EVALUATING THE EFFICACY OF AUTISM SCREENING TEST

STA 5939

Sam Coplin
Jessica Richardson

Abstract

Autism is a serious developmental disorder that impairs the ability of an individual to communicate and interact. Currently, autism affects over 3.5 million people in the United States alone and according to the CDC, those rates have been increasing alarmingly quickly. The current diagnostic procedure is to perform a CDE (Comprehensive Diagnostic Evaluation), which may include vision screening, genetic testing, neurological testing, and more. All of these tests are very expensive, potentially prohibiting underprivileged communities from being able to be tested and receive a diagnosis. In an effort to minimize unnecessary time and money spent on these procedures doctors often utilize a screening test beforehand to see if additional testing is warranted.

In this project, we set out to try to determine if an individual has autism based on a 10-question screening test in conjunction with demographic information that the medical professional administering the screening test would already have. We wanted to see if we could improve the prediction accuracy of preliminary basic screening tests and diminish the false-positive and false-negative rates through machine learning techniques. We found that through utilizing this additional information as well as more complex models, the prediction accuracy could be improved from about 72% to 85%. Furthermore, the false positive rates and false negative rates could be decreased by about 10% each. Through this project we demonstrate that future research should be pursued in this area because if more demographic information is able to be retrieved from the respondent, the accuracy of the test can be greatly improved upon, thus helping more individuals receive the correct diagnosis.

Data Description

The data we utilized in this project was downloaded from the UC Irvine (UCI) Machine Learning Repository. The data contained information about users gathered from an autism screening app completed by adults. Demographic information was included as well as scores for each question on the National Institute for Health Research's AQ-10 Screening Test. The individuals completing this quiz on the app already knew if they had autism or not - this information was gathered solely for research purposes. A full list of features is included in the table below.

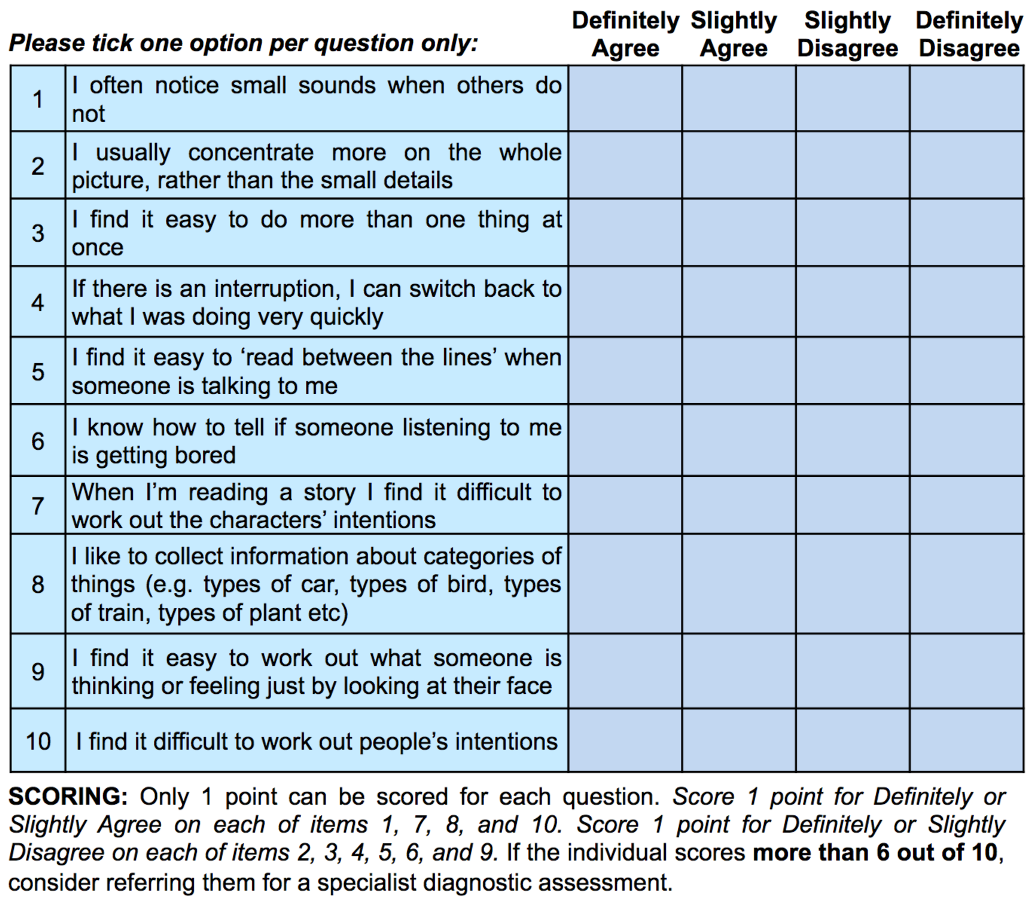
Table 1: Features and their descriptions

Attribute	**Type**	**Description**
Age	Number	Age in years
Gender	String	Male or Female
Ethnicity	String	List of common ethnicities in text format
Born with jaundice	Boolean (yes or no)	Whether the case was born with jaundice
Family member with PDD	Boolean (yes or no)	Whether any immediate family member has a PDD
Who is completing the test	String	Parent, self, caregiver, medical staff, clinician ,etc.
Country of residence	String	List of countries in text format
Used the screening app before	Boolean (yes or no)	Whether the user has used a screening app
Screening Method Type	Integer (0,1,2,3)	The type of screening methods chosen based on age category (0=toddler, 1=child, 2= adolescent, 3= adult)
Question 1 Answer	Binary (0, 1)	The answer code of the question based on the screening method used
Question 2 Answer	Binary (0, 1)	The answer code of the question based on the screening method used
Question 3 Answer	Binary (0, 1)	The answer code of the question based on the screening method used
Question 4 Answer	Binary (0, 1)	The answer code of the question based on the screening method used
Question 5 Answer	Binary (0, 1)	The answer code of the question based on the screening method used
Question 6 Answer	Binary (0, 1)	The answer code of the question based on the screening method used
Question 7 Answer	Binary (0, 1)	The answer code of the question based on the screening method used
Question 8 Answer	Binary (0, 1)	The answer code of the question based on the screening method used
Question 9 Answer	Binary (0, 1)	The answer code of the question based on the screening method used
Question 10 Answer	Binary (0, 1)	The answer code of the question based on the screening method used
Screening Score	Integer	The final score obtained based on the scoring algorithm of the screening method used. This was computed in an automated manner
Autism	Binary (0, 1)	The actual autism diagnosis of the respondent

AQ-10 Screening Test

The AQ-10, or Autism Spectrum Quotient, is a ten-question test and is a condensed version of the AQ fifty-question test. The test was developed due to a “need for brief self-administered instruments to determine an individual's position on the autism-normality continuum” (MIDSS). On the AQ-10, the respondent may choose a spectrum of categorical responses from “Definitely Disagree” to “Definitely Agree”. Depending on the respondent’s answer, a score of either zero or one is given for each question. The scores are summed and if the sum is greater than six, the respondent is considered a candidate for an autism diagnosis. The AQ-10 test is shown below.

Figure 1: AQ-10 Test



Please tick one option per question only:

		Definitely Agree	**Slightly Agree**	**Slightly Disagree**	**Definitely Disagree**
1	I often notice small sounds when others do not				
2	I usually concentrate more on the whole picture, rather than the small details				
3	I find it easy to do more than one thing at once				
4	If there is an interruption, I can switch back to what I was doing very quickly				
5	I find it easy to ‘read between the lines’ when someone is talking to me				
6	I know how to tell if someone listening to me is getting bored				
7	When I’m reading a story I find it difficult to work out the characters’ intentions				
8	I like to collect information about categories of things (e.g. types of car, types of bird, types of train, types of plant etc)				
9	I find it easy to work out what someone is thinking or feeling just by looking at their face				
10	I find it difficult to work out people’s intentions				

SCORING: Only 1 point can be scored for each question. *Score 1 point for Definitely or Slightly Agree on each of items 1, 7, 8, and 10. Score 1 point for Definitely or Slightly Disagree on each of items 2, 3, 4, 5, 6, and 9.* If the individual scores **more than 6 out of 10**, consider referring them for a specialist diagnostic assessment.

Data Pre-processing

Our first step was to process and organize our data. We first started off by doing an exploratory analysis of all the variables in our data set. We looked into the correlations between our variables as well as the format that each variable had been provided in. Furthermore, we looked into the differences in the variables between those who had autism and those who did not in order to come up with a general idea of which questions we believed would be indicative. Next, we dropped features from the dataset that we did not need. These included Age, Screening Score, Used the Screening App Before, Relation, and Country of Residence. All of these variables we either deemed to be not predictive of autism (Age, Used the Screening App Before, Relation), a linear combination of other factors (Screening Score), or were highly correlated with another feature (Country of Residence).

We then continued our data preprocessing by finding and handling missing data. After importing the data into SAS we found that only the Ethnicity feature had missing values. To deal with the missing values we used SAS's Proc MI for categorical data imputation. Because the missing data pattern was monotone and our data was nominal we used the MONOTONE LOGISTIC option as recommended by SAS's literature for Proc MI.

Our final step was to one-hot encode our Ethnicity variable and use SAS to establish baseline performance levels (overall accuracy, false-negative rate, false-positive rate) for the survey. After doing so, the dataset was exported from SAS into a .csv file and imported into Python for analysis.

Proposed Methods

In this project we first began by splitting our data set into testing and training sets. We did a simple random selection choosing 80% of our data to be in our training data set and using the remaining 20% for our testing data set. We then utilized five different machine learning algorithms to model and make predictions from our data. Since our project could be interpreted as a binary classification problem, we decided to utilize logistic regression, decision trees, random forests, LogitBoost, and neural networks. We then compare the five methods based on the accuracy levels of the training and testing data sets to choose which model had the most predictive power.

Analysis

We began with logistic regression. Using a logistic function to model our binary data, we were able to get a training set accuracy of 88% and a testing set accuracy of 87%. Both accuracies were fairly high and outperformed the original testing method model.

Next, we moved on to decision trees. Decision trees utilize a simple tree-like algorithm of decisions and their possible consequences. So we first began by running the decision tree algorithm for depths of trees ranging from 1 to 25 to determine the optimal tree depth. We found that when we set the depth of tree to equal 5 we obtained the highest test set accuracy before our model began to overfit. Thus, using decision trees and growing our trees out to a depth of 5 we obtained a training accuracy rate 87.23% and a testing accuracy rate of 88.65%. We found this very interesting as our model actually performed better on our testing data set then our training data set at this level and we feel that this is simply because of the random seed we happened to choose.

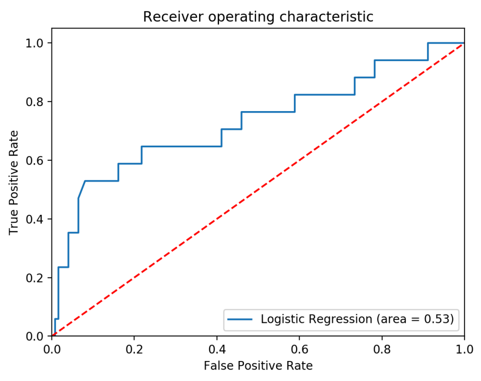
Then we moved on to creating a model utilizing random forests. A random forest is an ensemble learning method that constructs a multitude of decision trees and outputs the mean prediction of all the trees. We began by running various number of trees in order to find the optimal number. We found that when we ran 250 trees in our random forest algorithm we were able to obtain the highest testing accuracy. We received an accuracy of 97% for our training data set, but our testing data set is still right around 87% like we have been finding in the previous methods.

Next, we created a predictive model using LogitBoost. The LogitBoost algorithm is a boosting algorithm which continually adjusts weights to minimize the logistic loss. We ran 300 iterations, adjusting our weights for each iteration, and then plotted the logistic loss and were able to see that the plot did converge after approximately 45 iterations. From this model we were able to obtain accuracy levels of 86.86% and 87.94% for training and testing data sets respectively.

Lastly, we created various model using neural networks with varying levels of layers, while continually adjusting the parameters (i.e., batch size, learning rate, and the number of nodes in each hidden layer). Computationally, the neural networks took a long time to run and even after continuously readjusting our parameters we were still only able to achieve a training set accuracy of 82% and a test set accuracy of approximately 76%.

After completing our five proposed methods, we believed that random forests gave us our best model and we were impressed that we had improved our accuracy levels from the original testing methodology about 15%. However, as we completed further diagnostics our attention was brought to the ROC (receiver operating characteristic) curve – as shown below.

Figure 2: ROC Curve – Logistic Regression



Once we saw this ROC curve we realized that our model was not actually performing well as the area under the curve was only 0.53. We needed to look further into our models. Therefore, we looked into the confusion matrices for each of our five proposed models and realized the problem we had. Our testing data set only contained 17 instances of individuals with autism (about 12% of our data) and all of our models were essentially predicting that nearly no one had autism (at most three individuals with autism) and this explains why our accuracy levels were hovering around 87% accuracy.

Looking back at our original data set before the training/testing split, we realized that we had very imbalanced data. In the full data set we had only 91 instances of autism out of the 704 instances. We know that standard classifier algorithms like the ones we were utilizing have a bias towards the majority class. These algorithms tend to only predict the majority class data and the features of the minority class are treated as noise and are often ignored – hence, we were getting almost all of our testing data set instances to be predicted as not autistic. Thus, we realized that we needed to handle our issue of imbalanced data and then proceed with the proposed methods again.

Additional Methods

The first method we used to try to correct for imbalanced data was undersampling. Undersampling is a simple alternative sampling technique wherein instances of the majority data (all training data where the participant was not diagnosed as autistic) are randomly removed until there are an equal number of minority data and majority data (autistic and non-autistic respondents) in the training data set. Unfortunately, we began with a relatively small data set and this technique left us with an even smaller data set. Therefore, when we began running our models the performance of our final models suffered.

We then looked into oversampling. Oversampling, similarly to undersampling, selects random data points from the minority data set to duplicate until the number of minority instances is equal to the number of majority instances. Through utilizing oversampling we were left with a substantially larger training dataset to use for our models. We ran the ML algorithms on the new larger and balanced training dataset and it increased performance for almost all models substantially. However, we still thought we could do better so we pursued further models/techniques.

We also tried an alternative method of oversampling called SMOTE (Synthetic Minority Over-sampling Technique) to see if the training dataset that it provided could outperform the simply oversampled data set. SMOTE iterates through every point in the minority dataset and then utilizes the k-nearest neighbor algorithm and a random number between 0 and 1 to create synthetic data points until the minority and majority point sets are equal in size. SMOTE performed approximately equally as well as our random oversampling so we did not include the model performances in the results section

Logic Regression was the next alternative method we attempted, at the recommendation of Dr. Slate. Logic Regression can only be used with strictly binary data. It is essentially a decision tree model that uses Boolean logic to find new predictors that are logical combinations of the original predictors. Logic regression performed slightly worse than all of our other models, so we did not pursue it farther and also did not include it in the results section.

The final additional method we used was a simple change in threshold of the original logistic regression model using the oversampled training data. We began with a threshold of .7 and then continuously tried different thresholds to find the optimal threshold for our logistic model. Our optimal threshold turned out to be 0.74 rather than the default threshold of 0.5. This method performed significantly better than all of the models previous to this and is the one we ended up using as our final model.

Additional Method Results

In the table below we have compiled all of the results of the additional methods we utilized. We include the overall accuracy, the false-positive rate, the false-negative rate, and the Cohen-Kappa score (a statistic that measures inter-rater agreement) for each additional model. From this table you are able to see that the best model overall was the oversampled logistic regression with a threshold.

Table 2: Additional Results

Model	Prediction Accuracy	False-Positive Rate	False-Negative Rate	Cohen-Kappa Score
Baseline	0.7244	0.2382	0.5275	0.1607
Logistic Regression (Oversampled)	0.69	0.3065	0.3529	0.1880
Decision Tree (Oversampled)	0.8298	0.1129	0.6471	0.2106
Random Forest (Oversampled)	0.8582	0.0645	0.7059	0.2249
LogitBoost (Oversampled)	0.6809	0.3145	0.3529	0.1468
Neural Network (Oversampled)	0.8014	0.0645	0.7059	0.3286
Logistic Regression (Oversampled, Threshold)	0.8582	0.1048	0.4112	0.4095

Conclusion

Overall, we achieved marked improvement in every performance metric by utilizing demographic information in addition to survey questions, alternative sampling techniques, and modified logistic regression. This project emphasized the importance of secondary performance metrics such as ROC curves, Kappa-Cohen scores, and confusion matrices rather than accuracy alone to evaluate the performance of a predictive model.

As far as the efficacy of this screening test as a legitimate medical diagnostic tool goes, the false-positive and false-negative rates are too high for it to be of any use. Considering this, our recommendation to the medical community is to expand their prescreening survey to include more questions and more information. Then once more information is collected, they will need to utilize additional demographic information that all doctors should have, in conjunction with machine learning methods (logistic regression with custom threshold) to vastly improve the predictive power and reduce the false-negative and false-positive rates of the established screening tests. This change in procedure could easily prevent many unnecessary tests being performed, saving doctors and patients considerable amounts of money.

Works Cited

Autism Spectrum Disorder (ASD). (2018, November 15). Retrieved from https://www.cdc.gov/ncbddd/autism/data.html

Kooperberg, C. (2004, December 15). Logic Regression. Retrieved November 20, 2018, from http://kooperberg.fhcrc.org/logic/

Measurement Instrument Database for the Social Sciences. (n.d.). Retrieved November 10, 2018, from http://www.midss.org/content/autism-spectrum-quotient-aq-and-aq-10

SAS (2010, April 30). Proc MI Statement. Retrieved November 12, 2018, from https://support.sas.com/documentation/cdl/en/statug/63033/HTML/default/viewer.htm#statug_mi_sect004.htm

UCI Data Repository (n.d.). Autism Screening Adult Data Set. Retrieved October 20, 2018, from https://archive.ics.uci.edu/ml/datasets/Autism Screening Adult#

WordPress. (2017, October 12). Cohen's Kappa Statistic. Retrieved November 19, 2018, from https://www.statisticshowto.datasciencecentral.com/cohens-kappa-statistic/

Appendix (Code)

```
/* ////////////////////////////////////////////// */
/* //////////// SAS CODE *///////////// */
/* ////////////////////////////////////////////// */

/* Generated Code (IMPORT) */
/* Source File: AutismData.csv */
/* Source Path: /home/jrr17e0/sasuser.v94 */
/* Code generated on: 11/6/18, 2:54 PM */


%web_drop_table(WORK.IMPORT);

/* Importing the autism data set into SAS */
FILENAME REFFILE '/home/jrr17e0/sasuser.v94/AutismData.csv';
PROC IMPORT DATAFILE=REFFILE
	DBMS=CSV
	OUT=WORK.IMPORT;
	GETNAMES=YES;
RUN;

/* Running a proc contents on the data set to observe our variables */
PROC CONTENTS DATA=WORK.IMPORT;
RUN;

/* Identifying missing variables of numeric variables using proc means */
proc means data = work.import n nmiss ;
run;

/* Identifying missing variables of ALL variables using proc freq */
proc format;
 value $missfmt ' '='Missing' other='Not Missing';
 value  missfmt  . ='Missing' other='Not Missing';
run;

proc freq data=work.import;
	format _CHAR_ $missfmt.; /* apply format for the duration of this PROC */
	tables _CHAR_ / missing missprint nocum nopercent;
	format _NUMERIC_ missfmt.;
	tables _NUMERIC_ / missing missprint nocum nopercent;
run;
/* NOTE: Some variables simply say NA instead of being missing */

/* Calculating error rates and changing all of the binary variables to 0's and 1's */
data work.import;
```

```
            set work.import;
            rename ClassASD = prediction;
            rename austim = autism;
            if ethnicity = 'NA' then ethnicity = ' ';
            if (autism = 'yes' and prediction = 'YES') or (autism = 'no' and prediction = 'NO')
then correct = 1;
            else correct = 0;
            if (autism = 'yes' and prediction = 'NO') then falsenegative = 1;
            else falsenegative = 0;
            if (autism = 'no' and prediction = 'YES') then falsepositive = 1;
            else falsepositive = 0;
            if (gender = 'f') then gender = 0;
            else gender = 1;
            if (jundice = 'no') then jundice = 0;
            else jundice = 1;
            if (autism = 'no') then autism = 0;
            else autism = 1;
            if (prediction = 'NO') then prediction = 0;
            else prediction = 1;
proc freq data = work.import;
run;

/* Handling missing categorical data */
proc mi data = work.import out = imputed;
class gender jundice prediction ethnicity A1_Score A2_Score A3_Score A4_Score
A5_Score A6_Score A7_Score A8_Score A9_Score A10_Score;
var A1_Score A2_Score A3_Score A4_Score A5_Score A6_Score A7_Score A8_Score
A9_Score A10_Score gender jundice prediction ethnicity;
monotone logistic(ethnicity = A1_Score A2_Score A3_Score A4_Score A5_Score
A6_Score A7_Score A8_Score A9_Score A10_Score gender jundice prediction);

data work.imputed;
set work.imputed;
if ethnicity = 'White-European' then WhiteEuropean = 1;
    else WhiteEuropean = 0;
  if ethnicity = 'Latino' then Latino = 1;
    else Latino = 0;
  if ethnicity = 'Black' then Black = 1;
    else Black = 0;
  if ethnicity = 'Asian' then Asian = 1;
    else Asian = 0;
  if ethnicity = 'Middle Eastern' then MiddleEastern = 1;
    else MiddleEastern = 0;
    if ethnicity = 'Pasifika' then Pasifika = 1;
    else Pasifika = 0;
    if ethnicity = 'Hispanic' then Hispanic = 1;
```

```
   else Hispanic = 0;
   if ethnicity = 'South Asian' then SouthAsian = 1;
   else SouthAsian = 0;
   if ethnicity = 'Turkish' then Turkish = 1;
   else Turkish = 0;
   if ethnicity = 'Others' then Others = 1;
   else Others = 0;
run;

data work.autism (drop= _imputation_ age result used_app_before age_desc relation
correct falsenegative falsepositive contry_of_res ethnicity VAR1 prediction);
set work.imputed;
where _imputation_=1;
run;

/* Creating a data set with just the Y variable (has autism or not) */
data autismlabels (keep=autism);
        set work.autism;
run;

/* Creating a data set with all the explanatory variables */
data autismvars (drop = autism ethnicity contry_of_res);
        set work.autism;
run;

/* Creating a data set with just Y variable but -1/1 instead of 0/1 */
data autismlabels2 (keep=autism);
        set work.autism;
        if (autism = 0) THEN autism = -1;
run;


/* Work just to make exploratory plots for beginning presentation */
%web_drop_table(WORK.autism);
proc gchart data=work.autism ;
        where autism = "yes";
        vbar  A1_Score ;
        title 'A1_Score';
run;

proc gchart data=work.autism ;
        where autism = "yes";
        vbar  A2_Score ;
        title 'A2_Score';
run;
```

```
proc gchart data=work.autism ;
        where autism = "yes";
        vbar  A3_Score ;
        title 'A3_Score';
run;

proc gchart data=work.autism ;
        where autism = "yes";
        vbar  A4_Score ;
        title 'A4_Score';
run;
proc gchart data=work.autism ;
        where autism = "yes";
        vbar  A5_Score ;
        title 'A5_Score';
run;
proc gchart data=work.autism ;
        where autism = "yes";
        vbar  A6_Score ;
        title 'A6_Score';
run;
proc gchart data=work.autism ;
        where autism = "yes";
        vbar  A7_Score ;
        title 'A7_Score';
run;
proc gchart data=work.autism ;
        where autism = "yes";
        vbar  A8_Score ;
        title 'A8_Score';
run;
proc gchart data=work.autism ;
        where autism = "yes";
        vbar  A9_Score ;
        title 'A9_Score';
run;
proc gchart data=work.autism ;
        where autism = "yes";
        vbar  A10_Score ;
        title 'A10_Score';
run;

%web_open_table(WORK.AUTISM);
### /////////////////////////////////////////////////////////////////////////
### //////////////////////// Python Code   //////////////////////////
### ///////////////////////////////////////////////////////////////////////
```

```
# ALL DATA PROCESSING DONE IN SAS
# Packages needed for all data analysis
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from tabulate import tabulate
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn import preprocessing
from sklearn.metrics import roc_curve
from sklearn.linear_model import LogisticRegression
from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers
import time
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score, confusion_matrix, recall_score,
roc_auc_score, precision_score, cohen_kappa_score


# Reading in autism data set
autism = pd.read_csv("AUTISMVARS_FinalTAKE2.csv")
autism_labels = pd.read_csv("AUTISMLABELS_FinalTake2.csv")

# Splitting data set into test and train data set  (80%/20%)
X_train, X_test, y_train, y_test = train_test_split(autism, autism_labels, test_size=0.2,
random_state = 1234)


# Analysis to complete: Logistic Regression, Decision Trees,
# Random Forest, FSA, Neural Networks

### ////////////////////////////////////////////////////////////////////////
### //////////////////////// Logistic Regression //////////////////
### //////////////////////////////////////////////////////////////////////

start_time = time.time()
# Fitting a logistic regression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

# Making predictions
y_pred = logreg.predict(X_test)
```

```
print('Accuracy of logistic regression classifier on test set:
{:.2f}'.format(logreg.score(X_test, y_test)))
print('Accuracy of logistic regression classifier on train set:
{:.2f}'.format(logreg.score(X_train, y_train)))
print("--- %s seconds ---" % (time.time() - start_time))
# Plotting the ROC Curve
logit_roc_auc = roc_auc_score(y_test, logreg.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, logreg.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()

# Confusion matrix from model
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)

### ////////////////////////////////////////////////////////////////////////
### /////////////////////// Decision Trees ////////////////////////
### //////////////////////////////////////////////////////////////////////

# Creating a function called Misclassplot that will input the four sets
# of data and a depth and then plot the misclassification vs. depth of the
# tree for both the training and testing data set.

def DECISION_TREES(train, trainlab, test, testlab, depth, data):

    # Running a for loop to create data frames of the misclassification errors
    mis_train=[] # Initializing training misclassification list
    mis_test=[] # Initializing testing misclassification list
    for x in range(1,depth+1):
        dec_tree=DecisionTreeClassifier(max_depth=x) # Max_Depth attribute
        dec_tree=dec_tree.fit(train, trainlab) # Growing out the decision tree

        # Predicting on training data set / calculating misclassification error
        train_pred=dec_tree.predict(train)
        mis_train.append(100- (accuracy_score(trainlab,train_pred)*100))

        # Predicting on testing data set / calculating misclassification error
        test_pred=dec_tree.predict(test)
        mis_test.append(100- (accuracy_score(testlab,test_pred)*100))

    # Putting the misclassification errors into data frames
    mis_train = 100 - pd.DataFrame(mis_train)
    mis_train.columns=['Error']
    print(mis_train)
    mis_test = 100- pd.DataFrame(mis_test)
    mis_test.columns = ['Error']
    print(mis_test)

    # Plotting the misclassification error vs depth of tree for both testing/training
    plt.plot(range(1,1+len(mis_train)), mis_train, linestyle='-', marker='o')
    plt.plot(range(1,1+len(mis_test)), mis_test, linestyle='-', marker='o')
    plt.legend(['Training Data', 'Testing Data'], loc='upper left')
    plt.ylabel("Accuracy Rate (%)")
    plt.xlabel("Depth of Tree")
    plt.title("Accuracy Rates Vs. Depth of Trees - " + str(data))
    plt.axis([0,depth,80,100])
    plt.show()

    # Printing out the minimum classification error for testing data set
    max = np.array([[pd.DataFrame.max(mis_test.Error),
                pd.Series.idxmax(mis_test)+1]])
    headers = ['Maximum Accuracy Rate', 'Tree Depth']
    table = tabulate(max, headers, tablefmt="fancy_grid")
    print(table)
    return table;


# Running the function on the data with max depth from 1-24 decision trees
start_time = time.time()
a = DECISION_TREES(train=X_train, trainlab=y_train,
            test=X_test,testlab=y_test,
            depth=24, data = "Autism" )
print("--- %s seconds ---" % (time.time() - start_time))

# CHOSE TREE DEPTH 5, LARGEST ACCURACY BEFORE OVERFITTING

# Running with tree depth already chosen
start_time = time.time()

dec_tree=DecisionTreeClassifier(max_depth=5) # Max_Depth attribute
dec_tree=dec_tree.fit(X_train, y_train) # Growing out the decision tree

# Predicting on training data set / calculating misclassification error
```

```
train_pred=dec_tree.predict(X_train)
mis_train = (accuracy_score(y_train,train_pred) * 100)

# Predicting on testing data set / calculating misclassification error
test_pred=dec_tree.predict(X_test)
mis_test= (accuracy_score(y_test,test_pred)*100)

print(mis_train)
print(mis_test)
print("--- %s seconds ---" % (time.time() - start_time))

# Confusion matrix from model
confusion_matrix = confusion_matrix(y_test, test_pred)
print(confusion_matrix)


### ////////////////////////////////////////////////////////////////////////
### /////////////////////////// Random Forests ///////////////////
### //////////////////////////////////////////////////////////////////////

def RandomTreePlot(train, trainlab, valid, validlab, features):
   # Defining the number of trees (k)
   #k= [1,10, 20, 40,  60,  80,100, 150, 200, 250, 300]
   k = [250]
   # Initialization of misclassification error matrix
   d = np.zeros((len(k),3))
   d[:,0] = k
   Error_train = np.zeros(len(k))
   Error_test = np.zeros(len(k))

   # Creating random forests with k number of trees and the split attribute
   # specified for TRAINING data
   for x in range(0,len(k)):
      rand_forest=RandomForestClassifier(n_estimators=k[x], max_features=features)
      rand_forest = rand_forest.fit(train, trainlab.values.ravel())
      predictions = rand_forest.predict(train)
      Error_train[(x)] = metrics.accuracy_score(trainlab,predictions) * 100

   # Creating random forests with k number of trees and the split attribute
   # specified for TESTING data
   for x in range(0,len(k)):
      rand_forest=RandomForestClassifier(n_estimators=k[x], max_features=features)
      rand_forest = rand_forest.fit(train, trainlab.values.ravel())
      predictions = rand_forest.predict(valid)
      Error_test[(x)] = metrics.accuracy_score(validlab, predictions) * 100
```

```
    # Assignment of misclassification errors for training/testing data sets to
    # misclassification error matrix
    d[:,1] = Error_train
    d[:,2] = Error_test

    # Plotting the misclassification error vs number of trees for both
    # testing/training
    plt.plot(d[:,0], d[:,1], linestyle='-', marker='o')
    plt.plot(d[:,0], d[:,2], linestyle='-', marker='o')
    plt.legend(['Training Data', 'Testing Data'], loc='lower right')
    plt.ylabel("Accuracy Rate (%)")
    plt.xlabel("Number of Tree")
    plt.title("Accuracy Rate Vs. Number of Trees")
    plt.show()

    headers = ['Tree Depth', 'Accuracy Rate - Train', "Accuracy Rate - Test",]
    table = tabulate(d, headers, tablefmt="fancy_grid")
    print(table)
    return(predictions)


# The split attribute being chosen from a random subset of sqrt(500) features
start_time = time.time()
preds = RandomTreePlot(train=X_train, trainlab=y_train,
            valid=X_test,validlab=y_test, features="sqrt")
print("--- %s seconds ---" % (time.time() - start_time))

from sklearn.metrics import confusion_matrix

# Confusion matrix from model
confusion_matrix = confusion_matrix(y_test, preds)
print(confusion_matrix)


### ///////////////////////////////////////////////////////////////////////
### ///////////////////////////// LogitBoost //////////////////////
### //////////////////////////////////////////////////////////////////////

# Reading in labels as -1/1 instead of 0/1
autism_labels2 = pd.read_csv("AUTISMLABELS2_FinalTake2.csv")

# Splitting data set into test and train data set  (80%/20%)
X_train, X_test, y_train, y_test = train_test_split(autism, autism_labels2, test_size=0.2,
random_state=1234)

X_train.reset_index(drop=True, inplace=True)
```

```
X_test.reset_index(drop=True, inplace=True)


# Adding a column of 1's to xtrain and xtest data sets
X_train = pd.concat([pd.DataFrame([1]*len(X_train)), X_train], axis=1)
X_test = pd.concat([pd.DataFrame([1]*len(X_test)), X_test], axis=1)

# Creating a label column heading
y_train.columns = ['label']

# Initializing N and M
N = len(X_train)
M = X_train.shape[1]

# Iteration
weightsall = np.transpose(np.zeros(M))
loss = np.empty([300, 1])
accuracy_train = np.empty([4, 1]) # MAKE SURE THESE ARE ZERO BEFORE
RUNNING LOOP
accuracy_test = np.empty([4, 1]) # MAKE SURE THESE ARE ZERO BEFORE
RUNNING LOOP
ik = 1
z = np.zeros(N)

start_time = time.time()

# Loop
for i in range(1,301):
   H = np.dot(X_train, weightsall)
   p = (1/(1+np.exp(-2*H)))
   w = p*(1-p)
   y_train.reset_index(drop=True, inplace=True)
   for t in range(len(w)):
      if w[t] == 0:
         z[t] = 0
      else:
         z[t] = (((.5*(y_train.label[t]+1))-p[t])/w[t])

   coef = np.empty([2,M-1])
   newloss = np.empty([M-1, 1])

   for j in range(0,M-1):
      Xj = X_train.values[:,j+1]
      a = sum(w)
      b = sum(w*Xj)
      c = sum(w*Xj**2)
```

```
        d = sum(w*z)
        e = sum(w * Xj * z)

        # WLS estimation
        if (a*c)-b**2 ==0:
           Bj = np.array([(d/a), 0])
        else:
           Bj = ( 1 / (a*c-b**2) ) * np.array([c * d - b * e, a * e - b* d])

        # New H(X)
        Hj = H + .5*(Bj[0] + Bj[1] * Xj)

        # New loss function value
        lossj = sum(np.log(1 + np.exp(-2*y_train.label * Hj)))

        coef[:, j]=Bj
        newloss[j]=lossj

    Jhat = np.argmin(newloss) # [1:len(newloss)]

    weightsall[0] = weightsall[0] + .5*coef[0,Jhat]
    weightsall[Jhat+1] = weightsall[Jhat+1]+.5*coef[1,Jhat]
    loss[i-1] = newloss[Jhat]

    # Making predictions
    pred_train = np.sign(np.dot(X_train,weightsall))
    pred_test = np.sign(np.dot(X_test,weightsall))

    # Calculating error rate
    if i in (10,30, 100, 300):
        accuracy_train[ik-1]=metrics.accuracy_score(pred_train, y_train)
        accuracy_test[ik-1]=metrics.accuracy_score(pred_test, y_test)
        ik = ik+1

    if i==300:
        plt.plot(range(1,301), loss)
        plt.title("Loss Plot")

print("--- %s seconds ---" % (time.time() - start_time))

start_time = time.time()
# Computational time for just i = 300
H = np.dot(X_train, weightsall)
p = (1/(1+np.exp(-2*H)))
w = p*(1-p)
y_train.reset_index(drop=True, inplace=True)
```

```
for t in range(len(w)):
    if w[t] == 0:
        z[t] = 0
    else:
        z[t] = (((.5*(y_train.label[t]+1))-p[t])/w[t])

coef = np.empty([2,M-1])
newloss = np.empty([M-1, 1])

for j in range(0,M-1):
    Xj = X_train.values[:,j+1]
    a = sum(w)
    b = sum(w*Xj)
    c = sum(w*Xj**2)
    d = sum(w*z)
    e = sum(w * Xj * z)

    # WLS estimation
    if (a*c)-b**2 ==0:
        Bj = np.array([(d/a), 0])
    else:
        Bj = ( 1 / (a*c-b**2) ) * np.array([c * d - b * e, a * e - b* d])

    # New H(X)
    Hj = H + .5*(Bj[0] + Bj[1] * Xj)

    # New loss function value
    lossj = sum(np.log(1 + np.exp(-2*y_train.label * Hj)))

    coef[:, j]=Bj
    newloss[j]=lossj

Jhat = np.argmin(newloss) # [1:len(newloss)]

weightsall[0] = weightsall[0] + .5*coef[0,Jhat]
weightsall[Jhat+1] = weightsall[Jhat+1]+.5*coef[1,Jhat]
loss[300-1] = newloss[Jhat]

# Making predictions
pred_train = np.sign(np.dot(X_train,weightsall))
pred_test = np.sign(np.dot(X_test,weightsall))
accuracy_train=metrics.accuracy_score(pred_train, y_train)
accuracy_test=metrics.accuracy_score(pred_test, y_test)
print("--- %s seconds ---" % (time.time() - start_time))

# Confusion matrix from model
```

```
confusion_matrix = confusion_matrix(y_test, pred_test)
print(confusion_matrix)


### ///////////////////////////////////////////////////////////////////////
### ////////////////////// NeuralNetworks ///////////////////////
### //////////////////////////////////////////////////////////////////////
# Reading in labels as -1/1 instead of 0/1
autism_labels2 = pd.read_csv("AUTISMLABELS2_FinalTake2.csv")
def neuralnets(k, lr):
    accuracy_train = np.empty([10, 1]) # MAKE SURE THESE ARE ZERO BEFORE
RUNNING LOOP
    accuracy_test = np.empty([10, 1])


    # Split training and testing data set
    train_x, test_x, train_y, test_y = train_test_split(autism, autism_labels2, test_size=0.2)

    # Normalize data set
    scaler = preprocessing.StandardScaler().fit(train_x)
    train_x = pd.DataFrame(scaler.transform(train_x))

    # Transform test data set
    test_x = pd.DataFrame(scaler.transform(test_x))

    # Creating a model
    model = Sequential()
    model.add(Dense(k, input_dim=train_x.shape[1], activation='relu'))
    model.add(Dense(k, input_dim=train_x.shape[1], activation='relu'))
    model.add(Dense(1))

    # Compiling model
    sgd = optimizers.SGD(lr=lr)
    model.compile(loss='mean_squared_error', optimizer=sgd, metrics=['accuracy'])

    # Training a model
    results = model.fit(train_x, train_y, validation_data = (test_x, test_y), epochs=300,
batch_size=10)
    preds = model.predict_classes(test_x, verbose=1)
    accuracy_train = results.history['acc'][99]
    accuracy_test = results.history['val_acc'][99]
    return accuracy_train, accuracy_test, preds, test_y

# Batch size 10, epochs = 300
start_time = time.time()
mini_k32_2layers = neuralnets(256, .001)
```

```
print("--- %s seconds ---" % (time.time() - start_time))
# 82, 75
preds = mini_k32_2layers[2]
ytest = mini_k32_2layers[3]

# Confusion matrix from model
confusion_matrix = confusion_matrix(np.ravel(ytest), preds)
print(confusion_matrix)


### /////////////////////////////////////////////////////////////////////////////////////////////
### ////////////////// Take 2: Undersampling data set ////////////////////////
### //////////////////////////////////////////////////////////////////////////////////////////
# Reading in autism data set
autism = pd.read_csv("AUTISMVARS_FinalTAKE2.csv")
autism_labels = pd.read_csv("AUTISMLABELS_FinalTake2.csv")
autism_total = pd.concat([autism, autism_labels], axis=1)

# Splitting data set into test and train data set  (80%/20%)
X_train, X_test, y_train, y_test = train_test_split(autism, autism_labels, test_size=0.2,
random_state = 1234)

train = pd.concat([X_train, y_train], axis=1)
test = pd.concat([X_test, y_test], axis=1)

aut = len(train[train['autism'] == 1])
aut_indices = train[train.autism == 0].index
random_indices = np.random.choice(aut_indices, aut, replace=False)
no_aut_indices = train[train.autism == 1].index
under_sample_indices = np.concatenate([no_aut_indices,random_indices])
under_sample = train.loc[under_sample_indices]

no_aut = len(under_sample[under_sample['autism'] == 1])
aut = len(under_sample[under_sample['autism'] == 0])

X_train = under_sample.drop("autism", axis=1)
y_train = under_sample['autism']

### ///////////////////////////////////////////////////////////////////////////////////////
### ////////////////////////// Logistic Regression ///////////////////////////
### ////////////////////////////////////////////////////////////////////////////////////

start_time = time.time()
# Fitting a logistic regression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

```
# Making predictions
y_pred = logreg.predict(X_test)
print('Accuracy of logistic regression classifier on train set:
{:.2f}'.format(logreg.score(X_train, y_train)))
print('Accuracy of logistic regression classifier on test set:
{:.2f}'.format(logreg.score(X_test, y_test)))
print("--- %s seconds ---" % (time.time() - start_time))
# Plotting the ROC Curve
logit_roc_auc = roc_auc_score(y_test, logreg.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, logreg.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()

# Confusion matrix from model
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)

### ////////////////////////////////////////////////////////////////////////////////////////
### /////////////////////////// Random Forests ////////////////////////////
### ////////////////////////////////////////////////////////////////////////////////

def RandomTreePlot(train, trainlab, valid, validlab, features):
    # Defining the number of trees (k)
    #k= [1,10, 20, 40,  60,  80,100, 150, 200, 250, 300]
    k = [100]
    # Initialization of misclassification error matrix
    d = np.zeros((len(k),3))
    d[:,0] = k
    Error_train = np.zeros(len(k))
    Error_test = np.zeros(len(k))

    # Creating random forests with k number of trees and the split attribute
    # specified for TRAINING data
    for x in range(0,len(k)):
        rand_forest=RandomForestClassifier(n_estimators=k[x], max_features=features)
        rand_forest = rand_forest.fit(train, trainlab.values.ravel())
        predictions = rand_forest.predict(train)
        Error_train[(x)] = metrics.accuracy_score(trainlab,predictions) * 100

    # Creating random forests with k number of trees and the split attribute
    # specified for TESTING data
    for x in range(0,len(k)):
        rand_forest=RandomForestClassifier(n_estimators=k[x], max_features=features)
        rand_forest = rand_forest.fit(train, trainlab.values.ravel())
        predictions = rand_forest.predict(valid)
        Error_test[(x)] = metrics.accuracy_score(validlab, predictions) * 100

    # Assignment of misclassification errors for training/testing data sets to
    # misclassification error matrix
    d[:,1] = Error_train
    d[:,2] = Error_test

    # Plotting the misclassification error vs number of trees for both
    # testing/training
    plt.plot(d[:,0], d[:,1], linestyle='-', marker='o')
    plt.plot(d[:,0], d[:,2], linestyle='-', marker='o')
    plt.legend(['Training Data', 'Testing Data'], loc='lower right')
    plt.ylabel("Accuracy Rate (%)")
    plt.xlabel("Number of Tree")
    plt.title("Accuracy Rate Vs. Number of Trees")
    plt.show()

    headers = ['Tree Depth', 'Accuracy Rate - Train', "Accuracy Rate - Test",]
    table = tabulate(d, headers, tablefmt="fancy_grid")
    print(table)
    return(predictions)

# The split attribute being chosen from a random subset of sqrt(500) features
start_time = time.time()
preds = RandomTreePlot(train=X_train, trainlab=y_train,
            valid=X_test,validlab=y_test, features="sqrt")
print("--- %s seconds ---" % (time.time() - start_time))

# Confusion matrix from model
confusion_matrix = confusion_matrix(y_test, preds)
print(confusion_matrix)

# Realized that undersampling was not going to work and moved on to oversampling

### ////////////////////////////////////////////////////////////////////////////////
### ///////////////// Take 3: Oversampling data set //////////////////////
### ///////////////////////////////////////////////////////////////////////////////
```

```
# Reading in autism data set
autism = pd.read_csv("AUTISMVARS_FinalTAKE2.csv")
autism_labels = pd.read_csv("AUTISMLABELS_FinalTake2.csv")
autism_total = pd.concat([autism, autism_labels], axis=1)

#Split data into train and test
X_train_orig, X_test, y_train_orig, y_test = train_test_split(autism, autism_labels,
test_size=0.2, random_state = 1234)

# Oversample training data set
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
X_resampled, y_resampled = ros.fit_resample(X_train_orig, np.ravel(y_train_orig))

X_train = X_resampled
y_train = y_resampled

no_aut = len(y_resampled[y_resampled == 1]) #489
aut = len(y_resampled[y_resampled == 0]) #489


### ////////////////////////////////////////////////////////////////////////////////
### ///////////////////////// Logistic Regression /////////////////////////
### ///////////////////////////////////////////////////////////////////////////////

start_time = time.time()
# Fitting a logistic regression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

# Making predictions
y_pred = logreg.predict(X_test)
print('Accuracy of logistic regression classifier on train set:
{:.2f}'.format(logreg.score(X_train, y_train)))
print('Accuracy of logistic regression classifier on test set:
{:.2f}'.format(logreg.score(X_test, y_test)))
print("--- %s seconds ---" % (time.time() - start_time))
# Plotting the ROC Curve
logit_roc_auc = roc_auc_score(y_test, logreg.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, logreg.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()

# Confusion matrix from model
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
cohen_kappa_score(y_test, y_pred)

### ////////////////////////////////////////////////////////////////////////////////
### /////////////////////////// Decision Trees ////////////////////////////
### ////////////////////////////////////////////////////////////////////////////

# Creating a function called Misclassplot that will input the four sets
# of data and a depth and then plot the misclassification vs. depth of the
# tree for both the training and testing data set.

def DECISION_TREES(train, trainlab, test, testlab, depth, data):

    # Running a for loop to create data frames of the misclassification errors
    mis_train=[] # Initializing training misclassification list
    mis_test=[] # Initializing testing misclassification list
    for x in range(1,depth+1):
        dec_tree=DecisionTreeClassifier(max_depth=x) # Max_Depth attribute
        dec_tree=dec_tree.fit(train, trainlab) # Growing out the decision tree

        # Predicting on training data set / calculating misclassification error
        train_pred=dec_tree.predict(train)
        mis_train.append(100- (accuracy_score(trainlab,train_pred)*100))

        # Predicting on testing data set / calculating misclassification error
        test_pred=dec_tree.predict(test)
        mis_test.append(100- (accuracy_score(testlab,test_pred)*100))

    # Putting the misclassification errors into data frames
    mis_train = 100 - pd.DataFrame(mis_train)
    mis_train.columns=['Error']
    print(mis_train)
    mis_test = 100- pd.DataFrame(mis_test)
    mis_test.columns = ['Error']
    print(mis_test)

    # Plotting the misclassification error vs depth of tree for both testing/training
    plt.plot(range(1,1+len(mis_train)), mis_train, linestyle='-', marker='o')
    plt.plot(range(1,1+len(mis_test)), mis_test, linestyle='-', marker='o')
    plt.legend(['Training Data', 'Testing Data'], loc='upper left')
    plt.ylabel("Accuracy Rate (%)")
    plt.xlabel("Depth of Tree")
    plt.title("Accuracy Rates Vs. Depth of Trees - " + str(data))
    plt.axis([0,depth,50,100])
    plt.show()

    # Printing out the minimum classification error for testing data set
    max = np.array([[pd.DataFrame.max(mis_test.Error),
                pd.Series.idxmax(mis_test)+1]])
    headers = ['Maximum Accuracy Rate', 'Tree Depth']
    table = tabulate(max, headers, tablefmt="fancy_grid")
    print(table)
    return table;


# Running the function on the data with max depth from 1-24 decision trees
start_time = time.time()
a = DECISION_TREES(train=X_train, trainlab=y_train,
         test=X_test,testlab=y_test,
         depth=24, data = "Autism" )
print("--- %s seconds ---" % (time.time() - start_time))

# CHOSE TREE DEPTH 19, LARGEST ACCURACY BEFORE OVERFITTING

# Running with tree depth already chosen
start_time = time.time()

dec_tree=DecisionTreeClassifier(max_depth=19) # Max_Depth attribute
dec_tree=dec_tree.fit(X_train, y_train) # Growing out the decision tree

# Predicting on training data set / calculating misclassification error
train_pred=dec_tree.predict(X_train)
mis_train = (accuracy_score(y_train,train_pred) * 100)

# Predicting on testing data set / calculating misclassification error
test_pred=dec_tree.predict(X_test)
mis_test= (accuracy_score(y_test,test_pred)*100)

print(mis_train)
print(mis_test)
print("--- %s seconds ---" % (time.time() - start_time))

# Confusion matrix from model
confusion_matrix = confusion_matrix(y_test, test_pred)
```

```
print(confusion_matrix)
cohen_kappa_score(y_test, test_pred)

### ////////////////////////////////////////////////////////////////////////////////
### ////////////////////////// Random Forests ///////////////////////////
### ////////////////////////////////////////////////////////////////////////////////

def RandomTreePlot(train, trainlab, valid, validlab, features):
    # Defining the number of trees (k)
    #k= [1,10, 20, 40,  60,  80,100, 150, 200, 250, 300]
    k = [20]
    # Initialization of misclassification error matrix
    d = np.zeros((len(k),3))
    d[:,0] = k
    Error_train = np.zeros(len(k))
    Error_test = np.zeros(len(k))

    # Creating random forests with k number of trees and the split attribute
    # specified for TRAINING data
    for x in range(0,len(k)):
        rand_forest=RandomForestClassifier(n_estimators=k[x], max_features=features)
        rand_forest = rand_forest.fit(train, trainlab)
        predictions = rand_forest.predict(train)
        Error_train[(x)] = metrics.accuracy_score(trainlab,predictions) * 100

    # Creating random forests with k number of trees and the split attribute
    # specified for TESTING data
    for x in range(0,len(k)):
        rand_forest=RandomForestClassifier(n_estimators=k[x], max_features=features)
        rand_forest = rand_forest.fit(train, trainlab)
        predictions = rand_forest.predict(valid)
        Error_test[(x)] = metrics.accuracy_score(validlab, predictions) * 100

    # Assignment of misclassification errors for training/testing data sets to
    # misclassification error matrix
    d[:,1] = Error_train
    d[:,2] = Error_test

    # Plotting the misclassification error vs number of trees for both
    # testing/training
    plt.plot(d[:,0], d[:,1], linestyle='-', marker='o')
    plt.plot(d[:,0], d[:,2], linestyle='-', marker='o')
    plt.legend(['Training Data', 'Testing Data'], loc='lower right')
    plt.ylabel("Accuracy Rate (%)")
    plt.xlabel("Number of Tree")
    plt.title("Accuracy Rate Vs. Number of Trees")
    plt.show()

    headers = ['Tree Depth', 'Accuracy Rate - Train', "Accuracy Rate - Test",]
    table = tabulate(d, headers, tablefmt="fancy_grid")
    print(table)
    return predictions


# The split attribute being chosen from a random subset of sqrt(500) features
start_time = time.time()
preds = RandomTreePlot(train=X_train, trainlab=y_train,
            valid=X_test,validlab=y_test, features="sqrt")
print("--- %s seconds ---" % (time.time() - start_time))

# Confusion matrix from model
confusion_matrix = confusion_matrix(y_test, preds)
print(confusion_matrix)
cohen_kappa_score(y_test, preds)


### ///////////////////////////////////////////////////////////////////////////////
### /////////////////////////////// LogitBoost ///////////////////////////////
### ///////////////////////////////////////////////////////////////////////////////
# Reading in autism data set
autism = pd.read_csv("AUTISMVARS_FinalTAKE2.csv")
autism_labels = pd.read_csv("AUTISMLABELS_FinalTake2.csv")
# Changing labels to -1/1 instead of 0/1
autism_labels[autism_labels == 0] = -1

#Split data into train and test
X_train_orig, X_test, y_train_orig, y_test = train_test_split(autism, autism_labels,
test_size=0.2, random_state = 1234)

# Oversample training data set
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
X_resampled, y_resampled = ros.fit_resample(X_train_orig, np.ravel(y_train_orig))

X_train = X_resampled
y_train = y_resampled

#X_train.reset_index(drop=True, inplace=True)
X_test.reset_index(drop=True, inplace=True)

# Adding a column of 1's to xtrain and xtest data sets
X_train = pd.concat([pd.DataFrame([1]*len(X_train)), pd.DataFrame(X_train)], axis=1)
```

```
X_test = pd.concat([pd.DataFrame([1]*len(X_test)), X_test], axis=1)

# Creating a label column heading
y_train = pd.DataFrame(y_train)
y_train.columns = ['label']

# Initializing N and M
N = len(X_train)
M = X_train.shape[1]

# Iteration
weightsall = np.transpose(np.zeros(M))
loss = np.empty([300, 1])
accuracy_train = np.empty([4, 1]) # MAKE SURE THESE ARE ZERO BEFORE
RUNNING LOOP
accuracy_test = np.empty([4, 1]) # MAKE SURE THESE ARE ZERO BEFORE
RUNNING LOOP
ik = 1
z = np.zeros(N)

start_time = time.time()

# Loop
for i in range(1,301):
    H = np.dot(X_train, weightsall)
    p = (1/(1+np.exp(-2*H)))
    w = p*(1-p)
    y_train.reset_index(drop=True, inplace=True)
    for t in range(len(w)):
        if w[t] == 0:
            z[t] = 0
        else:
            z[t] = (((.5*(y_train.label[t]+1))-p[t])/w[t])

    coef = np.empty([2,M-1])
    newloss = np.empty([M-1, 1])

    for j in range(0,M-1):
        Xj = X_train.values[:,j+1]
        a = sum(w)
        b = sum(w*Xj)
        c = sum(w*Xj**2)
        d = sum(w*z)
        e = sum(w * Xj * z)

        # WLS estimation
        if (a*c)-b**2 ==0:
            Bj = np.array([(d/a), 0])
        else:
            Bj = ( 1 / (a*c-b**2) ) * np.array([c * d - b * e, a * e - b* d])

        # New H(X)
        Hj = H + .5*(Bj[0] + Bj[1] * Xj)

        # New loss function value
        lossj = sum(np.log(1 + np.exp(-2*y_train.label * Hj)))

        coef[:, j]=Bj
        newloss[j]=lossj

    Jhat = np.argmin(newloss) # [1:len(newloss)]

    weightsall[0] = weightsall[0] + .5*coef[0,Jhat]
    weightsall[Jhat+1] = weightsall[Jhat+1]+.5*coef[1,Jhat]
    loss[i-1] = newloss[Jhat]

    # Making predictions
    pred_train = np.sign(np.dot(X_train,weightsall))
    pred_test = np.sign(np.dot(X_test,weightsall))

    # Calculating error rate
    if i in (10,30, 100, 300):
        accuracy_train[ik-1]=metrics.accuracy_score(pred_train, y_train)
        accuracy_test[ik-1]=metrics.accuracy_score(pred_test, y_test)
        ik = ik+1

    if i==300:
        plt.plot(range(1,301), loss)
        plt.title("Loss Plot")

print("--- %s seconds ---" % (time.time() - start_time))

start_time = time.time()
# Computational time for just i = 300
H = np.dot(X_train, weightsall)
p = (1/(1+np.exp(-2*H)))
w = p*(1-p)
y_train.reset_index(drop=True, inplace=True)
for t in range(len(w)):
    if w[t] == 0:
        z[t] = 0
    else:
        z[t] = (((.5*(y_train.label[t]+1))-p[t])/w[t])

coef = np.empty([2,M-1])
newloss = np.empty([M-1, 1])

for j in range(0,M-1):
    Xj = X_train.values[:,j+1]
    a = sum(w)
    b = sum(w*Xj)
    c = sum(w*Xj**2)
    d = sum(w*z)
    e = sum(w * Xj * z)

    # WLS estimation
    if (a*c)-b**2 ==0:
        Bj = np.array([(d/a), 0])
    else:
        Bj = ( 1 / (a*c-b**2) ) * np.array([c * d - b * e, a * e - b* d])

    # New H(X)
    Hj = H + .5*(Bj[0] + Bj[1] * Xj)

    # New loss function value
    lossj = sum(np.log(1 + np.exp(-2*y_train.label * Hj)))

    coef[:, j]=Bj
    newloss[j]=lossj

Jhat = np.argmin(newloss) # [1:len(newloss)]

weightsall[0] = weightsall[0] + .5*coef[0,Jhat]
weightsall[Jhat+1] = weightsall[Jhat+1]+.5*coef[1,Jhat]
loss[300-1] = newloss[Jhat]

# Making predictions
pred_train = np.sign(np.dot(X_train,weightsall))
pred_test = np.sign(np.dot(X_test,weightsall))
accuracy_train=metrics.accuracy_score(pred_train, y_train)
accuracy_test=metrics.accuracy_score(pred_test, y_test)
print("--- %s seconds ---" % (time.time() - start_time))

# Confusion matrix from model
confusion_matrix = confusion_matrix(y_test, pred_test)
print(confusion_matrix)
cohen_kappa_score(y_test, pred_test)
```

```
### ////////////////////////////////////////////////////////////////////////////////
### //////////////////////////// NeuralNetworks ///////////////////////////
### ////////////////////////////////////////////////////////////////////////////////

# Reading in autism data set
autism = pd.read_csv("AUTISMVARS_FinalTAKE2.csv")
autism_labels = pd.read_csv("AUTISMLABELS_FinalTake2.csv")
# Changing labels to -1/1 instead of 0/1
autism_labels[autism_labels == 0] = -1

#Split data into train and test
X_train_orig, X_test, y_train_orig, y_test = train_test_split(autism, autism_labels,
test_size=0.2, random_state = 1234)

# Oversample training data set
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
X_resampled, y_resampled = ros.fit_resample(X_train_orig, np.ravel(y_train_orig))


X_train = X_resampled
y_train = y_resampled

def neuralnets(k, lr, train_x, test_x, train_y, test_y):
   accuracy_train = np.empty([10, 1]) # MAKE SURE THESE ARE ZERO BEFORE
RUNNING LOOP
   accuracy_test = np.empty([10, 1])

   # Normalize data set
   scaler = preprocessing.StandardScaler().fit(train_x)
   train_x = pd.DataFrame(scaler.transform(train_x))

   # Transform test data set
   test_x = pd.DataFrame(scaler.transform(test_x))

   # Creating a model
   model = Sequential()
   model.add(Dense(k, input_dim=train_x.shape[1], activation='relu'))
   model.add(Dense(k, input_dim=train_x.shape[1], activation='relu'))
   model.add(Dense(k, input_dim=train_x.shape[1], activation='relu'))
   model.add(Dense(k, input_dim=train_x.shape[1], activation='relu'))
   model.add(Dense(1))

   # Compiling model
   sgd = optimizers.SGD(lr=lr)
   model.compile(loss='mean_squared_error', optimizer=sgd, metrics=['accuracy'])
```

```
    # Training a model
    results = model.fit(train_x, train_y, validation_data = (test_x, test_y), epochs=100,
batch_size=10)
    preds = model.predict_classes(test_x, verbose=1)
    accuracy_train = results.history['acc'][99]
    accuracy_test = results.history['val_acc'][99]
    return accuracy_train, accuracy_test, preds, test_y

# FINAL ONE
#batch size 10 epochs 100
start_time = time.time()
mini_k32_2layers = neuralnets(32, .1, train_x=X_train, train_y=y_train,
                              test_x = X_test, test_y = y_test)
preds = mini_k32_2layers[2]
preds[preds == 0] = -1

ytest = mini_k32_2layers[3]

# Confusion matrix for model
confusion_matrix = confusion_matrix(np.ravel(ytest), preds)
print(confusion_matrix)
print("--- %s seconds ---" % (time.time() - start_time))
cohen_kappa_score(y_test, preds)


### ///////////////////////////////////////////////////////////////////////////////////////////////
### ///////////////// Take 4: Oversampling with SMOTE ////////////////////
### ///////////////////////////////////////////////////////////////////////////////////

# Reading in autism data set
autism = pd.read_csv("AUTISMVARS_FinalTAKE2.csv")
autism_labels = pd.read_csv("AUTISMLABELS_FinalTake2.csv")
autism_total = pd.concat([autism, autism_labels], axis=1)

#Split data into train and test
X_train_orig, X_test, y_train_orig, y_test = train_test_split(autism, autism_labels,
test_size=0.2, random_state = 1234)

# Oversampling using SMOTE
sm = SMOTE(random_state=12, ratio = 1.0)
X_resampled, y_resampled = sm.fit_sample(X_train_orig, np.ravel(y_train_orig))

X_train = X_resampled
y_train = y_resampled
```

```
no_aut = len(y_resampled[y_resampled == 1]) #489
aut = len(y_resampled[y_resampled == 0]) #489


### ////////////////////////////////////////////////////////////////////////////////
### ////////////////////////// Logistic Regression //////////////////////////
### ////////////////////////////////////////////////////////////////////////////////

start_time = time.time()
# Fitting a logistic regression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

# Making predictions
y_pred = logreg.predict(X_test)
print('Accuracy of logistic regression classifier on train set:
{:.2f}'.format(logreg.score(X_train, y_train)))
print('Accuracy of logistic regression classifier on test set:
{:.2f}'.format(logreg.score(X_test, y_test)))
print("--- %s seconds ---" % (time.time() - start_time))
# Plotting the ROC Curve
logit_roc_auc = roc_auc_score(y_test, logreg.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, logreg.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()

# Confusion matrix for model
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)

### ////////////////////////////////////////////////////////////////////////////////
### ////////////////////////////// Decision Trees //////////////////////////////
### ////////////////////////////////////////////////////////////////////////////////

# Creating a function called Misclassplot that will input the four sets
# of data and a depth and then plot the misclassification vs. depth of the
# tree for both the training and testing data set.
```

```
def DECISION_TREES(train, trainlab, test, testlab, depth, data):

    # Running a for loop to create data frames of the misclassification errors
    mis_train=[] # Initializing training misclassification list
    mis_test=[] # Initializing testing misclassification list
    for x in range(1,depth+1):
        dec_tree=DecisionTreeClassifier(max_depth=x) # Max_Depth attribute
        dec_tree=dec_tree.fit(train, trainlab) # Growing out the decision tree

        # Predicting on training data set / calculating misclassification error
        train_pred=dec_tree.predict(train)
        mis_train.append(100- (accuracy_score(trainlab,train_pred)*100))

        # Predicting on testing data set / calculating misclassification error
        test_pred=dec_tree.predict(test)
        mis_test.append(100- (accuracy_score(testlab,test_pred)*100))

    # Putting the misclassification errors into data frames
    mis_train = 100 - pd.DataFrame(mis_train)
    mis_train.columns=['Error']
    print(mis_train)
    mis_test = 100- pd.DataFrame(mis_test)
    mis_test.columns = ['Error']
    print(mis_test)

    # Plotting the misclassification error vs depth of tree for both testing/training
    plt.plot(range(1,1+len(mis_train)), mis_train, linestyle='-', marker='o')
    plt.plot(range(1,1+len(mis_test)), mis_test, linestyle='-', marker='o')
    plt.legend(['Training Data', 'Testing Data'], loc='upper left')
    plt.ylabel("Accuracy Rate (%)")
    plt.xlabel("Depth of Tree")
    plt.title("Accuracy Rates Vs. Depth of Trees - " + str(data))
    plt.axis([0,depth,50,100])
    plt.show()

    # Printing out the minimum classification error for testing data set
    max = np.array([[pd.DataFrame.max(mis_test.Error),
                pd.Series.idxmax(mis_test)+1]])
    headers = ['Maximum Accuracy Rate', 'Tree Depth']
    table = tabulate(max, headers, tablefmt="fancy_grid")
    print(table)
    return table;


# Running the function on the data with max depth from 1-24 decision trees
```

```
start_time = time.time()
a = DECISION_TREES(train=X_train, trainlab=y_train,
            test=X_test,testlab=y_test,
            depth=24, data = "Autism" )
print("--- %s seconds ---" % (time.time() - start_time))

# CHOSE TREE DEPTH 18, LARGEST ACCURACY BEFORE OVERFITTING

# Running with tree depth already chosen
start_time = time.time()

dec_tree=DecisionTreeClassifier(max_depth=18) # Max_Depth attribute
dec_tree=dec_tree.fit(X_train, y_train) # Growing out the decision tree

# Predicting on training data set / calculating misclassification error
train_pred=dec_tree.predict(X_train)
mis_train = (accuracy_score(y_train,train_pred) * 100)

# Predicting on testing data set / calculating misclassification error
test_pred=dec_tree.predict(X_test)
mis_test= (accuracy_score(y_test,test_pred)*100)

print(mis_train)
print(mis_test)
print("--- %s seconds ---" % (time.time() - start_time))

# Confusion matrix for model
confusion_matrix = confusion_matrix(y_test, test_pred)
print(confusion_matrix)

### ////////////////////////////////////////////////////////////////////////////////
### ////////////////////////// Random Forests ///////////////////////////
### ///////////////////////////////////////////////////////////////////////////////

def RandomTreePlot(train, trainlab, valid, validlab, features):
   # Defining the number of trees (k)
   #k= [1,10, 20, 40,  60,  80,100, 150, 200, 250, 300]
   k = [20]
   # Initialization of misclassification error matrix
   d = np.zeros((len(k),3))
   d[:,0] = k
   Error_train = np.zeros(len(k))
   Error_test = np.zeros(len(k))

   # Creating random forests with k number of trees and the split attribute
   # specified for TRAINING data
```

```
    for x in range(0,len(k)):
        rand_forest=RandomForestClassifier(n_estimators=k[x], max_features=features)
        rand_forest = rand_forest.fit(train, trainlab)
        predictions = rand_forest.predict(train)
        Error_train[(x)] = metrics.accuracy_score(trainlab,predictions) * 100

    # Creating random forests with k number of trees and the split attribute
    # specified for TESTING data
    for x in range(0,len(k)):
        rand_forest=RandomForestClassifier(n_estimators=k[x], max_features=features)
        rand_forest = rand_forest.fit(train, trainlab)
        predictions = rand_forest.predict(valid)
        Error_test[(x)] = metrics.accuracy_score(validlab, predictions) * 100

    # Assignment of misclassification errors for training/testing data sets to
    # misclassification error matrix
    d[:,1] = Error_train
    d[:,2] = Error_test

    # Plotting the misclassification error vs number of trees for both
    # testing/training
    plt.plot(d[:,0], d[:,1], linestyle='-', marker='o')
    plt.plot(d[:,0], d[:,2], linestyle='-', marker='o')
    plt.legend(['Training Data', 'Testing Data'], loc='lower right')
    plt.ylabel("Accuracy Rate (%)")
    plt.xlabel("Number of Tree")
    plt.title("Accuracy Rate Vs. Number of Trees")
    plt.show()

    headers = ['Tree Depth', 'Accuracy Rate - Train', "Accuracy Rate - Test",]
    table = tabulate(d, headers, tablefmt="fancy_grid")
    print(table)
    return predictions, rand_forest


# The split attribute being chosen from a random subset of sqrt(500) features
start_time = time.time()
preds = RandomTreePlot(train=X_train, trainlab=y_train,
            valid=X_test,validlab=y_test, features="sqrt")
print("--- %s seconds ---" % (time.time() - start_time))

# Confusion matrix for model
confusion_matrix = confusion_matrix(y_test, preds[0])
print(confusion_matrix)
```

```
### ///////////////////////////////////////////////////////////////////////////
### ////////////////////////////// LogitBoost /////////////////////////////
### ///////////////////////////////////////////////////////////////////////////
# Reading in autism data set
autism = pd.read_csv("AUTISMVARS_FinalTAKE2.csv")
autism_labels = pd.read_csv("AUTISMLABELS_FinalTake2.csv")
# Changing labels to -1/1 instead of 0/1
autism_labels[autism_labels == 0] = -1

#Split data into train and test
X_train_orig, X_test, y_train_orig, y_test = train_test_split(autism, autism_labels,
test_size=0.2, random_state = 1234)

# Oversample training data set
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
X_resampled, y_resampled = ros.fit_resample(X_train_orig, np.ravel(y_train_orig))

X_train = X_resampled
y_train = y_resampled

#X_train.reset_index(drop=True, inplace=True)
X_test.reset_index(drop=True, inplace=True)

# Adding a column of 1's to xtrain and xtest data sets
X_train = pd.concat([pd.DataFrame([1]*len(X_train)), pd.DataFrame(X_train)], axis=1)
X_test = pd.concat([pd.DataFrame([1]*len(X_test)), X_test], axis=1)

# Creating a label column heading
y_train = pd.DataFrame(y_train)
y_train.columns = ['label']

# Initializing N and M
N = len(X_train)
M = X_train.shape[1]

# Iteration
weightsall = np.transpose(np.zeros(M))
loss = np.empty([300, 1])
accuracy_train = np.empty([4, 1]) # MAKE SURE THESE ARE ZERO BEFORE
RUNNING LOOP
accuracy_test = np.empty([4, 1]) # MAKE SURE THESE ARE ZERO BEFORE
RUNNING LOOP
ik = 1
z = np.zeros(N)
```

```
start_time = time.time()

# Loop
for i in range(1,301):
    H = np.dot(X_train, weightsall)
    p = (1/(1+np.exp(-2*H)))
    w = p*(1-p)
    y_train.reset_index(drop=True, inplace=True)
    for t in range(len(w)):
        if w[t] == 0:
            z[t] = 0
        else:
            z[t] = (((.5*(y_train.label[t]+1))-p[t])/w[t])

    coef = np.empty([2,M-1])
    newloss = np.empty([M-1, 1])

    for j in range(0,M-1):
        Xj = X_train.values[:,j+1]
        a = sum(w)
        b = sum(w*Xj)
        c = sum(w*Xj**2)
        d = sum(w*z)
        e = sum(w * Xj * z)

        # WLS estimation
        if (a*c)-b**2 ==0:
            Bj = np.array([(d/a), 0])
        else:
            Bj = ( 1 / (a*c-b**2) ) * np.array([c * d - b * e, a * e - b* d])

        # New H(X)
        Hj = H + .5*(Bj[0] + Bj[1] * Xj)

        # New loss function value
        lossj = sum(np.log(1 + np.exp(-2*y_train.label * Hj)))

        coef[:, j]=Bj
        newloss[j]=lossj

    Jhat = np.argmin(newloss) # [1:len(newloss)]

    weightsall[0] = weightsall[0] + .5*coef[0,Jhat]
    weightsall[Jhat+1] = weightsall[Jhat+1]+.5*coef[1,Jhat]
    loss[i-1] = newloss[Jhat]
    # Making predictions
    pred_train = np.sign(np.dot(X_train,weightsall))
    pred_test = np.sign(np.dot(X_test,weightsall))

    # Calculating error rate
    if i in (10,30, 100, 300):
        accuracy_train[ik-1]=metrics.accuracy_score(pred_train, y_train)
        accuracy_test[ik-1]=metrics.accuracy_score(pred_test, y_test)
        ik = ik+1

    if i==300:
        plt.plot(range(1,301), loss)
        plt.title("Loss Plot")

print("--- %s seconds ---" % (time.time() - start_time))

start_time = time.time()
# Computational time for just i = 300
H = np.dot(X_train, weightsall)
p = (1/(1+np.exp(-2*H)))
w = p*(1-p)
y_train.reset_index(drop=True, inplace=True)
for t in range(len(w)):
    if w[t] == 0:
        z[t] = 0
    else:
        z[t] = (((.5*(y_train.label[t]+1))-p[t])/w[t])

coef = np.empty([2,M-1])
newloss = np.empty([M-1, 1])

for j in range(0,M-1):
    Xj = X_train.values[:,j+1]
    a = sum(w)
    b = sum(w*Xj)
    c = sum(w*Xj**2)
    d = sum(w*z)
    e = sum(w * Xj * z)

    # WLS estimation
    if (a*c)-b**2 ==0:
        Bj = np.array([(d/a), 0])
    else:
        Bj = ( 1 / (a*c-b**2) ) * np.array([c * d - b * e, a * e - b* d])

    # New H(X)
    Hj = H + .5*(Bj[0] + Bj[1] * Xj)

    # New loss function value
    lossj = sum(np.log(1 + np.exp(-2*y_train.label * Hj)))

    coef[:, j]=Bj
    newloss[j]=lossj

Jhat = np.argmin(newloss) # [1:len(newloss)]

weightsall[0] = weightsall[0] + .5*coef[0,Jhat]
weightsall[Jhat+1] = weightsall[Jhat+1]+.5*coef[1,Jhat]
loss[300-1] = newloss[Jhat]

# Making predictions
pred_train = np.sign(np.dot(X_train,weightsall))
pred_test = np.sign(np.dot(X_test,weightsall))
accuracy_train=metrics.accuracy_score(pred_train, y_train)
accuracy_test=metrics.accuracy_score(pred_test, y_test)
print("--- %s seconds ---" % (time.time() - start_time))

# Confusion matrix from model
confusion_matrix = confusion_matrix(y_test, pred_test)
print(confusion_matrix)

### ////////////////////////////////////////////////////////////////////////////////
### //////////////////////////// NeuralNetworks ////////////////////////////
### ///////////////////////////////////////////////////////////////////////////////

# Reading in autism data set
autism = pd.read_csv("AUTISMVARS_FinalTAKE2.csv")
autism_labels = pd.read_csv("AUTISMLABELS_FinalTake2.csv")
# Changing labels to -1/1 instead of 0/1
autism_labels[autism_labels == 0] = -1

#Split data into train and test
X_train_orig, X_test, y_train_orig, y_test = train_test_split(autism, autism_labels,
test_size=0.2, random_state = 1234)

# Oversample training data set
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
X_resampled, y_resampled = ros.fit_resample(X_train_orig, np.ravel(y_train_orig))


X_train = X_resampled
```

```
y_train = y_resampled

def neuralnets(k, lr, train_x, test_x, train_y, test_y):
    accuracy_train = np.empty([10, 1]) # MAKE SURE THESE ARE ZERO BEFORE
RUNNING LOOP
    accuracy_test = np.empty([10, 1])

    # Normalize data set
    scaler = preprocessing.StandardScaler().fit(train_x)
    train_x = pd.DataFrame(scaler.transform(train_x))

    # Transform test data set
    test_x = pd.DataFrame(scaler.transform(test_x))

    # Creating a model
    model = Sequential()
    model.add(Dense(k, input_dim=train_x.shape[1], activation='relu'))
    model.add(Dense(k, input_dim=train_x.shape[1], activation='relu'))
    model.add(Dense(k, input_dim=train_x.shape[1], activation='relu'))
    model.add(Dense(k, input_dim=train_x.shape[1], activation='relu'))
    model.add(Dense(1))

    # Compiling model
    sgd = optimizers.SGD(lr=lr)
    model.compile(loss='mean_squared_error', optimizer=sgd, metrics=['accuracy'])

    # Training a model
    results = model.fit(train_x, train_y, validation_data = (test_x, test_y), epochs=100,
batch_size=10)
    preds = model.predict_classes(test_x, verbose=1)
    accuracy_train = results.history['acc'][99]
    accuracy_test = results.history['val_acc'][99]
    return accuracy_train, accuracy_test, preds, test_y

# FINAL ONE
#batch size 10 epochs 100
start_time = time.time()
mini_k32_2layers = neuralnets(32, .1, train_x=X_train, train_y=y_train,
                              test_x = X_test, test_y = y_test)
preds = mini_k32_2layers[2]
ytest = mini_k32_2layers[3]

# Confusion matrix from model
confusion_matrix = confusion_matrix(np.ravel(ytest), preds)
print(confusion_matrix)
print("--- %s seconds ---" % (time.time() - start_time))
```

```
### ////////////////////////////////////////////////////////////////////////////////////////////////////
### ////// Take 5: Oversampling and Logistic Regression with Threshold /////
### ////////////////////////////////////////////////////////////////////////////////////////////////////

# Reading in autism data set
autism = pd.read_csv("AUTISMVARS_FinalTAKE2.csv")
autism_labels = pd.read_csv("AUTISMLABELS_FinalTake2.csv")
autism_total = pd.concat([autism, autism_labels], axis=1)

#Split data into train and test
X_train_orig, X_test, y_train_orig, y_test = train_test_split(autism, autism_labels,
test_size=0.2, random_state = 1234)

# Oversample training data set
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(ratio=float)
X_resampled, y_resampled = ros.fit_resample(X_train_orig, y_train_orig)

X_train = X_resampled
y_train = y_resampled

clf = LogisticRegression()
clf.fit(X_train, y_train)
THRESHOLD = 0.74
preds = np.where(clf.predict_proba(X_test)[:,1] > THRESHOLD, 1, 0)

pd.DataFrame(data=[accuracy_score(y_test, preds), recall_score(y_test, preds),
                   precision_score(y_test, preds), roc_auc_score(y_test, preds)],
             index=["accuracy", "recall", "precision", "roc_auc_score"])


confusion_matrix = confusion_matrix(y_test, preds)
print(confusion_matrix)

cohen_kappa_score(y_test, preds)
```