Autism Detection using Machine Learning

-Manohar B

Autism Detection Using Machine Learning

Project Report

Bachelor of Technology

(Computer Science Engineering)

Submitted to

LOVELY PREOFESSIONAL UNIVERSITY
PHAGWARA, PUNJAB

SUBMITTED BY

Name of the Student: Manohar Bethu
Registration Number: 12100789

Annexure-II

To whom so ever it may concern

I, **Manohar Bethu, 12100789**, hereby declare that the work done by me on "**Autism Detection using Machine Learning**", is a record of original work for the partial fulfillment of the requirements for the award of the degree, Bachelor of Technology.

Name of the Student: Manohar Bethu

Registration Number: 12100789

ACKNOWLEDGEMENT

I would like to express my special gratitude to the teacher and instructor of the course Machine Learning who provided me with the golden opportunity to learn a new technology.

I would like to thank my college Lovely Professional University for offering such a course which not only improves my programming skills, but also taught me other new technology.

Then I would like to thank my parents and friends who have helped me with their valuable suggestions and guidance for choosing this course.

Finally, I would like to thank everyone who has helped me a lot.

Table of Contents

ABSTRACT

Autism Spectrum Disorder (ASD) is a complex neurodevelopmental disorder that often goes undiagnosed due to the lack of early and accurate detection methods. This research aims to develop a machine learning model that can effectively detect autism in children using a combination of behavioral, cognitive, and demographic data. By leveraging advanced machine learning techniques, we aim to improve the early identification of ASD, enabling timely intervention and support for affected individuals. This research will contribute to the development of more efficient and accessible diagnostic tools for ASD.

OBJECTIVE

Autism Spectrum Disorder (ASD) is a complex neurological condition that affects an individual's social interactions, communication skills, and behavior. Traditional diagnostic methods, which rely on clinical assessments and standardized tests, can be time-consuming and costly. To address these challenges, machine learning techniques are being increasingly utilized to enhance the accuracy and efficiency of ASD detection. By analyzing various data inputs such as behavioral patterns, facial features, and genetic information, machine learning models can identify subtle indicators of autism that might be missed by conventional methods12.

Several machines learning algorithms, including Support Vector Machines (SVM), Random Forest Classifiers (RFC), and Convolutional Neural Networks (CNN), have shown promise in detecting ASD. These models can process large datasets to uncover patterns and correlations that are indicative of autism. For instance, deep learning models have been used to analyze facial images to detect autism with high accuracy2. The integration of machine learning in autism detection not only accelerates the diagnostic process but also reduces the burden on healthcare systems, making early intervention more accessible to those in need

INTRODUCTION

Introduction Autism Spectrum Disorder (ASD) is a developmental condition characterized by challenges in social interaction, communication, and repetitive behaviors. Early and accurate detection of ASD is crucial for providing timely interventions that can significantly improve the quality of life for individuals affected by this condition. Traditional diagnostic methods, which often involve lengthy clinical assessments and observations, can be both time-consuming and costly. To address these challenges, researchers are increasingly turning to machine learning techniques to enhance the detection and diagnosis of ASD.

Machine learning algorithms can analyze vast amounts of data to identify patterns and indicators of autism that might be overlooked by human evaluators. These algorithms utilize various data sources, including behavioral assessments, genetic information, and even facial recognition technology, to detect subtle signs of ASD. By leveraging the power of machine learning, it is possible to develop more efficient, accurate, and accessible diagnostic tools, ultimately leading to earlier diagnosis and better outcomes for individuals with autism

THEORETICAL BACKGROUND

1. What is **Machine Learning?**

Machine learning is a branch of artificial intelligence (AI) that focuses on the development of algorithms and statistical models that allow computers to learn from and make predictions or decisions based on data. Instead of explicitly programming a computer to perform a specific task, machine learning enables it to learn patterns and relationships within data to improve its performance over time.

2. Random Forest Classification

It constructs multiple decision trees during training and outputs the class that is the mode (classification) or mean (regression) of the predictions from individual trees It combines the predictions from multiple trees to improve generalization and robustness over a single decision tree. Random Forests help mitigate overfitting, improve prediction accuracy, and handle both classification and regression problems.

3. Decision Tree Classifier

A Decision Tree Classifier is a versatile machine learning algorithm that excels at making predictions based on a tree-like structure. This structure comprises nodes, where each internal

node represents a decision point based on a specific feature. These decisions branch out, leading to further nodes or leaf nodes. Leaf nodes, the terminal points of the tree, represent the final class label assigned to a data point.

4. K-Nearest Neighbor

K-Nearest Neighbors (KNN) is a straightforward, yet effective machine learning algorithm used for both classification and regression tasks. It classifies or predicts the value of a data point based on the majority class or average value of its k nearest neighbors in the feature space.

5. Support Vector Machine

At the core of SVMs is the concept of finding the optimal hyperplane that separates data points into different classes. This hyperplane is determined by a subset of data points known as support vectors, which are the data points closest to the decision boundary. The goal is to find the hyperplane that maximizes the margin between the two classes.

6. Logistic Regression

It models the probability of a binary outcome (e.g., 0 or 1, yes or no) based on one or more predictor variables. Unlike linear regression, which predicts a continuous numerical value, logistic regression predicts the probability of a specific outcome. The output of a logistic regression model is a probability value between 0 and 1, representing the likelihood of a data point belonging to a specific class.

Libraries

```
import numpy as np
import pandas as pd
from time import time
from IPython.display import display

# Import supplementary visualization code visuals.py
import visuals as vs

%matplotlib inline

data = pd.read_csv("autism_data.csv")
display(data.head(5))
```

	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	A10_Score	...	gender	ethnicity	jundice	austim	contry_
0	1	1	1	1	0	0	1	1	0	0	...	f	White-European	no	no	United
1	1	1	0	1	0	0	0	1	0	1	...	m	Latino	no	yes	
2	1	1	0	1	1	0	1	1	1	1	...	m	Latino	yes	yes	
3	1	1	0	1	0	0	1	1	0	1	...	f	White-European	no	yes	United
4	1	0	0	0	0	0	0	1	0	0	...	f	?	no	no	

5 rows × 21 columns

```
# Total number of records:
n_records = len(data.index)

# Total number of records with ASD
n_asd_yes = len(data[data['Class/ASD'] == 'YES'])

# Total number of records without ASD
n_asd_no = len(data[data['Class/ASD'] == 'NO'])

# Percentage of individuals with ASD
yes_percentage = float((n_asd_yes) / n_records * 100)

# Printing the outputs
print(f'Total number of records  : {n_records}')
print(f'Number of individuals with ASD : {n_asd_yes}')
print(f'Number of individuals without ASD : {n_asd_no}')
print("Percentage of individuals with ASD : {:.2f}%".format(yes_percentage))
```

```
Total number of records  : 704
Number of individuals with ASD : 189
Number of individuals without ASD : 515
Percentage of individuals with ASD : 26.85%
```

Feature set Exploration

```
1 data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 704 entries, 0 to 703
Data columns (total 21 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   A1_Score         704 non-null    int64
 1   A2_Score         704 non-null    int64
 2   A3_Score         704 non-null    int64
 3   A4_Score         704 non-null    int64
 4   A5_Score         704 non-null    int64
 5   A6_Score         704 non-null    int64
 6   A7_Score         704 non-null    int64
 7   A8_Score         704 non-null    int64
 8   A9_Score         704 non-null    int64
 9   A10_Score        704 non-null    int64
 10  age              702 non-null    float64
 11  gender           704 non-null    object
 12  ethnicity        704 non-null    object
 13  jundice          704 non-null    object
 14  austim           704 non-null    object
 15  contry_of_res    704 non-null    object
 16  used_app_before  704 non-null    object
 17  result           704 non-null    float64
 18  age_desc         704 non-null    object
 19  relation         704 non-null    object
 20  Class/ASD        704 non-null    object
dtypes: float64(2), int64(10), object(9)
memory usage: 115.6+ KB
```

```
1 data.describe()
```

	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	A10_Score	age	result
count	704.000000	704.000000	704.000000	704.000000	704.000000	704.000000	704.000000	704.000000	704.000000	704.000000	702.000000	704.000000
mean	0.721591	0.453125	0.457386	0.495739	0.498580	0.284091	0.417614	0.649148	0.323864	0.573864	29.698006	4.875000
std	0.448535	0.498152	0.498535	0.500337	0.500353	0.451301	0.493516	0.477576	0.468281	0.494866	16.507465	2.501493
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	17.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	21.000000	3.000000
50%	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	1.000000	27.000000	4.000000
75%	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	35.000000	7.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	383.000000	10.000000

Preparing the Data

```
1 data.isna().sum()
```

```
A1_Score           0
A2_Score           0
A3_Score           0
A4_Score           0
A5_Score           0
A6_Score           0
A7_Score           0
A8_Score           0
A9_Score           0
A10_Score          0
age                2
gender             0
ethnicity          0
jundice            0
austim             0
contry_of_res      0
used_app_before    0
result             0
age_desc           0
relation           0
Class/ASD          0
dtype: int64
```

Dropping Missing Values

```
data.dropna(inplace=True)
data.describe()
```

	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	A9_Score	A10_Score	age	result
count	702.000000	702.000000	702.000000	702.000000	702.000000	702.000000	702.000000	702.000000	702.000000	702.000000	702.000000	702.000000
mean	0.723647	0.452991	0.458689	0.497151	0.498575	0.284900	0.417379	0.650997	0.324786	0.574074	29.698006	4.883191
std	0.447512	0.498140	0.498646	0.500348	0.500354	0.451689	0.493478	0.476995	0.468629	0.494835	16.507465	2.498051
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	17.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	21.000000	3.000000
50%	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	1.000000	27.000000	4.000000
75%	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	35.000000	7.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	383.000000	10.000000

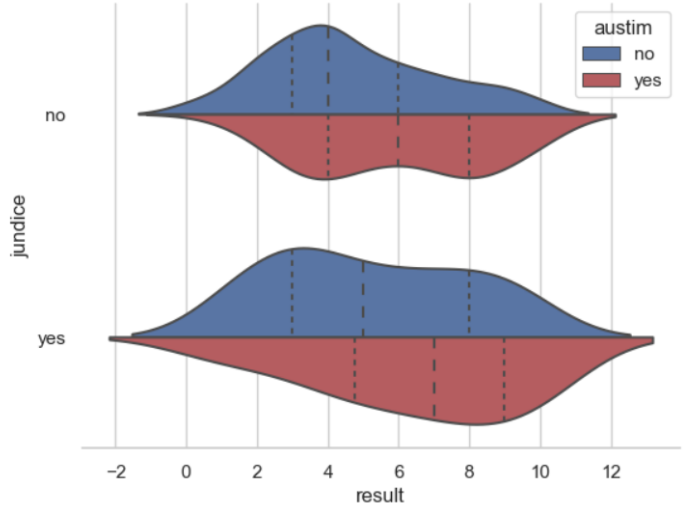
```
# After Data Cleaning

# Total number of records:
n_records = len(data.index)

# Total number of records with ASD
n_asd_yes = len(data[data['Class/ASD'] == 'YES'])

# Total number of records without ASD
n_asd_no = len(data[data['Class/ASD'] == 'NO'])

# Printing the outputs
print("AFTER REMOVING NULL VALUES : ")
print(f'Total number of records  : {n_records}')
print(f'Number of individuals with ASD : {n_asd_yes}')
print(f'Number of individuals without ASD : {n_asd_no}')

```

```
AFTER REMOVING NULL VALUES :
Total number of records  : 702
Number of individuals with ASD : 189
Number of individuals without ASD : 513
```

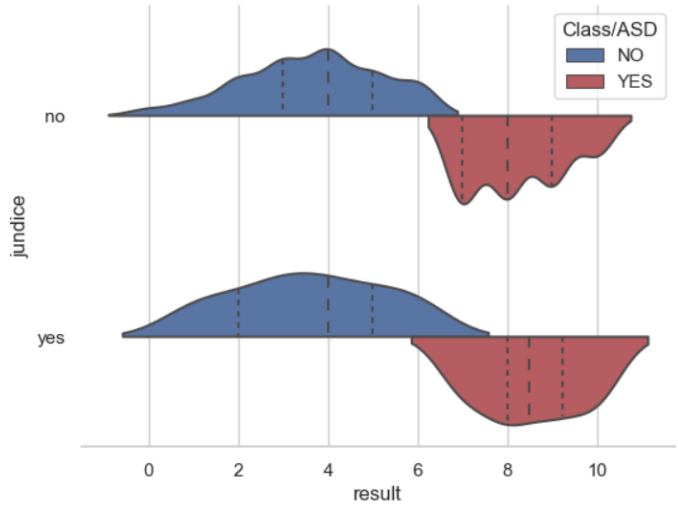
Visualizations with Seaborn

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="whitegrid", color_codes=True)
```

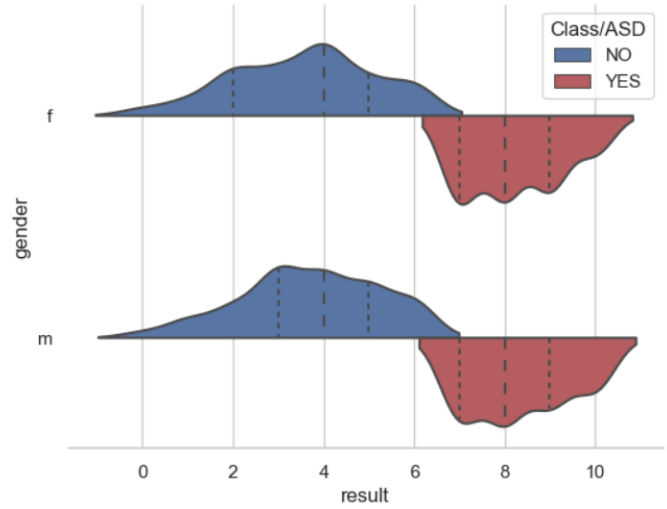
```
# Draw a nested violinplot and split the violins for easier comparison
sns.violinplot(x="result", y="jundice", hue="austim", data=data, split=True,inner="quart", palette={'yes': "r", 'no': "b"})
sns.despine(left=True)
```



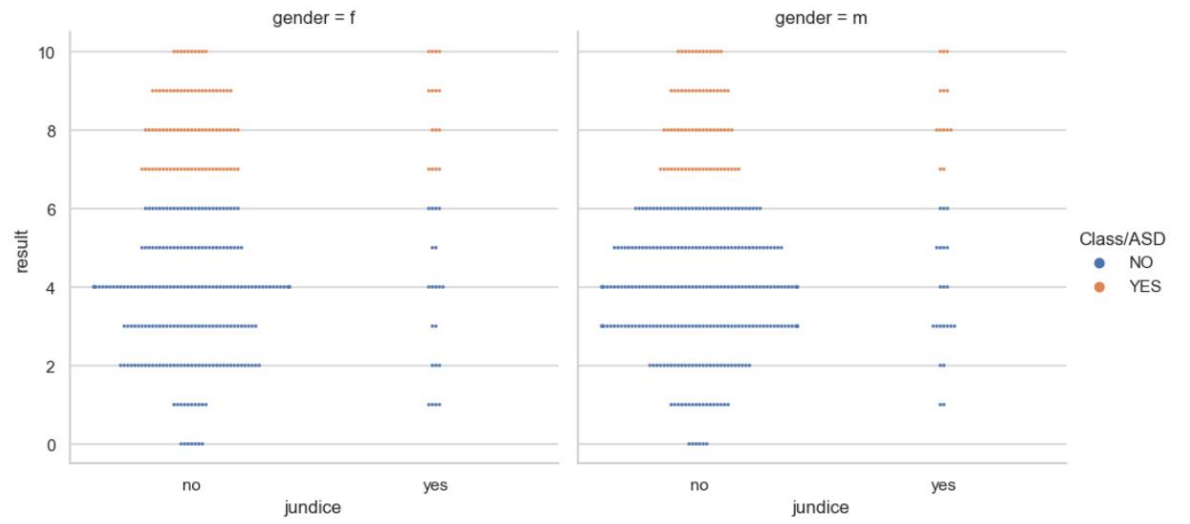
```
# Draw a nested violinplot and split the violins for easier comparison
sns.violinplot(x="result", y="jundice", hue="Class/ASD", data=data, split=True,
               inner="quart", palette={'YES': "r", 'NO': "b"})
sns.despine(left=True)
```




```
# Draw a nested violinplot and split the violins for easier comparison
sns.violinplot(x="result", y="gender", hue="Class/ASD", data=data, split=True,
               inner="quart", palette={'YES': "r", 'NO': "b"})
sns.despine(left=True)
```



```
sns.catplot(x="jundice", y="result", hue="Class/ASD", s = 5, col="gender", data=data, kind="swarm");
```



Convert the Pandas dataframes into numpy arrays that can be used by scikit_learn. Let's create an array that extracts only the feature data we want to work with and another array that contains the classes (class/ASD).

```
data_raw = data['Class/ASD']
features_raw = data[['age', 'gender', 'ethnicity', 'jundice', 'austim', 'contry_of_res', 'result',
                     'relation','A1_Score','A2_Score','A3_Score','A4_Score','A5_Score','A6_Score','A7_Score','A8_Score',
                     'A9_Score','A10_Score']]
```

Data Preprocessing

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
num = ['age','result']
features_minmax_transform = pd.DataFrame(data = features_raw)
features_minmax_transform[num] = scaler.fit_transform(features_raw[num])
```

```
display(features_minmax_transform.head(5))
```

	age	gender	ethnicity	jundice	austim	contry_of_res	result	relation	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_
0	0.024590	f	White-European	no	no	United States	0.6	Self	1	1	1	1	0	0	1	
1	0.019126	m	Latino	no	yes	Brazil	0.5	Self	1	1	0	1	0	0	0	
2	0.027322	m	Latino	yes	yes	Spain	0.8	Parent	1	1	0	1	1	0	1	
3	0.049180	f	White-European	no	yes	United States	0.6	Self	1	1	0	1	0	0	1	
4	0.062842	f	?	no	no	Egypt	0.2	?	1	0	0	0	0	0	0	

One-Hot Encoding on features_minmax_transform ¶

```
features_final = pd.get_dummies(features_minmax_transform)
features_final.head(5)
```

	age	result	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score	...	contry_of_res_United Kingdom	contry_of_res_United States	cont
0	0.024590	0.6	1	1	1	1	0	0	1	1	...	0	1	
1	0.019126	0.5	1	1	0	1	0	0	0	1	...	0	0	
2	0.027322	0.8	1	1	0	1	1	0	1	1	...	0	0	
3	0.049180	0.6	1	1	0	1	0	0	1	1	...	0	1	
4	0.062842	0.2	1	0	0	0	0	0	0	1	...	0	0	

5 rows × 103 columns

Encode all classes data to numerical data

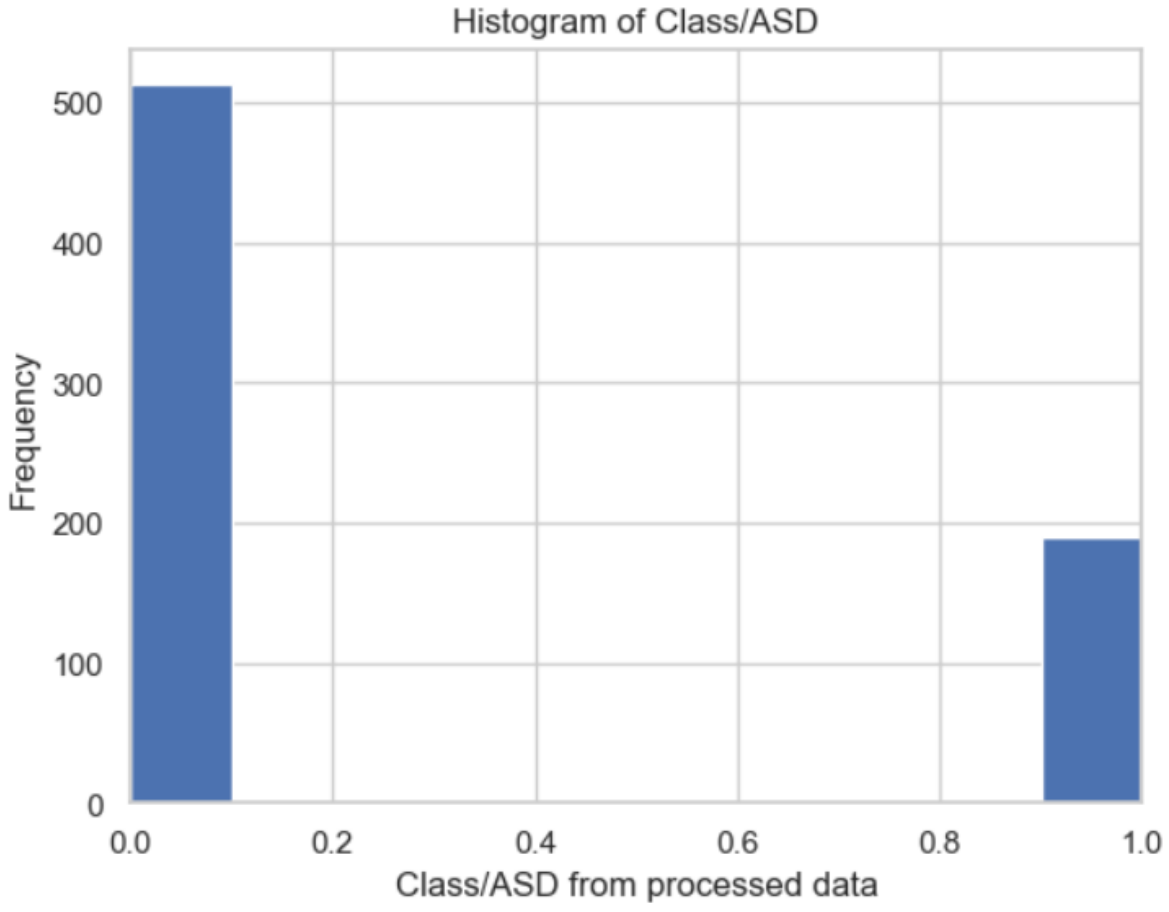
```
data_classes = data_raw.apply(lambda x : 1 if x == 'YES' else 0)
```

```
encoded = list(features_final.columns)
print("{} total features after one-hot encoding".format(len(encoded)))
print(encoded)
```

```
103 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', 'gender_f', 'gender_m', 'ethnicity_?', 'ethnicity_Asian', 'ethnicity_Black', 'ethnicity_Hispanic', 'ethnicity_Lati
no', 'ethnicity_Middle Eastern ', 'ethnicity_Others', 'ethnicity_Pasifika', 'ethnicity_South Asian', 'ethnicity_Turkish', 'ethn
icity_White-European', 'ethnicity_others', 'jundice_no', 'jundice_yes', 'austim_no', 'austim_yes', 'contry_of_res_Afghanistan',
'contry_of_res_AmericanSamoa', 'contry_of_res_Angola', 'contry_of_res_Argentina', 'contry_of_res_Armenia', 'contry_of_res_Arub
a', 'contry_of_res_Australia', 'contry_of_res_Austria', 'contry_of_res_Azerbaijan', 'contry_of_res_Bahamas', 'contry_of_res_Ban
gladesh', 'contry_of_res_Belgium', 'contry_of_res_Bolivia', 'contry_of_res_Brazil', 'contry_of_res_Burundi', 'contry_of_res_Can
ada', 'contry_of_res_Chile', 'contry_of_res_China', 'contry_of_res_Costa Rica', 'contry_of_res_Cyprus', 'contry_of_res_Czech Re
public', 'contry_of_res_Ecuador', 'contry_of_res_Egypt', 'contry_of_res_Ethiopia', 'contry_of_res_Finland', 'contry_of_res_Fran
ce', 'contry_of_res_Germany', 'contry_of_res_Hong Kong', 'contry_of_res_Iceland', 'contry_of_res_India', 'contry_of_res_Indones
ia', 'contry_of_res_Iran', 'contry_of_res_Iraq', 'contry_of_res_Ireland', 'contry_of_res_Italy', 'contry_of_res_Japan', 'contry
_of_res_Jordan', 'contry_of_res_Kazakhstan', 'contry_of_res_Lebanon', 'contry_of_res_Malaysia', 'contry_of_res_Mexico', 'contry
_of_res_Nepal', 'contry_of_res_Netherlands', 'contry_of_res_New Zealand', 'contry_of_res_Nicaragua', 'contry_of_res_Niger', 'co
ntry_of_res_Oman', 'contry_of_res_Pakistan', 'contry_of_res_Philippines', 'contry_of_res_Portugal', 'contry_of_res_Romania', 'c
ontry_of_res_Russia', 'contry_of_res_Saudi Arabia', 'contry_of_res_Serbia', 'contry_of_res_Sierra Leone', 'contry_of_res_South
Africa', 'contry_of_res_Spain', 'contry_of_res_Sri Lanka', 'contry_of_res_Sweden', 'contry_of_res_Tonga', 'contry_of_res_Turke
y', 'contry_of_res_Ukraine', 'contry_of_res_United Arab Emirates', 'contry_of_res_United Kingdom', 'contry_of_res_United State
s', 'contry_of_res_Uruguay', 'contry_of_res_Viet Nam', 'relation_?', 'relation_Health care professional', 'relation_Others', 'r
elation_Parent', 'relation_Relative', 'relation_Self']
```

```
plt.hist(data_classes, bins=10)
plt.xlim(0,1)
plt.title('Histogram of Class/ASD')
plt.xlabel('Class/ASD from processed data')
plt.ylabel('Frequency')
```

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



Splitting Data

```
from sklearn.model_selection import train_test_split
np.random.seed(123)
X_train, X_test, y_train, y_test = train_test_split(features_final,data_classes,test_size=0.2, random_state=1)
print("Train set has {} enteries.".format(X_train.shape[0]))
print("Test set has {} enteries.".format(X_test.shape[0]))
```

```
Train set has 561 enteries.
Test set has 141 enteries.
```

Models

1. Decision Making Tree

```
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
dec_model = DecisionTreeClassifier()
dec_model.fit(X_train.values, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
y_pred = dec_model.predict(X_test.values)
print('True : ', y_test.values[0:25])
print('False :', y_pred[0:25])
```

```
True :  [1 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0]
False : [1 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0]
```

```
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, y_pred)
print(cm)
TP = cm[1,1]
FP = cm[0,1]
TN = cm[0,0]
FN = cm[1,0]
```

```
[[101   0]
 [  0  40]]
```

```
print('Accuracy:')
print((TN+TP)/float(TP+TN+FP+FN))
print('Error:')
print((FP+FN)/float(TP+TN+FP+FN))
print('Precision:')
print(metrics.precision_score(y_test,y_pred))
print('Score:')
print(dec_model.score(X_test.values, y_test))
```

```
Accuracy:
1.0
Error:
0.0
Precision:
1.0
Score:
1.0
```

2. Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
rndm_model = RandomForestClassifier(n_estimators=5, random_state=1)
cv_score = cross_val_score(rndm_model, features_final, data_classes, cv =10)
cv_score.mean()
```

```
0.9900603621730383
```

```
# F-beta Score
rndm_model.fit(X_train.values, y_train)
from sklearn.metrics import fbeta_score
y_pred = rndm_model.predict(X_test.values)
fbeta_score(y_test, y_pred, average='binary', beta=0.5)
```

```
1.0
```

3. Support Vector Machine

```
from sklearn import svm
svm_model = svm.SVC(kernel='linear', C=1, gamma=2)
cv_score = cross_val_score(svm_model, features_final, data_classes, cv =10)
cv_score.mean()
```

```
1.0
```

```
#F-beta Score
svm_model.fit(X_train.values, y_train)
from sklearn.metrics import fbeta_score
y_pred = svm_model.predict(X_test.values)
fbeta_score(y_test, y_pred, average='binary', beta=0.5)
```

```
1.0
```

4. K-Nearest-Neighbors (KNN)

```
from sklearn import neighbors
knn_model = neighbors.KNeighborsClassifier(n_neighbors=10)
cv_score = cross_val_score(knn_model, features_final, data_classes, cv =10)
cv_score.mean()
```

```
0.9458752515090543
```

```
#F-beta Score
knn_model.fit(X_train.values, y_train)
from sklearn.metrics import fbeta_score
y_pred = knn_model.predict(X_test.values)
fbeta_score(y_test, y_pred, average='binary', beta=0.5)
```

```
0.9183673469387756
```

```
for n in range(10,30):
    knn_model = neighbors.KNeighborsClassifier(n_neighbors=n)
    cv_scores = cross_val_score(knn_model, features_final, data_classes, cv=10)
    print (n, cv_scores.mean())
```

```
10 0.9458752515090543
11 0.9473239436619719
12 0.9444869215291751
13 0.9501609657947686
14 0.9458953722334005
15 0.9458953722334004
16 0.951569416498994
17 0.951549295774648
18 0.9529778672032194
19 0.9572635814889336
20 0.9529778672032194
21 0.9529778672032194
22 0.9486921529175051
23 0.9472635814889336
24 0.9486921529175051
25 0.9486720321931589
26 0.9515090543259557
27 0.9501006036217303
28 0.9486720321931589
29 0.9472434607645874
```

Hence, K is not making any significant difference on accuracy of our predictions.

5. Naive Bayes

```
from sklearn.naive_bayes import MultinomialNB
nb_model = MultinomialNB()
cv_score = cross_val_score(nb_model, features_final, data_classes, cv =10)
cv_score.mean()
```

```
0.8746277665995976
```

```
#F-beta Score
nb_model.fit(X_train.values, y_train)
from sklearn.metrics import fbeta_score
y_pred = nb_model.predict(X_test.values)
fbeta_score(y_test, y_pred, average='binary', beta=0.5)
```

```
0.7675438596491228
```

6. Logistic Regression

```
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression()
cv_score = cross_val_score(lr_model, features_final, data_classes, cv =10)
cv_score.mean()
```

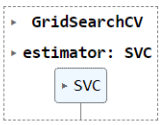
```
0.9971428571428571
```

```
#F-beta Score
lr_model.fit(X_train.values, y_train)
from sklearn.metrics import fbeta_score
y_pred = lr_model.predict(X_test.values)
fbeta_score(y_test, y_pred, average='binary', beta=0.5)
```

```
0.9948979591836734
```

Model Tuning

```
from sklearn.metrics import fbeta_score
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

def f_beta_score(y_true, y_predict):
    return fbeta_score(y_true, y_predict, beta=0.5)

clf = SVC(random_state=1)
parameters = {'C': range(1, 6),
              'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
              'degree': range(1, 6)}

scorer = make_scorer(f_beta_score)

# Now you can use GridSearchCV to find the best hyperparameters
grid_search = GridSearchCV(clf, parameters, scoring=scorer)
grid_search.fit(X_train, y_train)
```




```
grid_obj = GridSearchCV(estimator = clf, param_grid = parameters, scoring = scorer)
grid_fit = grid_obj.fit(X_train.values, y_train)
best_clf = grid_fit.best_estimator_
```

```
predictions = (clf.fit(X_train.values, y_train)).predict(X_test.values)
best_predictions = best_clf.predict(X_test.values)
```

```
print ("Unoptimized model\n------")
print ("Accuracy score on testing data: {:.4f}".format(accuracy_score(y_test, predictions)))
print ("F-score on testing data: {:.4f}".format(fbeta_score(y_test, predictions, beta = 0.5)))
print ("\nOptimized Model\n------")
print ("Final accuracy score on the testing data: {:.4f}".format(accuracy_score(y_test, best_predictions)))
print ("Final accuracy score on the testing data: {:.4f}".format(accuracy_score(y_test, best_predictions)))
print ("Final F-score on the testing data: {:.4f}".format(fbeta_score(y_test, best_predictions, beta = 0.5)))
```

```
Unoptimized model
------
Accuracy score on testing data: 0.9645
F-score on testing data: 0.9574

Optimized Model
------
Final accuracy score on the testing data: 1.0000
Final accuracy score on the testing data: 1.0000
Final F-score on the testing data: 1.0000
```

CONCLUSION

In conclusion, the application of machine learning in autism detection represents a significant advancement in the field of medical diagnostics. By leveraging sophisticated algorithms and vast datasets, machine learning models can identify subtle patterns and indicators of autism spectrum disorder (ASD) that might be missed by traditional diagnostic methods. This not only enhances the accuracy and efficiency of early detection but also makes the diagnostic process more accessible and less burdensome for patients and healthcare providers.

The integration of machine learning techniques, such as Support Vector Machines (SVM), Random Forest Classifiers (RFC), and Convolutional Neural Networks (CNN), into autism detection systems holds great promise for the future. These technologies can process diverse data inputs, from behavioral assessments to genetic information, providing a comprehensive approach to diagnosing ASD. As research and technology continue to evolve, machine learning is poised to play an increasingly vital role in improving the lives of individuals with autism through earlier and more precise diagnosis.