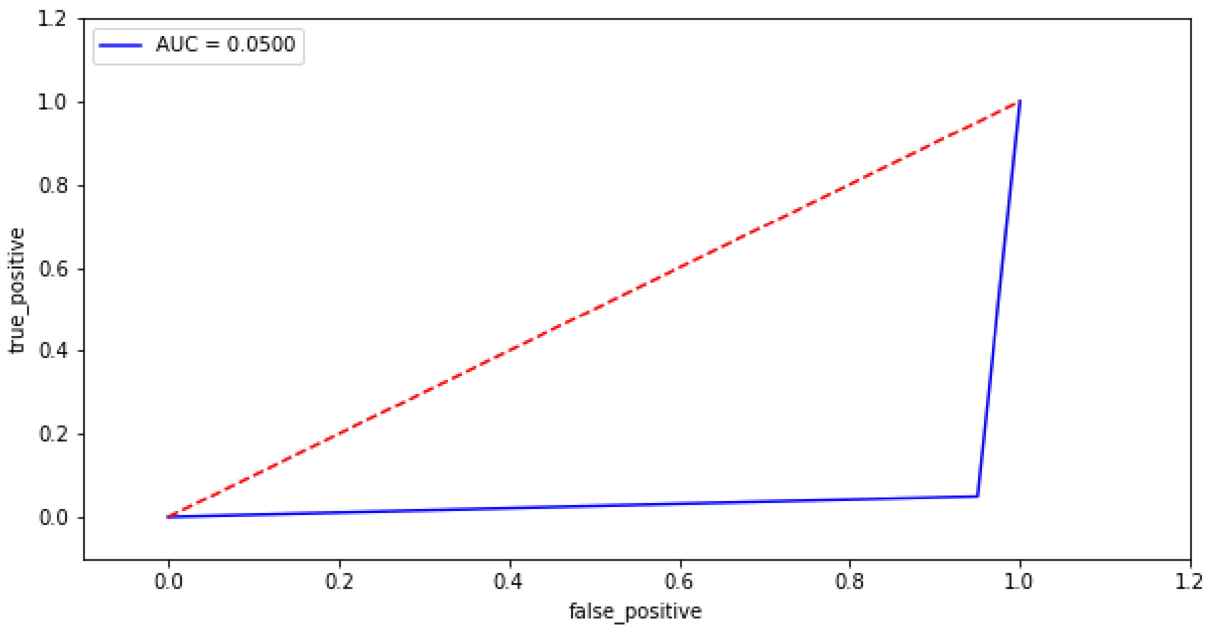
Out[38]:
```
(array([[0.48794118, 0.51205882],
        [0.4940819 , 0.5059181 ],
        [0.4821248 , 0.5178752 ],
        [0.48519306, 0.51480694],
        [0.48780587, 0.51219413],
        [0.48240689, 0.51759311],
        [0.48858664, 0.51141336],
        [0.48296553, 0.51703447],
        [0.48401319, 0.51598681],
        [0.48677223, 0.51322777],
        [0.48170546, 0.51829454],
        [0.50815177, 0.49184823],
        [0.4792603 , 0.5207397 ],
        [0.48260273, 0.51739727],
        [0.4817959 , 0.5182041 ],
        [0.48808939, 0.51191061],
        [0.4911751 , 0.5088249 ],
        [0.47810329, 0.52189671],
        [0.4806426 , 0.5193574 ],
        [0.48668581, 0.51331419],
        [0.52791721, 0.47208279],
        [0.52225151, 0.47774849],
        [0.52495547, 0.47504453],
        [0.52529976, 0.47470024],
        [0.50895182, 0.49104818],
        [0.52297273, 0.47702727],
        [0.51940368, 0.48059632],
        [0.51008762, 0.48991238],
        [0.52530679, 0.47469321],
        [0.51497166, 0.48502834],
        [0.52554705, 0.47445295],
        [0.53031293, 0.46968707],
        [0.51748775, 0.48251225],
        [0.52008793, 0.47991207],
        [0.5220058 , 0.4779942 ],
        [0.52470192, 0.47529808],
        [0.51035794, 0.48964206],
        [0.49881237, 0.50118763],
        [0.52086713, 0.47913287],
        [0.52807072, 0.47192928]]),
 array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0], dtype=int64))
```

In [39]:
```python
plt.clf()
plt.figure(figsize = (20,4))
plt.plot(Ttest)
plt.plot(Y)
cm = confusion_matrix(Ttest, Y)
print("\n Confusion Matrix: \n", cm)
cr = classification_report(Ttest,Y)
print("\n Classification Report: \n", cr)
print("Accuracy: ", 100 - np.mean(np.abs(Ttest - Y)) * 100, "%")
```

```
 Confusion Matrix: 
 [[ 1 19]
 [19  1]]

 Classification Report: 
               precision    recall  f1-score   support

         0.0       0.05      0.05      0.05        20
         1.0       0.05      0.05      0.05        20

    accuracy                           0.05        40
   macro avg       0.05      0.05      0.05        40
weighted avg       0.05      0.05      0.05        40

Accuracy:  50.0 %

<Figure size 1440x720 with 0 Axes>
```

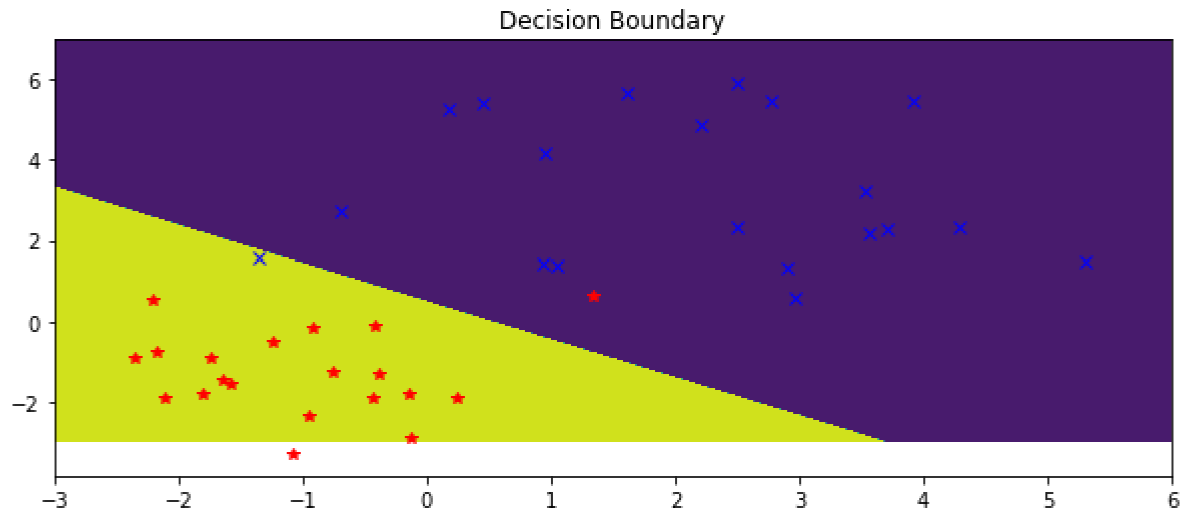


In [40]:
```
false_positive, true_positive, thresholds = roc_curve(Ttest, Y)
roc_auc = auc(false_positive, true_positive)
plt.figure(figsize = (10,5))
plt.plot(false_positive, true_positive, 'b',
label='AUC = %0.4f'% roc_auc)
plt.legend(loc='upper left')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.1,1.2])
plt.ylim([-0.1,1.2])
plt.ylabel('true_positive')
plt.xlabel('false_positive')
plt.show()
```



In [41]:
```
x = np.linspace(-3, 6, 1000)
y = np.linspace(-3, 7, 1000)
xs, ys = np.meshgrid(x, y)
X = np.vstack((xs.flat, ys.flat)).T
classes, _ = clsf.use(X)
zs = _.reshape(xs.shape)

plt.figure(figsize=(10,4))
plt.contourf(xs, ys, zs.reshape(xs.shape))
plt.title("Decision Boundary")

plt.plot(Ct1[:, 0], Ct1[:, 1], '*r')
plt.plot(Ct2[:, 0], Ct2[:, 1], 'xb')
```

Out[41]:
```
[<matplotlib.lines.Line2D at 0x22eec83f748>]
```



Explanation of correct implementation with the toy data

I sucessfully trained the toy data with non-linear logistic regression. I have used softmax as an activation function to train the neural network. The toy data is trained with 2 input nodes, 4 hidden layers and 2 output node. It achieves the accuracy of 50%. The low accuracy is due to less amount of data. The above decison boundary plot shows the misclassification of 1 or samples of the given dataset.

III.C Testing on classification dataset and its Plots

In [42]:
```
X_train1,X_test1,y_train1,y_test1 = model_selection.train_test_split(np.array(df_
```

In [43]:
```
print(X_train1.shape,y_train1.shape,X_test1.shape,y_test1.shape)
```

```
(9864, 7) (9864, 1) (2466, 7) (2466, 1)
```

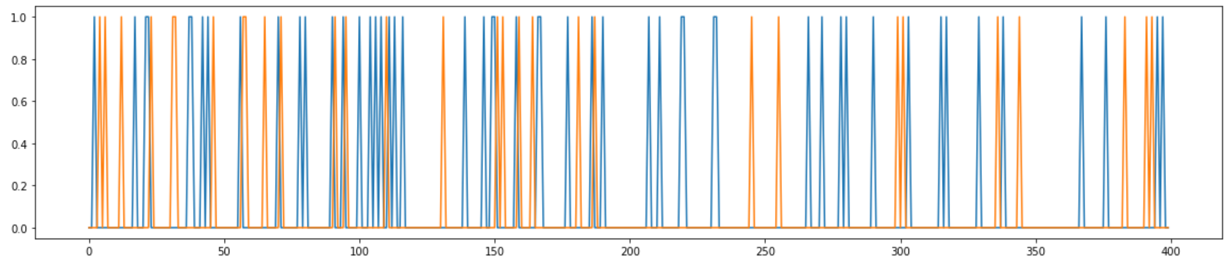
In [125]:
```
nn = NeuralNetLogReg([7,3,2])
nn.train(X_train1, y_train1, niter=1000)
probability,label = nn.use(X_test1)
```

In [126]:
```
print(probability[0:10],label[0:10])
```

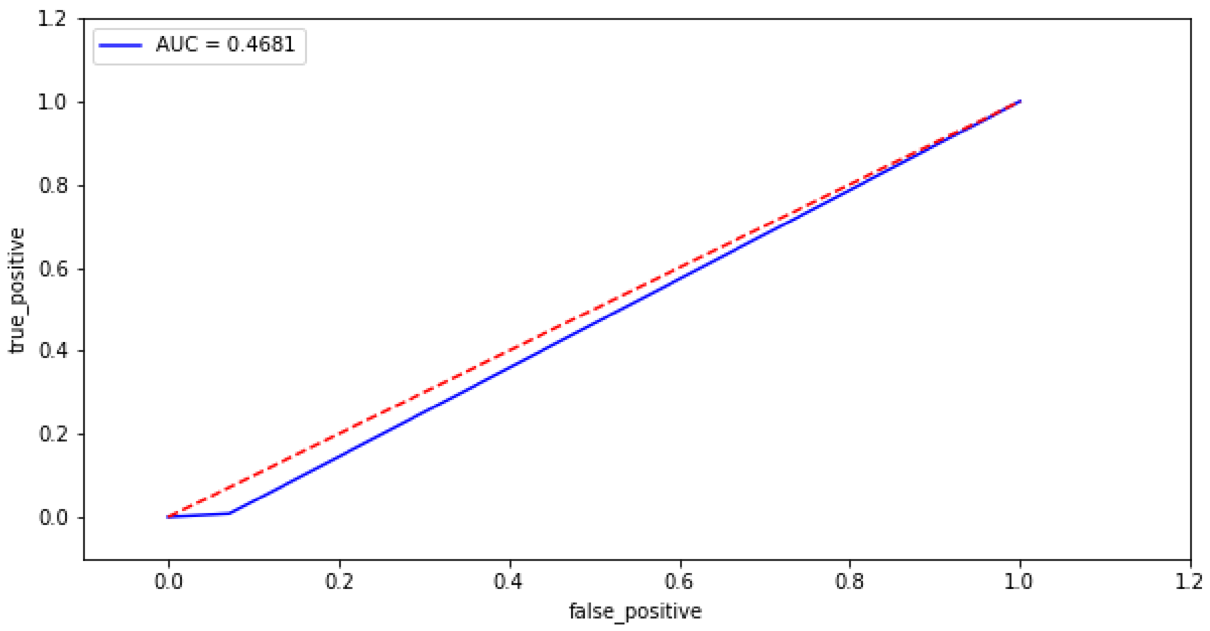
```
[[0.55869453 0.44130547]
 [0.51745884 0.48254116]
 [0.55869453 0.44130547]
 [0.55868972 0.44131028]
 [0.49501688 0.50498312]
 [0.55869453 0.44130547]
 [0.49442648 0.50557352]
 [0.55869453 0.44130547]
 [0.55869453 0.44130547]
 [0.55869453 0.44130547]] [0 0 0 0 1 0 1 0 0 0]
```

In [127]:
```
plt.figure(figsize = (20,4))
plt.plot(y_test1[:400])
plt.plot(label[:400])
cm = confusion_matrix(y_test1,label)
print("Confusion Matrix: \n", cm)

cr = classification_report(y_test1,label)
print("\nClassification Report: \n", cr)
print("Testing Accuracy: ", 100 - np.mean(np.abs(y_test1 - label)) * 100, "%")
```

```
Confusion Matrix: 
 [[1949  151]
 [ 363    3]]

Classification Report: 
               precision    recall  f1-score   support

           0       0.84      0.93      0.88      2100
           1       0.02      0.01      0.01       366

    accuracy                           0.79      2466
   macro avg       0.43      0.47      0.45      2466
weighted avg       0.72      0.79      0.75      2466

Testing Accuracy:  80.7669462845551 %
```



In [128]:
```
false_positive, true_positive, thresholds = roc_curve(y_test1, label)
roc_auc = auc(false_positive, true_positive)
plt.figure(figsize = (10,5))
plt.plot(false_positive, true_positive, 'b',
label='AUC = %0.4f'% roc_auc)
plt.legend(loc='upper left')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.1,1.2])
plt.ylim([-0.1,1.2])
plt.ylabel('true_positive')
plt.xlabel('false_positive')
plt.show()
```



In [95]:
```
def k_fold_1(s):
    k_val_1 = len(s) / 5.0
    partioned_data_1 = []
    count_1 = 0.0
    while count_1 < len(s):
        partioned_data_1.append(s[int(count_1):int(count_1 + k_val_1)])
        count_1 += k_val_1
    return partioned_data_1
```

In [103]:
```
final_list=[]
best_paramlist=[]
def cross_validate(X, T, parameters):
    feature_partition_1 = k_fold_1(X)
    target_partition_1 = k_fold_1(T)
    for i in range(5):
        print("Iteration: ",i)
        Xtest = feature_partition_1[i]
        Ttest = target_partition_1[i]

        history_rmse=[]
        params=[]
        for k in range(5):
            print("k fold: ",k)
            if i == k: continue

            Xval = feature_partition_1[k]
            Tval = target_partition_1[k]

            Xtrain = feature_partition_1[not i and not k]
            Ttrain = target_partition_1[not i and not k]

            flist=[]
            paramlist=[]
            for param in parameters:

                model = NeuralNetLogReg(param)
                model.train(Xtrain, Ttrain)
                prob,pred = model.use(Xval)
                Tval1=Tval[:,0]
                Tval1=np.where(Tval1==0,1,0)
                valAcc = f1_score(Tval1, pred)
                flist.append(valAcc)
                paramlist.append(param)

            print(max(flist))
            print(paramlist[np.argmax(flist)])
            history_rmse.append(max(flist))
            params.append(paramlist[np.argmax(flist)])


        bestParam = params[np.argmax(history_rmse)]
        bestModel = NeuralNetLogReg(bestParam)

        X_train1 = feature_partition_1[not i]
        T_train1 = target_partition_1[not i]
        bestModel.train(X_train1, T_train1)

        prob1,finalPred = bestModel.use(X_test1)
        Ttest1=Ttest[:,0]
        Ttest1=np.where(Ttest1==0,1,0)
        myf1 = f1_score(Ttest1, finalPred)
        final_list.append(myf1)
        best_paramlist.append(bestParam)

    return final_list,best_paramlist
```

Presentation of CV result for NonLinear Logistic Regression

I have explained the results and inference in the bottom section

In [104]:
```
models = [[7,4,2], [7,5,2],[7,6,2], [7,7,2], [7,8,2]]
bestAccuracy,bestParam = (cross_validate(np.array(df_classification_new),np.array
```

```
Iteration:  0
k fold:  0
k fold:  1

C:\Users\jeetj\Miniconda3\envs\senses\lib\site-packages\ipykernel_launcher.py:1
4: RuntimeWarning: invalid value encountered in true_divide


0.94389721627409
[7, 7, 2]
k fold:  2
0.9100552486187845
[7, 5, 2]
k fold:  3
0.8511432571161922
[7, 4, 2]
k fold:  4
0.8237263812485052
[7, 5, 2]
Iteration:  1
k fold:  0
0.9497444633730835
[7, 7, 2]
k fold:  1
k fold:  2
0.9100552486187845
[7, 5, 2]
k fold:  3
0.8849197377345692
[7, 5, 2]
k fold:  4
0.8886885984677783
[7, 6, 2]
Iteration:  2
k fold:  0
0.9497444633730835
[7, 6, 2]
k fold:  1
0.94389721627409
[7, 7, 2]
k fold:  2
k fold:  3
0.9075511644318983
[7, 6, 2]
k fold:  4
0.8886885984677783
[7, 5, 2]
Iteration:  3
k fold:  0
0.9497444633730835
[7, 8, 2]
k fold:  1
0.94389721627409
[7, 5, 2]
```

```
k fold:  2
0.9100552486187845
[7, 5, 2]
k fold:  3
k fold:  4
0.8886885984677783
[7, 4, 2]
Iteration:  4
k fold:  0

C:\Users\jeetj\Miniconda3\envs\senses\lib\site-packages\ipykernel_launcher.py:1
3: RuntimeWarning: overflow encountered in exp
  del sys.path[0]
C:\Users\jeetj\Miniconda3\envs\senses\lib\site-packages\ipykernel_launcher.py:3
5: RuntimeWarning: divide by zero encountered in log
C:\Users\jeetj\3D Objects\Machine Learning\Assignment 3\grad.py:128: RuntimeWar
ning: invalid value encountered in double_scalars
  Delta = 2. * (fnew - fold) / (alpha*mu)

0.939582885400989
[7, 6, 2]
k fold:  1
0.94389721627409
[7, 8, 2]
k fold:  2
0.9100552486187845
[7, 8, 2]
k fold:  3
0.8849197377345692
[7, 4, 2]
k fold:  4
```

In [105]:
```
print(bestAccuracy,bestParam)
```

```
[0.008928571428571428, 0.1328289197141656, 0.1616871704745167, 0.1526364477335
8, 0.8886885984677783] [[7, 7, 2], [7, 7, 2], [7, 6, 2], [7, 8, 2], [7, 8, 2]]
```

In [109]:
```
nn = NeuralNetLogReg([7,7,2])
nn.train(X_train1, y_train1, niter=1000)
probability1,label1 = nn.use(X_test1)
```

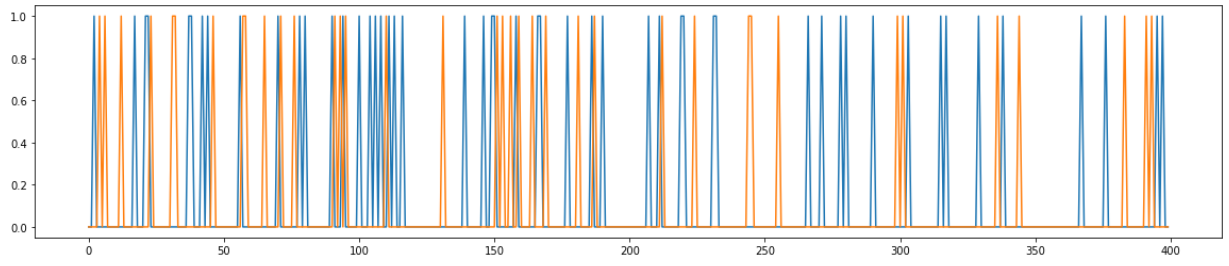
```
C:\Users\jeetj\Miniconda3\envs\senses\lib\site-packages\ipykernel_launcher.py:1
4: RuntimeWarning: invalid value encountered in true_divide
```

In [110]:
```
print(probability1,label1)
```

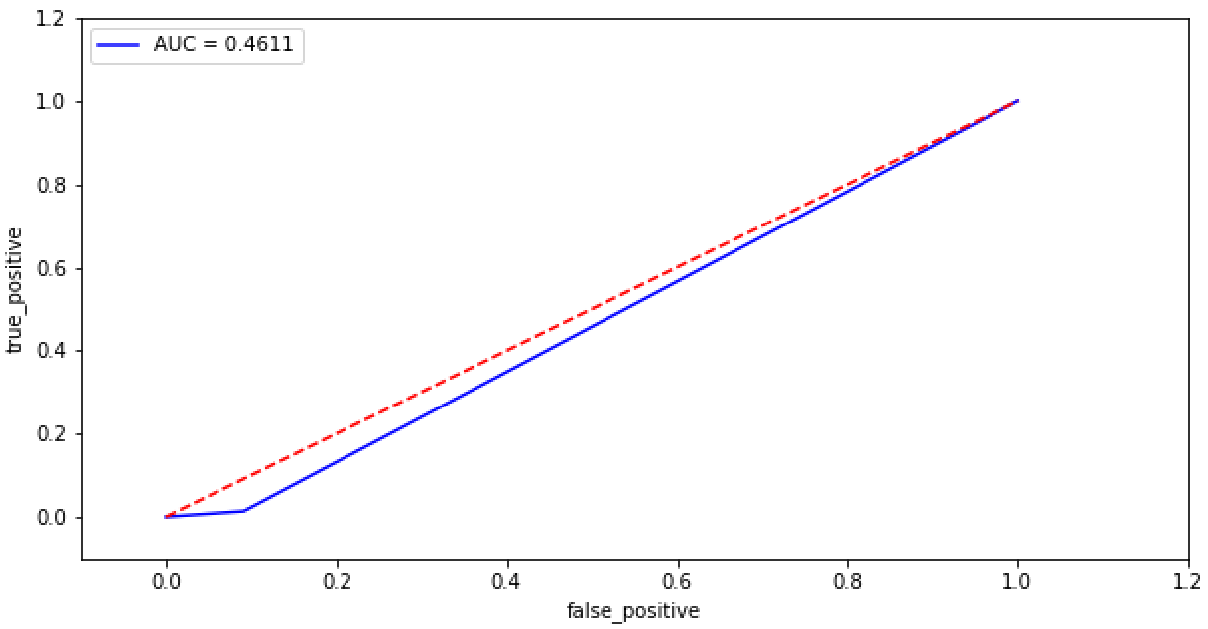
```
[[0.51679152 0.48320848]
 [0.50288119 0.49711881]
 [0.51679152 0.48320848]
 ...
 [0.51663811 0.48336189]
 [0.51679152 0.48320848]
 [0.51679152 0.48320848]] [0 0 0 ... 0 0 0]
```

In [111]:
```
plt.figure(figsize = (20,4))
plt.plot(y_test1[:400])
plt.plot(label1[:400])
cm = confusion_matrix(y_test1,label1)
print("Confusion Matrix: \n", cm)

cr = classification_report(y_test1,label1)
print("\nClassification Report: \n", cr)
print("Testing Accuracy: ", 100 - np.mean(np.abs(y_test1 - label1)) * 100, "%")
```

```
Confusion Matrix:
 [[1908  192]
 [ 361    5]]

Classification Report:
               precision    recall  f1-score   support

           0       0.84      0.91      0.87      2100
           1       0.03      0.01      0.02       366

    accuracy                           0.78      2466
   macro avg       0.43      0.46      0.45      2466
weighted avg       0.72      0.78      0.75      2466

Testing Accuracy:  79.54083072363215 %
```



In [112]:
```
false_positive, true_positive, thresholds = roc_curve(y_test1, label1)
roc_auc = auc(false_positive, true_positive)
plt.figure(figsize = (10,5))
plt.plot(false_positive, true_positive, 'b',
label='AUC = %0.4f'% roc_auc)
plt.legend(loc='upper left')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.1,1.2])
plt.ylim([-0.1,1.2])
plt.ylabel('true_positive')
plt.xlabel('false_positive')
plt.show()
```



Discussion about various plots

The plots and the inference of the plots are mentioned above while implementing the different methods. Following plots are plotted:

1. Plots for visualization of the Regression data
2. Plots for Non Linear Regression of actual and predicted output, Residual plot, and Testing accuracy plot.
3. Plots for visualization of the classification data
4. Plots for visualization of Non Linear Logistic Regression such as ROC curve, classification report, confusion matrix, and actual and predicted value curve.
5. Plots for the visualization of the Toy dataset.

Discussion about the parameter network structure

- In this assignment, Initially I formed the 5 fold cross validation which helped me to select the optimal and best network parameters. From the process of 5 fold Cross Validation it gave me two best parametes one for my Non-linear regression model and another for Non-Linear Logistic Regression model.
- For Non-Linear Regression model, I designed the neural network with 6 input nodes as there were 6 different features for my data. It had 3 input layers and One output node. I performed 5 fold CV on the model by passing this variables [[6,2,1],[6,3,1],[6,4,1], [6,5,1], [6,6,1]]. As I said above her First value is the Input features(Input nodes), second parameter is the number of layers and last parameter is output node. The accuracy that I achieved using these paramater is: 99.376166 %, RMSE error is: 0.0112 and the R2 Score is: 0.99.
- Similarly, For Non-Linear Logistic Regression model, I designed the neural network with 7 input nodes as there were 7 different features for my data. It had 7 input layers and Two output node. I performed 5 fold CV on the model by passing this variables [[7,4,2], [7,5,2], [7,6,2], [7,7,2], [7,8,2]].As I said above her First value is the Input features(Input nodes), second parameter is the number of layers and last parameter is output node. The accuracy that I achieved using these paramater is: 79.54% and the AUC value is: 0.4611.

Discussion about results

I Implemented the Neural Network for the Nonlinear Regression and Nonlinear Logistic regression. The best optimum parameter for NonLinear Regression is 6 input nodes, 3 layers and 1 output node. Using this paramters I trained a neural netowrk to predict the pm2.5 parameter of the dataset. Whereas, the best optimal parameter for NonLinear Logistic Regression is 7 input node, 7 layers and 2 output nodes. Using this parameter I trained the NeuralNetLogReg for classifying the Revenue. The accuracy achieved in NonLinear Regression is 99.37%. The model was overfitted. It does not mean that by adding number of layers in the network it will increase the accuracy of the model. Whereas the accuracy achieved in the NonLinear Logistic regression is 79.54%.

Conclusions

In this assignment, I learned about how to select the best parameters for implementation of Neural Network from scratch. The method which suggests the best parameter for Neural Network is also implemented from scratch which is called Cross Validation. Adding to it, I also learned about how the complex network keeps track of minor information of the data and which is very important for the prediction purpose. Some of the challenges faced during the implementation of the network was how to infer the outcome of the data based on the prediction. As quantifying the results of the neural network is tough to prove it. Also, Implementing the K fold cross validation was challenging task but by refering to Reference[1] it gave some glimpse about how to design the code for the k fold cross validation. Overall it was very interesting implementing the neural network from scratch and it motivated me to do research in this field deeply.

References

1. Brownlee, Jason. “A Gentle Introduction to k-Fold Cross-Validation.” Machine Learning Mastery, 8 Aug. 2019, https://machinelearningmastery.com/k-fold-cross-validation/

(https://machinelearningmastery.com/k-fold-cross-validation/).
2. Liang, X., Zou, T., Guo, B., Li, S., Zhang, H., Zhang, S., Huang, H. and Chen, S. X. (2015). Assessing Beijing's PM2.5 pollution: severity, weather impact, APEC and winter heating. Proceedings of the Royal Society A, 471, 20150257.
3. “Seaborn.heatmap¶.” Seaborn.heatmap - Seaborn 0.9.0 Documentation, https://seaborn.pydata.org/generated/seaborn.heatmap.html (https://seaborn.pydata.org/generated/seaborn.heatmap.html).
4. “Seaborn.pairplot¶.” Seaborn.pairplot - Seaborn 0.9.0 Documentation, https://seaborn.pydata.org/generated/seaborn.pairplot.html (https://seaborn.pydata.org/generated/seaborn.pairplot.html).
5. Narenkmanoharan. “Narenkmanoharan/K-Cross-Fold-Validation.” GitHub, https://github.com/narenkmanoharan/K-Cross-fold-Validation/blob/master/k_fold.py (https://github.com/narenkmanoharan/K-Cross-fold-Validation/blob/master/k_fold.py).
6. “Notebook on Nbviewer.” Jupyter Notebook Viewer, https://nbviewer.jupyter.org/url/webpages.uncc.edu/mlee173/teach/itcs6156/notebooks/notes/Nc ML (https://nbviewer.jupyter.org/url/webpages.uncc.edu/mlee173/teach/itcs6156/notebooks/notes/N ML) Methodology.Sol.ipynb.
7. Sakar, C.O., Polat, S.O., Katircioglu, M. et al. Neural Comput & Applic (2018).
8. Sharma, Himanshu. “Activation Functions : Sigmoid, ReLU, Leaky ReLU and Softmax Basics for Neural Networks and Deep...” Medium, Medium, 9 Feb. 2019, https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e (https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e).

Extra Credit

Now you are testing various **activation functions** in this section. Use the best neural network structure and explore 3 different activation functions of your choice (one should be *tanh* that you used in the previous sections). You should use cross validation to discover the best model (with activation function). One extra credit is assigned when you finish the work completely.

Sigmoid

The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is especially used for models where we have to predict the probability as an output.Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice. The logistic sigmoid function can cause a neural network to get stuck at the training time.

In [160]:
```
class NeuralNet:
    """ neural network class for regression

        Parameters
        ----------
        nunits: list
            the number of inputs, hidden units, and outputs

        Methods
        -------
        set_hunit
            update/initiate weights

        pack
            pack multiple weights of each layer into one vector

        forward
            forward processing of neural network

        backward
            back-propagation of neural network

        train
            train the neural network

        use
            appply the trained network for prediction

        Attributes
        ----------
        _nLayers
            the number of hidden unit layers

        rho
            learning rate

        _W
            weights
        _weights
            weights in one dimension (_W is referencing _weight)

        stdX
            standardization class for data
        stdT
            standardization class for target

        Notes
        -----

    """

    def __init__(self, nunits):
        self._nLayers=len(nunits)-1
        self.rho = [1] * self._nLayers
        self._W = []
        wdims = []
        lenweights = 0
        for i in range(self._nLayers):
            nwr = nunits[i] + 1
            nwc = nunits[i+1]
            wdims.append((nwr, nwc))
            lenweights = lenweights + nwr * nwc

        self._weights = np.random.uniform(-0.1,0.1, lenweights)
        start = 0  # fixed index error 20110107
        for i in range(self._nLayers):
            end = start + wdims[i][0] * wdims[i][1]
            self._W.append(self._weights[start:end])
            self._W[i].resize(wdims[i])
            start = end

        self.stdX = None
        self.stdT = None
        self.stdTarget = True

    def add_ones(self, w):
        return np.hstack((np.ones((w.shape[0], 1)), w))

    def get_nlayers(self):
        return self._nLayers

    def set_hunit(self, w):
        for i in range(self._nLayers-1):
            if w[i].shape != self._W[i].shape:
                print("set_hunit: shapes do not match!")
                break
            else:
                self._W[i][:] = w[i][:]

    def pack(self, w):
        return np.hstack(map(np.ravel, w))

    def unpack(self, weights):
        self._weights[:] = weights[:]  # unpack

    def cp_weight(self):
        return copy(self._weights)

    def RBF(self, X, m=None,s=None):
        if m is None: m = np.mean(X)
        if s is None: s = 2 #np.std(X)
        r = 1. / (np.sqrt(2*np.pi)* s)
        return r * np.exp(-(X - m) ** 2 / (2 * s ** 2))

    def Sigma(self, X):
        return (1/(1+(np.exp(-(X)))))

    def dSigma(self, X):
        return ((np.exp(-x))/(1+(np.exp(-x)))^2)

    def forward(self,X):
        t = X
        Z = []
        for i in range(self._nLayers):
            Z.append(t)
            if i == self._nLayers - 1:
                t = np.dot(self.add_ones(t), self._W[i])
            else:
                t = self.Sigma(np.dot(self.add_ones(t), self._W[i]))

        return (t, Z)

    def backward(self, error, Z, T, lmb=0):
        delta = error
        N = T.size
        dws = []
        for i in range(self._nLayers - 1, -1, -1):
            rh = float(self.rho[i]) / N
            if i==0:
                lmbterm = 0
            else:
                lmbterm = lmb * np.vstack((np.zeros((1, self._W[i].shape[1])),
                            self._W[i][1:,]))

            #print(Z[i].T.shape)
            dws.insert(0,(-rh * np.dot(self.add_ones(Z[i]).T, delta) + lmbterm))
            if i != 0:
                delta = np.dot(delta, self._W[i][1:, :].T) * (np.exp(-Z[i])/(1+n
        return self.pack(dws)

    def _errorf(self, T, Y):
        return T - Y

    def _objectf(self, T, Y, wpenalty):
        return 0.5 * np.mean(np.square(T - Y)) + wpenalty

    def train(self, X, T, **params):

        verbose = params.pop('verbose', False)
        # training parameters
        _lambda = params.pop('Lambda', 0.)

        #parameters for scg
        niter = params.pop('niter', 1000)
        wprecision = params.pop('wprecision', 1e-10)
        fprecision = params.pop('fprecision', 1e-10)
        wtracep = params.pop('wtracep', False)
        ftracep = params.pop('ftracep', False)

        # optimization
        optim = params.pop('optim', 'scg')

        if self.stdX == None:
            explore = params.pop('explore', False)
            self.stdX = Standardizer(X, explore)
        Xs = self.stdX.standardize(X)
        if self.stdT == None and self.stdTarget:
            self.stdT = Standardizer(T)
            T = self.stdT.standardize(T)
        def gradientf(weights):
            self.unpack(weights)
            Y,Z = self.forward(Xs)

            error = self._errorf(T, Y)
            return self.backward(error, Z, T, _lambda)

        def optimtargetf(weights):
            """ optimization target function : MSE
            """
            self.unpack(weights)
            #self._weights[:] = weights[:]  # unpack
            Y,_ = self.forward(Xs)
            Wnb=np.array([])
            for i in range(self._nLayers):
                if len(Wnb)==0: Wnb=self._W[i][1:,].reshape(self._W[i].size-self
                else: Wnb = np.vstack((Wnb,self._W[i][1:,].reshape(self._W[i].si
            wpenalty = _lambda * np.dot(Wnb.flat ,Wnb.flat)
            return self._objectf(T, Y, wpenalty)

        if optim == 'scg':
            result = scg(self.cp_weight(), gradientf, optimtargetf,
                                            wPrecision=wprecision, fPrecision=fpreci
                                            nIterations=niter,
                                            wtracep=wtracep, ftracep=ftracep,
                                            verbose=False)
            self.unpack(result['w'][:])
            self.f = result['f']
        elif optim == 'steepest':
            result = steepest(self.cp_weight(), gradientf, optimtargetf,
                                    nIterations=niter,
                                    xPrecision=wprecision, fPrecision=fprecision,
                                    xtracep=wtracep, ftracep=ftracep )
            self.unpack(result['w'][:])
        if ftracep:
            self.ftrace = result['ftrace']
        if 'reason' in result.keys() and verbose:
            print(result['reason'])

        return result

    def use(self, X, retZ=False):
        if self.stdX:
            Xs = self.stdX.standardize(X)
        else:
            Xs = X
        Y, Z = self.forward(Xs)
        if self.stdT is not None:
            Y = self.stdT.unstandardize(Y)
        if retZ:
            return Y, Z
        return Y
```

In [161]:
```
def k_fold_1(s):
    k_val_1 = len(s) / 5.0
    partioned_data_1 = []
    count_1 = 0.0
    while count_1 < len(s):
        partioned_data_1.append(s[int(count_1):int(count_1 + k_val_1)])
        count_1 += k_val_1
    return partioned_data_1

final_list=[]
best_paramlist=[]
def cross_validate(X, T, parameters):
    feature_partition_1 = k_fold_1(X)
    target_partition_1 = k_fold_1(T)
    for i in range(5):
        print("Iteration: ",i)
        Xtest = feature_partition_1[i]
        Ttest = target_partition_1[i]

        history_rmse=[]
        params=[]
        for k in range(5):
            print("k fold: ",k)
            if i == k: continue

            Xval = feature_partition_1[k]
            Tval = target_partition_1[k]

            Xtrain = feature_partition_1[not i and not k]
            Ttrain = target_partition_1[not i and not k]

            flist=[]
            paramlist=[]
            for param in parameters:

                model = NeuralNetLogReg(param)
                model.train(Xtrain, Ttrain)
                prob,pred = model.use(Xval)
                Tval1=Tval[:,0]
                Tval1=np.where(Tval1==0,1,0)
                valAcc = f1_score(Tval1, pred)
                flist.append(valAcc)
                paramlist.append(param)

            print(max(flist))
            print(paramlist[np.argmax(flist)])
            history_rmse.append(max(flist))
            params.append(paramlist[np.argmax(flist)])


        bestParam = params[np.argmax(history_rmse)]
        bestModel = NeuralNetLogReg(bestParam)

        X_train1 = feature_partition_1[not i]
        T_train1 = target_partition_1[not i]
        bestModel.train(X_train1, T_train1)
        prob1,finalPred = bestModel.use(X_test1)
        Ttest1=Ttest[:,0]
        Ttest1=np.where(Ttest1==0,1,0)
        myf1 = f1_score(Ttest1, finalPred)
        final_list.append(myf1)
        best_paramlist.append(bestParam)

    return final_list,best_paramlist

models = [[7,4,2], [7,5,2],[7,6,2], [7,7,2], [7,8,2]]
bestAccuracy,bestParam = (cross_validate(np.array(df_classification_new),np.array
```

```
Iteration:  0
k fold:  0
k fold:  1


C:\Users\jeetj\3D Objects\Machine Learning\Assignment 3\nn.py:113: FutureWarnin
g: arrays to stack must be passed as a "sequence" type such as list or tuple. S
upport for non-sequence iterables such as generators is deprecated as of NumPy
1.16 and will raise an error in the future.
  return np.hstack(map(np.ravel, w))
C:\Users\jeetj\Miniconda3\envs\senses\lib\site-packages\ipykernel_launcher.py:1
4: RuntimeWarning: invalid value encountered in true_divide

C:\Users\jeetj\Miniconda3\envs\senses\lib\site-packages\sklearn\metrics\classif
ication.py:1437: UndefinedMetricWarning: F-score is ill-defined and being set t
o 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)


0.26865671641791045
[7, 5, 2]
k fold:  2


C:\Users\jeetj\Miniconda3\envs\senses\lib\site-packages\ipykernel_launcher.py:3
5: RuntimeWarning: divide by zero encountered in log

0.10105649977032613
[7, 4, 2]
k fold:  3
0.8849197377345692
[7, 4, 2]
k fold:  4
0.8634686346863469
[7, 4, 2]
Iteration:  1
k fold:  0
0.9497444633730835
[7, 5, 2]
k fold:  1
k fold:  2
0.9100552486187845
[7, 4, 2]
k fold:  3
0.8849197377345692
```

```
[7, 5, 2]
k fold:  4
0.8886885984677783
[7, 6, 2]
Iteration:  2
k fold:  0
0.8986154533273784
[7, 5, 2]
k fold:  1
0.94389721627409
[7, 8, 2]
k fold:  2
k fold:  3
0.8484565014031805
[7, 8, 2]
k fold:  4
0.8886885984677783
[7, 6, 2]
Iteration:  3
k fold:  0
0.9497444633730835
[7, 4, 2]
k fold:  1
0.9343128781331028
[7, 4, 2]
k fold:  2
0.885817852288174
[7, 8, 2]
k fold:  3
k fold:  4
0.8886885984677783
[7, 6, 2]
Iteration:  4
k fold:  0
0.9497444633730835
[7, 8, 2]
k fold:  1
```

```
C:\Users\jeetj\Miniconda3\envs\senses\lib\site-packages\ipykernel_launcher.py:1
3: RuntimeWarning: overflow encountered in exp
  del sys.path[0]
```

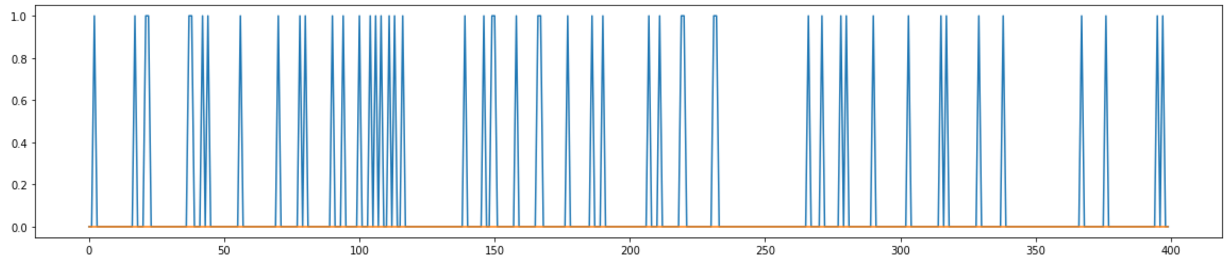
```
0.9324675324675326
[7, 8, 2]
k fold:  2
0.9100552486187845
[7, 8, 2]
k fold:  3
0.8849197377345692
[7, 6, 2]
k fold:  4
```

In [162]:
```
X_train1,X_test1,y_train1,y_test1 = model_selection.train_test_split(np.array(df_
```

In [163]:
```
nn = NeuralNetLogReg([7,3,2])
nn.train(X_train1, y_train1, niter=1000)
probability,label = nn.use(X_test1)
```

In [164]:
```
plt.figure(figsize = (20,4))
plt.plot(y_test1[:400])
plt.plot(label[:400])
cm = confusion_matrix(y_test1,label)
print("Confusion Matrix: \n", cm)

cr = classification_report(y_test1,label)
print("\nClassification Report: \n", cr)
print("Testing Accuracy: ", 100 - np.mean(np.abs(y_test1 - label)) * 100, "%")
```

```
Confusion Matrix:
 [[2100    0]
 [ 366    0]]

Classification Report:
               precision    recall  f1-score   support

           0       0.85      1.00      0.92      2100
           1       0.00      0.00      0.00       366

    accuracy                           0.85      2466
   macro avg       0.43      0.50      0.46      2466
weighted avg       0.73      0.85      0.78      2466

Testing Accuracy:  85.15815085158151 %

C:\Users\jeetj\Miniconda3\envs\senses\lib\site-packages\sklearn\metrics\classif
ication.py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined
and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```



In [165]:
```
rmse = np.sqrt(mean_squared_error(y_test1, label))
print("RMSE Error: ",rmse)
```

```
RMSE Error:  0.3852512056881651
```

Tanh

Tanh is also like logistic sigmoid but better. The range of the tanh function is from (-1 to 1). tanh is also sigmoidal. The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph.

In [166]:
```
class NeuralNet:
    """ neural network class for regression
        
        Parameters
        ----------
        nunits: list
            the number of inputs, hidden units, and outputs

        Methods
        -------
        set_hunit    
            update/initiate weights

        pack
            pack multiple weights of each layer into one vector

        forward
            forward processing of neural network

        backward
            back-propagation of neural network

        train
            train the neural network

        use
            appply the trained network for prediction

        Attributes
        ----------
        _nLayers
            the number of hidden unit layers

        rho
            learning rate

        _W
            weights
        _weights
            weights in one dimension (_W is referencing _weight)

        stdX
            standardization class for data
        stdT
            standardization class for target

        Notes
        -----
        
    """

    def __init__(self, nunits):
        self._nLayers=len(nunits)-1
        self.rho = [1] * self._nLayers
        self._W = []
        wdims = []
        lenweights = 0
        for i in range(self._nLayers):
            nwr = nunits[i] + 1
            nwc = nunits[i+1]
            wdims.append((nwr, nwc))
            lenweights = lenweights + nwr * nwc

        self._weights = np.random.uniform(-0.1,0.1, lenweights)
        start = 0  # fixed index error 20110107
        for i in range(self._nLayers):
            end = start + wdims[i][0] * wdims[i][1]
            self._W.append(self._weights[start:end])
            self._W[i].resize(wdims[i])
            start = end

        self.stdX = None
        self.stdT = None
        self.stdTarget = True

    def add_ones(self, w):
        return np.hstack((np.ones((w.shape[0], 1)), w))

    def get_nlayers(self):
        return self._nLayers

    def set_hunit(self, w):
        for i in range(self._nLayers-1):
            if w[i].shape != self._W[i].shape:
                print("set_hunit: shapes do not match!")
                break
            else:
                self._W[i][:] = w[i][:]

    def pack(self, w):
        return np.hstack(map(np.ravel, w))

    def unpack(self, weights):
        self._weights[:] = weights[:]  # unpack

    def cp_weight(self):
        return copy(self._weights)

    def RBF(self, X, m=None,s=None):
        if m is None: m = np.mean(X)
        if s is None: s = 2 #np.std(X)
        r = 1. / (np.sqrt(2*np.pi)* s)
        return r * np.exp(-(X - m) ** 2 / (2 * s ** 2))

    def Sigma(self, X):
        return (1/(1+(np.exp(-(X)))))

    def forward(self,X):
        t = X
        Z = []

        for i in range(self._nLayers):
            Z.append(t)
```

```
                if i == self._nLayers - 1:
                    t = np.dot(self.add_ones(t), self._W[i])
                else:
                    t = np.tanh(np.dot(self.add_ones(t), self._W[i]))

            return (t, Z)

        def backward(self, error, Z, T, lmb=0):
            delta = error
            N = T.size
            dws = []
            for i in range(self._nLayers - 1, -1, -1):
                rh = float(self.rho[i]) / N
                if i==0:
                    lmbterm = 0
                else:
                    lmbterm = lmb * np.vstack((np.zeros((1, self._W[i].shape[1])),
                                  self._W[i][1:,]))

                #print(Z[i].T.shape)
                dws.insert(0,(-rh * np.dot(self.add_ones(Z[i]).T, delta) + lmbterm))
                if i != 0:
                    delta = np.dot(delta, self._W[i][1:, :].T) * (1 - Z[i]**2)
            return self.pack(dws)

        def _errorf(self, T, Y):
            return T - Y

        def _objectf(self, T, Y, wpenalty):
            return 0.5 * np.mean(np.square(T - Y)) + wpenalty

        def train(self, X, T, **params):

            verbose = params.pop('verbose', False)
            # training parameters
            _lambda = params.pop('Lambda', 0.)

            #parameters for scg
            niter = params.pop('niter', 1000)
            wprecision = params.pop('wprecision', 1e-10)
            fprecision = params.pop('fprecision', 1e-10)
            wtracep = params.pop('wtracep', False)
            ftracep = params.pop('ftracep', False)

            # optimization
            optim = params.pop('optim', 'scg')

            if self.stdX == None:
                explore = params.pop('explore', False)
                self.stdX = Standardizer(X, explore)
            Xs = self.stdX.standardize(X)
            if self.stdT == None and self.stdTarget:
                self.stdT = Standardizer(T)
                T = self.stdT.standardize(T)

            def gradientf(weights):
                self.unpack(weights)
                Y,Z = self.forward(Xs)

                error = self._errorf(T, Y)
                return self.backward(error, Z, T, _lambda)

            def optimtargetf(weights):
                """ optimization target function : MSE
                """
                self.unpack(weights)
                #self._weights[:] = weights[:]  # unpack
                Y,_ = self.forward(Xs)
                Wnb=np.array([])
                for i in range(self._nLayers):
                    if len(Wnb)==0: Wnb=self._W[i][1:,].reshape(self._W[i].size-self
                    else: Wnb = np.vstack((Wnb,self._W[i][1:,].reshape(self._W[i].si
                wpenalty = _lambda * np.dot(Wnb.flat ,Wnb.flat)
                return self._objectf(T, Y, wpenalty)

            if optim == 'scg':
                result = scg(self.cp_weight(), gradientf, optimtargetf,
                                            wPrecision=wprecision, fPrecision=fpreci
                                            nIterations=niter,
                                            wtracep=wtracep, ftracep=ftracep,
                                            verbose=False)
                self.unpack(result['w'][:])
                self.f = result['f']
            elif optim == 'steepest':
                result = steepest(self.cp_weight(), gradientf, optimtargetf,
                                    nIterations=niter,
                                    xPrecision=wprecision, fPrecision=fprecision,
                                    xtracep=wtracep, ftracep=ftracep )
                self.unpack(result['w'][:])
            if ftracep:
                self.ftrace = result['ftrace']
            if 'reason' in result.keys() and verbose:
                print(result['reason'])

            return result

        def use(self, X, retZ=False):
            if self.stdX:
                Xs = self.stdX.standardize(X)
            else:
                Xs = X
            Y, Z = self.forward(Xs)
            if self.stdT is not None:
                Y = self.stdT.unstandardize(Y)
            if retZ:
                return Y, Z
            return Y
```

In [167]:
```
def k_fold_1(s):
    k_val_1 = len(s) / 5.0
    partioned_data_1 = []
    count_1 = 0.0
    while count_1 < len(s):
        partioned_data_1.append(s[int(count_1):int(count_1 + k_val_1)])
        count_1 += k_val_1
    return partioned_data_1

final_list=[]
best_paramlist=[]
def cross_validate(X, T, parameters):
    feature_partition_1 = k_fold_1(X)
    target_partition_1 = k_fold_1(T)
    for i in range(5):
        print("Iteration: ",i)
        Xtest = feature_partition_1[i]
        Ttest = target_partition_1[i]

        history_rmse=[]
        params=[]
        for k in range(5):
            print("k fold: ",k)
            if i == k: continue

            Xval = feature_partition_1[k]
            Tval = target_partition_1[k]

            Xtrain = feature_partition_1[not i and not k]
            Ttrain = target_partition_1[not i and not k]

            flist=[]
            paramlist=[]
            for param in parameters:

                model = NeuralNetLogReg(param)
                model.train(Xtrain, Ttrain)
                prob,pred = model.use(Xval)
                Tval1=Tval[:,0]
                Tval1=np.where(Tval1==0,1,0)
                valAcc = f1_score(Tval1, pred)
                flist.append(valAcc)
                paramlist.append(param)

            print(max(flist))
            print(paramlist[np.argmax(flist)])
            history_rmse.append(max(flist))
            params.append(paramlist[np.argmax(flist)])


        bestParam = params[np.argmax(history_rmse)]
        bestModel = NeuralNetLogReg(bestParam)

        X_train1 = feature_partition_1[not i]
        T_train1 = target_partition_1[not i]
        bestModel.train(X_train1, T_train1)
        prob1,finalPred = bestModel.use(X_test1)
        Ttest1=Ttest[:,0]
        Ttest1=np.where(Ttest1==0,1,0)
        myf1 = f1_score(Ttest1, finalPred)
        final_list.append(myf1)
        best_paramlist.append(bestParam)

    return final_list,best_paramlist

models = [[7,4,2], [7,5,2],[7,6,2], [7,7,2], [7,8,2]]
bestAccuracy,bestParam = (cross_validate(np.array(df_classification_new),np.array
```

```
Iteration:  0
k fold:  0
k fold:  1

C:\Users\jeetj\3D Objects\Machine Learning\Assignment 3\nn.py:113: FutureWarnin
g: arrays to stack must be passed as a "sequence" type such as list or tuple. S
upport for non-sequence iterables such as generators is deprecated as of NumPy
1.16 and will raise an error in the future.
  return np.hstack(map(np.ravel, w))
C:\Users\jeetj\Miniconda3\envs\senses\lib\site-packages\sklearn\metrics\classif
ication.py:1437: UndefinedMetricWarning: F-score is ill-defined and being set t
o 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\jeetj\Miniconda3\envs\senses\lib\site-packages\ipykernel_launcher.py:1
4: RuntimeWarning: invalid value encountered in true_divide

C:\Users\jeetj\Miniconda3\envs\senses\lib\site-packages\ipykernel_launcher.py:1
3: RuntimeWarning: overflow encountered in exp
  del sys.path[0]
```

```
0.94389721627409
[7, 6, 2]
k fold:  2
0.9100552486187845
[7, 4, 2]
k fold:  3
0.8849197377345692
[7, 4, 2]
k fold:  4
0.8830992297236068
[7, 6, 2]
Iteration:  1
k fold:  0
0.9497444633730835
[7, 4, 2]
k fold:  1
k fold:  2
0.9076002658985154
[7, 8, 2]
k fold:  3
0.8849197377345692
[7, 5, 2]
k fold:  4
```

```
0.8886885984677783
[7, 5, 2]
Iteration:  2
k fold:  0
0.9497444633730835
[7, 5, 2]
k fold:  1
0.94389721627409
[7, 6, 2]
k fold:  2
k fold:  3
0.8849197377345692
[7, 8, 2]
k fold:  4
0.8890892696122633
[7, 6, 2]
Iteration:  3
k fold:  0
0.9497444633730835
[7, 6, 2]
k fold:  1
0.9432183415470323
[7, 8, 2]
k fold:  2
0.9042435014441236
[7, 6, 2]
k fold:  3
k fold:  4
```

```
C:\Users\jeetj\Miniconda3\envs\senses\lib\site-packages\ipykernel_launcher.py:3
5: RuntimeWarning: divide by zero encountered in log
```

```
0.8886885984677783
[7, 5, 2]
Iteration:  4
k fold:  0
0.9497444633730835
[7, 6, 2]
k fold:  1
0.94389721627409
[7, 7, 2]
k fold:  2
0.9100552486187845
[7, 8, 2]
k fold:  3
0.8849197377345692
[7, 5, 2]
k fold:  4
```

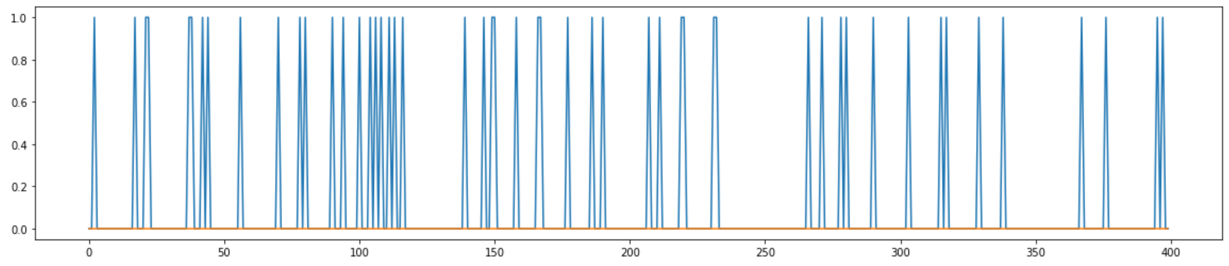
In [168]:
```
X_train1,X_test1,y_train1,y_test1 = model_selection.train_test_split(np.array(df_
```

In [169]:
```
nn = NeuralNetLogReg([7,3,2])
nn.train(X_train1, y_train1, niter=1000)
probability,label = nn.use(X_test1)
```

```
C:\Users\jeetj\Miniconda3\envs\senses\lib\site-packages\ipykernel_launcher.py:1
4: RuntimeWarning: invalid value encountered in true_divide
```

In [170]:
```
plt.figure(figsize = (20,4))
plt.plot(y_test1[:400])
plt.plot(label[:400])
cm = confusion_matrix(y_test1,label)
print("Confusion Matrix: \n", cm)

cr = classification_report(y_test1,label)
print("\nClassification Report: \n", cr)
print("Testing Accuracy: ", 100 - np.mean(np.abs(y_test1 - label)) * 100, "%")
```

```
Confusion Matrix:
 [[2100    0]
 [ 366    0]]

Classification Report:
               precision    recall  f1-score   support

           0       0.85      1.00      0.92      2100
           1       0.00      0.00      0.00       366

    accuracy                           0.85      2466
   macro avg       0.43      0.50      0.46      2466
weighted avg       0.73      0.85      0.78      2466

Testing Accuracy:  85.15815085158151 %

C:\Users\jeetj\Miniconda3\envs\senses\lib\site-packages\sklearn\metrics\classif
ication.py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined
and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```



In [171]:
```
rmse = np.sqrt(mean_squared_error(y_test1, label))
print("RMSE Error: ",rmse)
```

```
RMSE Error:  0.3852512056881651
```

ReLu

A Rectified Linear Unit (A unit employing the rectifier is also called a rectified linear unit ReLU) has output 0 if the input is less than 0, and raw output otherwise. That is, if the input is greater than 0, the output is equal to the input. The operation of ReLU is closer to the way our biological neurons work. ReLU is non-linear and has the advantage of not having any backpropagation errors unlike the sigmoid function, also for larger Neural Networks, the speed of building models based off on ReLU is very fast opposed to using Sigmoids.[8]

In [172]:
```
class NeuralNet:
    """ neural network class for regression

        Parameters
        ----------
        nunits: list
            the number of inputs, hidden units, and outputs

        Methods
        -------
        set_hunit
            update/initiate weights

        pack
            pack multiple weights of each layer into one vector

        forward
            forward processing of neural network

        backward
            back-propagation of neural network

        train
            train the neural network

        use
            appply the trained network for prediction

        Attributes
        ----------
        _nLayers
            the number of hidden unit layers

        rho
            learning rate

        _W
            weights
        _weights
            weights in one dimension (_W is referencing _weight)

        stdX
            standardization class for data
        stdT
            standardization class for target

        Notes
        -----

    """

    def __init__(self, nunits):
        self._nLayers=len(nunits)-1
        self.rho = [1] * self._nLayers
        self._W = []
        wdims = []
        lenweights = 0
        for i in range(self._nLayers):
            nwr = nunits[i] + 1
            nwc = nunits[i+1]
            wdims.append((nwr, nwc))
            lenweights = lenweights + nwr * nwc

        self._weights = np.random.uniform(-0.1,0.1, lenweights)
        start = 0  # fixed index error 20110107
        for i in range(self._nLayers):
            end = start + wdims[i][0] * wdims[i][1]
            self._W.append(self._weights[start:end])
            self._W[i].resize(wdims[i])
            start = end

        self.stdX = None
        self.stdT = None
        self.stdTarget = True

    def add_ones(self, w):
        return np.hstack((np.ones((w.shape[0], 1)), w))

    def get_nlayers(self):
        return self._nLayers

    def set_hunit(self, w):
        for i in range(self._nLayers-1):
            if w[i].shape != self._W[i].shape:
                print("set_hunit: shapes do not match!")
                break
            else:
                self._W[i][:] = w[i][:]

    def pack(self, w):
        return np.hstack(map(np.ravel, w))

    def unpack(self, weights):
        self._weights[:] = weights[:]  # unpack

    def cp_weight(self):
        return copy(self._weights)

    def RBF(self, X, m=None,s=None):
        if m is None: m = np.mean(X)
        if s is None: s = 2 #np.std(X)
        r = 1. / (np.sqrt(2*np.pi)* s)
        return r * np.exp(-(X - m) ** 2 / (2 * s ** 2))

    def Sigma(self, X):
        return (1/(1+(np.exp(-(X)))))

    def dSigma(self, X):
        return ((np.exp(-x))/(1+(np.exp(-x)))^2)

    def ReLU(self, x):
        return x * (x > 0)
    def dReLU(self, x):
        return 1. * (x > 0)

    def forward(self,X):
        t = X
        Z = []

        for i in range(self._nLayers):
            Z.append(t)
            if i == self._nLayers - 1:
                t = np.dot(self.add_ones(t), self._W[i])
            else:
                t = self.ReLU(np.dot(self.add_ones(t), self._W[i]))

        return (t, Z)

    def backward(self, error, Z, T, lmb=0):
        delta = error
        N = T.size
        dws = []
        for i in range(self._nLayers - 1, -1, -1):
            rh = float(self.rho[i]) / N
            if i==0:
                lmbterm = 0
            else:
                lmbterm = lmb * np.vstack((np.zeros((1, self._W[i].shape[1])),
                            self._W[i][1:,]))

            #print(Z[i].T.shape)
            dws.insert(0,(-rh * np.dot(self.add_ones(Z[i]).T, delta) + lmbterm))
            if i != 0:
                delta = np.dot(delta, self._W[i][1:, :].T) * (1. * Z[i])
        return self.pack(dws)

    def _errorf(self, T, Y):
        return T - Y

    def _objectf(self, T, Y, wpenalty):
        return 0.5 * np.mean(np.square(T - Y)) + wpenalty

    def train(self, X, T, **params):

        verbose = params.pop('verbose', False)
        # training parameters
        _lambda = params.pop('Lambda', 0.)

        #parameters for scg
        niter = params.pop('niter', 1000)
        wprecision = params.pop('wprecision', 1e-10)
        fprecision = params.pop('fprecision', 1e-10)
        wtracep = params.pop('wtracep', False)
        ftracep = params.pop('ftracep', False)

        # optimization
        optim = params.pop('optim', 'scg')

        if self.stdX == None:
            explore = params.pop('explore', False)
            self.stdX = Standardizer(X, explore)
        Xs = self.stdX.standardize(X)
        if self.stdT == None and self.stdTarget:
            self.stdT = Standardizer(T)
            T = self.stdT.standardize(T)

        def gradientf(weights):
            self.unpack(weights)
            Y,Z = self.forward(Xs)

            error = self._errorf(T, Y)
            return self.backward(error, Z, T, _lambda)

        def optimtargetf(weights):
            """ optimization target function : MSE
            """
            self.unpack(weights)
            #self._weights[:] = weights[:]  # unpack
            Y,_ = self.forward(Xs)
            Wnb=np.array([])
            for i in range(self._nLayers):
                if len(Wnb)==0: Wnb=self._W[i][1:,].reshape(self._W[i].size-self
                else: Wnb = np.vstack((Wnb,self._W[i][1:,].reshape(self._W[i].si
            wpenalty = _lambda * np.dot(Wnb.flat ,Wnb.flat)
            return self._objectf(T, Y, wpenalty)

        if optim == 'scg':
            result = scg(self.cp_weight(), gradientf, optimtargetf,
                                        wPrecision=wprecision, fPrecision=fpreci
                                        nIterations=niter,
                                        wtracep=wtracep, ftracep=ftracep,
                                        verbose=False)
            self.unpack(result['w'][:])
            self.f = result['f']
        elif optim == 'steepest':
            result = steepest(self.cp_weight(), gradientf, optimtargetf,
                                nIterations=niter,
                                xPrecision=wprecision, fPrecision=fprecision,
                                xtracep=wtracep, ftracep=ftracep )
            self.unpack(result['w'][:])
        if ftracep:
            self.ftrace = result['ftrace']
        if 'reason' in result.keys() and verbose:
            print(result['reason'])

        return result

    def use(self, X, retZ=False):
        if self.stdX:
            Xs = self.stdX.standardize(X)
        else:
            Xs = X
        Y, Z = self.forward(Xs)
        if self.stdT is not None:
            Y = self.stdT.unstandardize(Y)
        if retZ:
            return Y, Z
        return Y
```

In [173]:
```
def k_fold_1(s):
    k_val_1 = len(s) / 5.0
    partioned_data_1 = []
    count_1 = 0.0
    while count_1 < len(s):
        partioned_data_1.append(s[int(count_1):int(count_1 + k_val_1)])
        count_1 += k_val_1
    return partioned_data_1

final_list=[]
best_paramlist=[]
def cross_validate(X, T, parameters):
    feature_partition_1 = k_fold_1(X)
    target_partition_1 = k_fold_1(T)
    for i in range(5):
        print("Iteration: ",i)
        Xtest = feature_partition_1[i]
        Ttest = target_partition_1[i]

        history_rmse=[]
        params=[]
        for k in range(5):
            print("k fold: ",k)
            if i == k: continue

            Xval = feature_partition_1[k]
            Tval = target_partition_1[k]

            Xtrain = feature_partition_1[not i and not k]
            Ttrain = target_partition_1[not i and not k]

            flist=[]
            paramlist=[]
            for param in parameters:

                model = NeuralNetLogReg(param)
                model.train(Xtrain, Ttrain)
                prob,pred = model.use(Xval)
                Tval1=Tval[:,0]
                Tval1=np.where(Tval1==0,1,0)
                valAcc = f1_score(Tval1, pred)
                flist.append(valAcc)
                paramlist.append(param)

            print(max(flist))
            print(paramlist[np.argmax(flist)])
            history_rmse.append(max(flist))
            params.append(paramlist[np.argmax(flist)])


        bestParam = params[np.argmax(history_rmse)]
        bestModel = NeuralNetLogReg(bestParam)

        X_train1 = feature_partition_1[not i]
        T_train1 = target_partition_1[not i]
        bestModel.train(X_train1, T_train1)
        prob1,finalPred = bestModel.use(X_test1)
        Ttest1=Ttest[:,0]
        Ttest1=np.where(Ttest1==0,1,0)
        myf1 = f1_score(Ttest1, finalPred)
        final_list.append(myf1)
        best_paramlist.append(bestParam)

    return final_list,best_paramlist

models = [[7,4,2], [7,5,2],[7,6,2], [7,7,2], [7,8,2]]
bestAccuracy,bestParam = (cross_validate(np.array(df_classification_new),np.array
```

```
Iteration:  0
k fold:  0
k fold:  1

C:\Users\jeetj\3D Objects\Machine Learning\Assignment 3\nn.py:113: FutureWarnin
g: arrays to stack must be passed as a "sequence" type such as list or tuple. S
upport for non-sequence iterables such as generators is deprecated as of NumPy
1.16 and will raise an error in the future.
  return np.hstack(map(np.ravel, w))
C:\Users\jeetj\Miniconda3\envs\senses\lib\site-packages\ipykernel_launcher.py:1
4: RuntimeWarning: invalid value encountered in true_divide

C:\Users\jeetj\Miniconda3\envs\senses\lib\site-packages\sklearn\metrics\classif
ication.py:1437: UndefinedMetricWarning: F-score is ill-defined and being set t
o 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)

0.94389721627409
[7, 4, 2]
k fold:  2
0.8952210006730985
[7, 4, 2]
k fold:  3
0.8849197377345692
[7, 4, 2]
k fold:  4
0.8886885984677783
[7, 5, 2]
Iteration:  1
k fold:  0
0.9497444633730835
[7, 5, 2]
k fold:  1
k fold:  2
0.9100552486187845
[7, 4, 2]
k fold:  3
0.8849197377345692
[7, 7, 2]
k fold:  4
0.8886885984677783
[7, 4, 2]
Iteration:  2
k fold:  0
0.9497444633730835
```

```
[7, 4, 2]
k fold:  1

C:\Users\jeetj\Miniconda3\envs\senses\lib\site-packages\ipykernel_launcher.py:3
5: RuntimeWarning: divide by zero encountered in log

0.94389721627409
[7, 6, 2]
k fold:  2
k fold:  3
0.8788360991134349
[7, 5, 2]
k fold:  4
0.8886885984677783
[7, 8, 2]
Iteration:  3
k fold:  0
0.9477277576274803
[7, 6, 2]
k fold:  1
0.94389721627409
[7, 4, 2]
k fold:  2
0.9100552486187845
[7, 5, 2]
k fold:  3
k fold:  4
0.8886885984677783
[7, 5, 2]
Iteration:  4
k fold:  0
0.9497444633730835
[7, 4, 2]
k fold:  1
0.94389721627409
[7, 8, 2]
k fold:  2
0.9100552486187845
[7, 4, 2]
k fold:  3
0.8849197377345692
[7, 4, 2]
k fold:  4
```

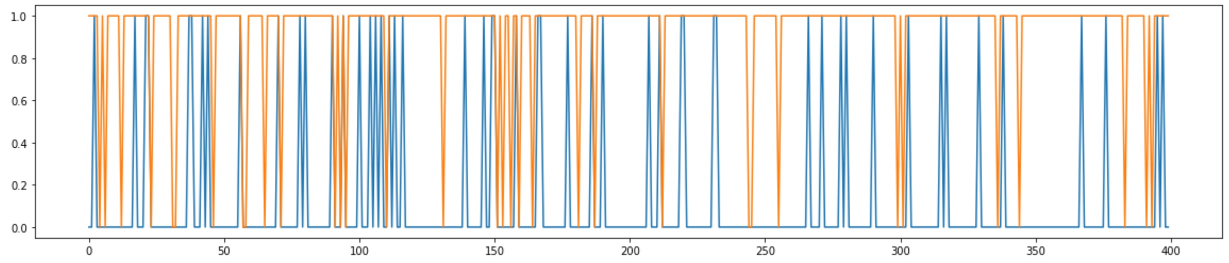
In [174]:
```
X_train1,X_test1,y_train1,y_test1 = model_selection.train_test_split(np.array(df
```

In [175]:
```
nn = NeuralNetLogReg([7,3,2])
nn.train(X_train1, y_train1, niter=1000)
probability,label = nn.use(X_test1)
```

```
C:\Users\jeetj\Miniconda3\envs\senses\lib\site-packages\ipykernel_launcher.py:1
4: RuntimeWarning: invalid value encountered in true_divide
```

In [176]:
```
plt.figure(figsize = (20,4))
plt.plot(y_test1[:400])
plt.plot(label[:400])
cm = confusion_matrix(y_test1,label)
print("Confusion Matrix: \n", cm)
cr = classification_report(y_test1,label)
print("\nClassification Report: \n", cr)
print("Testing Accuracy: ", 100 - np.mean(np.abs(y_test1 - label)) * 100, "%")
```

```
Confusion Matrix: 
 [[ 184 1916]
 [   5  361]]

Classification Report: 
               precision    recall  f1-score   support

           0       0.97      0.09      0.16      2100
           1       0.16      0.99      0.27       366

    accuracy                           0.22      2466
   macro avg       0.57      0.54      0.22      2466
weighted avg       0.85      0.22      0.18      2466

Testing Accuracy:  20.231054753405445 %
```



In [177]:
```
rmse = np.sqrt(mean_squared_error(y_test1, label))
print("RMSE Error: ",rmse)
```

```
RMSE Error:  0.8826065503892112
```

Conlusion

I have trained the model using three different activation function such as Sigmoid, Tanh and Relu. The training was performed by selecting the best parameters from 5 fold cross validation. I have plotted the graph of the actual and predicted value. Also, calculated the RMSE error for all the activation functions. RMSE error of sigmoid and Tanh is nearly 0.35 where as the RMSE error for Relu is 0.88. As we know that in the hidden layers nonlinear activation function should be used to make the network Non-Linear and to solve the vanishing gradients problem.

How to start?

- Download [a3.tgz (http://webpages.uncc.edu/mlee173/teach/itcs6156/notebooks/assign/a3.tgz)](http://webpages.uncc.edu/mlee173/teach/itcs6156/notebooks/assign/a3.tgz).
- Unzip the python files to current working directory.
- Import the necessary classes or functions including NeuralNet.

Grading

DO NOT forget to submit your data! Your notebook is supposed to run fine after running your codes.

** Note: this is a WRITING assignment. Proper writing is REQUIRED. Comments are not considered as writing. **

points		description
5	Overview	states the objective and the appraoch
10	Data	
	2	Includes description of your data
	3	Plots to visualize data
	5	Reading and analyzing the plots
40	Methods	
	10	Summary of CV & correctness of implementation
	5	Summary of nonlinear regression
	5	Explanation of codes
	5	Summary of nonlinear logistic regression
	5	Explanation of codes
	10	Examination of correct implementation (NonlinearLogReg) with toy data.
40	Results	Your Data
	10	Presentaion of CV results
	10	Discussions about parameter/network structure choice
	10	plots for results
	10	Discussion about the prediction results. Try to analyze what nonlinear regression model learned.
5	Conclusions	