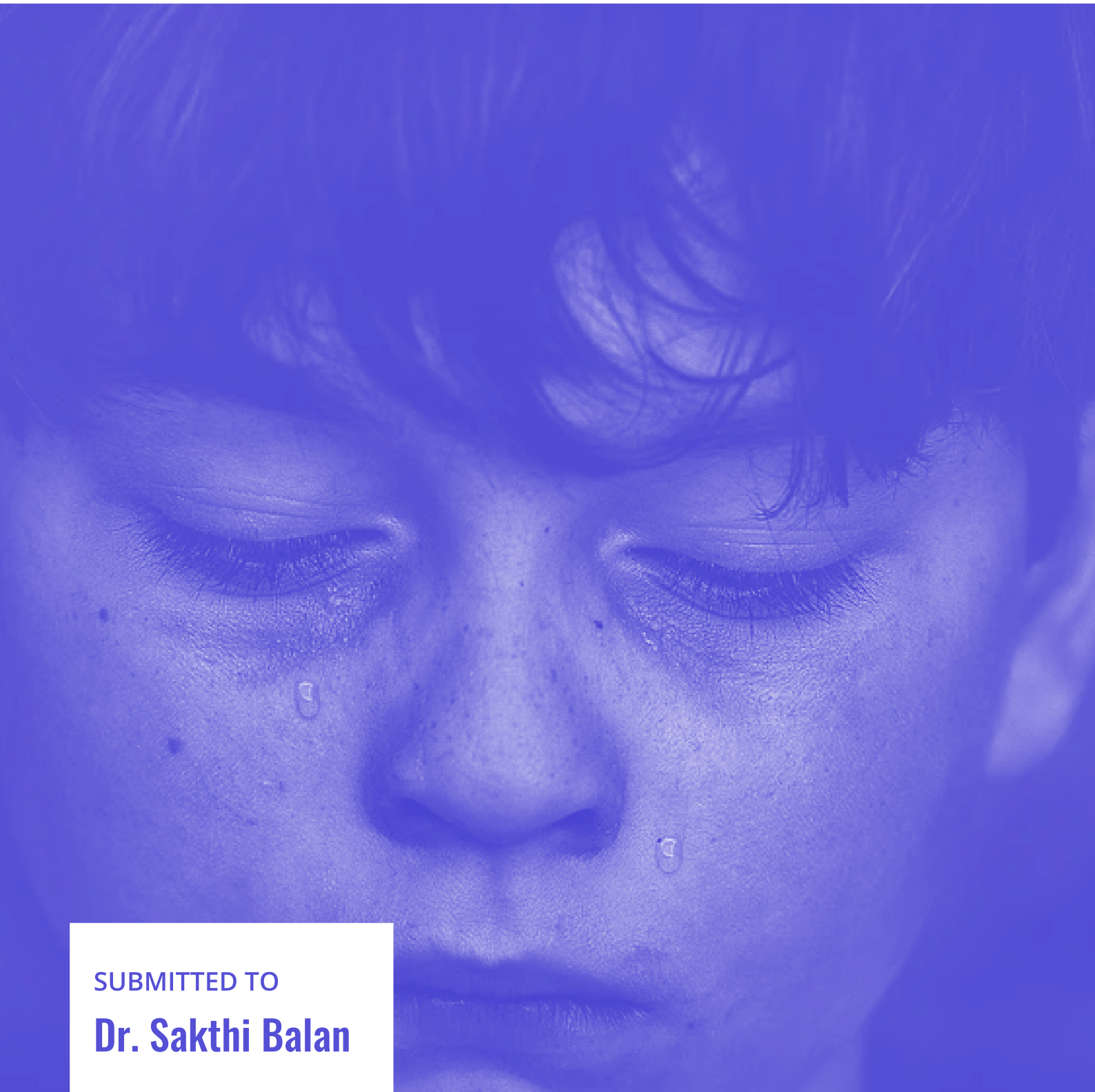
Machine Learning approaches to the

Autism Spectrum Disorder Classification



SUBMITTED TO

Dr. Sakthi Balan

Group Members:

Shreshth Bhatt(18ucc012)
Aashish Methani(18ucc085)
Shreyans Jain (18ucc151)
Divyansh Jain(18ucs218)

INDEX

Objective of the Project

For the autism screening of adults with 20 traits, a new data collection for more research in particular the classification of prominent autistic symptoms and the further classification of ASD cases, is proposed. Ten behavioural characteristics (AQ-10-Adult) were identified, plus ten-person characteristics in the detection of ASD cases that were the product of behavioural science controls.

Dataset Name and Source

Dataset : Adult Data Set of Autism Screening.
Source: UCI Machine Learning Repository.
Dataset URL :
https://raw.githubusercontent.com/Shreyans145/IDS/main/csv_result-Autism-Adult-Data.csv

Specifications of the Dataset

Abstract: Autistic Spectrum Disorder Screening Data for Adult. This dataset is related to classification and predictive tasks.

Data Set Characteristics:	N/A	**Number of Instances:**	704	**Area:**	Social
Attribute Characteristics:	Integer	**Number of Attributes:**	21	**Date Donated**	2017-12-24
Associated Tasks:	Classification	**Missing Values?**	Yes	**Number of Web Hits:**	65143

Libraries used

```
import numpy as np
import pandas as pd
from time import time
import matplotlib.pyplot as plt

%matplotlib inline
```

Data Preprocessing

Data preprocessing is a technique for data mining that involves the translation of raw data into a comprehensible format. In this process, the information used was pre-processed and all NaN values were eliminated..

Labeling Data(one hot coding)

We will see that each record has several non-numeric attributes, such as country of residence, nationality, etc., from the available data collection. Generally, learning algorithms assume the data to be numerical, also known as categorical factors, such as the translation of non-numeric characteristics. With the one-hot encoding method, one common way to convert categorical variables is.
For each potential shape of each non-numeric function, "A "dummy" vector is generated by One-Hot Encoding. For example, assume all possible entries for functions: A, B, or C. Then inside someFeatureA, someFeatureB, and someFeatureC, this functionality is encoded.

	someFeature		someFeature_A	someFeature_B	someFeature_C
0	B		0	1	0
1	C	----> one-hot encode ---->	0	0	1
2	A		1	0	0

```
[19] #One-hot encode the 'features_minmax_transform' data using pandas.get_dummies()
     features_final = pd.get_dummies(features_minmax_transform)
     display(features_final.head(5))


     # Encode the 'all_classes_raw' data to numerical values
     asd_classes = asd_raw.apply(lambda x: 1 if x == 'YES' else 0)



     # Print the number of features after one-hot encoding
     encoded = list(features_final.columns)
     print "{} total features after one-hot encoding.".format(len(encoded))

     # Uncomment the following line to see the encoded feature names
     print encoded
```

	age	result	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	...	contry_of_res_United Arab Emirates	contry_of_res_
0	0.024590	0.6	1	1	1	1	0	0	1	1	...	0	
1	0.019126	0.5	1	1	0	1	0	0	0	1	...	0	
2	0.027322	0.8	1	1	0	1	1	0	1	1	...	0	
3	0.049180	0.6	1	1	0	1	0	0	1	1	...	0	
5	0.051913	0.9	1	1	1	1	1	0	1	1	...	0	

5 rows × 94 columns

```
94 total features after one-hot encoding.
['age', 'result', 'A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score', 'A6_Score', 'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score'
```

Implementing the algorithms

1. Decision Tree

We will start building a DecisionTreeClassifier and add it to research performance.

The decision tree is the most effective and common process for classification and estimation. A Decision tree is a tree-like flowchart where each internal node is an attribute query, each branch is a test result, and there is a class name for each leaf node (terminal node).

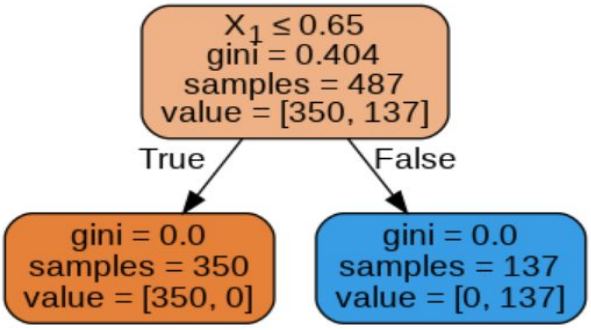
```
[22] from sklearn import tree
     from sklearn.tree import DecisionTreeClassifier

     dectree = DecisionTreeClassifier(random_state=1)

     # Train the classifier on the training set
     dectree.fit(X_train, y_train)
```

```
     DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                            max_features=None, max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_impurity_split=None,
                            min_samples_leaf=1, min_samples_split=2,
                            min_weight_fraction_leaf=0.0, presort=False, random_state=1,
                            splitter='best')
```

Depiction of the Decision Tree algorithm

Decision trees classify the instances by sorting them down the tree to a certain root leaf node, which provides the case name. The example is defined by evaluating, beginning with the tree root node, the attribute defined by this node, but instead moving down the tree branch corresponding to the value of the attribute, as seen in the figure above. This procedure is then replicated for a subtree rooted in a new node.

```
[64] import pydotplus


     dot_data = tree.export_graphviz(dectree,
                                     out_file=None,
                                     filled=True,
                                     rounded=True,
                                     special_characters=True)
     graph = pydotplus.graph_from_dot_data(dot_data)

     from IPython.display import Image
     Image(graph.create_png())
```



Evaluating Model Performance

Metrics

The F-beta score can be used as a measure that considers both accuracy and memory.

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

In fact, when β=0.5 is given more emphasis on precision. This is called the score of the F- (or F-score for simplicity).

Note: Retrieval of accuracy, precision, recall

Accuracy

It checks the degree to which the proper inference is made by the classifier. That is the ratio between the number of reliable predictions and the total number of forecasts (the number of test data points).

Precision

It tells us what percentage of the messages we called spam is in fact spam. That is the ratio of real positive terms to all positive ones (words that are simply spam and labelled as spam) (all words categorised as spam, regardless of whether it was the right classification), in other words, it is the ratio of

[True Positives/(True Positives + False Positives)]

Sensitivity

It shows us how many of the messages that were actually spam were categorised as spam by us. This is the proportion of true positives of all phrases that were theoretically spam , in other words, the spam ratio.

[True Positives/(True Positives + False Negatives)]

Issues in classification that are biassed in their classification distributions, as in our case where we have

- a total of 609 records with
- 180 individuals diagnosed with ASD and
- 429 individuals not diagnosed with ASD

Accuracy in itself is not a really good metric. In this case, however, precision and memory are very helpful. To obtain the F1 score, which is the weighted average (harmonic mean) of the precision and recall results, these two parameters can be mixed. This ranking will vary from 0 to 1, with 1 being the highest possible F1 rating .

```
[140] # make class predictions for the testing set
      y_pred_class = dectree.predict(X_test)
```

```
[141] # print the first 25 true and predicted responses
      print('True:', y_test.values[0:25])
      print('False:', y_pred_class[0:25])

      ('True:', array([1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1,
             0, 1, 0]))
      ('False:', array([1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1,
             0, 1, 0]))
```

```
[142] from sklearn import metrics
      # IMPORTANT: first argument is true values, second argument is predicted values
      # this produces a 2x2 numpy array (matrix)
      #print(metrics.confusion_matrix(y_test, y_pred_class))

      # save confusion matrix and slice into four pieces
      confusion = metrics.confusion_matrix(y_test, y_pred_class)
      print(confusion)
      #[row, column]
      TP = confusion[1, 1]
      TN = confusion[0, 0]
      FP = confusion[0, 1]
      FN = confusion[1, 0]
```

```
[[79  0]
 [ 0 43]]
```

Metrics computed from a confusion matrix

Accuracy of Classification:

In general, how much is the classifier correct?

```
[143] # use float to perform true division, not integer division
      print((TP + TN) / float(TP + TN + FP + FN))

      1.0
```

Error while Classifying:

How much in general, is the classifier incorrect?

```
[144] classification_error = (FP + FN) / float(TP + TN + FP + FN)

      print(classification_error)

      0.0
```

Sensitivity:

How far is the forecast accurate because the actual value is positive?

```
[145] sensitivity = TP / float(FN + TP)

      print(sensitivity)
      print(metrics.recall_score(y_test, y_pred_class))

      1.0
      1.0
```

Specificity:

If the true value is negative, how much is the right forecast?

```
[146] specificity = TN / (TN + FP)

      print(specificity)

      1
```

False Positive Rate:

If the actual value is negative, how much is wrong with the prediction?

```
[147] false_positive_rate = FP / float(TN + FP)

      print(false_positive_rate)
      #print(1 - specificity)

      0.0
```

Precision:

If a positive value is expected, how much is the prediction correct?

```
[148] precision = TP / float(TP + FP)

      #print(precision)
      print(metrics.precision_score(y_test, y_pred_class))

      1.0
```

Visualizing the prediction for classification:

```
[149] # print the first 10 predicted responses
      # 1D array (vector) of binary values (0, 1)
      dectree.predict(X_test)[0:10]

      array([1, 0, 0, 1, 1, 0, 1, 1, 0, 0])
```

```
[150] # print the first 10 predicted probabilities of class membership
      dectree.predict_proba(X_test)[0:10]
```

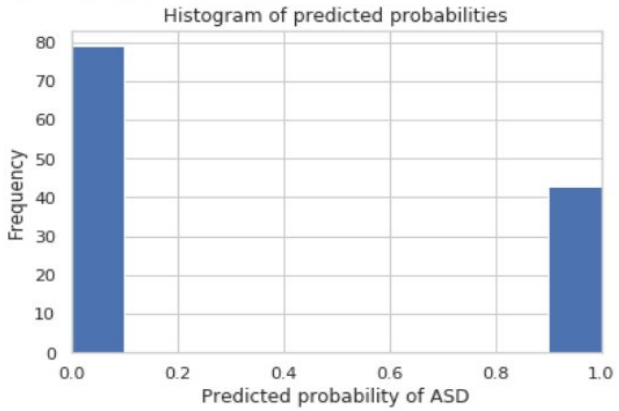
```
array([[0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.]])
```

```
[151] # store the predicted probabilities for class 1
      y_pred_prob = dectree.predict_proba(X_test)[:, 1]
```

```
[152] # allow plots to appear in the notebook

      import matplotlib.pyplot as plt

      # adjust the font size
      plt.rcParams['font.size'] = 12
```

```
[153] # histogram of predicted probabilities

      # 8 bins
      plt.hist(y_pred_prob, bins=10)

      # x-axis limit from 0 to 1
      plt.xlim(0,1)
      plt.title('Histogram of predicted probabilities')
      plt.xlabel('Predicted probability of ASD')
      plt.ylabel('Frequency')
```

```
Text(0,0.5,'Frequency')
```



Cross-validation

Now, instead of a single train/test break, we use K-Fold cross-validation to get a better estimate of the accuracy of your model (K=10).

```
[156] from sklearn.model_selection import cross_val_score

      dectree = DecisionTreeClassifier(random_state=1)

      cv_scores = cross_val_score(dectree, features_final, asd_classes, cv=10)

      cv_scores.mean()

      1.0
```

AUC Score:

AUC is the proportion of the ROC plot below the curve.

```
[157] # calculate cross-validated AUC
      from sklearn.model_selection import cross_val_score
      cross_val_score(dectree, features_final, asd_classes, cv=10, scoring='roc_auc').mean()

      1.0
```

F-Beta Score:

```
[158] dectree.fit(X_train, y_train)
      from sklearn.metrics import fbeta_score
      predictions_test = dectree.predict(X_test)
      fbeta_score(y_test, predictions_test, average='binary', beta=0.5)

      1.0
```

2. SVM

We're going to try using SVM next. SVC to see how strong it is compared to the decision tree for a linear kernel. A supervised machine learning model that uses classification algorithms for two-group classification problems is the Support Vector Machine (SVM). You will group the new text by including a list of training data named for each category in the SVM model. But you're working on a problem with text classification.

```
[162] from sklearn import svm

      C = 1.0
      svc = svm.SVC(kernel='linear', C=C, gamma=2)
```

```
[163] cv_scores = cross_val_score(svc, features_final, asd_classes, cv=10)

      cv_scores.mean()
```

```
      1.0
```

AUC Score:

AUC is the proportion of the ROC plot below the curve.

```
[164] # calculate cross-validated AUC
      from sklearn.model_selection import cross_val_score
      cross_val_score(svc, features_final, asd_classes, cv=10, scoring='roc_auc').mean()
```

```
      1.0
```

F-Beta Score:

```
[165] svc.fit(X_train, y_train)
      from sklearn.metrics import fbeta_score
      predictions_test = svc.predict(X_test)
      fbeta_score(y_test, predictions_test, average='binary', beta=0.5)
```

```
      1.0
```

3. K-Nearest-Neighbors (KNN)

Next, we learn the **K-Nearest-Neighbors algorithm** with a starting value of K=10. Note that K is an example of a hyperparameter-a parameter on the model itself that needs to be adapted to your particular data set for the maximum performance.
A basic algorithm that stores all available cases and classifies new cases on the basis of a similarity measure is K nearest neighbours (e.g., distance functions). As a device that is non-parametric,

```
[166] from sklearn import neighbors

      knn = neighbors.KNeighborsClassifier(n_neighbors=10)
      cv_scores = cross_val_score(knn, features_final, asd_classes, cv=10)

      cv_scores.mean()
```

```
0.9474590163934427
```

AUC Score:

AUC is the proportion of the ROC plot below the curve.

```
[167] # calculate cross-validated AUC
      from sklearn.model_selection import cross_val_score
      cross_val_score(knn, features_final, asd_classes, cv=10, scoring='roc_auc').mean()
```

```
0.9930078749846192
```

F-Beta Score:

```
[168] knn.fit(X_train, y_train)
      from sklearn.metrics import fbeta_score
      predictions_test = knn.predict(X_test)
      fbeta_score(y_test, predictions_test, average='binary', beta=0.5)
```

```
0.9192825112107623
```

It's tricky to pick K, so we can't discard KNN until we've tried multiple K values. Therefore, we write a KNN loop with K values ranging from 10 to 50 to see if K makes a major difference.

```
[169] for n in range(10, 50):
          knn = neighbors.KNeighborsClassifier(n_neighbors=n)
          cv_scores = cross_val_score(knn, features_final, asd_classes, cv=10)
          print (n, cv_scores.mean())
```

```
(10, 0.9474590163934427)
(11, 0.9507377049180328)
(12, 0.9507377049180328)
(13, 0.9540437158469945)
(14, 0.9507650273224044)
(15, 0.944207650273224)
(16, 0.9507650273224044)
(17, 0.9523770491803278)
(18, 0.9523770491803278)
(19, 0.9540163934426229)
(20, 0.9523770491803278)
(21, 0.9523770491803278)
(22, 0.9474590163934424)
(23, 0.9490983606557375)
(24, 0.9507377049180326)
(25, 0.9507377049180328)
(26, 0.9523770491803278)
(27, 0.9507377049180328)
(28, 0.9507377049180326)
(29, 0.9507377049180328)
(30, 0.9523770491803278)
(31, 0.9474863387978143)
(32, 0.9491256830601094)
(33, 0.9474863387978143)
(34, 0.9507650273224044)
(35, 0.9491256830601094)
(36, 0.9491256830601091)
(37, 0.9507650273224044)
(38, 0.9540710382513661)
(39, 0.9524316939890708)
(40, 0.9524316939890708)
(41, 0.9524316939890708)
(42, 0.9507923497267757)
(43, 0.9507923497267757)
(44, 0.9507923497267757)
(45, 0.9507923497267757)
(46, 0.9524316939890708)
(47, 0.9524316939890708)
(48, 0.9557103825136611)
(49, 0.9524316939890708)
```

4. Naive Bayes

I'm trying Naive Bayes now. MultinomialNB classifier and the question of how its performance holds up. A set of classification algorithms based on the Bayes' Theorem are **Naive Bayes classifiers**. It is not a single algorithm, but a family of algorithms where a general concept is shared by all of them, i.e. each pair of features classified is **independent** of each other.

```
[170] from sklearn.naive_bayes import MultinomialNB

      #scaler = preprocessing.MinMaxScaler()
      #all_features_minmax = scaler.fit_transform(all_features)

      nb = MultinomialNB()
      cv_scores = cross_val_score(nb, features_final, asd_classes, cv=10)

      cv_scores.mean()
```

```
0.885
```

AUC Score:

AUC is the proportion of the ROC plot below the curve.

```
[171] # calculate cross-validated AUC
      from sklearn.model_selection import cross_val_score
      cross_val_score(nb, features_final, asd_classes, cv=10, scoring='roc_auc').mean()
```

```
0.9445090439276485
```

F-Beta Score:

```
[172] nb.fit(X_train, y_train)
      from sklearn.metrics import fbeta_score
      predictions_test = nb.predict(X_test)
      fbeta_score(y_test, predictions_test, average='binary', beta=0.5)
```

```
0.8370044052863436
```

5. Logistic regression

Logistic regression is a mathematical model that, while several more sophisticated extensions exist, uses a logistic function to model a binary dependent variable in its simple form. Logistic regression analyses the logistic model parameters in regression analysis (a form of binary regression).

```
[186] from sklearn.linear_model import LogisticRegression

      logreg = LogisticRegression()
      cv_scores = cross_val_score(logreg, features_final, asd_classes, cv=10)
      cv_scores.mean()
```

```
0.9704371584699454
```

AUC Score:

AUC is the percentage of the ROC plot that is underneath the curve

```
[187] # calculate cross-validated AUC
      from sklearn.model_selection import cross_val_score
      cv_scores_roc = cross_val_score(logreg, features_final, asd_classes, cv=10, scoring='roc_auc').mean()
      cv_scores_roc.mean()
```

```
0.9974098683401008
```

F-Beta Score:

```
[188] logreg.fit(X_train, y_train)
      from sklearn.metrics import fbeta_score
      predictions_test = logreg.predict(X_test)
      fbeta_score(y_test, predictions_test, average='binary', beta=0.5)
```

```
0.9307359307359306
```

Conclusion

We have come to the conclusion that both models function exceptionally well with the data after exploring the ASD dataset with various types of learning algorithms.

We have used three different metrics (such as precision, AUC score and F-score) to assess the model's efficiency, and it seems that all the metrics suggested an almost flawless classification of the ASD instances. Here the number of instances after the data was cleaned was not adequate enough to assert that this model is optimal.

References

- Autism Screening Adult Data Set
- sklearn.neighbors.KNeighborsClassifier — scikit-learn 0.23.2 documentation
- K Nearest Neighbor | KNN Algorithm | KNN in Python & R
- Machine Learning Classifiers. What is classification? | by Sidath Asiri
- sklearn.metrics.roc_auc_score — scikit-learn 0.23.2 documentation
- sklearn.metrics.fbeta_score — scikit-learn 0.23.2 documentation