Student ID Number: 19075153

Assignment 1
Part B

Semester 2 2021

Student Name: Bernard O'Leary
Student ID: 19075153

PAPER NAME: Data Mining and Machine Learning

PAPER CODE: COMP809

Due Date: Friday 10 Sep 2021 (midnight)

TOTAL MARKS: 100

INSTRUCTIONS:

1. **The following actions may be deemed to constitute a breach of the General Academic Regulations Part 7: Academic Discipline,**
   - Communicating with or collaborating with another person regarding the Assignment
   - Copying from any other student work for your Assignment
   - Copying from any third-party websites unless it is an open book Assignment
   - Uses any other unfair means
2. **Please email DCT.EXAM@AUT.AC.NZ if you have any technical issues with your Assessment/Assignment/Test submission on Blackboard immediately**
3. **Attach your code for all the datasets in the appendix section.**

Study Area I (Dataset is bank.csv use the Bank.zip)

(a) Pre-processing

All of the libraries necessary for the assignment were imported in the first step, including thoe necessary for pre-rocessing the data.

Preprocesing data required the application of semi-colon delimitaion, removal of any records that were classified "unknown" and turn all parameters into categorical (numerical) values so that they can be processed by the various classifiers that we are using for this assignment.

(b) Top five most influential features

The top five most influential features were selected by creating a method that will test the accuracy of models created using a range of different feature extraction processes. The processes used are:

- Extra Trees Classifier
- RFE (Recursive Feature Extraction) using Logistic Regression
- Chi-squared test
- Principal Component Analysis (PCA)

The features were then ordered in order of influence, or captured using the algorithm's built-in feature selection capacity. The "transform" method on the fitted model object is then used to enable the top five features to be selected from the result.

The best performing approach was RFE. The RFE process has in-built ability to refine the feature-set to five most influential features. The process for feature selection using RFE is as follows:

```
# Feature Extraction with RFE
model = LogisticRegression()
rfe = RFE(model, 5)
fit = rfe.fit(X, Y)
features = fit.transform(X)
pred_features = features[:, 0:5]
```

Once the top five features have been identified, they are run through a "get_accuracy" method which creates a MLPClassifier model and returns an accuracy score. These scores are then compared. As above, the best scoring approach was RFE.

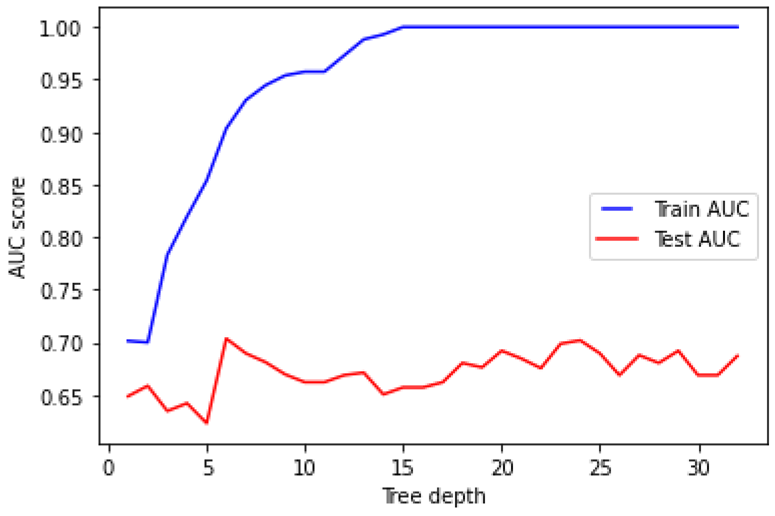
(c) Decision Tree algorithm

The two parameter that proved to be most influential in ability reduce the number of nodes in resulting tree were the min_samples_leaf and min_samples_split parameters. The best performance was obtained when min_samples_split=0.3 and min_samples_leaf=0.2.

The min_samples_leaf parameter produced smaller trees with an optimised value, however neither parameter seemd to have any effect on the accuracy of the model training process, despite reducing the number of nodes in the tree.

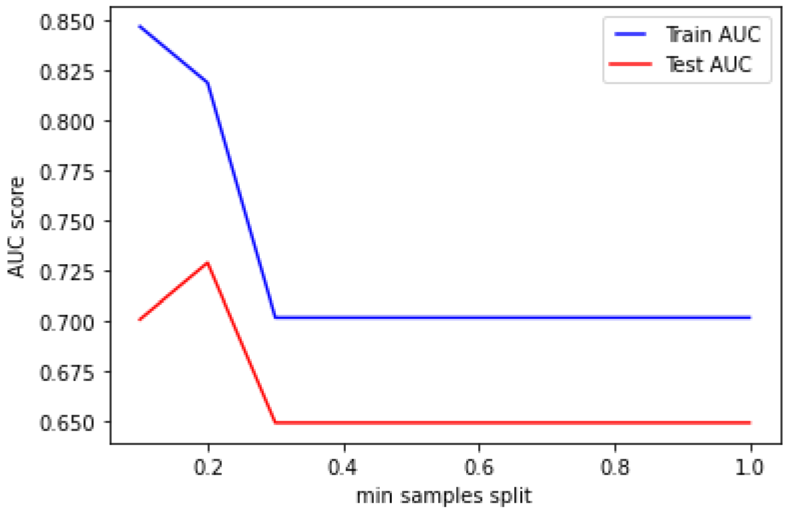
The following images show the train and test AUC performance for each different type of five parameters tested. The resulting model accuracy tested with 10x cross-validation and number of nodes in the resulting tree is listed. An optimised model was trained for each parameter to test for accuracy based on what appeared to be an optimal value for the variable as is shown by the charts produced (i.e. values that appear to reduce the risk of under/over-fitting).

max_depth=3



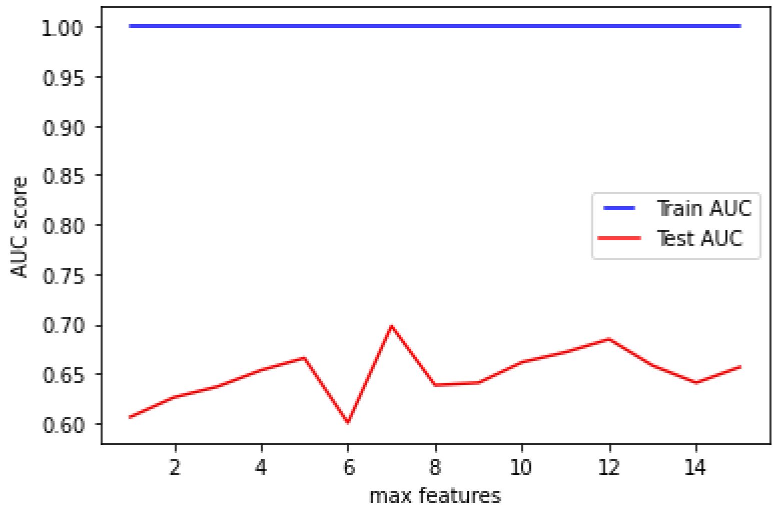
```
0.8301572897761644
15
```

min_samples_split=0.3



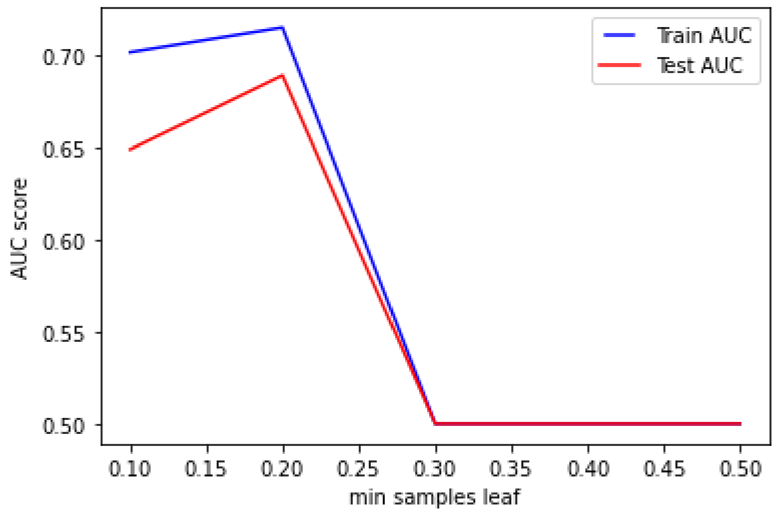
```
0.8196612220205687
13
```

max_features=16



```
0.7968844525105868
151
```

min_samples_leaf=0.2



```
0.7827888687235329
7
```

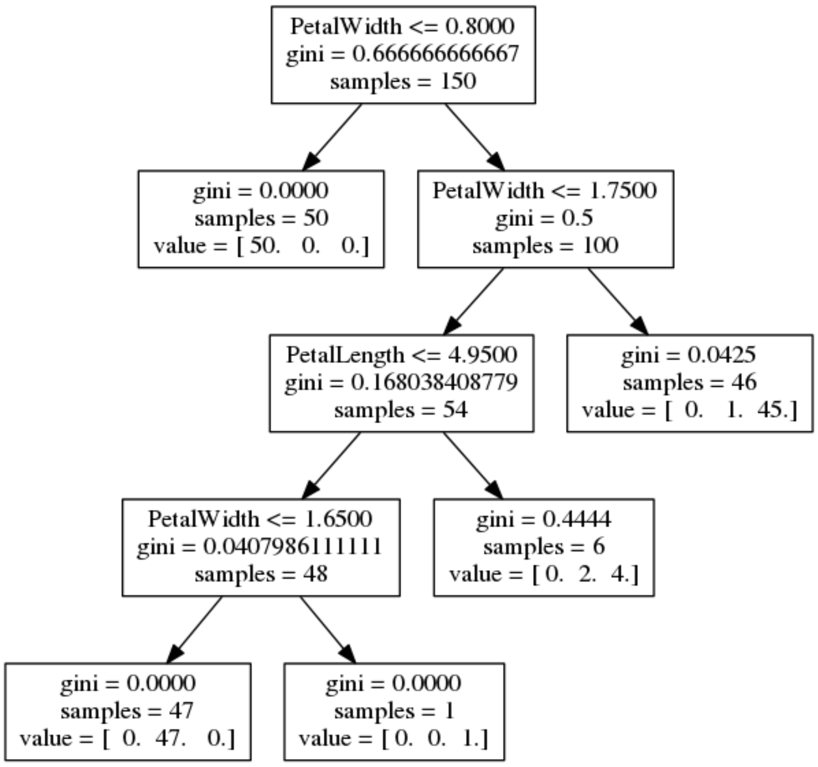
(d) Describe the role of the two parameters

As can be seen from the above results, the best performing parameters and values for the prameters as obtained when min_samples_split=0.3 and min_samples_leaf=0.2.

min_samples_split defines the minimum number of samples that are required to be analysed before the model generation algorithm will elect to split an *internal* node of the tree.For example, if min_samples_split=2, then a minimum of two observations need to be considered to split the node.

min_samples_leaf parameter works the same way, but for external (leaf) nodes. The difference between this parameter and min_samples_split is min_samples_leaf specifies a minimum number of samples in a leaf, while min_samples_split can create as many sub-leaves as it's setting allows, dependent on the value specified for min_samples_leaf.

The following diagram illustrates the behaviour and subtle differences between the two parameters:



This tree was constructed against the Iris dataset with min_samples_split=10. We can see that the internals nodes of the tree have no samples less than 10, whereaes the are two leafs that considered less than 10 samples. min_samples_leaf=1 for this tree.

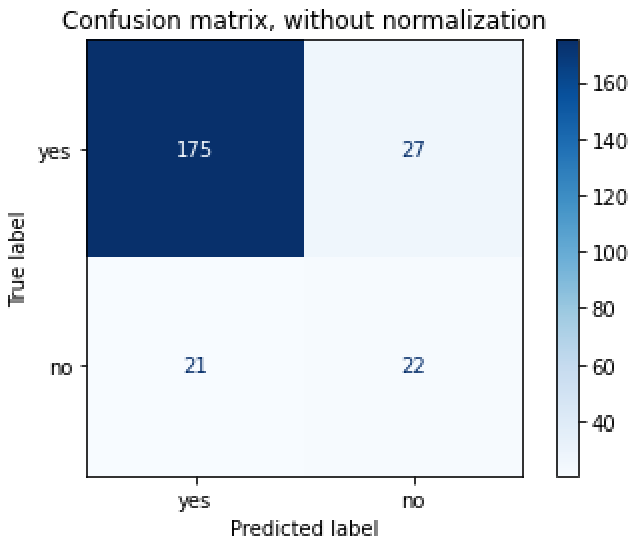
Documentation for the parameters is included in the Decision Tree Classifier documentation – here: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

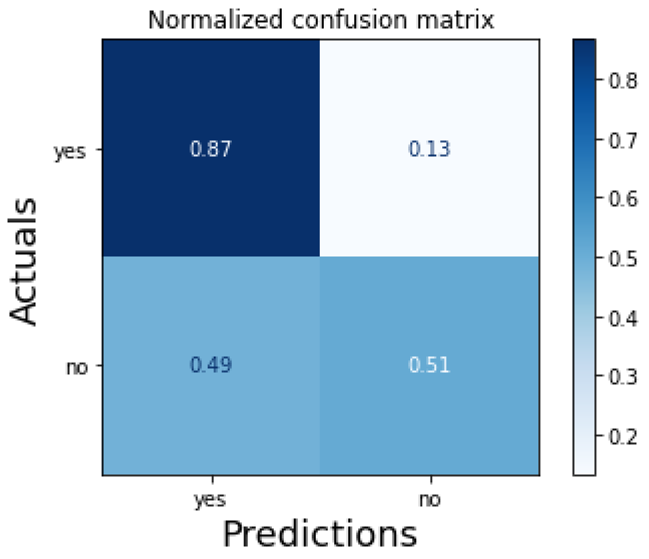
Depending on the size, shape and nature of the content of the dataset, the parameters that had an effect here may not have as strong of an effect as they have had here. For example, for large datasets such as for image processing or audio processing applications, performance of these parameters might be outperformed in terms of influence on the model by adjustment of some other parameters.

(e) Examine the Confusion Matrix

The final score, number of nodes and confusion matrix with an without normalisation are provided below. We can see that of the observations considered, we have 87% true-positives and 51% true negatives. False positive is sitting at 49% whereas false negatives is sitting at 13%.

```
0.7827888687235329
7
Confusion matrix, without normalization
[[175  27]
 [ 21  22]]
Normalized confusion matrix
[[0.866 0.134]
 [0.488 0.512]]
```





The model was trained and tested with 10x cross-validation. The overall accuracy of the model is reported as 78%, which is significantly less than the percentage of true positives, but is influenced by the low number of true negatives. Our equation for accuracy is as follows:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

It is likely that the number of "yes" results for outcomes is larger as it is in the interest of banks to have customers make term deposits, so the marketing campaign conducted might have been very effective. Also this dataset might have been taken from a time when interest rates for term deposits were unusually high, making it a comparatively good investment. Due to globally low interest rates for lending, this is not necessarily the case nowadays.

Study Area II (Dataset is Autism-Child-Data.arff)

(a) Describe the autism spectrum disorder

The Autism-Spectrum Disorder (ASD) refers to and quantifies autistic traits in adults [1]. Because Autism is a condition the influences aspects of behaviour, such as high neuroticism, low extraversion and low agreeableness [2], it can lead to difficulty successing in environments such as educational institutions that do not cater for variances in behaviour that may be exhibited in someone who is identified as being higher than average on the Autism Disorder Spectrum. On the other hand, there is evidence to sugget that individuals who are identified as being higher than average on the Autism Disorder Spectrum, may favour vocations such as engineering, physics or mathematics [2], which allows education institutions to support children who score higher than average on the Autism Disorder Spectrum by providing options to focus more in these areas. It is therefore of interest to be able to identify individuals who may be on the Autism Disorder Spectrum earlier rather than later.

The test for ASD is called the Autism Spectrum Quotient (AQ). The intention of the AQ is to provide a quick and easy way to quantify how many autistic traits and adult has [1]. The AQ is a self-reporting test that consists of a 50-item questionnaire [2]. The questionanaire is split evently in to five different types of questions covering social skill, attention switching, attention to detail, communication and imagination. Answers to questions are designed so as an answer can be classified as either autistic or non-autistic. Questions that my be included are for example "I find it hard to make new friends", or "I am fascinated by numbers". Although the AQ test is designed originally for adults, there is no significant difference in it's ability to indicate where an adolescent is on the Autism Disorder Spectrum [1].

[1] Baron-Cohen, S., Hoekstra, R.A., Knickmeyer, R. et al. The Autism-Spectrum Quotient (AQ)—Adolescent Version. J Autism Dev Disord 36, 343 (2006).

[2] Elizabeth J. Austin, Personality correlates of the broader autism phenotype as assessed by the Autism Spectrum Quotient (AQ), Personality and Individual Differences, Volume 38, Issue 2, 2005, Pages 451-460

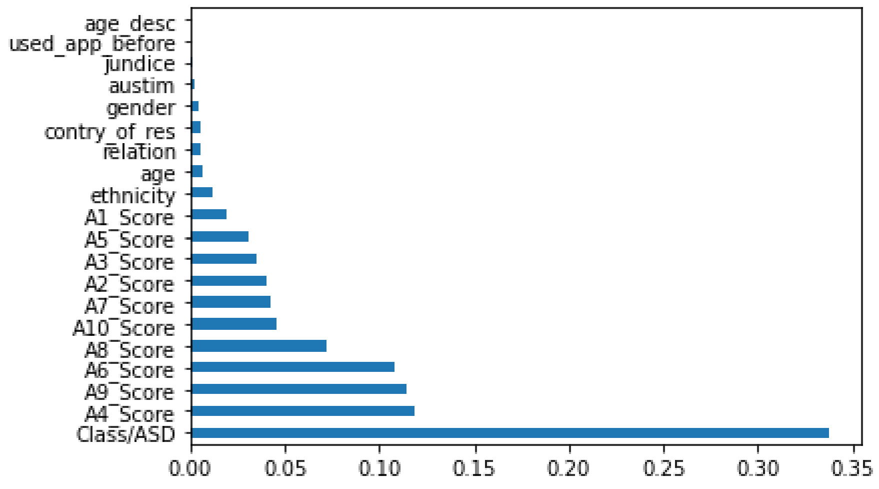
(b) Identify the top five significant features

The same approach for feature selection was applied in this section as it was for the first section. The top five features were identified using the Extra Trees Classifier method. This method was selected because of the four different feature selection methods that were tested, Extra Trees Classifier performed slightly better than the rest. A typical result was as follows:

```
Accuracy score of our model without feature selection : 0.05
Accuracy score of our model with chi square feature selection : 0.15
Accuracy score of our model with RFE selection : 0.15
Accuracy score of our model with PCA selection : 0.15
Accuracy score of our model with Extra Trees selection : 0.16
```

Docuemntation for the method can be found here: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html

Several data cleansing step needed to be applied prior to application of the Extra Trees Classifier. Cleansing included changing the data to be categorical and changing the imported to remove quite marks around all of the data that were imported so that it could be transformed to categorical. Finally rows that has negative values were removed from the dataset. The result of applying the Extra Trees Classifier is illustrated in the following chart. In this case, the five highest scoring features were A8_Score, A6_Score, A9_Score, A4_Score and Class/ASD. The results varied easch time the process was run, but Class/ASD always shows up as most influential, with a combination of other features thereafter.



The independence assumption states that each feature is conditionally independent of any other feature. This is a requirement of the mathematical theory behind the Naïve Bayes algorithm, however in practice, it will often not always be the case that all features are entirely statistically independent of each other. If a large numbers of features are dependent on each other though, accuracy may drop.

Because there are only 10 questions asked by the AQ test given in the study, and we are not give the details of what the questions were (just the index) it is difficult to just by looking at the questions how likely it is that they are properly independent of each other. It is likely that the final score (Class/ASD) is not fully independent from any of the individual the questions though.

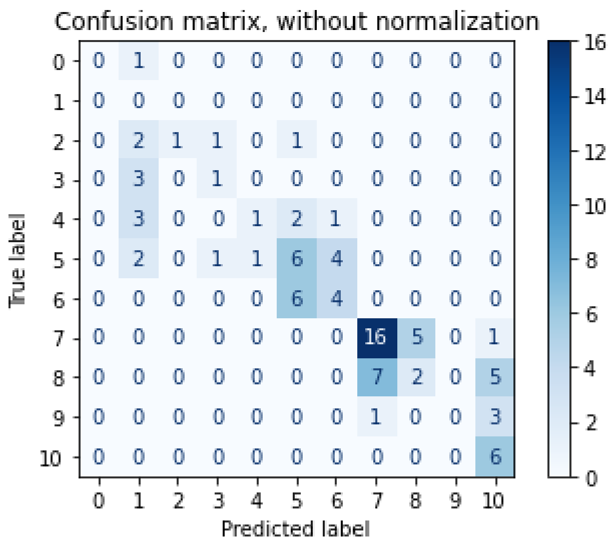
(c) Naïve Bayes algorithm

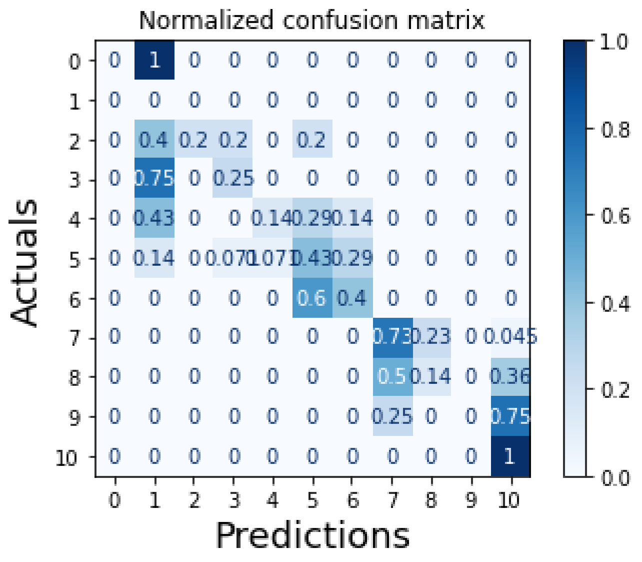
The accuracy for the NB algorithm was consistently sitting at about 35% accuracy. This indicates that the model would be able to product the correct AQ score (a number between 1 and 10) about 35% of the time. This is not a great result, and the model is therefore not considered a good predictor.

This might be because of the smaller number of features that are included in this version of the AQ model, which is for the intention of using with chileren, to make it easier for them to complete. The original AQ model has 50 features in it, whereas this modified one for use with chideren has only 10. Where the independence assumption is not completely met by all the features for example, with less feature, the issues caused by this might be more prominent.

A way to reduce this effect might be to include more data, rather than the 292 records included for this study. With more data, the accuracy might improve.

We can see from the confusion matrix that there is the start of some strong correlation between features through the horizonal of the diagram, which is what we would want to see to indicate a high proportion of true positives and true negatives, but for fault positives and false negatives, it looks like there is just not enough data for the model to give a reliable estimate for how often these situations might occur.





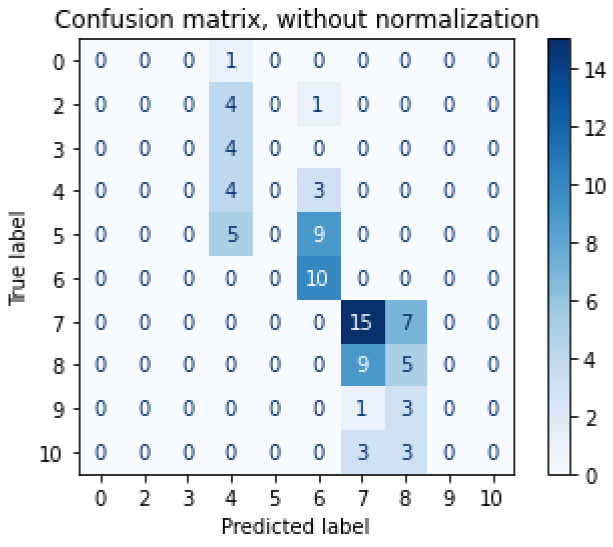
```
0.3580952380952381
```

(d) Decision Tree Classifier algorithm

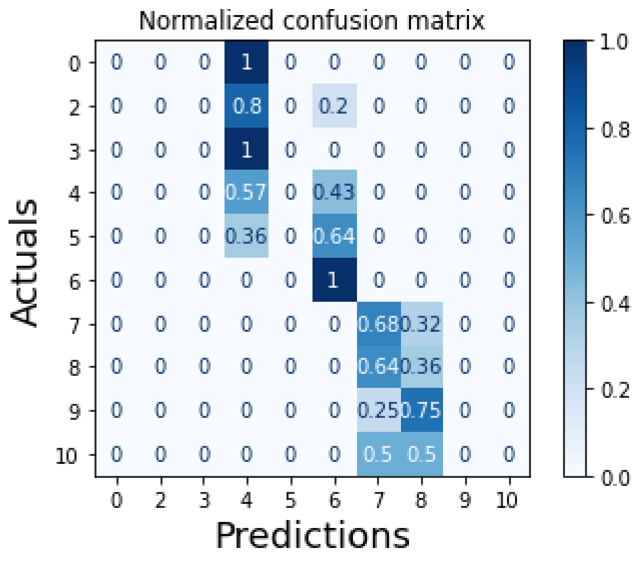
Result of the Decision Tree Classifier algorithm was significantly less accurate then the NB, although clearly neither is a good predictor, the DTC algorithm did worse than the NB one. Accuracy was consistently reported as being about 25% - about 10% less accurate than for NB. This indicates that the model would be able to product the correct

AQ score (a number between 1 and 10) about 25% of the time. This is not a great result, and the model is therefore not considered a good predictor.

As we can see from the confusion matrix, there is some clustering to the mid-diagonal of the chart, but the data are clumped and less well distributed, indicating less predictive capability than for NB just by looking at the quality of the confusion matrix.



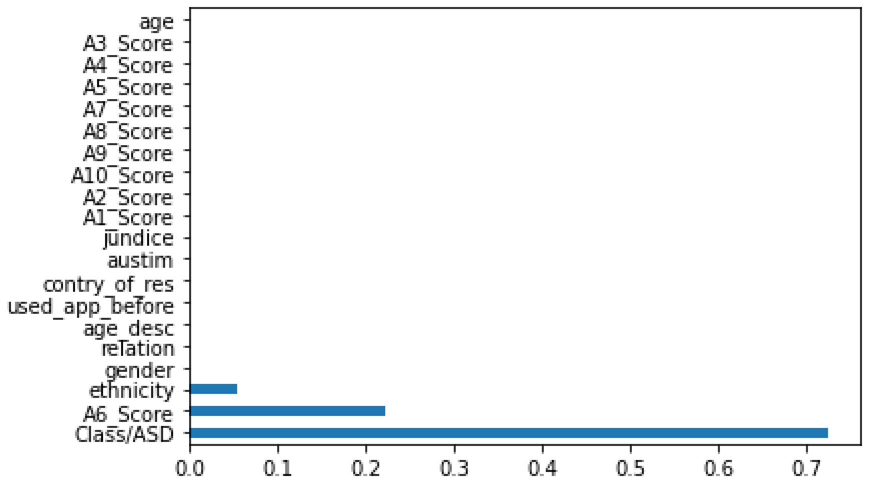
True label	0	2	3	4	5	6	7	8	9	10
0	0	0	0	1	0	0	0	0	0	0
2	0	0	0	4	0	1	0	0	0	0
3	0	0	0	4	0	0	0	0	0	0
4	0	0	0	4	0	3	0	0	0	0
5	0	0	0	5	0	9	0	0	0	0
6	0	0	0	0	0	10	0	0	0	0
7	0	0	0	0	0	0	15	7	0	0
8	0	0	0	0	0	0	9	5	0	0
9	0	0	0	0	0	0	1	3	0	0
10	0	0	0	0	0	0	3	3	0	0



Actuals	0	2	3	4	5	6	7	8	9	10
0	0	0	0	1	0	0	0	0	0	0
2	0	0	0	0.8	0	0.2	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
4	0	0	0	0.57	0	0.43	0	0	0	0
5	0	0	0	0.36	0	0.64	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0.68	0.32	0	0
8	0	0	0	0	0	0	0.64	0.36	0	0
9	0	0	0	0	0	0	0.25	0.75	0	0
10	0	0	0	0	0	0	0.5	0.5	0	0

```
0.27809523809523806
```

The top five features for the DTC model were Class/ASD, A6 _Score, ethnicity, gender, relation. The influence of each feature is less well distributed, however Class/ASD in this case is also still the feature with the most influence, same as for NB. The DTC seems to place less significance on the binary features in the dataset, i.e. the questionnaire answers (y/n) and places more emphasis on the categorical features such as ethnicity. This is possibly because the NB model is better suited to binary data.



Appendix

Two appendixes are provided

19075153_O'Leary_PartB1: extract from Jupyter-Labs for answers to Part B, section 1

19075153_O'Leary_PartB2: extract from Jupyter-Labs for answers to Part B, section 2

19075153_O'Leary_PartB1

September 11, 2021

[1]:
```
# Feature Extraction with Univariate Statistical Tests (Chi-squared for␣
 ↪classification)
import pandas
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
```

1 Data pre-processing performed

Apply semi-colon delimitaion, remove the records that were classified "unknown"

Turn all parameters into categorical (numerical) values

Parameters adjusted are min_samples_leaf and min_samples_split

[2]:
```
# load data
path = "bank/bank.csv"
rawdata = pandas.read_csv(path, sep=';')

# filter to exclude records where the outcome of the marketing campaign was␣
 ↪unknown
known_outcomes = rawdata[rawdata["poutcome"] != "unknown"]

# Convert to categorical
list_of_columns = known_outcomes.columns
known_outcomes[list_of_columns] = known_outcomes[list_of_columns].apply(lambda␣
 ↪col:pandas.Categorical(col).codes)

array = known_outcomes.values
nrow, ncol = known_outcomes.shape
X = array[:, 0:16]
Y = array[:, 16]
```

```
/home/bernard/anaconda3/lib/python3.8/site-packages/pandas/core/frame.py:3191:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self[k1] = value[k2]
```

2 Feature selection

Feature Extraction with RFE with LogisticRegression Wrapper gives the best accuracy of 0.8

[3]:
```
# generate model and get accuracy
def get_accuracy(target_train, target_test, predicted_test, predicted_train):
    clf = MLPClassifier(activation='logistic', solver='sgd',␣
 ↪learning_rate_init=0.1, alpha=1e-5, hidden_layer_sizes=(5, 2),␣
 ↪random_state=1,max_iter=2000)
    clf.fit(predicted_train, np.ravel(target_train, order='C'))
    predictions = clf.predict(predicted_test)
    return accuracy_score(target_test, predictions)

pred_train, pred_test, tar_train, tar_test = train_test_split(X, Y, test_size=.
 ↪3, random_state=4)
print("Accuracy score of our model without feature selection : %.2f" %␣
 ↪get_accuracy(tar_train, tar_test, pred_test, pred_train))

# feature extraction
test = SelectKBest(score_func=chi2, k=5)
fit = test.fit(X, Y)
# summarize scores
np.set_printoptions(precision=3)
print(fit.scores_)
features = fit.transform(X)
# summarize selected features
print(features[0:5, :],"summerize features")
print()
# Now apply only the K most significant features according to the chi square␣
 ↪method
pred_features = features[:, 0:5]
pred_train, pred_test, tar_train, tar_test = train_test_split(pred_features, Y,␣
 ↪test_size=.3, random_state=2)
print("Accuracy score of our model with chi square feature selection : %.2f" %␣
 ↪get_accuracy(tar_train, tar_test, pred_test,pred_train))
print()

## Feature Importance with Recursive Feature Extraction
```

```
from sklearn.feature_selection import SelectFromModel
# Feature Extraction with RFE
model = LogisticRegression()  # Logistic regression is the Wrapper classifier␣
 ↪here
rfe = RFE(model, 5)
fit = rfe.fit(X, Y)
## summarize components
#print("Num Features: %d" % (fit.n_features_))
#print("Selected Features: %s" % (fit.support_))
#print("Feature Ranking: %s" % (fit.ranking_))
## Now apply only the K most significant features according to the RFE feature␣
 ↪selection method
features = fit.transform(X)
pred_features = features[:, 0:5]
pred_train, pred_test, tar_train, tar_test = train_test_split(pred_features, Y,␣
 ↪test_size=.3, random_state=2)
print("Accuracy score of our model with RFE selection : %.2f" %␣
 ↪get_accuracy(tar_train, tar_test, pred_test,pred_train))
print()

## Feature Extraction with PCA
## feature extraction
pca = PCA(n_components=5)
fit = pca.fit(X)
features = fit.transform(X)
## summarize components
#print("Explained Variance: %s" % (fit.explained_variance_ratio_))
#print(fit.components_)
## Now apply only the K most significant faetures (components) according to the␣
 ↪PCA feature selection method
#features = fit.transform(X)
pred_features = features[:, 0:5]
pred_train, pred_test, tar_train, tar_test = train_test_split(pred_features, Y,␣
 ↪test_size=.3, random_state=2)
print("Accuracy score of our model with PCA selection : %.2f" %␣
 ↪get_accuracy(tar_train, tar_test, pred_test,pred_train))
print()

## Feature Importance with Extra Trees Classifier
from sklearn.ensemble import ExtraTreesClassifier
## feature extraction
model = ExtraTreesClassifier(max_depth=3,min_samples_leaf=2)
fit = model.fit(X, Y)
print(model.feature_importances_)
print()
t = SelectFromModel(fit, prefit=True)
```

```
features = t.transform(X)
pred_features = features[:, 0:5]
pred_train, pred_test, tar_train, tar_test = train_test_split(pred_features, Y,␣
↪test_size=.3, random_state=2)
print("Accuracy score of our model with Extra Trees selection : %.2f" %␣
↪get_accuracy(tar_train, tar_test, pred_test, pred_train))
print()
```

```
Accuracy score of our model without feature selection : 0.76
[2.742e+01 8.645e+00 4.448e-04 1.461e+00 1.860e-02 1.339e+03 1.719e+01
 5.028e+00 1.071e-02 8.913e+00 1.487e+00 9.006e+03 5.781e+00 1.119e+03
 1.089e+00 1.294e+02]
[[ 13 569 188 227   0]
 [ 15 404 155 218   0]
 [ 15 295 115 108   0]
 [ 16 173 278 218   1]
 [ 23  25 258  83   0]] summerize features

Accuracy score of our model with chi square feature selection : 0.74


/home/bernard/anaconda3/lib/python3.8/site-
packages/sklearn/utils/validation.py:70: FutureWarning: Pass
n_features_to_select=5 as keyword args. From version 1.0 (renaming of 0.25)
passing these as positional arguments will result in an error
  warnings.warn(f"Pass {args_msg} as keyword args. From version "
/home/bernard/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/bernard/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/bernard/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/bernard/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/bernard/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/bernard/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/bernard/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
Accuracy score of our model with RFE selection : 0.80

Accuracy score of our model with PCA selection : 0.76

[9.726e-03 1.254e-02 9.055e-04 8.011e-03 6.200e-05 2.384e-02 1.703e-01
 3.132e-02 2.523e-03 1.170e-02 1.699e-02 2.554e-01 3.225e-03 7.409e-02
 2.356e-03 3.770e-01]

Accuracy score of our model with Extra Trees selection : 0.74
```

3 Adjusting two suitable DTC parameters

3.1 Two best parameters

min_samples_leaf and min_samples_split

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

3.2 Which of the two parameters yielded better accuracy while producing smaller trees?

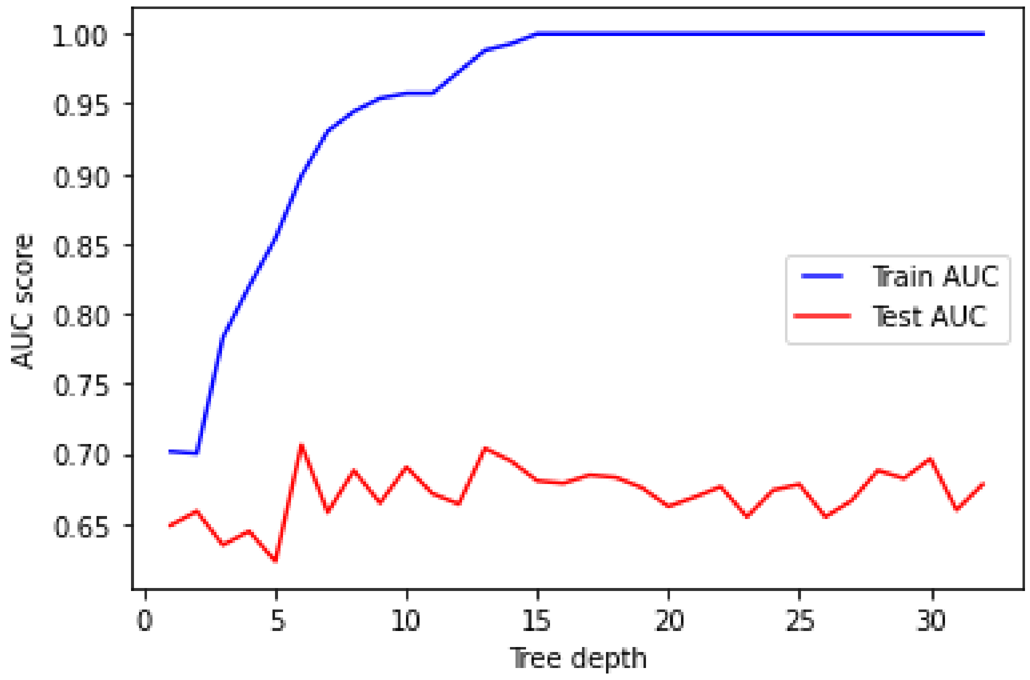
min_samples_leaf produced smaller trees with an optimised value, however neither parameter seemd to have any effect on the accuracy of the model training process, despite reducing the number of nodes in the tree.

[4]:
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import precision_score, recall_score, auc, roc_curve
from sklearn.metrics import accuracy_score,confusion_matrix
from matplotlib.legend_handler import HandlerLine2D
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold

# Get the average of the list
def Average(lst):
    return sum(lst) / len(lst)

# Get the train/test split
pred_train, pred_test, tar_train, tar_test = train_test_split(X, Y, test_size=.
 ↪3, random_state=4)
```

```
## Feature Importance with Extra Trees Classifier
max_depths = np.linspace(1, 32, 32, endpoint=True)
train_results = []
test_results = []

for max_depth in max_depths:
   dt = DecisionTreeClassifier(max_depth=max_depth)
   dt.fit(pred_train, tar_train)
   train_pred = dt.predict(pred_train)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(tar_train,␣
 ↪train_pred)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   # Add auc score to previous train results
   train_results.append(roc_auc)
   y_pred = dt.predict(pred_test)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(tar_test,␣
 ↪y_pred)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   # Add auc score to previous test results
   test_results.append(roc_auc)
   print(max_depth, accuracy_score(tar_test, y_pred))

line1, = plt.plot(max_depths, train_results, 'b', label="Train AUC")
line2, = plt.plot(max_depths, test_results, 'r', label="Test AUC")
plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.ylabel('AUC score')
plt.xlabel('Tree depth')
plt.show()

# Optimun for avoiding over/underfitting but keeping below 15 nodes is 3
dt = DecisionTreeClassifier(max_depth=3)
kf = KFold(n_splits=10)
dt.fit(pred_train, tar_train)
print(Average(cross_val_score(dt, pred_train, tar_train, cv=kf,␣
 ↪scoring='accuracy')))
print(dt.tree_.node_count)
```

```
1.0 0.8285714285714286
2.0 0.8448979591836735
3.0 0.8204081632653061
4.0 0.8367346938775511
5.0 0.8163265306122449
6.0 0.8326530612244898
7.0 0.8285714285714286
8.0 0.8326530612244898
9.0 0.8244897959183674
10.0 0.8367346938775511
```

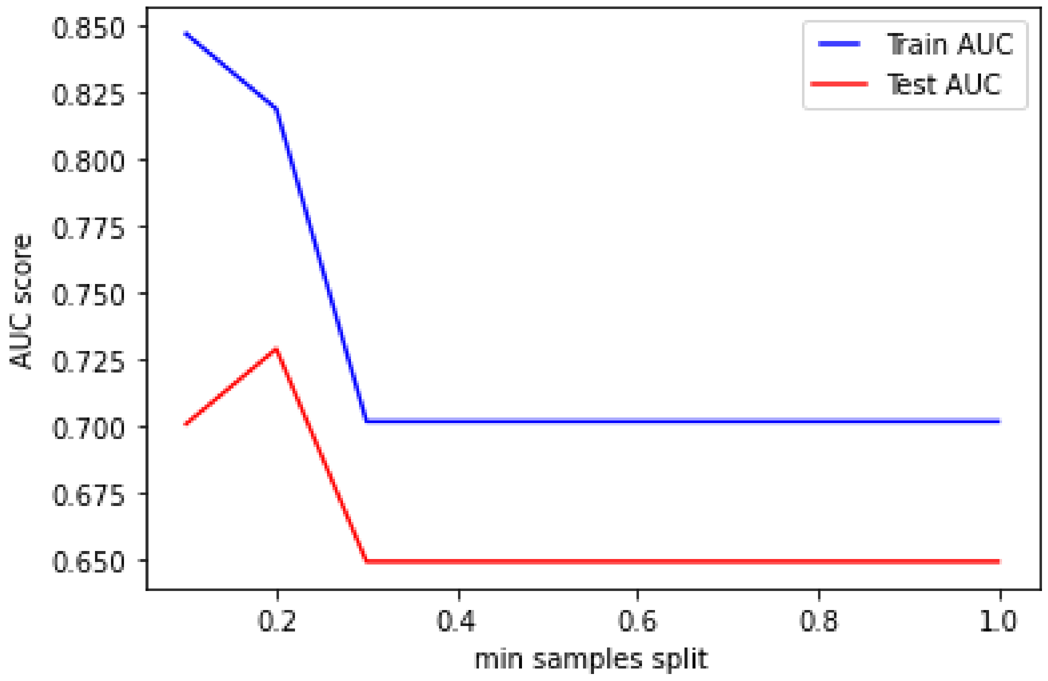
```
11.0 0.8204081632653061
12.0 0.8081632653061225
13.0 0.8285714285714286
14.0 0.8285714285714286
15.0 0.8204081632653061
16.0 0.8326530612244898
17.0 0.8122448979591836
18.0 0.8244897959183674
19.0 0.8122448979591836
20.0 0.8204081632653061
21.0 0.8163265306122449
22.0 0.8285714285714286
23.0 0.8081632653061225
24.0 0.8244897959183674
25.0 0.8163265306122449
26.0 0.8081632653061225
27.0 0.8122448979591836
28.0 0.8326530612244898
29.0 0.8081632653061225
30.0 0.8163265306122449
31.0 0.8163265306122449
32.0 0.8163265306122449
```



```
0.8301572897761644
15
```

[5]:
```
min_samples_splits = np.linspace(0.1, 1.0, 10, endpoint=True)
min_samples_splits
train_results = []
test_results = []

for min_samples_split in min_samples_splits:
   dt = DecisionTreeClassifier(min_samples_split=min_samples_split)
   dt.fit(pred_train, tar_train)
   train_pred = dt.predict(pred_train)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(tar_train,␣
 ↪train_pred)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   # Add auc score to previous train results
   train_results.append(roc_auc)
   y_pred = dt.predict(pred_test)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(tar_test,␣
 ↪y_pred)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   # Add auc score to previous test results
   test_results.append(roc_auc)
   print(min_samples_split, accuracy_score(tar_test, y_pred))

line1, = plt.plot(min_samples_splits, train_results, 'b', label="Train AUC")
line2, = plt.plot(min_samples_splits, test_results, 'r', label="Test AUC")
plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.ylabel('AUC score')
plt.xlabel('min samples split')
plt.show()

# Optimun for avoiding over/underfitting but keeping below 15 nodes is 0.3
dt = DecisionTreeClassifier(min_samples_split=0.3)
kf = KFold(n_splits=10)
dt.fit(pred_train, tar_train)
print(Average(cross_val_score(dt, pred_train, tar_train, cv=kf,␣
 ↪scoring='accuracy')))
print(dt.tree_.node_count)
```

```
0.1 0.8081632653061225
0.2 0.8244897959183674
0.30000000000000004 0.8285714285714286
0.4 0.8285714285714286
0.5 0.8285714285714286
0.6 0.8285714285714286
0.7000000000000001 0.8285714285714286
0.8 0.8285714285714286
0.9 0.8285714285714286
1.0 0.8285714285714286
```

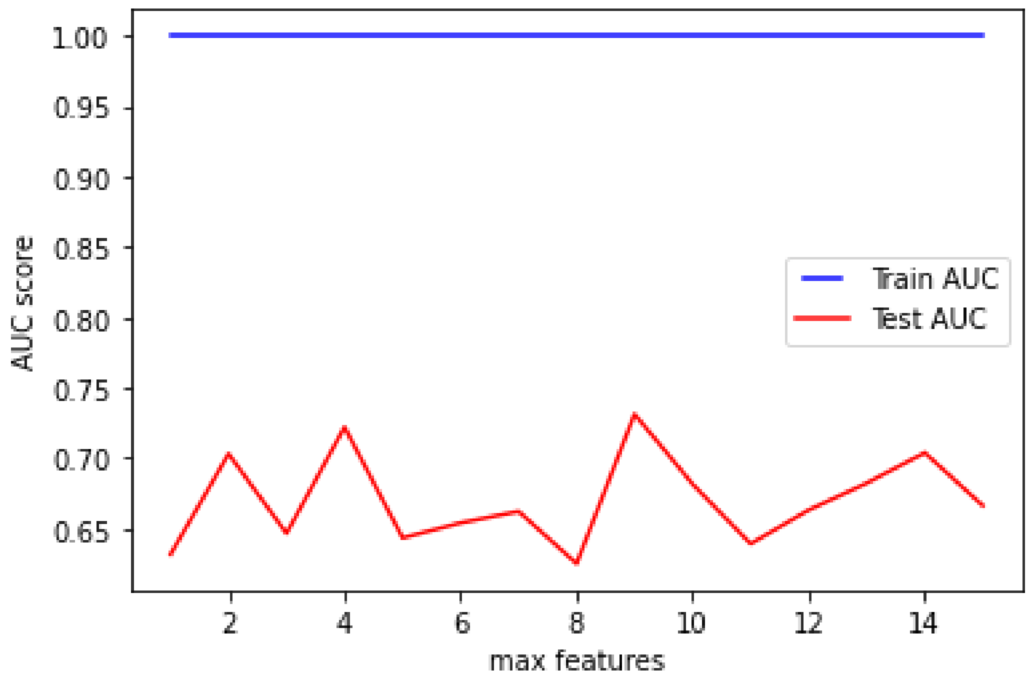


```
0.8196612220205687
13
```

[6]:
```
max_features = list(range(1,X.shape[1]))
train_results = []
test_results = []

for max_feature in max_features:
   dt = DecisionTreeClassifier(max_features=max_feature)
   dt.fit(pred_train, tar_train)
   train_pred = dt.predict(pred_train)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(tar_train,␣
 ↪train_pred)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   # Add auc score to previous train results
   train_results.append(roc_auc)
   y_pred = dt.predict(pred_test)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(tar_test,␣
 ↪y_pred)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   # Add auc score to previous test results
   test_results.append(roc_auc)
   print(max_feature, accuracy_score(tar_test, y_pred))

line1, = plt.plot(max_features, train_results, 'b', label="Train AUC")
```

```
line2, = plt.plot(max_features, test_results, 'r', label="Test AUC")
plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.ylabel('AUC score')
plt.xlabel('max features')
plt.show()

# Unable be stay nnder 15 nodes by a long sot, but the more features the less␣
 ↪nodes
dt = DecisionTreeClassifier(max_features=16)
kf = KFold(n_splits=10)
dt.fit(pred_train, tar_train)
print(Average(cross_val_score(dt, pred_train, tar_train, cv=kf,␣
 ↪scoring='accuracy')))
print(dt.tree_.node_count)
```

```
1 0.7551020408163265
2 0.8122448979591836
3 0.7795918367346939
4 0.8285714285714286
5 0.7591836734693878
6 0.7918367346938775
7 0.8204081632653061
8 0.7591836734693878
9 0.8285714285714286
10 0.7918367346938775
11 0.7673469387755102
12 0.7918367346938775
13 0.8081632653061225
14 0.8285714285714286
15 0.8122448979591836
```

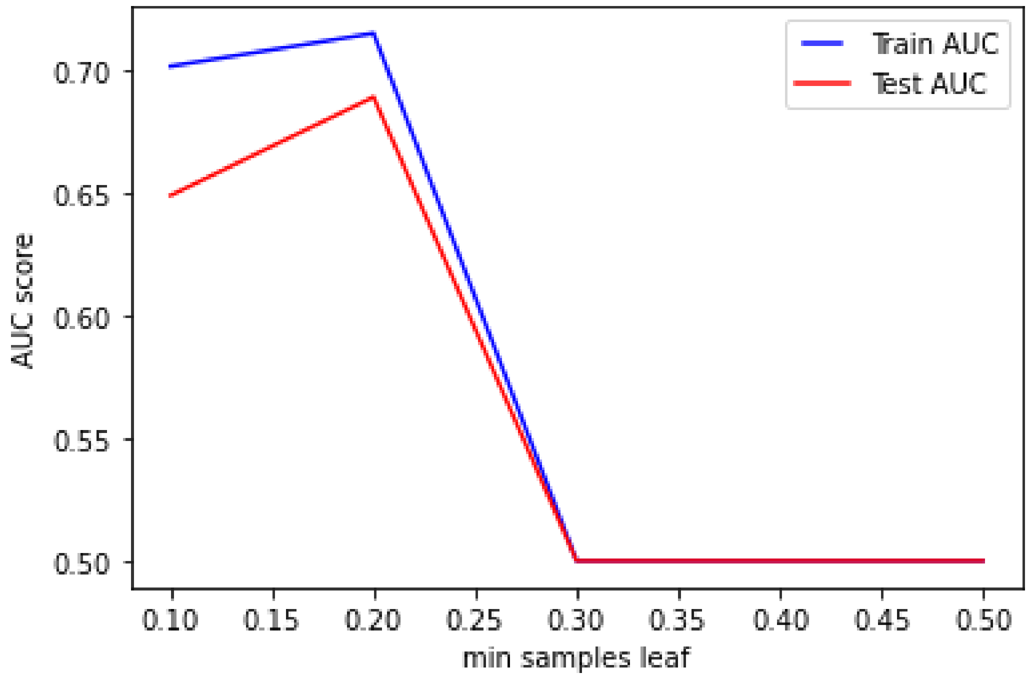


```
0.7810647307924985
151
```

[7]:
```
min_samples_leafs = np.linspace(0.1, 0.5, 5, endpoint=True)
train_results = []
test_results = []

for min_samples_leaf in min_samples_leafs:
   dt = DecisionTreeClassifier(min_samples_leaf=min_samples_leaf)
   dt.fit(pred_train, tar_train)
   train_pred = dt.predict(pred_train)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(tar_train,␣
 ↪train_pred)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   # Add auc score to previous train results
   train_results.append(roc_auc)
   y_pred = dt.predict(pred_test)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(tar_test,␣
 ↪y_pred)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   # Add auc score to previous test results
   test_results.append(roc_auc)
   print(min_samples_leaf, accuracy_score(tar_test, y_pred))

line1, = plt.plot(min_samples_leafs, train_results, 'b', label="Train AUC")
```

```
line2, = plt.plot(min_samples_leafs, test_results, 'r', label="Test AUC")
plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.ylabel('AUC score')
plt.xlabel('min samples leaf')
plt.show()

# Optimun for avoiding over/underfitting but keeping below 15 nodes is 0.2
dt = DecisionTreeClassifier(min_samples_leaf=0.2)
kf = KFold(n_splits=10)
dt.fit(pred_train, tar_train)
print(Average(cross_val_score(dt, pred_train, tar_train, cv=kf,␣
 ↪scoring='accuracy')))
print(dt.tree_.node_count)
```

```
0.1 0.8285714285714286
0.2 0.8040816326530612
0.30000000000000004 0.8244897959183674
0.4 0.8244897959183674
0.5 0.8244897959183674
```



```
0.7827888687235329
7
```

4 Describe the role of the two parameters

4.1 min_samples_split (int or float, default=2)

The minimum number of samples required to split an internal node:

If int, then consider min_samples_split as the minimum number.

If float, then min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.

Changed in version 0.18: Added float values for fractions.

4.2 min_samples_leaf (int or float, default=1)

The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

If int, then consider min_samples_leaf as the minimum number.

If float, then min_samples_leaf is a fraction and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.

Changed in version 0.18: Added float values for fractions.

4.3 Will the same apply for other dataset?

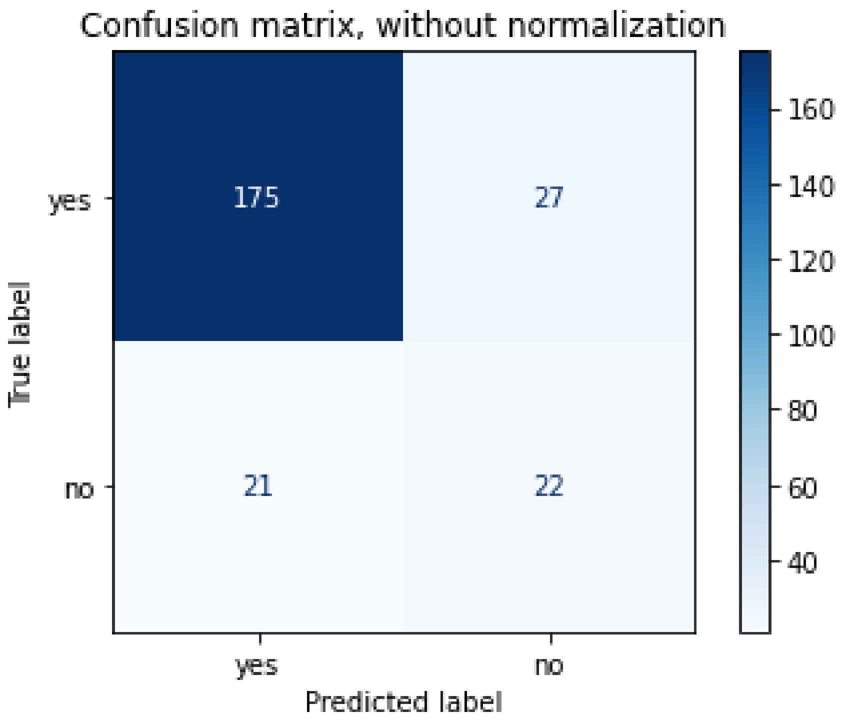
Depends on the type of dataset, how

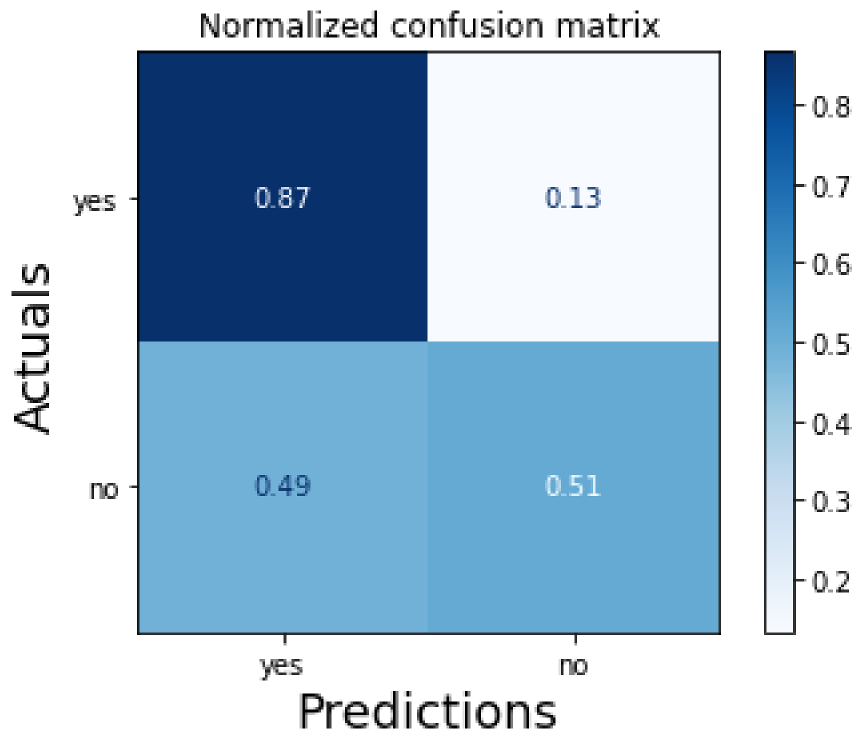
5 Confusion Matrix

[8]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import plot_confusion_matrix

# Get the train/test split
pred_train, pred_test, tar_train, tar_test = train_test_split(X, Y, test_size=.
 ↪3, random_state=4)

dt = DecisionTreeClassifier(min_samples_leaf=0.2, min_samples_split=0.3)
kf = KFold(n_splits=10)
dt.fit(pred_train, tar_train)
train_pred = dt.predict(pred_train)
print(Average(cross_val_score(dt, pred_train, tar_train, cv=kf,␣
 ↪scoring='accuracy')))
print(dt.tree_.node_count)
```

```python
# Plot non-normalized confusion matrix
titles_options = [("Confusion matrix, without normalization", None),
                  ("Normalized confusion matrix", 'true')]
for title, normalize in titles_options:
    disp = plot_confusion_matrix(dt, pred_test, tar_test,
                                 display_labels=['yes','no'],
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
    disp.ax_.set_title(title)

    print(title)
    print(disp.confusion_matrix)

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.show()
```

```
0.7827888687235329
7
Confusion matrix, without normalization
[[175  27]
 [ 21  22]]
Normalized confusion matrix
[[0.866 0.134]
 [0.488 0.512]]
```




Normalized confusion matrix
Actuals
Predictions
yes
no
yes
no
0.87
0.13
0.49
0.51
0.8
0.7
0.6
0.5
0.4
0.3
0.2

```
[ ]:
```

19075153_O'Leary_PartB2

September 11, 2021

[1]:
```
# Feature Extraction with Univariate Statistical Tests (Chi-squared for␣
 ↪classification)
import pandas
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.feature_selection import RFE
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
```

1 Data pre-processing performed

[2]:
```
# load data
path = "Autism-Screening-Child-Data Plus Description/Autism-Child-Data.arff"
from scipy.io.arff import loadarff
raw_data = loadarff(path)
df_data = pandas.DataFrame(raw_data[0])

# Remove the b'' from the data
str_df = df_data.select_dtypes([np.object])
str_df = str_df.stack().str.decode('utf-8').unstack()
for col in str_df:
    df_data[col] = str_df[col]

# Make the whole lot categorical
list_of_columns = df_data.columns
df_data[list_of_columns] = df_data[list_of_columns].apply(lambda col:pandas.
 ↪Categorical(col).codes)

# Remove all rows with negative values
df_data = df_data[(df_data >= 0).all(1)]
```

```
# split the data to X and y
X = df_data.loc[:, df_data.columns != 'result']
Y = df_data.iloc[:, 17]
#with pandas.option_context('display.max_rows', None, 'display.max_columns',␣
 ↪None):
#     print(df_data)
df_data
```

```
<ipython-input-2-b56ce8af1f47>:8: DeprecationWarning: `np.object` is a
deprecated alias for the builtin `object`. To silence this warning, use `object`
by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  str_df = df_data.select_dtypes([np.object])
```

[2]:
```
     A1_Score  A2_Score  A3_Score  A4_Score  A5_Score  A6_Score  A7_Score  \
0           1         1         0         0         1         1         0
1           1         1         0         0         1         1         0
2           1         1         0         0         0         1         1
3           0         1         0         0         1         1         0
4           1         1         1         1         1         1         1
..        …        …        …        …        …        …        …
287         1         1         1         1         1         1         1
288         1         0         0         0         1         0         1
289         1         0         1         1         1         1         1
290         1         1         1         0         1         1         1
291         0         0         1         0         1         0         1

     A8_Score  A9_Score  A10_Score  …  gender  ethnicity  jundice  austim  \
0           1         0          0  …       1          6        0       0
1           1         0          0  …       1          5        0       0
2           1         0          0  …       1          0        0       0
3           0         0          1  …       0          0        1       0
4           1         1          1  …       1          6        1       0
..        …        …         …  …  …     …        …      …
287         1         1          1  …       0         10        1       1
288         0         0          1  …       0         10        1       1
289         0         0          1  …       1          4        0       0
290         1         1          1  …       1          8        0       0
291         0         0          0  …       0          8        0       0

     contry_of_res  used_app_before  result  age_desc  relation  Class/ASD
0               24                0       5         0         2          0
1               24                0       5         0         2          0
2               24                1       5         0         0          0
3               24                0       4         0         0          0
4               51                0      10         0         2          1
```

```
..           …         …    …     …     …    …
287          50        0   10     0     2    1
288           3        0    4     0     2    0
289           8        0    7     0     2    1
290          18        0    9     0     2    1
291          18        0    3     0     2    0

[288 rows x 21 columns]
```

2 Feature selection

[3]:
```
# generate model and get accuracy
def get_accuracy(target_train, target_test, predicted_test, predicted_train):
    clf = MLPClassifier(activation='logistic', solver='sgd',␣
 ↪learning_rate_init=0.1, alpha=1e-5, hidden_layer_sizes=(5, 2),␣
 ↪random_state=1,max_iter=2000)
    clf.fit(predicted_train, np.ravel(target_train, order='C'))
    predictions = clf.predict(predicted_test)
    return accuracy_score(target_test, predictions)

pred_train, pred_test, tar_train, tar_test = train_test_split(X, Y, test_size=.
 ↪3, random_state=4)
print("Accuracy score of our model without feature selection : %.2f" %␣
 ↪get_accuracy(tar_train, tar_test, pred_test, pred_train))

# feature extraction
test = SelectKBest(score_func=chi2, k=5)
fit = test.fit(X, Y)
# summarize scores
np.set_printoptions(precision=3)
print(fit.scores_)
features = fit.transform(X)
# summarize selected features
print(features[0:5, :],"summerize features")
print()
# Now apply only the K most significant features according to the chi square␣
 ↪method
pred_features = features[:, 0:5]
pred_train, pred_test, tar_train, tar_test = train_test_split(pred_features, Y,␣
 ↪test_size=.3, random_state=2)
print("Accuracy score of our model with chi square feature selection : %.2f" %␣
 ↪get_accuracy(tar_train, tar_test, pred_test,pred_train))
print()

## Feature Importance with Extra Trees Classifier
from sklearn.feature_selection import SelectFromModel
```

```
# Feature Extraction with RFE
model = LogisticRegression()  # Logistic regression is the Wrapper classifier␣
↪here
rfe = RFE(model, 5)
fit = rfe.fit(X, Y)
## summarize components
#print("Num Features: %d" % (fit.n_features_))
#print("Selected Features: %s" % (fit.support_))
#print("Feature Ranking: %s" % (fit.ranking_))
## Now apply only the K most significant features according to the RFE feature␣
↪selection method
features = fit.transform(X)
pred_features = features[:, 0:5]
pred_train, pred_test, tar_train, tar_test = train_test_split(pred_features, Y,␣
↪test_size=.3, random_state=2)
print("Accuracy score of our model with RFE selection : %.2f" %␣
↪get_accuracy(tar_train, tar_test, pred_test,pred_train))
print()

## Feature Extraction with PCA
## feature extraction
pca = PCA(n_components=5)
fit = pca.fit(X)
features = fit.transform(X)
## summarize components
#print("Explained Variance: %s" % (fit.explained_variance_ratio_))
#print(fit.components_)
## Now apply only the K most significant faetures (components) according to the␣
↪PCA feature selection method
#features = fit.transform(X)
pred_features = features[:, 0:5]
pred_train, pred_test, tar_train, tar_test = train_test_split(pred_features, Y,␣
↪test_size=.3, random_state=2)
print("Accuracy score of our model with PCA selection : %.2f" %␣
↪get_accuracy(tar_train, tar_test, pred_test,pred_train))
print()

## Feature Importance with Extra Trees Classifier
from sklearn.ensemble import ExtraTreesClassifier
## feature extraction
model = ExtraTreesClassifier(max_depth=3,min_samples_leaf=2)
fit = model.fit(X, Y)
print(model.feature_importances_)
print()
t = SelectFromModel(fit, prefit=True)
features = t.transform(X)
```

```
pred_features = features[:, 0:5]
pred_train, pred_test, tar_train, tar_test = train_test_split(pred_features, Y,␣
 ↪test_size=.3, random_state=2)
print("Accuracy score of our model with Extra Trees selection : %.2f" %␣
 ↪get_accuracy(tar_train, tar_test, pred_test, pred_train))
print()
```

```
Accuracy score of our model without feature selection : 0.05
[ 20.321  22.921  21.164  51.545  18.469  33.979  18.496  43.588  56.682
  20.841  26.617   1.917  56.078   5.255   5.473 121.11    1.619     nan
   4.792 149.    ]
[[ 0  0  6 24  0]
 [ 0  0  5 24  0]
 [ 0  0  0 24  0]
 [ 0  0  0 24  0]
 [ 1  1  6 51  1]] summerize features

Accuracy score of our model with chi square feature selection : 0.15


/home/bernard/anaconda3/lib/python3.8/site-
packages/sklearn/utils/validation.py:70: FutureWarning: Pass
n_features_to_select=5 as keyword args. From version 1.0 (renaming of 0.25)
passing these as positional arguments will result in an error
  warnings.warn(f"Pass {args_msg} as keyword args. From version "
/home/bernard/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/bernard/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/bernard/anaconda3/lib/python3.8/site-
```

```
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/bernard/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/bernard/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/bernard/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/bernard/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/bernard/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/bernard/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/bernard/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/bernard/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
```

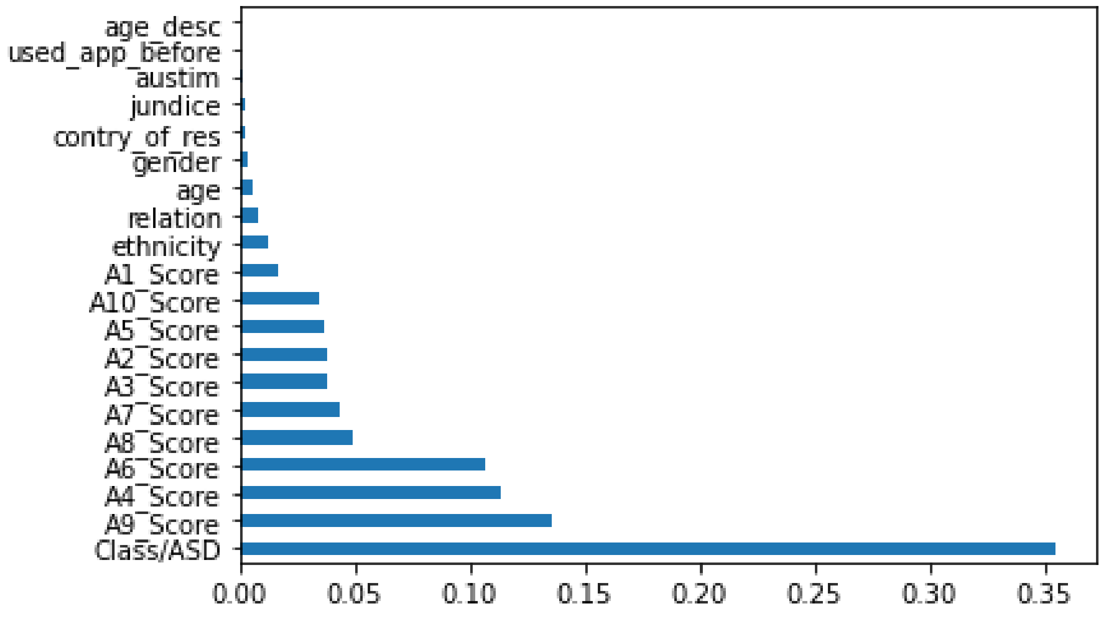
```
regression
  n_iter_i = _check_optimize_result(
/home/bernard/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
Accuracy score of our model with RFE selection : 0.15

Accuracy score of our model with PCA selection : 0.15

[0.026 0.036 0.053 0.115 0.043 0.091 0.04  0.059 0.133 0.038 0.008 0.003
 0.007 0.002 0.002 0.003 0.001 0.    0.01  0.329]

Accuracy score of our model with Extra Trees selection : 0.15
```

3 Get five most important features

[4]:
```
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt

# Get the average of the list
def Average(lst):
    return sum(lst) / len(lst)

## feature extraction
model = ExtraTreesClassifier(max_depth=3,min_samples_leaf=2)
fit = model.fit(X, Y)
print(model.feature_importances_)
print()
feat_importances = pandas.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(20).plot(kind='barh')

t = SelectFromModel(fit, prefit=True)
features = t.transform(X)
pred_features = features[:, 0:5]
pred_train, pred_test, tar_train, tar_test = train_test_split(pred_features, Y,␣
↪test_size=.3, random_state=4)
```

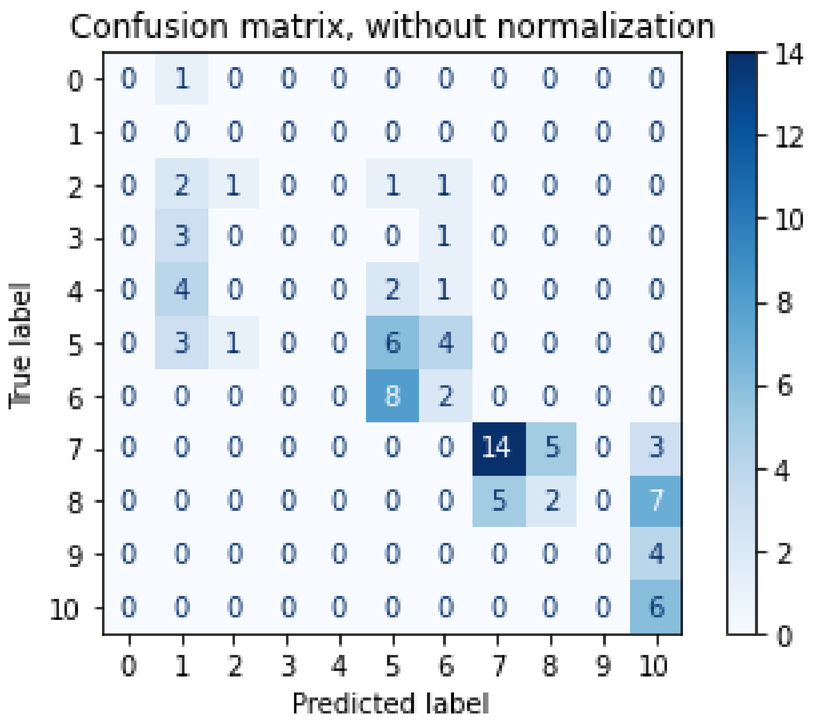
```
[0.017 0.038 0.038 0.114 0.036 0.107 0.043 0.049 0.136 0.035 0.006 0.004
 0.012 0.002 0.001 0.002 0.    0.    0.008 0.355]
```



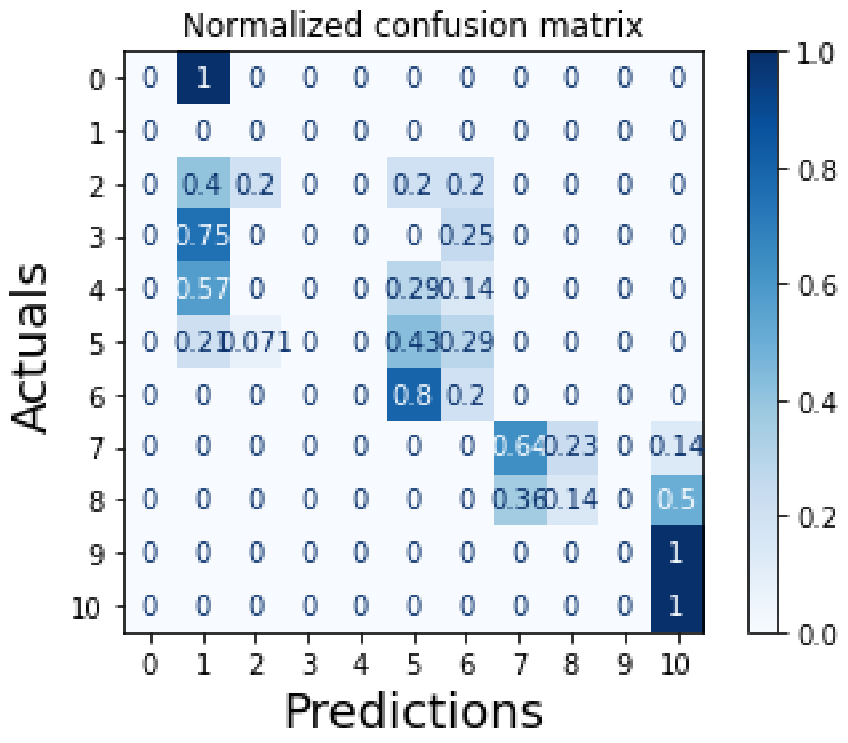
4 Naive Bayes Model

```
[5]: from sklearn.model_selection import KFold
     import numpy as np
     import matplotlib.pyplot as plt
     from sklearn.metrics import precision_score, recall_score, auc, roc_curve

     from sklearn import svm, datasets
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import plot_confusion_matrix

     gnb = GaussianNB() #suitable for numeric features
     gnb.fit(pred_train, np.ravel(tar_train,order='C'))
     kf = KFold(n_splits=10)
     gnb.fit(pred_train, tar_train)

     # Plot non-normalized confusion matrix
     titles_options = [("Confusion matrix, without normalization", None),
                       ("Normalized confusion matrix", 'true')]
     for title, normalize in titles_options:
         disp = plot_confusion_matrix(gnb, pred_test, tar_test,
                                      #display_labels=['yes','no'],
                                      cmap=plt.cm.Blues,
                                      normalize=normalize)
         disp.ax_.set_title(title)

         print(title)
         print(disp.confusion_matrix)

     plt.xlabel('Predictions', fontsize=18)
     plt.ylabel('Actuals', fontsize=18)
     plt.show()

     # Print accuracy
     print(Average(cross_val_score(gnb, pred_train, tar_train, cv=kf,␣
      ↪scoring='accuracy')))
```

```
Confusion matrix, without normalization
[[ 0  1  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0]
 [ 0  2  1  0  0  1  1  0  0  0  0]
 [ 0  3  0  0  0  0  1  0  0  0  0]
 [ 0  4  0  0  0  2  1  0  0  0  0]
 [ 0  3  1  0  0  6  4  0  0  0  0]
 [ 0  0  0  0  0  8  2  0  0  0  0]
 [ 0  0  0  0  0  0  0 14  5  0  3]
 [ 0  0  0  0  0  0  0  5  2  0  7]
 [ 0  0  0  0  0  0  0  0  0  0  4]
 [ 0  0  0  0  0  0  0  0  0  0  6]]
Normalized confusion matrix
[[0.    1.    0.    0.    0.    0.    0.    0.    0.    0.    0.   ]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.   ]
 [0.    0.4   0.2   0.    0.    0.2   0.2   0.    0.    0.    0.   ]
 [0.    0.75  0.    0.    0.    0.    0.25  0.    0.    0.    0.   ]
 [0.    0.571 0.    0.    0.    0.286 0.143 0.    0.    0.    0.   ]
 [0.    0.214 0.071 0.    0.    0.429 0.286 0.    0.    0.    0.   ]
 [0.    0.    0.    0.    0.    0.8   0.2   0.    0.    0.    0.   ]
 [0.    0.    0.    0.    0.    0.    0.    0.636 0.227 0.    0.136]
 [0.    0.    0.    0.    0.    0.    0.    0.357 0.143 0.    0.5  ]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    1.   ]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    1.   ]]
```



	0	1	2	3	4	5	6	7	8	9	10
0	0	1	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0
2	0	2	1	0	0	1	1	0	0	0	0
3	0	3	0	0	0	0	1	0	0	0	0
4	0	4	0	0	0	2	1	0	0	0	0
5	0	3	1	0	0	6	4	0	0	0	0
6	0	0	0	0	0	8	2	0	0	0	0
7	0	0	0	0	0	0	0	14	5	0	3
8	0	0	0	0	0	0	0	5	2	0	7
9	0	0	0	0	0	0	0	0	0	0	4
10	0	0	0	0	0	0	0	0	0	0	6



	0	1	2	3	4	5	6	7	8	9	10
0	0	1	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0.4	0.2	0	0	0.2	0.2	0	0	0	0
3	0	0.75	0	0	0	0	0.25	0	0	0	0
4	0	0.57	0	0	0	0.29	0.14	0	0	0	0
5	0	0.21	0.071	0	0	0.43	0.29	0	0	0	0
6	0	0	0	0	0	0.8	0.2	0	0	0	0
7	0	0	0	0	0	0	0	0.64	0.23	0	0.14
8	0	0	0	0	0	0	0	0.36	0.14	0	0.5
9	0	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	0	0	1

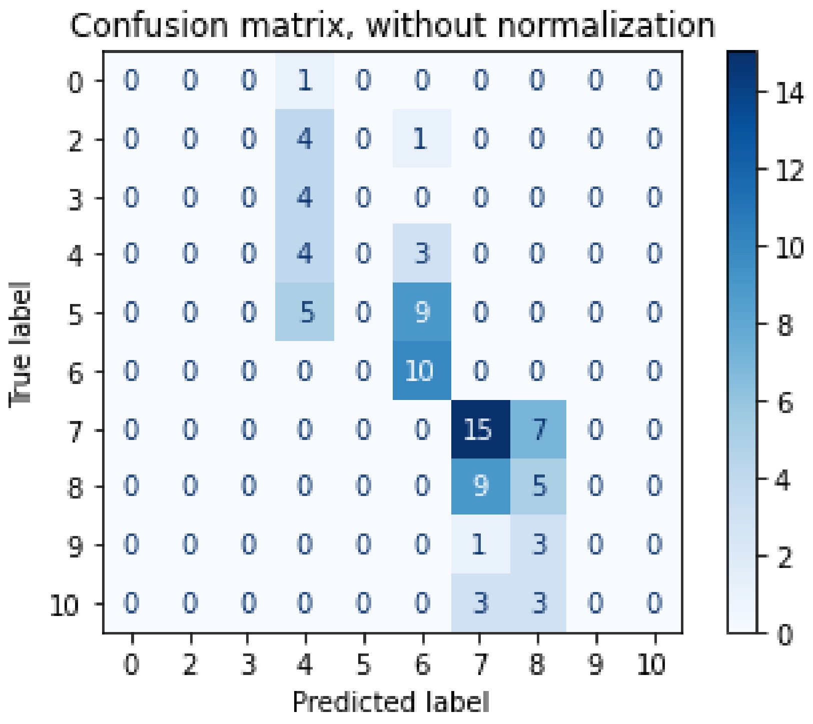
```
0.35261904761904767
```

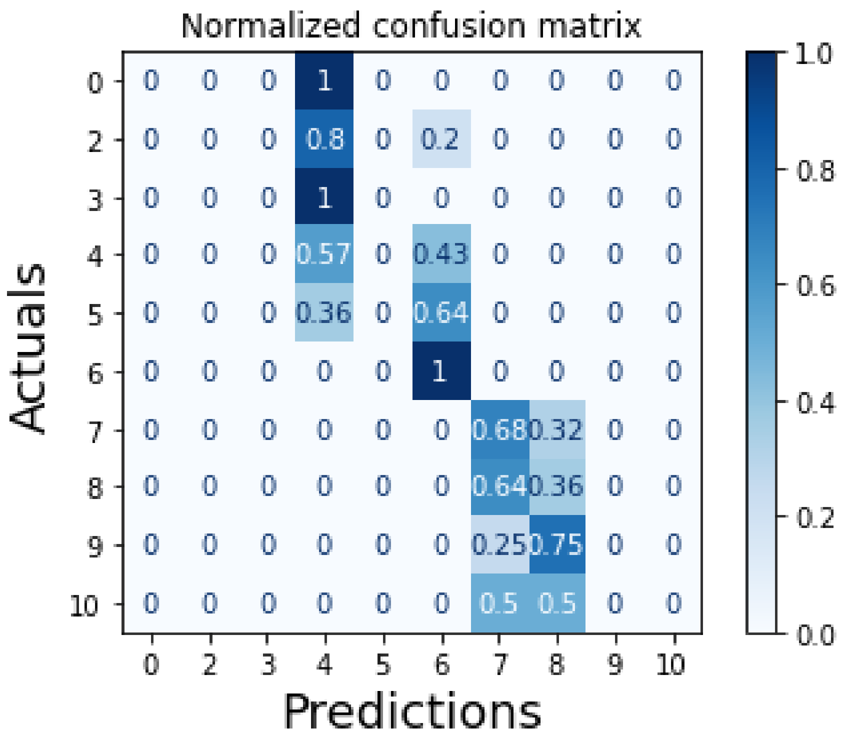
5 Decision Tree Classifier Model

[6]:
```
from sklearn.tree import DecisionTreeClassifier
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import plot_confusion_matrix

# Reset the train/test-split
pred_train, pred_test, tar_train, tar_test = train_test_split(X, Y, test_size=.
 ↪3, random_state=4)

# Train the DTC
dt = DecisionTreeClassifier(min_samples_leaf=0.2, min_samples_split=0.3)
kf = KFold(n_splits=10)
dt.fit(pred_train, tar_train)
train_pred = dt.predict(pred_train)

# Plot non-normalized confusion matrix
titles_options = [("Confusion matrix, without normalization", None),
                  ("Normalized confusion matrix", 'true')]
for title, normalize in titles_options:
    disp = plot_confusion_matrix(dt, pred_test, tar_test,
                                 #display_labels=['yes','no'],
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
    disp.ax_.set_title(title)

    print(title)
    print(disp.confusion_matrix)

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.show()

# print acccuracy
print(Average(cross_val_score(dt, pred_train, tar_train, cv=kf,␣
 ↪scoring='accuracy')))


# get importance
print(dt.feature_importances_)
print()
```

```
feat_importances = pandas.Series(dt.feature_importances_, index=X.columns)
feat_importances.nlargest(20).plot(kind='barh')
```

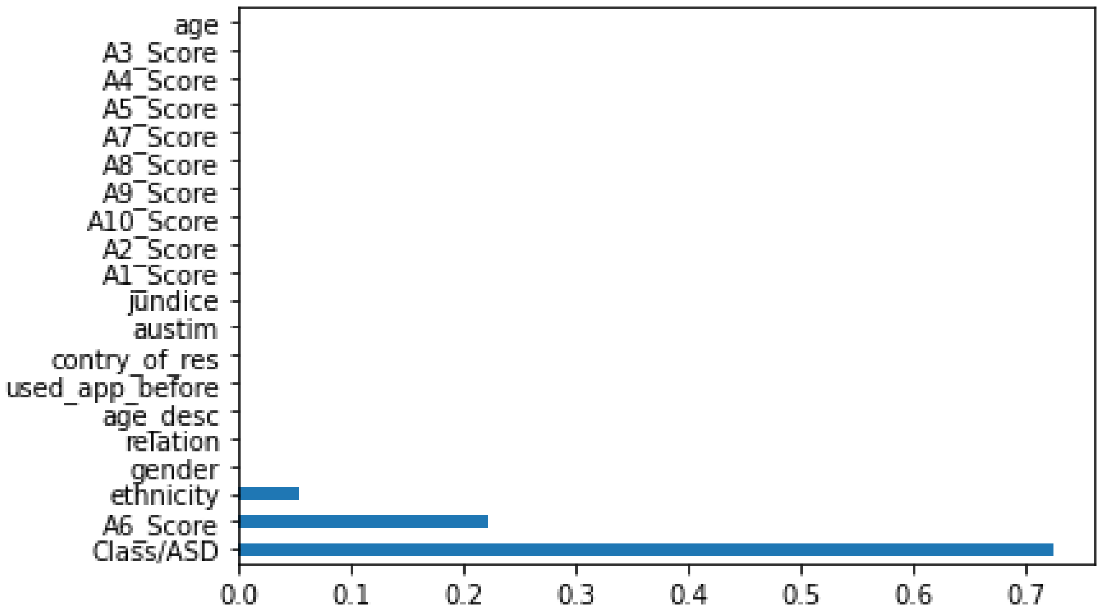
```
Confusion matrix, without normalization
[[ 0  0  0  1  0  0  0  0  0  0]
 [ 0  0  0  4  0  1  0  0  0  0]
 [ 0  0  0  4  0  0  0  0  0  0]
 [ 0  0  0  4  0  3  0  0  0  0]
 [ 0  0  0  5  0  9  0  0  0  0]
 [ 0  0  0  0  0 10  0  0  0  0]
 [ 0  0  0  0  0  0 15  7  0  0]
 [ 0  0  0  0  0  0  9  5  0  0]
 [ 0  0  0  0  0  0  1  3  0  0]
 [ 0  0  0  0  0  0  3  3  0  0]]
Normalized confusion matrix
[[0.    0.    0.    1.    0.    0.    0.    0.    0.    0.   ]
 [0.    0.    0.    0.8   0.    0.2   0.    0.    0.    0.   ]
 [0.    0.    0.    1.    0.    0.    0.    0.    0.    0.   ]
 [0.    0.    0.    0.571 0.    0.429 0.    0.    0.    0.   ]
 [0.    0.    0.    0.357 0.    0.643 0.    0.    0.    0.   ]
 [0.    0.    0.    0.    0.    1.    0.    0.    0.    0.   ]
 [0.    0.    0.    0.    0.    0.    0.682 0.318 0.    0.   ]
 [0.    0.    0.    0.    0.    0.    0.643 0.357 0.    0.   ]
 [0.    0.    0.    0.    0.    0.    0.25  0.75  0.    0.   ]
 [0.    0.    0.    0.    0.    0.    0.5   0.5   0.    0.   ]]
```





```
0.27809523809523806
[0.    0.    0.    0.    0.    0.223 0.    0.    0.    0.    0.    0.
 0.054 0.    0.    0.    0.    0.    0.    0.723]
```

[6]: <AxesSubplot:>


age
A3_Score
A4_Score
A5_Score
A7_Score
A8_Score
A9_Score
A10_Score
A2_Score
A1_Score
jundice
austim
contry_of_res
used_app_before
age_desc
relation
gender
ethnicity
A6_Score
Class/ASD
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7

```
[ ]:
```