2019

Machine Learning

ASSIGNMENT 2

BY GABRIELLA DI GREGORIO 15624188

SCHOOL OF COMPUTER SCIENCE | CMP3751M Machine Learning

Contents

Task 1 2
Importing Libraries, Reading the Data, and Assigning Variables: 2
Data Pre-processing: 2
Describing the Data: 2
Boxplot: 3
Density Plot: 4
Task 2 5
Task 3 7
Task 4 9
References 11

Task 1

Importing Libraries, Reading the Data, and Assigning Variables:

```
#CMP3751M Machine Learning Assignment 2
#By Gabriella Di Gregorio 15624188
#Task 1
#Description and Data Preprocessing

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

#reads in the dataset into dataframe using pandas.read_csv
df = pd.read_csv('CMP3751M_CMP9772M_ML_Assignment 2-dataset-nuclear_plants_final.csv')
df_val = df.values
#assigns all the columns except the first column to the variable X
X = df_val[:,1:13]
#assigns only the first column as variable Y
Y = df_val[:,0]
```

Data Pre-processing:

```
#LabelEncoder encodes target labels with value between 0 and n_classes-1. (scikit-learn.org, n.d.)
#it can be used to normalize labels and transform non-numerical labels to numerical labels (scikit-learn.org, n.d.)
#The status column is not numerical and since machine learning models are based on mathematical equations,
#categorical variables need to be changed to numbers.
#OneHotEncoder encodes categorical features as a one-hot numeric array. (scikit-learn.org, n.d.)
#So, the data in the status column is transformed to numerical values, then put into an array.
labelencoder_X = LabelEncoder()
X[:, 0] = labelencoder_X.fit_transform(X[:, 0])
onehotencoder = OneHotEncoder(categorical_features=[0])
X = onehotencoder.fit_transform(X).toarray()
labelencoder_Y = LabelEncoder()
Y = labelencoder_Y.fit_transform(Y)

#train_test_split is used to split arrays or matrices into random train and test subsets. (scikit-learn.org, n.d.)
#The test size of 0.1 means that 90% of the data will be the training data to train the model,
#and 10% will be the test data to test the model predictions.
#The random state shuffles the data before it is split. 0 uses np.random
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.1, random_state=0)

#StandardScaler standardizes features by removing the mean and scaling to unit variance. (scikit-learn.org, n.d.)
#This process is known as Feature Scaling
#It is important to ensure that all features are on the same scale to avoid issues in the model.
#So this process converts different scales to a standard scale to make it easier for Machine Learning Algorithms (Gupta, 2019)
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

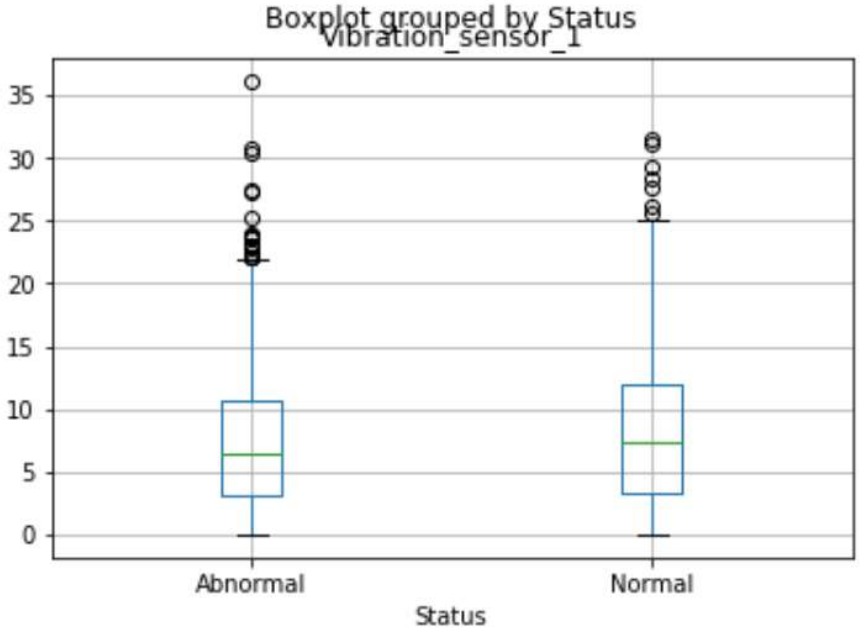
Describing the Data:

```
#pandas.DataFrame.describe generates descriptive statistics that summarize the central tendency,
#dispersion and shape of a dataset's distribution, excluding NaN values. (pandas.pydata.org, n.d.)
#It analyzes both numeric and object series, as well as DataFrame column sets of mixed data types. (pandas.pydata.org, n.d.)
print(df.describe())
#pandas.DataFrame.size returns an int representing the number of elements in this object. (pandas.pydata.org, n.d.)
#Returns the number of rows times number of columns if DataFrame. (pandas.pydata.org, n.d.)
print('\nThe size of the data is: ',df.size)
#pandas.DataFrame.shape returns a tuple representing the dimensionality of the DataFrame. (pandas.pydata.org, n.d.)
print('The shape of the data is: ', df.shape)
#pandas.DataFrame.columns simply counts the number of columns in the DataFrame, and therefore represents the number of features.
print('There are ',len(df.columns), ' features in the dataset')

#pandas.DataFrame.isnull().values.any() checks if any values in the dataframe are null, and returns a boolean.
print('\nAre there any missing values? ', df.isnull().values.any())

print('\nAll columns are of float64 type except for Status...')
#pandas.DataFrame.select_dtypes returns a subset of columns in a dataframe given a type inclusion or exclusion.
#pandas.DataFrame.dtypes returns the data type
print(df.select_dtypes(exclude=['float64']).dtypes)
```

```
       Power_range_sensor_1  Power_range_sensor_2  Power_range_sensor_3   \
count            996.000000            996.000000            996.000000
mean               4.999574              6.379273              9.228112
std                2.764856              2.312569              2.532173
min                0.008200              0.040300              2.583966
25%                2.892120              4.931750              7.511400
50%                4.881100              6.470500              9.348000
75%                6.794557              8.104500             11.046800
max               12.129800             11.928400             15.759900

       Power_range_sensor_4  Pressure _sensor_1  Pressure _sensor_2  \
count            996.000000          996.000000          996.000000
mean               7.355272           14.199127            3.077958
std                4.354778           11.680045            2.126091
min                0.062300            0.024800            0.008262
25%                3.438141            5.014875            1.415800
50%                7.071550           11.716802            2.672400
75%               10.917400           20.280250            4.502500
max               17.235858           67.979400           10.242738

       Pressure _sensor_3  Pressure _sensor_4  Vibration_sensor_1  \
count          996.000000          996.000000          996.000000
mean             5.749234            4.997002            8.164563
std              2.526136            4.165490            6.173261
min              0.001224            0.005800            0.000000
25%              4.022800            1.581625            3.190292
50%              5.741357            3.859200            6.752900
75%              7.503578            7.599900           11.253300
max             12.647500           16.555620           36.186438

       Vibration_sensor_2  Vibration_sensor_3  Vibration_sensor_4
count          996.000000          996.000000          996.000000
mean            10.001593           15.187982            9.933591
std              7.336233           12.159625            7.282383
min              0.018500            0.064600            0.009200
25%              4.004200            5.508900            3.842675
50%              8.793050           12.185650            8.853050
75%             14.684055           21.835000           14.357400
max             34.867600           53.238400           43.231400

The size of the data is:  12948
The shape of the data is:  (996, 13)
There are  13  features in the dataset

Are there any missing values?  False

All columns are of float64 type except for Status...
Status    object
dtype: object
```

Boxplot:

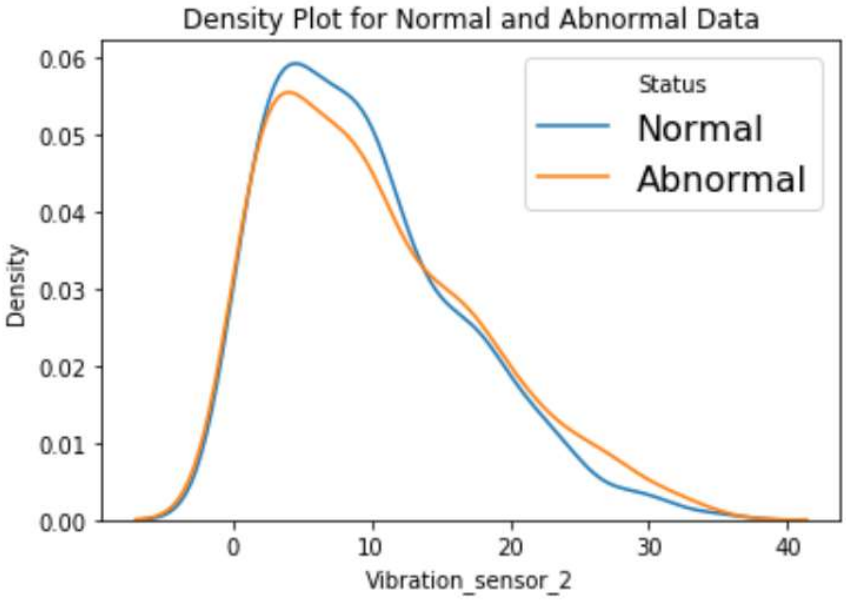
```
#pandas.DataFrame.boxplot makes a box-and-whisker plot from DataFrame columns, optionally grouped by some other columns. (pandas.
#A box plot displays groups of data based on their qualities. They show the upper and lower quartiles, the median,
#and the range of the data.
plt.figure(1)
boxplot = df.boxplot(by='Status', column=['Vibration_sensor_1'])
plt.show()
```



The box plot shows that vibration levels of 0 are within the acceptable range for both abnormal and normal cases for vibrations sensor 1. As well of this, the lower quartiles of these readings are the same for both abnormal and abnormal cases too. However, the data begins to differ with the mean. The mean vibration levels are actually slightly higher in normal cases than in abnormal cases, but since this difference is very small, the mean would not be the best way to detect whether the readings are abnormal or normal because they could fluctuate. Similarly, the upper quartile is only slightly higher in the normal cases, which again would not make it a useful measure to determine the status of the readings. However, the acceptable range for normal cases is higher than those of normal cases, resulting in much more data being classed as outliers for abnormal data. The upper range of normal data is 25 and the upper range of abnormal data is approximately 22 but there is a large cluster of outliers in the abnormal data between 22 and 25. Furthermore, there is one outlier of around 36 in the abnormal data which is higher than any of the normal data, this suggests that high vibration readings might be signs of an abnormal situation. However, since the two plots are relatively similar overall, it may nut be the best indicator of the status.

Density Plot:

```
#The following code is used to make a density plot to show Vibration_sensor_2...
Status = ['Normal', 'Abnormal']

#This for loop goes through each value in the Status column
for stat in Status:
    subset = df[df['Status'] == stat]

    #seaborn.distplot flexibly plots a univariate distribution of observations. (seaborn.pydata.org, n.d.)
    #This function combines the matplotlib hist function (with automatic calculation of a good default bin size),
    #with the seaborn kdeplot() and rugplot() functions.(seaborn.pydata.org, n.d.)
    #Since there are two plots, the histogram is not displayed to avoid confusion, a kde plots a gaussian kernel density estimate,
    #and a label is a legend for the relevant plot component.
    sns.distplot(subset['Vibration_sensor_2'], hist = False, kde = True, label = stat)

#This simply formats the density plot to give it a title, a key, and labeled axis
plt.legend(prop={'size': 16}, title = 'Status')
plt.title('Density Plot for Normal and Abnormal Data')
plt.xlabel('Vibration_sensor_2')
plt.ylabel('Density')
```



A Density Plot visualises the distribution of data over a continuous interval. This chart is a variation of a Histogram that uses kernel smoothing to plot values, allowing for smoother distributions by smoothing out the noise. The peaks of a Density Plot help display where values are concentrated over the interval. (Datavizcatalogue.com, n.d.) This density plot shows that the majority of both abnormal and normal vibration readings for sensor 2 lie around 5-15 because this is where they both peak. However, it shows that there are more lower readings of around 5-15 for normal data because the peak is higher than that of normal data. Furthermore, there are more higher readings of around 15-35 for abnormal data since the density line is higher than normal data. This would also suggest that higher vibration levels could indicate an abnormal situation.

Task 2

Although an accuracy score of 90% is relatively high and might be the result of the best model, there are several other factors to consider when deciding which model to use besides this.

Firstly, the way the data is split can have an impact on the accuracy score. The intern split the data into 70% training, 20% validation, and 10% testing. The training set is the data that is used to train the model, it learns from this data. Obviously, the larger the training data, the more material the model has been taught with so this set should always be the largest. A 70% split into the training set is reasonable, though it could be higher. The validation set is the sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration. (Shah, 2017) The validation set is not always necessary depending on the model hyperparameters and because a cross-validation method can be used instead. Since the models/classifiers that have been used are not specified, the hyperparameters are unknown. However, since the model neither learns or gets tested by this set, 20% of the data seems too much in this case. Furthermore, the test set is key in evaluating the model. Once the model has been fully trained, it must be carefully tested to evaluate how well it fits the data. Since this step is so important and always necessary, a 10% split may be too small. The test set should probably be the same size as the validation set if not larger. The size of the test set can have a significant impact on the accuracy result. For example, 10% of the data might not fit the model well but there is chance that this proportion of data is anomalous, therefore 20% of the data may fit better and generate a

higher accuracy score. Based on this data set and the limited information, I think a split of 80% training, 20% testing, and no validation set but instead use cross-validation would be the best solution.

To avoid the need to cut down the size of the training set for a validation set, which can cause underfitting models, k-fold cross validation can be used instead. This is where the data is divided into k subsets and the holdout method is repeated k times, such that each time, one of the k subsets is used as the test/validation set and the other subsets are put together to form a training set. The error estimation is averaged over all k trials to get the total effectiveness of the model. Every data point gets to be in a validation set exactly once and gets to be in a training set k-1 times. This significantly reduces bias as most of the data is used for fitting, and also significantly reduces variance as most of the data is also being used in validation set. Interchanging the training and test sets also adds to the effectiveness of this method. (Gupta, 2017) This proves that using an 80/20 train-test split and replacing the need for a validation set by performing k-fold cross-validation instead would undoubtably lead to more accurate results since the model will have been trained with more data and been tested with more data too.

Furthermore, the training methods used may have had an impact on the results. The intern sometimes used just a subset of the available features, or by using different types of classifiers. Whilst varying the classifiers is a good way to compare different models, since some classifiers may be better in this case than others, varying the data being used is not. The purpose is to assess how all of the variables can be used to detect a normal or abnormal situation, not just a select few. For example, if vibration level readings were left out of the model and this data happens to be a key indicator of the reactor status, the model would not be useful or helpful. It is not a fair test to compare different models that are not being trained with the same data. Therefore, the result that scored the highest accuracy might not be the best since if it did not include all the features, it is not as useful as a model that did but perhaps scored lower accuracy.

Lastly, there are many other types of evaluation metrics that can be used to evaluate a model besides accuracy. Whilst the classification accuracy is the most used metric, it is not enough to truly judge a Machine Learning model, others should be assessed. The classification accuracy is the ratio of the number of correct predictions to the total number of input samples. However, it only works well if there are equal number of samples belonging to each class. (Mishra, 2018) In this scenario, the sample distribution is equal which does make accuracy a good metric to assess the model, but that does not mean that others shouldn't be considered. Logarithmic Loss or Log Loss works by penalising the false classifications. It works well for multi-class classification. When working with Log Loss, the classifier must assign probability to each class for all the samples. In general, minimising Log Loss gives greater accuracy for the classifier. (Mishra, 2018) The data for this scenario is multi-class classification since the data can only belong to either a normal or abnormal status. Therefore, Log Loss would be another valid way of evaluating the model. Another way to evaluate a model is by using a Confusion Matrix which outputs a matrix that describes the performance of the model. It is used in cases where there are two classes, but the classifier also predicts a class for a given input sample. In this case, these would be 'actual normal', 'actual abnormal', 'predicted normal' and 'predicted abnormal'. It is calculated by dividing the sum of true positives and false negatives by the total number of samples. (Mishra, 2018) Since it applies to this scenario well, it would have also been an interesting metric to evaluate the model.

Overall, there is no way of definitively saying wither the model that scored 90% accuracy is the best to use or not based on the information available. However, it is evident that the best method carried out to come to this conclusion was by far the best. The intern's methods could have been

significantly improved by using larger train and test sets, using k-fold cross validation rather than a validation set, using all features to fairly train all models, and using a variety of evaluation metrics besides accuracy to determine the best model. With all of these changes in place, there is significant chance that the model selected to be the best would be different from the one that scored 90% accuracy using his relatively poor methods.

Task 3

```
#CMP3751M Machine Learning Assignment 2
#By Gabriella Di Gregorio 15624188
#Task 3
#Artificial Neural Networks

from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
```

```
#sklearn.neural_network.MLPClassifier: MLP stands for a multi-layer perceptron classifier.
#This model optimizes the log-loss function using LBFGS or stochastic gradient descent. (scikit-learn.org)
#Solver is the solver for weight optimisation. lbfgs is an optimizer in the family of quasi-Newton methods. (scikit-learn.org)
#This solver was chosen because it performs relatively quickly
#Activation function for the hidden layer: 'logistic', the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).(scikit-le
#hidden_layer_sizes: the ith element represents the number of neurons in the ith hidden layer. (scikit-learn.org)
clf = MLPClassifier(solver='lbfgs', activation='logistic', hidden_layer_sizes=(500,500))
#Fits the model to data matrix X and target Y.
clf.fit(X_train,Y_train)
#Predicts using the multi-layer perceptron classifier
Y_predict = clf.predict(X_test)

#sklearn.metrics.confusion_matrix computes confusion matrix to evaluate the accuracy of a classification.
#By definition a confusion matrix C is such that Cij is equal to the number of observations known to be in group i
#and predicted to be in group j.(scikit-learn.org)
#Takes the groud truth/real values and the estimated targets returned by the classifier
confusion_matrix(Y_test, Y_predict)

#Since all the data was converted to numerical, it was helpful to give them labels to better understand the displayed data.
target_names = ['normal', 'abnormal']
#sklearn.metrics.classification_report build a text report showing the main classification metrics.
#It returns a text summary of the precision, recall, and F1 score for each class, as well as the averages of these.
#It takes the groud truth/real values and the estimated targets returned by the classifier
#and target_names which are display names matching the labels
print('Artifical Neural Network:')
print(classification_report(Y_test, Y_predict, target_names=target_names))

#sklearn.metrics.accuracy_score
#In multilabel classification, this function computes subset accuracy:
#the set of labels predicted for a sample must exactly match the corresponding set of labels in y_true.(scikit-learn.org)
#Takes the ground truth/correct labels and the predicted labels returned by the classifier.
print('accuracy = ', accuracy_score(Y_test, Y_predict))
```

```
#sklearn.ensemble.RandomForestClassifier
#A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset
#and uses averaging to improve the predictive accuracy and control over-fitting.(scikit-learn.org)
#The sub-sample size is always the same as the original input sample size
#but the samples are drawn with replacement because bootstrap=true is left by default. (scikit-learn.org)
#The n_estimators is the number of trees in the forest.
#The min_samples_leaf is 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. (scikit-learn.org)
rfc5 = RandomForestClassifier(n_estimators=1000,min_samples_leaf=5)
#Builds a forest of trees from the training set (X, y).
rfc5.fit(X_train, Y_train)
#Predict class for X.
#The predicted class of an input sample is a vote by the trees in the forest, weighted by their probability estimates.
#That is, the predicted class is the one with highest mean probability estimate across the trees. (scikit-learn.org)
#The same steps are repeated that were performed on the MLP Classifier
rfc5_predict = rfc5.predict(X_test)
confusion_matrix(Y_test, rfc5_predict)
print('\nRandom Forest for 5 leaf nodes:')
print(classification_report(Y_test, rfc5_predict, target_names=target_names))
print('accuracy = ', accuracy_score(Y_test, rfc5_predict))
#The same is repeated for 50 leaf nodes instead of 5.
rfc50 = RandomForestClassifier(n_estimators=1000,min_samples_leaf=50)
rfc50.fit(X_train, Y_train)
rfc50_predict = rfc50.predict(X_test)
confusion_matrix(Y_test, rfc50_predict)
print('\nRandom Forest for 50 leaf nodes:')
print(classification_report(Y_test, rfc50_predict, target_names=target_names))
print('accuracy = ', accuracy_score(Y_test, rfc50_predict))
```

```
Artifical Neural Network:
             precision    recall  f1-score   support

     normal       0.58      0.78      0.66        49
   abnormal       0.68      0.45      0.54        51

avg / total       0.63      0.61      0.60       100

accuracy =  0.61

Random Forest for 5 leaf nodes:
             precision    recall  f1-score   support

     normal       0.76      0.76      0.76        49
   abnormal       0.76      0.76      0.76        51

avg / total       0.76      0.76      0.76       100

accuracy =  0.76

Random Forest for 50 leaf nodes:
             precision    recall  f1-score   support

     normal       0.71      0.71      0.71        49
   abnormal       0.73      0.73      0.73        51

avg / total       0.72      0.72      0.72       100

accuracy =  0.72
```

As the name suggests, a multi-layer perceptron (MLP) is composed of multiple perceptrons. They are composed of an input layer to receive the signal, an output layer that makes a decision or prediction about the input, and in between, an arbitrary number of hidden layers that are the true computational engine of the MLP. (Pathmind, n.d.) An MLP can be viewed as a logistic regression classifier where the input is first transformed using a learnt non-linear transformation. This transformation projects the input data into a space where it becomes linearly separable, and this intermediate layer is referred to as a hidden layer. (Deeplearning.net, 2018) Multilayer perceptrons are often applied to supervised learning problems; they train on a set of input-output pairs and learn to model the correlation between those inputs and outputs. Training involves adjusting the parameters, or the weights and biases, of the model in order to minimize error. Backpropagation is used to make those weigh and bias adjustments relative to the error, and the error itself can be measured in a variety of ways. (Pathmind, n.d.)

A Random Forest consists of a large number of individual decision trees that operate as one, and each individual tree generates a class predication and the class with the most ‘votes’ becomes the model’s prediction. The reason that a random forest model works well is based on the idea that a large number of relatively uncorrelated models (trees) operating together will outperform any of the individual constituent models. (Yiu, 2019) It differs from a forest model which would average the prediction of the tress because random sampling of training data points is used when the trees are built, and random subsets of features are considered when splitting the nodes. (Koehrsen, 2018) So, each tree learns from a random sample of data points during training and this is done by bootstrapping which means that the samples are drawn with replacement, so some samples will be used several times in a tree. Although each tree could have a high variance, by training each tree on different samples, the entire forest will have lower variance without the cost of increasing bias. During testing, predictions are made by calculating the average of the predictions of each decision tree.

The output presents several metrics on the performance of both algorithms; the precision, recall, f1-score, support, and accuracy. The precision is the ratio of true positives to the number of true positives plus the number of false negatives. The precision represents the ability of the classifier to not label a negative sample as a positive one. (Scikit-learn.org, n.d.) So, a higher precision score is better because this would mean that more samples have been correctly labelled. The classification reports show that the MLP classifier had the lowest precision score of around 0.61 and the Random Forest classifiers both scored much higher, with the 5-leaf node model scoring the highest at around 0.76. This means that more data was correctly labelled using this model. The recall is the ratio of true positives to the sum of true positives and false negatives, this represents the ability of the classifier to find all of the positive samples. (Scikit-learn.org, n.d.) Again, a higher recall score means better model results. These scores were often the same as the precision scores for all models, which again suggests that the random forest classifiers performed better than the MLP, and the RFC with 5 leaf nodes had the best results. The f1-score can be interpreted as a weighted harmonic mean of the precision and recall. (Scikit-learn.org, n.d.) This is almost a way to summarise the results of both previous scores, so again the results are similar, with the RFC with 5 leaf nodes scoring the highest. The support is simply the number of occurrences of each class in the true values of y, since this will be the same for all models as it is an interpretation of the dataset, this is not useful in representing the success of a model. However, an accuracy score was also generated which computes subset accuracy in multi-label classification, so that the set of labels predicted true for a sample must exactly match the corresponding set of labels in the true y. (Scikit-learn.org, n.d.) As discovered in the previous section, the accuracy score is the most popular evaluation metric but does not always guarantee to represent the best model. However, like all of the previous metrics, the accuracy score suggests that the random forest classifier with 5 leaf nodes is the best out of the 3 models, since both RFC models scored higher accuracy than the MLP and the fewer leaf nodes gave the best result.

Task 4

```
from sklearn.model_selection import cross_val_score
import numpy as np

#Using the cross-validation (CV) function replaces the need to sacrifice some training data for a validation set
#In k-fold, the training set is split into k smaller sets and the model is trained using k-1 of the folds as training data
#and the other as test. k is 10 in this case.
#The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop.
#By default, the score computed at each CV iteration is the score method of the estimator. (scikit-learn.org)
#The CV is used on MLP and RFC models, varying the number of hidden layers and the number of trees respectively.
#It is calculated by passing the classifiers from the section above, changing the hidden layer sizes for MLP
#and the n estimators for RFC. the x, y, and the 10 folds is also passed.
#The mean score is then calculated for each model.

print('Cross Validation Scores for MLP Classifier with 50 hidden layers:')
scores_ANN50 = cross_val_score(MLPClassifier(solver='lbfgs', activation='logistic', hidden_layer_sizes=(50,50)), X, Y, cv=10)
print(scores_ANN50)
print('mean = ', np.mean(scores_ANN50))

print('\nCross Validation Scores for MLP Classifier with 500 hidden layers:')
scores_ANN500 = cross_val_score(MLPClassifier(solver='lbfgs', activation='logistic', hidden_layer_sizes=(500,500)), X, Y, cv=10)
print(scores_ANN500)
print('mean = ', np.mean(scores_ANN500))

print('\nCross Validation Scores for MLP Classifier with 1000 hidden layers:')
scores_ANN1000 = cross_val_score(MLPClassifier(solver='lbfgs', activation='logistic', hidden_layer_sizes=(1000,1000)), X, Y, cv=1
print(scores_ANN1000)
print('mean = ', np.mean(scores_ANN1000))

print('\nCross Validation Scores for Random Forest Classifier with 20 trees:')
scores_rfc20 = cross_val_score(RandomForestClassifier(n_estimators=20), X, Y, cv=10)
print(scores_rfc20)
print('mean = ', np.mean(scores_rfc20))

print('\nCross Validation Scores for Random Forest Classifier with 500 trees:')
scores_rfc500 = cross_val_score(RandomForestClassifier(n_estimators=500), X, Y, cv=10)
print(scores_rfc500)
print('mean = ', np.mean(scores_rfc500))

print('\nCross Validation Scores for Random Forest Classifier with 10000 trees:')
scores_rfc10000 = cross_val_score(RandomForestClassifier(n_estimators=10000), X, Y, cv=10)
print(scores_rfc10000)
print('mean = ', np.mean(scores_rfc10000))
```

```
Cross Validation Scores for MLP Classifier with 50 hidden layers:
[ 0.37        0.47        0.51        0.56        0.81        0.57        0.52
  0.49        0.52040816  0.5       ]
mean =  0.532040816327

Cross Validation Scores for MLP Classifier with 500 hidden layers:
[ 0.42        0.49        0.68        0.55        0.81        0.58        0.42
  0.54        0.54081633  0.52040816]
mean =  0.55512244898

Cross Validation Scores for MLP Classifier with 1000 hidden layers:
[ 0.45        0.52        0.6         0.62        0.72        0.48        0.51
  0.58        0.45918367  0.52040816]
mean =  0.545959183673

Cross Validation Scores for Random Forest Classifier with 20 trees:
[ 0.31        0.51        0.72        0.66        0.84        0.6         0.69
  0.78        0.47959184  0.59183673]
mean =  0.618142857143

Cross Validation Scores for Random Forest Classifier with 500 trees:
[ 0.31        0.49        0.69        0.7         0.86        0.53        0.66
  0.77        0.53061224  0.66326531]
mean =  0.620387755102

Cross Validation Scores for Random Forest Classifier with 10000 trees:
[ 0.29        0.49        0.7         0.71        0.86        0.52        0.66
  0.75        0.51020408  0.63265306]
mean =  0.612285714286
```

Cross Validation is a technique used to test the effectiveness of machine learning models. It works by randomly shuffling the dataset then splitting it into k groups. For each of the number of groups, the model is trained used k-1 sets and the other group as the test set. The model is fit on the training set and evaluated on the test set, and the evaluation score is retained each time in order to summarize the skill of the model using the sample of model evaluation scores. (Brownlee, 2018) So for this case, the data is split into 10 equal parts and the process is repeated so that each group is in the test set once.

The results of the cross-validation scores suggest that 500 is the best number of hidden layers to use for MLP classifier and for the number of trees in the Random Forest. A lower number of hidden layers or trees scored lower than higher amounts in both cases, however, very large numbers such as 1000 decreased the score in comparison to 500. Although 500 was the best out of the 3 in these cases, it doesn't definitely mean that this is the best number to use. For example, for the MLP classifier, 50 hidden layers scored around 0.02 less than 500, but 1000 only scored around 0.01 less. This may mean that a number in between 500 and 1000, such as 700 may give an even better score. However, the difference between the score for 20 and 1000 trees compared to 500 is approximately the same, which may mean that 500 is more likely to be the best number for RFC classifiers.

After assessing the classification reports, the accuracy scores, and the cross-validation scores, it is clear that the random forest classifier is a significantly better model for this scenario. The accuracy score when using RFC was at least 10% better than MLP. Since accuracy is a commonly used metric which works well for this dataset, these results are significant and strongly suggest that RFC is better. Furthermore, various combinations of the number of trees or leaf nodes always scored better in the precision, recall, f1, and cross-validation scores. Although RFC is clearly the better choice in this scenario, it is not always the case. ANNs work with more data types such as images, audio, and text whereas RFC can only work with tabular data. (KDnuggets, 2019) But since this data is indeed tabular, RFC is the better choice not only because of the better results it generated, but also because it tends to be easier to work with. Random Forests not only achieve similarly good performance results in practical application in many areas, they also have some advantages compared to Neural Networks in specific cases. This includes their robustness as well as benefits in

cost and time, due to the ability to vary many parameters such as the number of tress and maximum depth of a tree, and since they require less input preparation and are quick to train and optimise. They are particularly advantageous in terms of interpretability. (Roßbach, 2018) They are easier to understand and require less expertise to evaluate, so even if an ANN model scored slightly higher in accuracy, an RFC may still be preferred to avoid unnecessary complications.

References

Brownlee, J. (2018). *A Gentle Introduction to k-fold Cross-Validation*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/k-fold-cross-validation/ [Accessed 17 Dec. 2019].

Datavizcatalogue.com. (n.d.). *Density Plot - Learn about this chart and tools to create it*. [online] Available at: https://datavizcatalogue.com/methods/density_plot.html [Accessed 14 Dec. 2019].

Deeplearning.net. (2018). *Multilayer Perceptron — DeepLearning 0.1 documentation*. [online] Available at: http://deeplearning.net/tutorial/mlp.html [Accessed 17 Dec. 2019].

Gupta, P. (2017). Cross-Validation in Machine Learning. [online] Towards Data Science | Medium. Available at: https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f [Accessed 14 Dec. 2019].

Gupta, T. (2019). *Data Preprocessing in Python*. [online] Towards Data Science | Medium. Available at: https://towardsdatascience.com/data-preprocessing-in-python-b52b652e37d5 [Accessed 12 Dec. 2019].

KDnuggets. (2019). *Random Forests vs Neural Networks: Which is Better, and When?* [online] Available at: https://www.kdnuggets.com/2019/06/random-forest-vs-neural-network.html [Accessed 17 Dec. 2019].

Koehrsen, W. (2018). *An Implementation and Explanation of the Random Forest in Python*. [online] Towards Data Science | Medium. Available at: https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf308a9b76 [Accessed 17 Dec. 2019].

Mishra, A. (2018). *Metrics to Evaluate your Machine Learning Algorithm*. [online] Towards Data Science | Medium. Available at: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234 [Accessed 14 Dec. 2019].

Pandas.pydata.org. (2019). *pandas: powerful Python data analysis toolkit — pandas 0.25.3 documentation*. [online] Available at: https://pandas.pydata.org/pandas-docs/stable/index.html [Accessed 12 Dec. 2019].

Pathmind. (n.d.). *A Beginner's Guide to Multilayer Perceptrons (MLP)*. [online] Available at: https://pathmind.com/wiki/multilayer-perceptron [Accessed 17 Dec. 2019].

Roßbach, P. (2018). *Neural Networks vs. Random Forests – Does it always have to be Deep Learning?* [online] Frankfurt School Blog. Available at: https://blog.frankfurt-school.de/neural-networks-vs-random-forests-does-it-always-have-to-be-deep-learning/# [Accessed 17 Dec. 2019].

Scikit-learn.org. (2019). *scikit-learn: machine learning in Python — scikit-learn 0.22 documentation*. [online] Available at: https://scikit-learn.org/stable/index.html [Accessed 12 Dec. 2019].

Seaborn.pydata.org. (2018). *seaborn.distplot — seaborn 0.9.0 documentation*. [online] Available at: https://seaborn.pydata.org/generated/seaborn.distplot.html [Accessed 12 Dec. 2019].

Shah, T. (2017). *About Train, Validation and Test Sets in Machine Learning*. [online] Towards Data Science | Medium. Available at: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7 [Accessed 14 Dec. 2019].

Yiu, T. (2019). *Understanding Random Forest*. [online] Towards Data Science | Medium. Available at: https://towardsdatascience.com/understanding-random-forest-58381e0602d2 [Accessed 17 Dec. 2019].