**Machine Learning Assignment 2: Benjamin Terrill**

# Table of Contents

# Section 1: Data Import, Summary, Pre-processing and Visualisation (20%)

## Step 1 – Import the Data

The first step needed to be taken was to import the data set from the .csv file into my python IDE. For this assignment the IDE I will be using is Spyder. To import the data I used the pandas library and used the code shown below to import the data into a variable called "df".

```
df = pd.read_csv('NuclearPowerDataset.csv')
#print('dataset', df)
```

## Step 2 – Data Summary

I created a data summary for each feature in the dataset. This meant finding the mean, minimum, maximum, variance, etc. for each of the features.

Before doing this, I also checked for any missing values in the data. It is important to remove any missing values in the dataset before continuing as it can cause any data processing to become biased and inaccurate. To do this I used the following code:

```
#Check for missing data
df.isnull()
#NO MISSING VALUES FOUND#
```

I did not find any missing data in the dataset so no further action needed to be taken. If there was missing data I would remove it before continuing.

After this I created a function called "data_summary" that will be used to get the summary of each feature.

I used the pandas library to make this step easier. This meant all I had to do was create a variable for each value and use the df.mean/sum/etc. to get the values.

A code snippet with some examples are shown to the right. After finding the values for each I print them out for the user to see.

```
def data_summary(df):
    #Task 1 - Data Summary#
    #Mean of each feature
    Means = df.mean()
    print("Mean of each feauture: ")
    print(Means)
    print(" ")
    #Sum of each feature
    Sums = df.sum()
    print("Sum of each feature: ")
    print(Sums)
    print(" ")
    #Min of each feature
    Mins = df.min()
    print("Min of each feature: ")
    print(Mins)
    print(" ")
    #Max of each feature
    Max = df.max()
    print("Max of each feature: ")
    print(Max)
    print(" ")
```

## Results

Here I will display the results for the data summary.

```
Mean of each feauture:
Power_range_sensor_1       4.999574
Power_range_sensor_2       6.379273
Power_range_sensor_3       9.228112
Power_range_sensor_4       7.355272
Pressure_sensor_1         14.199127
Pressure_sensor_2          3.077958
Pressure_sensor_3          5.749234
Pressure_sensor_4          4.997002
Vibration_sensor_1         8.164563
Vibration_sensor_2        10.001593
Vibration_sensor_3        15.187982
Vibration_sensor_4         9.933591
dtype: float64
```

```
Min of each feature:
Status                     Abnormal
Power_range_sensor_1         0.0082
Power_range_sensor_2         0.0403
Power_range_sensor_3        2.58397
Power_range_sensor_4         0.0623
Pressure_sensor_1            0.0248
Pressure_sensor_2          0.008262
Pressure_sensor_3          0.001224
Pressure_sensor_4            0.0058
Vibration_sensor_1                0
Vibration_sensor_2           0.0185
Vibration_sensor_3           0.0646
Vibration_sensor_4           0.0092
dtype: object
```

```
Max of each feature:
Status                      Normal
Power_range_sensor_1       12.1298
Power_range_sensor_2       11.9284
Power_range_sensor_3       15.7599
Power_range_sensor_4       17.2359
Pressure_sensor_1          67.9794
Pressure_sensor_2          10.2427
Pressure_sensor_3          12.6475
Pressure_sensor_4          16.5556
Vibration_sensor_1         36.1864
Vibration_sensor_2         34.8676
Vibration_sensor_3         53.2384
Vibration_sensor_4         43.2314
dtype: object
```

```
Standard Deviation of each feature:
Power_range_sensor_1       2.764856
Power_range_sensor_2       2.312569
Power_range_sensor_3       2.532173
Power_range_sensor_4       4.354778
Pressure_sensor_1         11.680045
Pressure_sensor_2          2.126091
Pressure_sensor_3          2.526136
Pressure_sensor_4          4.165490
Vibration_sensor_1         6.173261
Vibration_sensor_2         7.336233
Vibration_sensor_3        12.159625
Vibration_sensor_4         7.282383
dtype: float64
```

```
Variance of each feature:
Power_range_sensor_1       7.644426
Power_range_sensor_2       5.347974
Power_range_sensor_3       6.411900
Power_range_sensor_4      18.964093
Pressure_sensor_1        136.423460
Pressure_sensor_2          4.520263
Pressure_sensor_3          6.381361
Pressure_sensor_4         17.351306
Vibration_sensor_1        38.109156
Vibration_sensor_2        53.820310
Vibration_sensor_3       147.856474
Vibration_sensor_4        53.033107
dtype: float64
```

```
Sum of each feature:
Status                   rmalNorm...
Power_range_sensor_1         4979.58
Power_range_sensor_2         6353.76
Power_range_sensor_3          9191.2
Power_range_sensor_4         7325.85
Pressure_sensor_1            14142.3
Pressure_sensor_2            3065.65
Pressure_sensor_3            5726.24
Pressure_sensor_4            4977.01
Vibration_sensor_1            8131.9
Vibration_sensor_2           9961.59
Vibration_sensor_3           15127.2
Vibration_sensor_4           9893.86
dtype: object
```

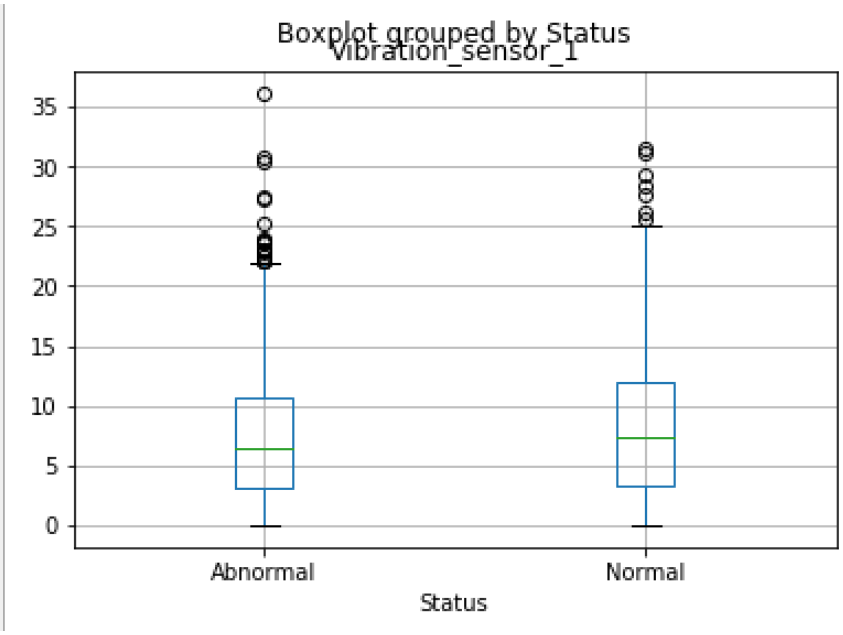
## Pre-Processing

To find the size of the data, I used a count to find the number of features and the number of rows for each feature. From this I got the output of 13 features all with 996 rows. This comes to a total of 12948 cells in the data set.

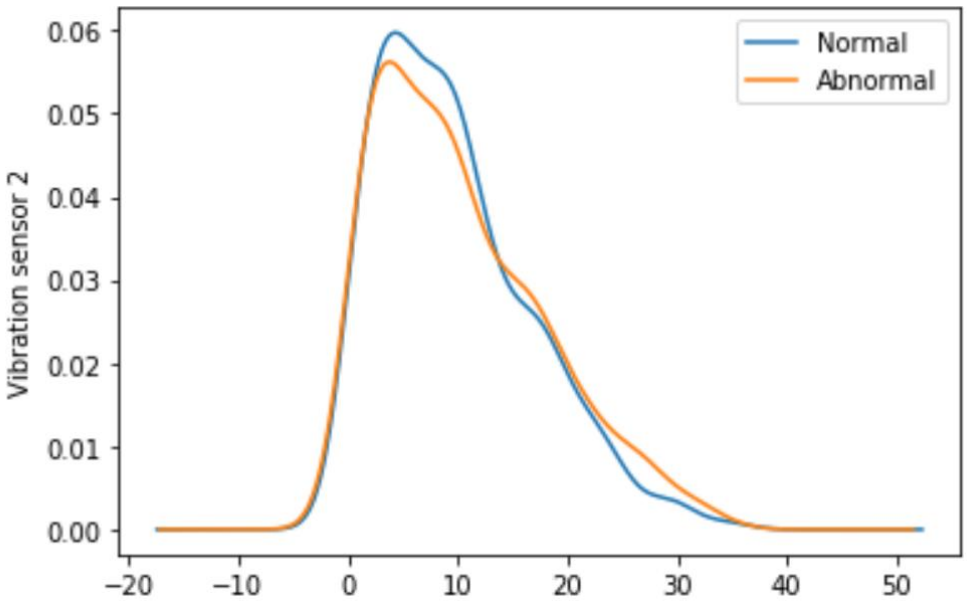
I also found that there is one categorical variable. This is the status feature. For this dataset, this variable holds one of two values, normal or abnormal compared to every other feature in the dataset which holds a numerical value.

## Visualisation

### *Box Plot*




### *Density Plot*



*Code for Plots*

Below is a code snippet for how I created bot the box plot and density plot.

```
#Plots
STA1 = df.iloc[:, [0,9]]

#Box Plot
bp = STA1.boxplot(column='Vibration_sensor_1', by='Status')
plt.show()

STA2 = df.groupby("Status")["Vibration_sensor_2"].plot.kde()
```

From these plots I can gather some information about the differences between the Normal and Abnormal data. For example, in the box plot for vibration_sensor 1, I can see that the abnormal data has several more outliers, whilst having a lower mean and a lower range. This would suggest that these outliers are what cause the sensor to detect abnormal results. This continues onto the density plot which shows a higher density towards the higher values on the graph. This demonstrates that the abnormal data is reporting higher levels of vibration in the reactor.

## Section 2: Discussion on selecting an algorithm (30%)

### Case Study

When discussing the performance of a classification model there are three main points to consider. These are the accuracy, sensitivity and specificity. In this case study the intern decided to try 10 different models which used 70% of the data as training data, 20% as validation data and 10% as test data and only looked at the accuracy's of each model.

To find the best model we must first discuss the difference between accuracy, sensitivity and specificity. Accuracy is the overall measure of how often the classification is done correctly by the model using the test data. A higher score means that the model classified the data correctly, a higher percentage of the time and is arguably the most important measure of how good a model is. Sensitivity is the models ability to identify true positive data, in this case, how good the model is at identifying the abnormal data. Specificity is the opposite and measure the models ability to identify the true negative rate which is the normal data.

When deciding on the best model to use for any specific case all three of these measures should be discussed because a high sensitivity with a low specificity would result in a higher chance of the model giving false positive classification, and a high specificity and low sensitivity would produce classifications with many abnormal reading being missed. This is particularly important for something like a nuclear reactor because if an abnormal pressure reading is missed is could cause catastrophic problems. That is why even if the accuracy of the model is high, the other measures should also be taken into account to get the best model for this particular use case.

Furthermore, the intern used 10% of the data as test data and 70% as training. This is not the ideal method of training and testing a classification model. A better method would be to use something like CV-Cross validation. This method will run the model many times using a different group of the dataset

as the test data each time and gets an accuracy for each. Once this is done a mean of the accuracies can be returned to get a better look at the how the good the model really is.

## Section 3: Designing Algorithms (30%)

### Step 1 – Training and Test Data

The first step for creating the artificial neural net (ANN) is to split the dataset into training and test data. For this task I was asked to make this a 90%-10% split.

To begin I created two new data frames. One will contain the training data and the other will contain the test data. I used frac = 0.90 to load 90% of the data from the original dataset into the training set. This also allowed me to automatically shuffle the data so a random 90% is used.

Next I assigned the test set the original data and used drop to remove the data in the training set. This left me with a test set only containing the data not in the training set.

Following this I needed to further split the data. The task asks us to classify the pressurized water so I only need to use the pressure sensor features for the ANN. This was done by creating an X variable which contains every row from the 4 pressure sensor features and a Y variable that stores the status.

I renamed these as trainX, trainY, testX and testY to make is clear what each variable is for.

```
data_summary(df)

"""Task 3"""

df_copy = df.copy()


df_train = df_copy.sample(frac = 0.90)
#train dataset gets 90% of the data.

df_test = df_copy.drop(df_train.index)
# test dataset will get the remaining 30%.
print("train", df_train.shape)
print("test", df_test.shape)


print('test', df.keys())
df.iloc[:, [0,9]]
x = df_train.iloc[:, [5,6,7,8]]
y = df_train['Status']

x2 = df_test.iloc[:, [5,6,7,8]]
y2 = df_test['Status']
#print("Y: ", x)

trainX = x
trainY = y
testX = x2
testY = y2
```

## Step 2 – Training the ANN

After creating the training and testing data I moved on to building the artificial neural net. To do this I used the scikit-learn python libraries and utilised the following functions from that library.

```
20 import sklearn
21 from sklearn.preprocessing import StandardScaler
22 from sklearn.neural_network import MLPClassifier
23 from sklearn.ensemble import RandomForestClassifier
24 from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
25 from sklearn.model_selection import KFold
26 from sklearn.model_selection import ShuffleSplit
```

Next I designed a new function called “ANN”. This function accepts 6 variables, these are the training and test data, as well as, the number of neurons and number of trees for the random forest classifier.

```
def ANN(df, trainX, trainY, testX, testY, no_neurons, no_trees):
    scaler = StandardScaler()
    scaler.fit(trainX)

    trainX = scaler.transform(trainX)
    testX = scaler.transform(testX)

    mlp = MLPClassifier(hidden_layer_sizes=(no_neurons,no_neurons))
    ANN = mlp.fit(trainX, trainY)
    print("mlp", ANN)
```

The data needs to be normalised. This means that the values in the dataset need to be measured on the same scale. In this case they need to be the same scale as the trainX data. I used the StandardScaler function from the sklearn python library for this.

After the data is normalised I created an MLP Classifier. This stands for Multi-Layer Perceptron Classifier. This again uses the sklearn python library. The MLPClassifier takes in the number of hidden layers and the number of neurons in each layer.

In a neural net the hidden layers are in between the input and output layers and perform computations of the weighted inputs to create something that the output layer can use. For example, the hidden layer will create a detector that will look for a specific feature in the data input to determine what it is so it can be classified. Many hidden layers can work together on the data to detect different features of the data and together can improve the accuracy of the neural net.

Each hidden layer is made up of a number of neurons. These neurons typically compute the weighted average of the input data. This is then passed through an activation function such as a sigmoid function. If it is activated it has classified the data as one of the outputs it is looking for. This allows a neural net to classify more complex systems by combining many small inputs that can make up the system.

Once the ANN has been created it needs to be fit using the training data. In this case trainX contains the pressure features data and trainY contains the status of the data (normal or abnormal).

After fitting the data we get the following output from the system. This shows us the values of each parameter in the model.

```
mlp MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(500, 500), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
```

## Step 3 – Testing the ANN

To test the ANN and get an accuracy score I first needed to get the predictions for the test data. To do this I used the .predict function and passed the testX data.

Using this I created a confusion matrix of the predictions and the actual test data to get the following output. I also created a classification report that contain the accuracy of the ANN.

```
predictions = ANN.predict(testX)
print("Confusion Matrix: ")
print(confusion_matrix(testY,predictions))
print(" ")
print("Classification Report: ")
print(classification_report(testY,predictions))
```

```
Classification Report:
             precision    recall  f1-score   support

   Abnormal       0.77      0.82      0.80        57
     Normal       0.74      0.67      0.71        43

avg / total       0.76      0.76      0.76       100
```

```
Confusion Matrix:
[[47 10]
 [14 29]]
```

My artificial neural net got an accuracy of 0.80 for classifying abnormal data and 0.71 for classifying normal data and had an overall accuracy of 0.76.

## Step 4 – Training Random Forest Classifiers

A random forest classifier (also known as a random decision forest) is made up of a large number of trees which themselves are made up of leaf nodes. Each tree in the forest provides a classification prediction and the class with the most predictions becomes the one the models outputs. Its success comes from the idea that a large number of unrelated models (in this case trees) will outperform an individual model.

To train the Random Forest Classifier I used the sklearn python library. Using this I passed the function the number of trees that I want to use and the minimum number of samples needed for a leaf node. I then train the classifier using .fit and pass the training data.

```
"""Task 3.5 Random Forests"""

Classifier = RandomForestClassifier(n_estimators=no_trees, random_state=0, min_samples_leaf = 5)
Classifier.fit(trainX, trainY)
```

The data used for this has already been normalised and scaled from the artificial neural net so the data is ready to be used in the random forest classifier.

## Step 5 – Testing the Random Forest Classifier

To test the random forest classifier, I used the same method as for the artificial neural net. This meant creating a prediction variable that stored the classifiers predictions of the test data. I then created a confusion matrix and a classification report to find the accuracy of the random forest classifier.

```
y_pred = Classifier.predict(testX)
#print(y_pred)
print("Confusion Matrix Random Forest: ")
print(confusion_matrix(testY, y_pred))
print(" ")
print("Classification Report Random Forest: ")
print(classification_report(testY, y_pred))
print(accuracy_score(testY, y_pred))
ForestAccuracy = accuracy_score(testY, y_pred)
```

```
Confusion Matrix Random Forest:
[[36  6]
 [15 43]]

Classification Report Random Forest:
             precision    recall  f1-score   support

   Abnormal       0.71      0.86      0.77        42
     Normal       0.88      0.74      0.80        58

avg / total       0.81      0.79      0.79       100

0.79
Ann Accuracy:  0.68
Forest Accuracy:  0.79
```

The output for both of these are shown in the screenshot to the left.

With 1000 trees the accuracy of the classifier is 0.79.

# Section 4: Model Selection (20%)

## Cross-validation

For this task I will perform 10 Fold Cross-validation (CV) to compare different parameters of both the artificial neural net and the random forest classifier. KFold cross validation is the process of splitting the dataset into a set number of folds (in this case 10) and then run the neural net 10 times, using a different fold as the test set each time. From this I can get the mean accuracy of the classifier and get a more accurate picture of how well the model is performing.

The first step is to determine the amounts of folds based on the size of the dataset. For this I used the KFold function shown below.

```
X = df_copy_2.iloc[:, [1,2,3,4,5,6,7,8,9,10,11,12]]
y = df_copy_2.iloc[:, [0]]


kf = KFold(n_splits=10,shuffle=True, random_state = 1)
kf.get_n_splits(X)
```

For this, X holds the data from each feature and y holds the label 'Status'. When the data is split it will be shuffled in order to get a mix of normal and abnormal data in the training sets.

I then used a 'for' loop to split the data X number of times (10) and create new training and test data.

```
for train_index, test_index in kf.split(X):
    trainX, testX = X.iloc[train_index], X.iloc[test_index]
    trainY, testY = y.iloc[train_index], y.iloc[test_index]
    trainY=trainY.squeeze()
    testY=testY.squeeze()
```

Using these training and test sets I can call the ANN function I made in the previous task again each time the 'for' loops runs through.

Each time the 'for' loops goes through with a new set of training and test data, the accuracy of each set will be found using the ANN functions. Set 1 is for 50 neurons and 20 trees, set 2 is 500 neurons and 500 tree and set 3 is for 1000 neurons and 10000 trees. The scores for each of these is added to a list to store all of the results.

```
ANNAccuracySet1, ForestAccuracySet1 = ANN(df_copy_2, trainX, trainY, testX, testY, 50, 20, epochs)
AccuracyANNSet1.append(ANNAccuracySet1)
AccuracyRFCSet1.append(ForestAccuracySet1)

ANNAccuracySet2, ForestAccuracySet2 = ANN(df_copy_2, trainX, trainY, testX, testY, 500, 500, epochs)
AccuracyANNSet2.append(ANNAccuracySet2)
AccuracyRFCSet2.append(ForestAccuracySet2)

ANNAccuracySet3, ForestAccuracySet3 = ANN(df_copy_2, trainX, trainY, testX, testY, 1000, 10000, epochs)
AccuracyANNSet3.append(ANNAccuracySet3)
AccuracyRFCSet3.append(ForestAccuracySet3)
```

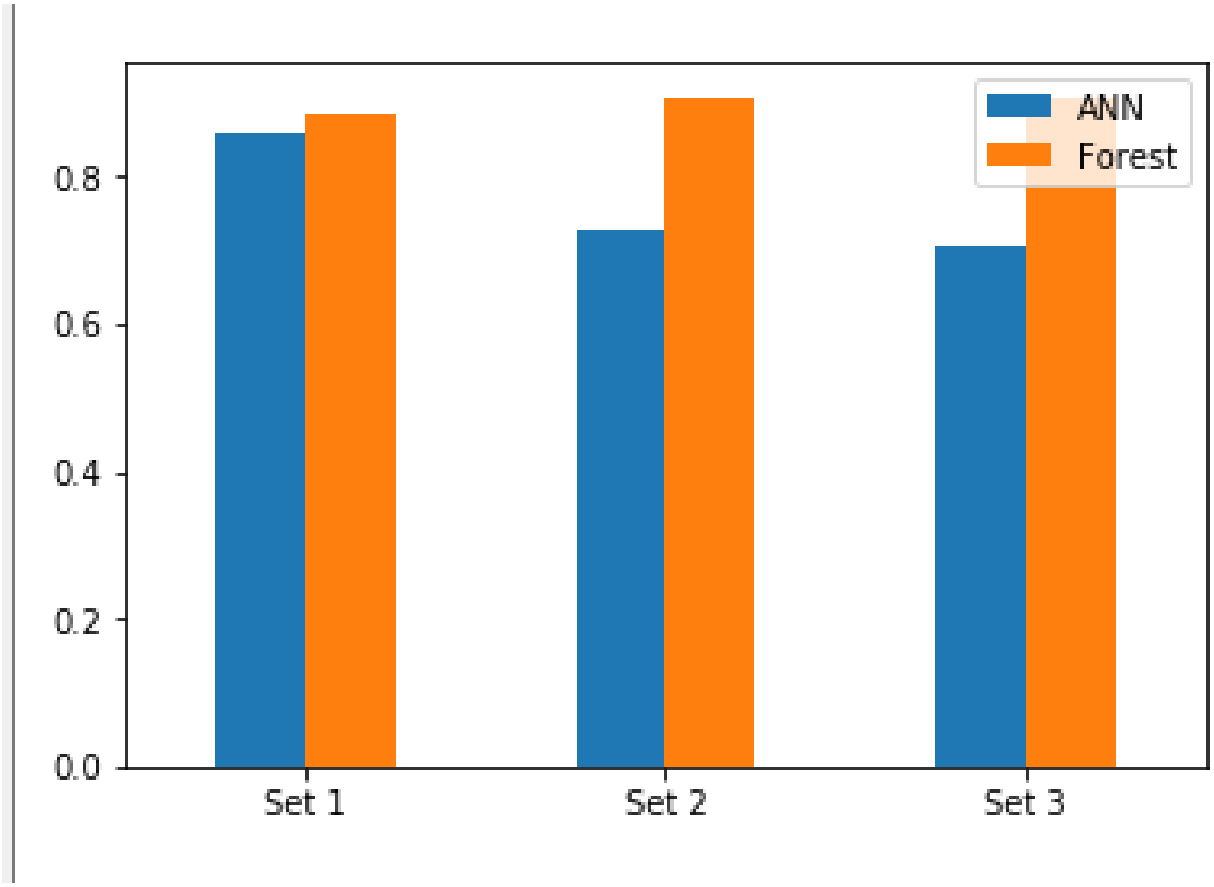
After the loop has run its course I will have 60 results. I use the following code to get the mean results from each set.

The outputs for this are shown below.

```
ANN Mean Set1 0.856494949495
Forest Mean Set1 0.885565656566
ANN Mean Set2 0.72795959596
Forest Mean Set2 0.906656565657
ANN Mean Set3 0.704878787879
Forest Mean Set3 0.907646464646
```

```
annMeanSet1 = np.mean(AccuracyANNSet1)
forestMeanSet1 = np.mean(AccuracyRFCSet1)
print(annMeanSet1)
print("ANN Mean Set1", annMeanSet1)
print("Forest Mean Set1", forestMeanSet1)

annMeanSet2 = np.mean(AccuracyANNSet2)
forestMeanSet2 = np.mean(AccuracyRFCSet2)

print("ANN Mean Set2", annMeanSet2)
print("Forest Mean Set2", forestMeanSet2)

annMeanSet3 = np.mean(AccuracyANNSet3)
forestMeanSet3 = np.mean(AccuracyRFCSet3)

print("ANN Mean Set3", annMeanSet3)
print("Forest Mean Set3", forestMeanSet3)
```

Finally, I decided to create a bar graph to help visualise the results.



## Evaluation

Based on the results in the bar graph it seems that the ideal parameters for the artificial network are 50 neurons on two layers. This has an average accuracy of 0.85. My results here indicate that as the number of neurons in the hidden layers the accuracy actually decreases, this is likely due to overfitting and reduce the neural networks effectiveness for this data set. For the random forest classifier, the accuracy increases as the number of trees is increased with a significant increase coming when increasing it to 500 trees. Changing the amount of trees again to 10000 improves the classifier but only by a marginal amount.

Across all models with different combination of parameters, my most accurate model was the random forest classifier with 10,000 trees. However, this is based purely of the accuracy score. If we take into account the amount of time it takes for the model to be created, fit and teste then 500 trees seems like the ideal candidate as the accuracy was also very high and it ran significantly faster than other models. This may be important in the scenario of a nuclear reactor if the values change rapidly then model needs to be able to update and reclassify the data quickly so the faster model may be more suitable.

It is also important to take into account the sensitivity and specificity when discussing which model is best for this data set. The ANN had more unpredictable sensitivity and specificity score, compared to the random forest classifier which has much higher sensitivity which means that is it more accurate when classifying abnormal data. This is good for this data set as it is a nuclear reactor so any abnormal data needs to be found to stop any malfunctions/problems within the reactor.

Overall, I would choose the 1000 Random Forest Classifier because it provides a very high accuracy of 0.907 with a relatively high sensitivity compared to the other models. It also runs much faster than the ANN with 1000 neurons on two hidden layers. In my opinion, these factors make it the best model for this data set specifically but not for all data sets.

# References

KDnuggets. (2019). *A Beginner's Guide to Neural Networks with Python and SciKit Learn 0.18! - KDnuggets*. [online] Available at: https://www.kdnuggets.com/2016/10/beginners-guide-neural-networks-python-scikit-learn.html [Accessed 16 Dec. 2019].

Medium. (2019). *Understanding Random Forest*. [online] Available at: https://towardsdatascience.com/understanding-random-forest-58381e0602d2 [Accessed 16 Dec. 2019].

MIT News. (2019). *Explained: Neural networks*. [online] Available at: http://news.mit.edu/2017/explained-neural-networks-deep-learning-0414 [Accessed 16 Dec. 2019].

Pathmind. (2019). *A Beginner's Guide to Neural Networks and Deep Learning*. [online] Available at: https://pathmind.com/wiki/neural-network [Accessed 16 Dec. 2019].

work?, W. and Idden, H. (2019). *What are neurons in neural networks / how do they work?*. [online] Cross Validated. Available at: https://stats.stackexchange.com/questions/241888/what-are-neurons-in-neural-networks-how-do-they-work/241904 [Accessed 16 Dec. 2019].