|  |  |
| --- | --- |
|  | |
| Python Machine Learning Project  Tools for Data Analytics CA TWO – Report | |
|  | |
| Ciaran Finnegan / Radoslav Durina / Dermot Madsen  Student No : 10524150 / 10524378 / 10522567  11/10/2019 |  |
|  |  |

List of contents

1 Project Overview 3

1.1 High Level Description 3

1.2 Environment Assumptions 3

1.3 Project Execution Instructions 3

2 Machine Learning Process Flow Description 4

2.1 High Level Description of Machine Learning Workflow 4

3 Data Pre-Processing and Preparation 6

3.1 Python Libraries 6

3.2 What is data pre-processing? Why do we do it? 6

3.3 Load and Visualise Data 6

3.4 Types of Data and performing pre-processing 7

4 Model Evaluation Strategy 9

4.1 Benefit of Random Forest 9

4.2 Objective of Model evaluation 9

4.3 Justification for approach to analysis 11

5 Model Building and Testing 12

5.1 Dividing the original dataset into train/validation/test sets 12

5.2 Tuning the RandomForest model for the Spruce dataset 13

5.3 Determine the subset of ‘significant features’ 16

6 Identifying The Best Model 18

6.1 Run RandomForest using ‘significant’ dataset 18

6.2 Final Classification Model Characteristics 18

7 Guidelines Generated by Prediction Model 23

7.1 Focus on a smaller Feature set of data 23

7.2 Check the ‘Missing Data’ 23

8 Principal Component Analysis (PCA) 24

8.1 Code and Data Preparation 24

8.2 PCA Implementation 25

9 K-Means Clustering 26

9.1 Elbow Plot Creation 26

9.2 K-Means Implementation 27

10 Summary : How we used Python to Analyse the Spruce Dataset 29

# Project Overview

## High Level Description

This document covers the design, implementation and observations on all parts of CA Two for the Tools for Data Analytics module.

Although it is primarily a group report, containing the collaborative design/coding decisions and general observations, there are also sub sections for the individual reflections of each team member; Dermot, Radoslav, and Ciaran. (Submitted in separate documents).

## Environment Assumptions

The IDEs used for development were a combination of Microsoft Visual Studio Community 2019, PyCharm Community 2016.3, and Jupyter Notebook (Anaconda 3). The code is submitted in one Python ***Surname\_surname\_Surname.py*** file (*Madsen\_Durina\_Finnegan.py*), along with the source ‘Spruce’ csv file, and this report document (in /pdf format).

.

There were very minor code snippets added to the Python code base to improve graph presentations, depending on which IDE the individual contributors were using to develop and test the CA Two code and analysis.

## Project Execution Instructions

The procedure for the project solution is executed from the ‘*Madsen\_Durina\_Finnegan.py*’ file. For Visual Studio;

1. Add the *.py* file into a VS Project, include the *Spruce.csv* file in the same project folder.
2. Right click in the Solution Explorer list
3. Choose the option to ‘Set as Startup File’.
4. Run the file

The problem solution files follow the following structure;

* The main function is named with the format: *MainProg\_CATwo()*.
* The main program contains all the high level function calls to invoke the Machine Learning workflow steps. The code for all functions is included in the ‘*Madsen\_Durina\_Finnegan.py*’*.py*’ file.

# Machine Learning Process Flow Description

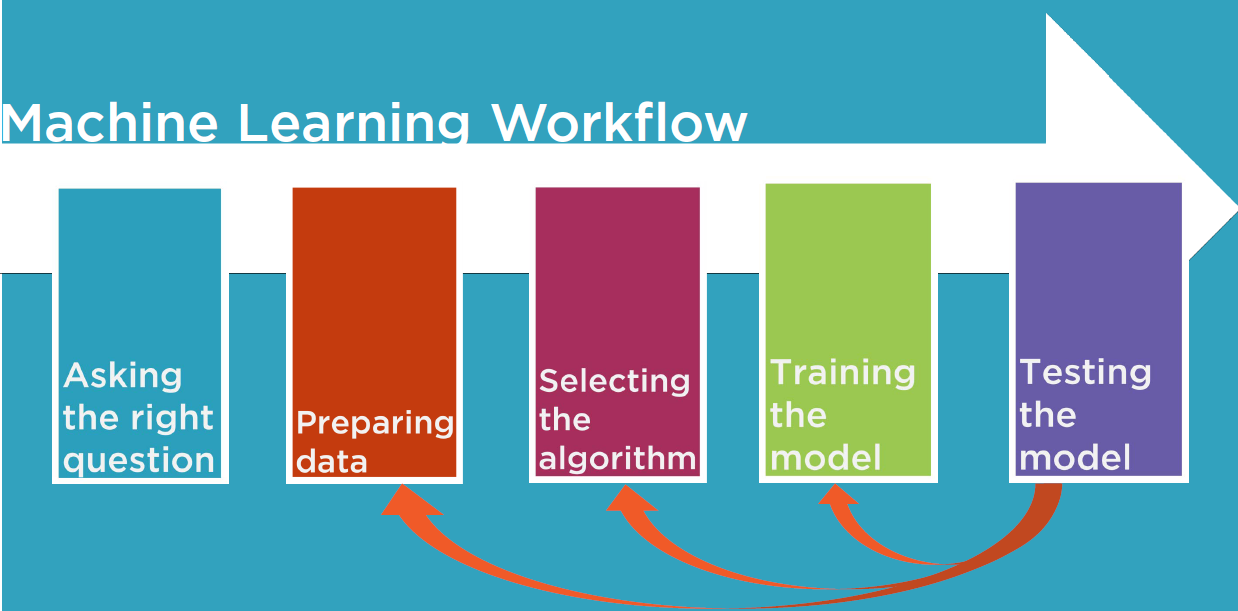
## High Level Description of Machine Learning Workflow

### Process Flow Diagram for Machine Learning

This continuous assessment is an exercise in the ability to analyse datasets using Python.

Research shows that this process, whether using Python or other tools, tends to follow orchestrated and repeatable patterns. This leads the analyst to systemtatically transform and process information to create prediction solutions.

A representation of this process workflow can be given as..

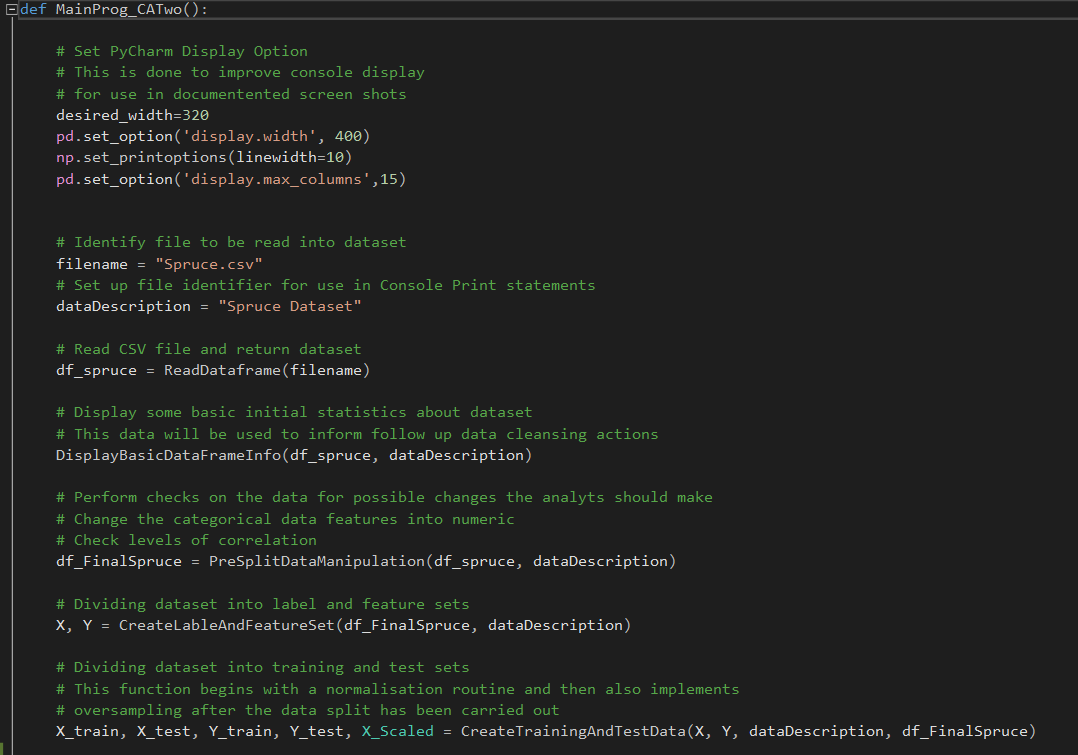


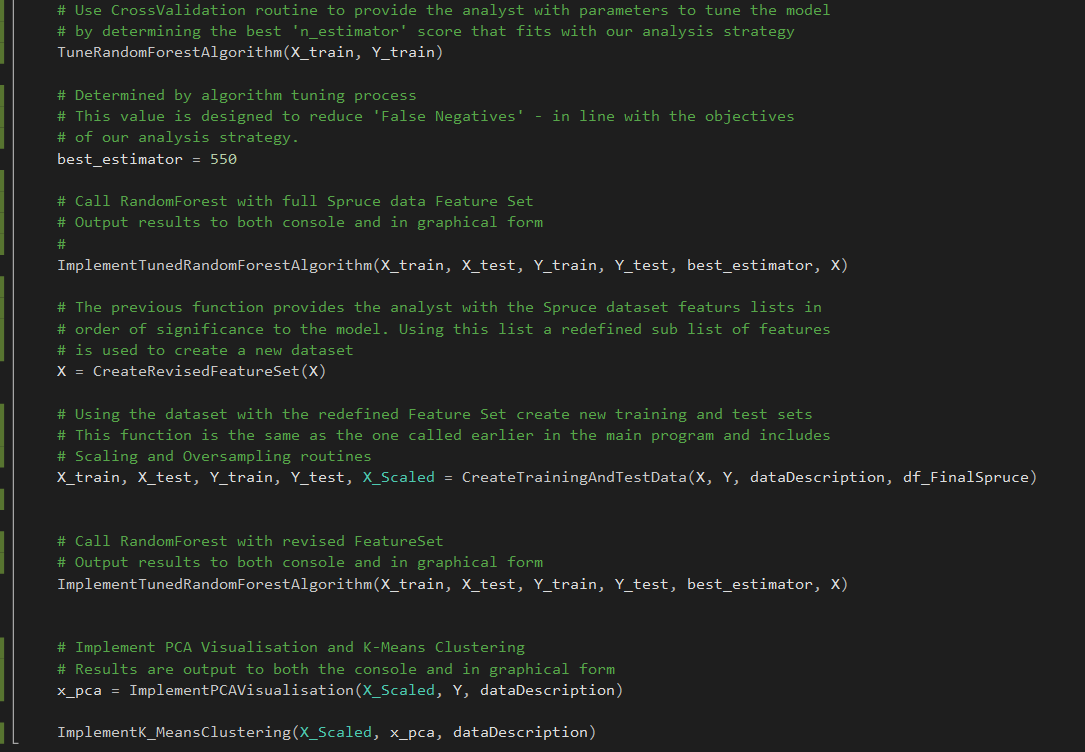
*\*credit to Jerry Kurata - Pluralsight*

* The Right Question : For the Spruce dataset analysis we were given the question - how to predict locations to plan Spruce trees for the Canadian Forest Department. We needed to supplement this by determining if we should focus on accuracy, or reducing certain types of false predictions for planting Spruce trees.
* Preparing the Data : What routines do we need to code in Python to manipulate the Spruce data? We need to make the data work effectively while constructing a model using the RandomForest algorithm in Python.
* Selecting the Algorithm : The RandomForest approach is a good fit for this problem. We expand on this statement with more detail later in this report.
* Training the Model : Python provides many libraries to split and train the model using the Spruce data. This is elaborated on in the report.
* Testing the Model : Python routines execute tests on the trained model and represent out both in the console and graphically.

The output from various Python functions we wrote helped inform decisions to refine parameters for Python functions and reiterate through the training the testing process.

### High Level Program Flow Sequence (code snippet from main function).



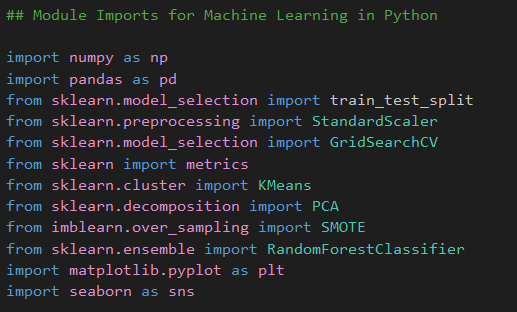


# Data Pre-Processing and Preparation

## Python Libraries

There are a number of libraries which can be imported into a Python application to provide access to pre-built Machine Learning functions.

The libraries imported into our CA applications are:



## What is data pre-processing? Why do we do it?

Data pre-processing is a process of cleaning the ‘raw’ data, that is the data is collected in the real world, and converting to a ‘clean’ data set.

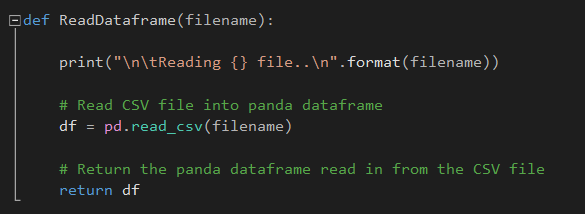
The Spruce data in the ***Spruce.csv*** file is reasonably well formed but still requires certain steps to be executed to convert the data into a small clean data set.

Data pre-processing is required to achieve good results from the training and application of the subsequent model in our Spruce machine learning project.

## 

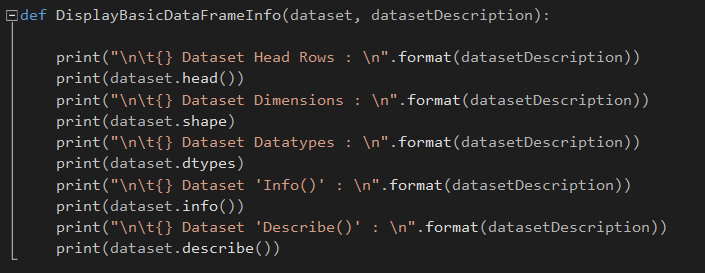
## Load and Visualise Data

The Python code used the Panda library to read the Spuce data into a Panda dataframe.

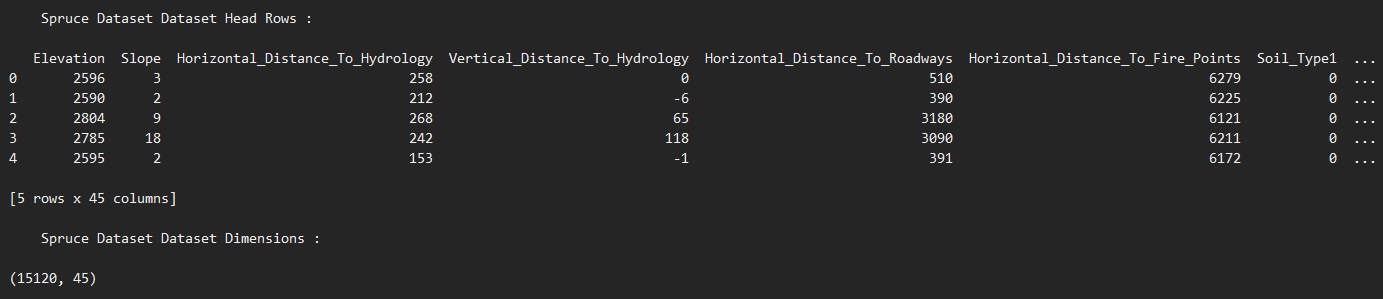


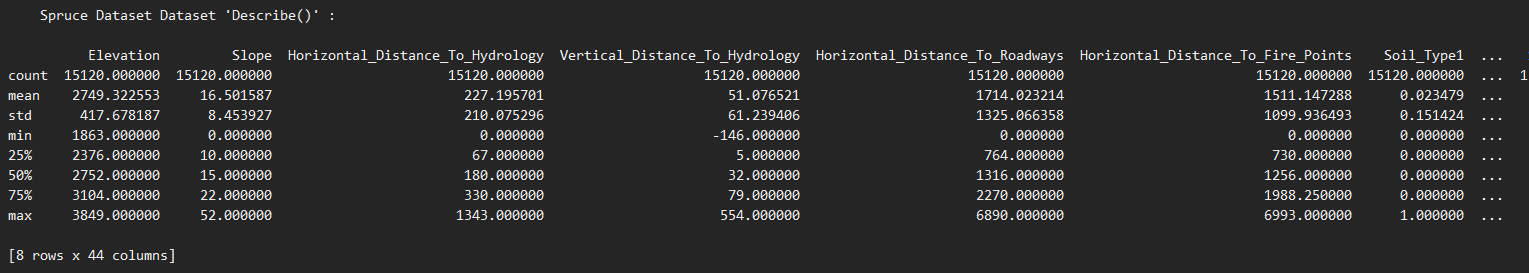
Once the data is in a dataframe format we can start to display and manipulate the data in our Spuce tree Python project.

We wrote the following function in Python to display basic information on the initial Panda dataset created from the direct read of the ***Spruce.csv*** file.



A partial view of the Visual Studio console output looks like this…





## Types of Data and performing pre-processing

Once our Python application loaded data from the CSV file into the Panda dataframe, we were able to proceed with further analysis to determine what ‘clean up’ wa required.

### Three Types of Data

In general terms ‘real world’ data types we would be concerned with are:

1. Numeric e.g. income, age

2. Categorical e.g. gender, nationality

3. Ordinal e.g. low/medium/high

In the ***Spruce.csv*** file we only need to contend with numerical and categorical date.

### How can data pre-processing be performed?

These are some of the basic pre — processing techniques that can be used to convert raw data.

1. **Conversion of data:** As we know that Machine Learning models can only handle numeric features, hence categorical and ordinal data must be somehow converted into numeric features. For the ***Spruce.csv*** file, we only have one categorical colum to consider : ‘Tree\_Type’.

2. **Ignoring the missing values:** Whenever we encounter missing data in the data set then we can remove the row or column of data depending on our need. This method is known to be efficient but it shouldn’t be performed if there are a lot of missing values in the dataset. There are no NULL values in the ***Spruce.csv*** file. There are a number of zero values that could potentially be evaluated as missing but we will choose to not take any actions for this data.

3. **Filling the missing values:** Whenever we encounter missing data in the data set then we can fill the missing data manually, most commonly the mean, median or highest frequency value is used. This could be an option for the ‘zero’ values in the Spruce dataset but we choose not to take any actions in this project.

4.**Machine learning:** If we have some missing data then we can predict what data shall be present at the empty position by using the existing data. As above, we take no specific actions to address the potentially ‘missing’ values – the zero entires.

5. **Outliers detection:** There are some error data that might be present in our data set that deviates drastically from other observations in a data set. [Example: human weight = 800 Kg; due to mistyping of extra 0]. We did not check for outliers in the Spruce dataset.

### Python data processing code used on Spruce dataset

We coded the following analysis to check the Spruce data for clean up actions:

Null Data

< Add text to above for Spruce.csv>

# Model Evaluation Strategy

## Benefit of Random Forest

The dataset ***Spruce.csv*** contains cartographic data for observations made over different 30m × 30m patches in the forests of Alberta, Canada. This dataset has 15,120 observations, with 44 input variables (cartographic variables) and 1 target variable (Tree\_Type).

This Spruce dataset allows for Supervised Learning and is a ‘classification’ problem because we are required to advise Canada’s Forest Department whether they should plant Spruce or another tree type in a given 30m x 30m plot.

In Machine Learning, Random Forest is a commonly used and versatile algorithm to generate a model for this type of problem.

Given the characteristic (list of features) for a new forest plot we are aiming to generate a prediction on whether Spruce trees should be planted, based on previous real world data.

## Objective of Model evaluation

Section 5 of this document describes the process of ‘training’ a model to predict results.

Once the model is trained we can use the same trained model to predict using the testing data i.e. the unseen data. Once this is done we can develop a confusion matrix, this tells us how well our model is trained. A confusion matrix has 4 parameters, which are ‘**True positives’**,**‘True Negatives’**,**‘False Positives’**and ‘**False Negative’**. We prefer that we get more values in the True negatives and true positives to get a more accurate model. The size of the Confusion matrix completely depends upon the number of classes.

* **True positives:** These are cases in which we predicted TRUE and our predicted output is correct.
* **True negatives:** We predicted FALSE and our predicted output is correct.
* **False positives:** We predicted TRUE, but the actual predicted output is FALSE.
* **False negatives:** We predicted FALSE, but the actual predicted output is TRUE.

We can also find out the accuracy of the model using the confusion matrix.

*Accuracy = (True Positives +True Negatives) / (Total number of classes)*

The objectives to consider are:

* Should we look for the highest possible classification accuracy?

*Give a reason why we would want to focus on this metric…*

* Should we look to minimise ‘false positives’?

*Give a reason why we would want to focus on this metric…*

* Should we look to minimise ‘false negatives’?

*Give a reason why we would want to focus on this metric…*

.

## Justification for approach to analysis

<…text…>

# Model Building and Testing

## Dividing the original dataset into train/validation/test sets

<text..make relevant to Spruce>

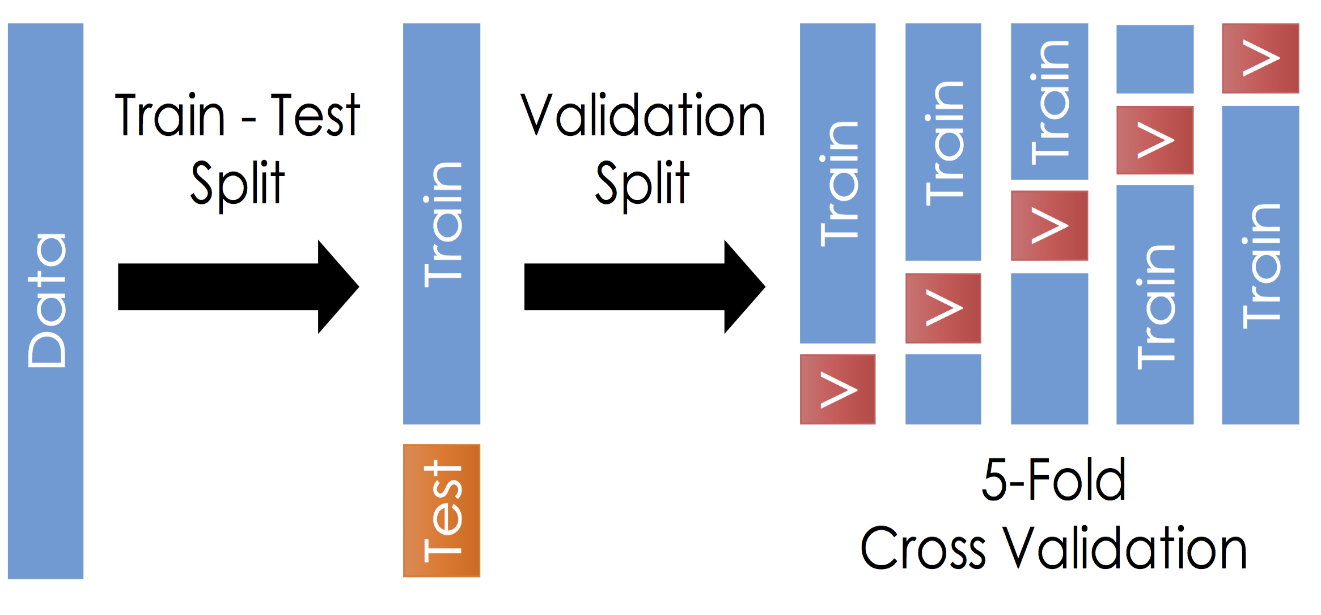
For training a model we initially split the model into 3 three sections which are ‘**Training data**’ ,‘**Validation data**’ and ‘**Testing data**’.

You train the classifier using ‘**training data set**’, tune the parameters using ‘**validation set**’ and then test the performance of your classifier on unseen ‘**test data set**’. An important point to note is that during training the classifier only the training and/or validation set is available. The test data set must not be used during training the classifier. The test set will only be available during testing the classifier.

**Training set:** The training set is the material through which the computer learns how to process information. Machine learning uses algorithms to perform the training part. A set of data used for learning that is to fit the parameters of the classifier.

**Validation set:** Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. A set of unseen data is used from the training data to tune the parameters of a classifier.

**Test set:** A set of unseen data used only to assess the performance of a fully-specified classifier.



Once the data is divided into the 3 given segments we can start the training process.

In a data set, a training set is implemented to build up a model, while a test (or validation) set is to validate the model built. Data points in the training set are excluded from the test (validation) set. Usually, a data set is divided into a training set, a validation set (some people use ‘test set’ instead) in each iteration, or divided into a training set, a validation set and a test set in each iteration.

## Tuning the RandomForest model for the Spruce dataset

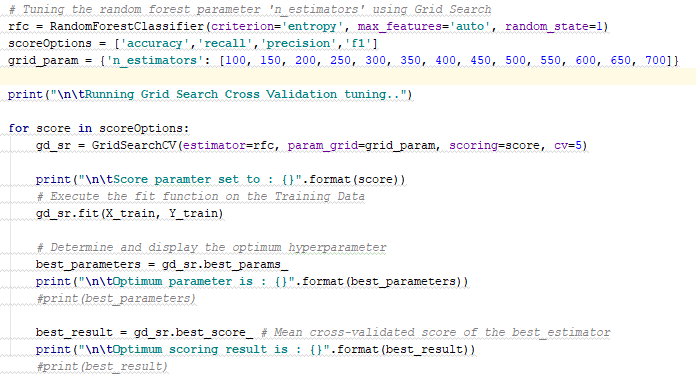
In our Python code – ***TuneRandomForestAlgorithm(X\_train, Y-train)*** - a CrossValidation routines was invoked on the training set to implement a 5-fold cross validation exercise.

The code snippet below shows the main body of our function providing a value for a hyperparameter to be applied to the model in order to optimise results.

The FOR LOOP runs through each of the following scoring options:

* Scoring = 'accuracy' when you want to maximize prediction accuracy
* Scoring = 'recall' when you want to minimize false negatives
* Scoring = 'precision' when you want to minimize false positives
* Scoring = 'f1' when you want to balance false positives and false negatives (place equal emphasis on minimizing both)

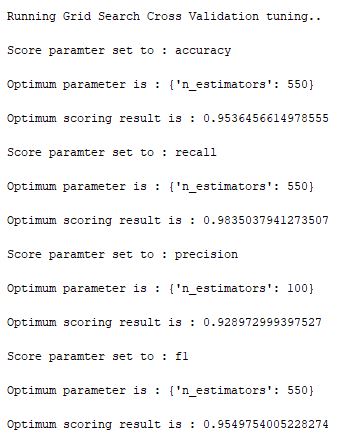
‘Scoring’ is a parameter option used in the *GridSearchCV* function and will return a hyperparameter value optimise the model for a particular objective goal.



The ‘n-estimator’ value list is an incremental list reflecting the staggered list of option for the tree depth for the algorithm.

Each ‘scoring’ parameter is executed against this range of steps to provide and ‘n-estimator’ value that will optimise the model for the scoring objective.

This code took some time to run when executed and produced the following console output:



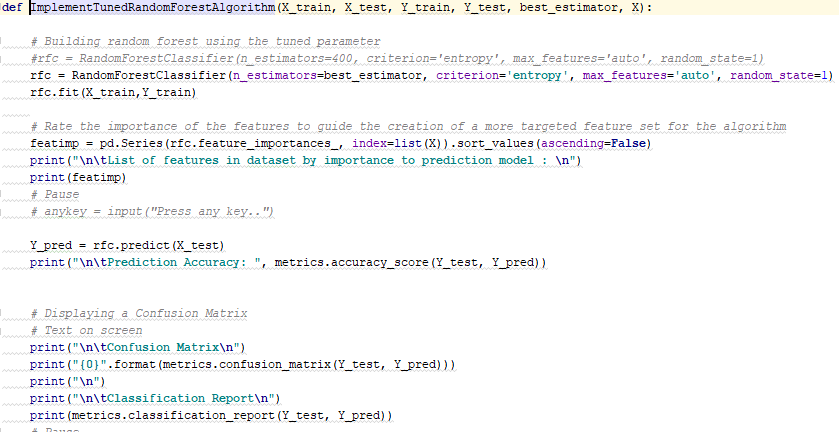
The ‘n-estimator’ value will provide the max tree depth tuned for use in the RandomForest model when it is applied against the Spruce training dataset with a particular scoring objective.

This use and purpose of this ‘hyperparameter; is further described in Section 6.2 of this document.

## Determine the subset of ‘significant features’

### Run the RandomForest Algorithm against Full Spruce Training set

<…add text..alter image as required..>



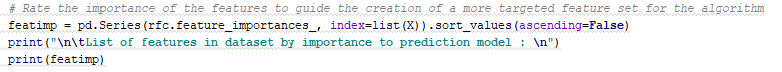
<…add text..>

### Create Revised Featureset

<…text…>

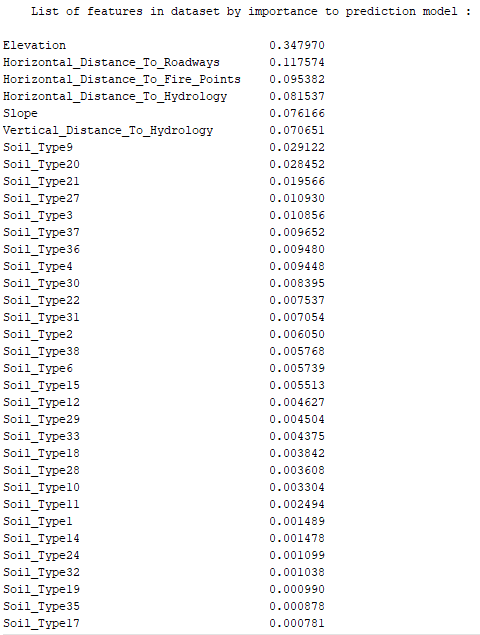
The RandomForest code invokes a function to generate a list of features, which is sorted by the relevance to the production model.

<..code snippet image…update a necessary…>



The screenshot below shows a partial list of features produced in the PyCharm console. It represent a scoring of all features.

The features with most relevance are sorted at the top of the list, which assists the analyst in making an informed decision on which features are most ‘relevant’.



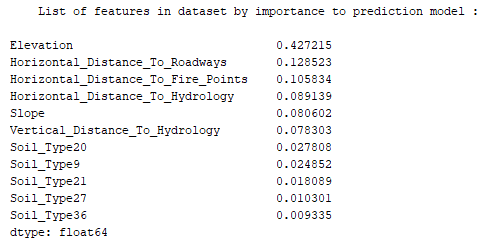
# Identifying The Best Model

## Run RandomForest using ‘significant’ dataset

We made an assumption that the features with a scare less than 0.01 could be ignored for the purposes of building a model.

Therefore our revised data set looks like:

<screenshot..>



This will allow us to run a more performant analysis and we can compare the accuracy of the model predictions against with test data to ensure that no significant accuracy has been lost by using this reduced dataset.

The output of the model when run against the reduced

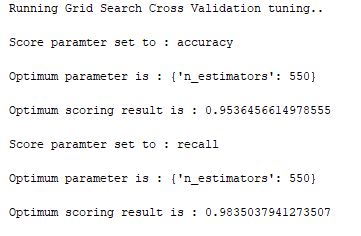
## Final Classification Model Characteristics

### Features and Parameters

<text – relate date to Spruce.csv>

Using the Feature set described above in Section 6.1, we will also run the final model using the n\_estimator’ value as described in the tuning process in Section 5.2 of this document.

In the console window, after the Grid Search Cross Validation tuning was executed, the follow value for the n\_estimator was displayed.



Our objective is to improve overall model accuracy and the reduction of false negatives (‘recall’).

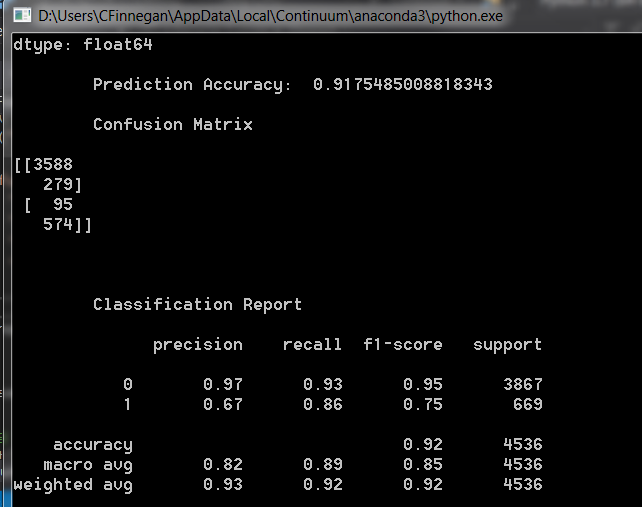
The ‘n\_estimators’ value of 550, as taken from our incremental list of options from 100 – 700 (increasing in increments of 50), will provide the optimum parameter value for both accuracy and recall.

This value represents the optimum tree depth to be used for the RandomForest algorithm. This hyperparameter will improve the accuracy of the model run against the test data (or any ‘real world’ data).

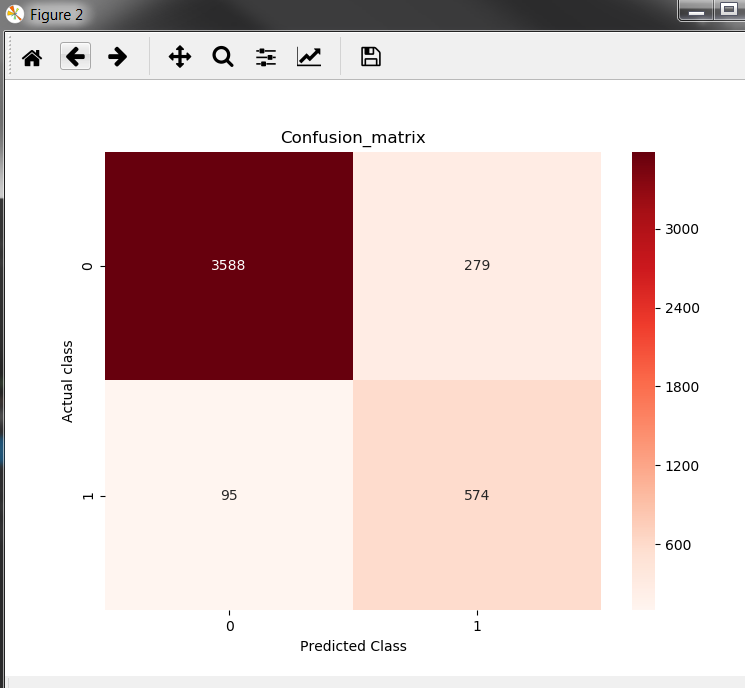
This hyperparameter should prevent ‘overfitting’ of the data by the RandomForest algorithm (the concept will to many ML algorithms). Thus we are attempting to avoid a situation that the model fits the training data very well but is a poor predictor against the test data, and hence other real world examples.

### Performance Results on Test Set

The performance results from the tuned model are output to the PyCharm console as follows;



The Graph results were presented by our program like this:



The Model prediction scores 91.75% against the test data, with an 86% success on minimising false negatives.

The confusion matrix seen in the graph shows the following figures:

**Confusion matrix:**

**[[3588 279]**

**[ 95 574]]**

**TP: 3588**

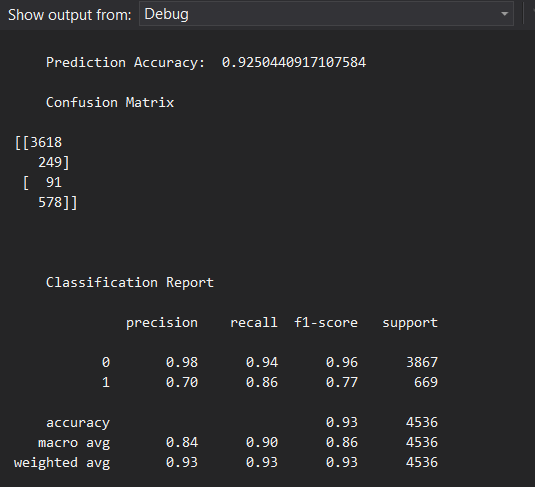
**TN: 574**

**FP: 95**

**FN: 279**

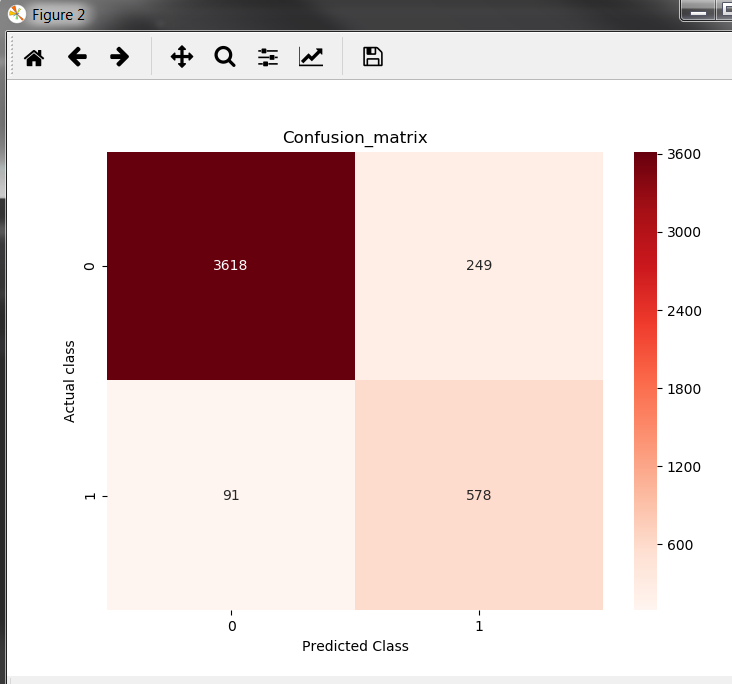
### Comparison Against Full Dataset

An interesting comparison can be made with the model performance when the full data set is used:



Model accuracy is 92.5% but the false negative performance is slighltly worse.

The graph of the Confusion Matrix for this run looks like this:



The model performs marginally better with a full dataset feature list but the difference is not large. The efficiency in using a smaller set of features outweighs the minor drop in efficiency.

# Guidelines Generated by Prediction Model

## Focus on a smaller Feature set of data

<….>

## Check the ‘Missing Data’

<….>

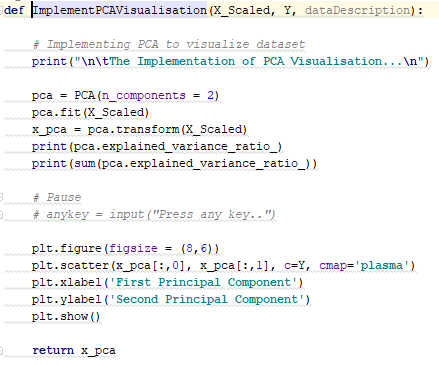
# Principal Component Analysis (PCA)

## Code and Data Preparation

PCA is a technique for dimensionality reduction that is used to visualise datasets with more than 2 or 3 dimensions, which is the case with the Spruce dataset.

The PCA visualisation for the Spruce dataset will attempt to represent on two axis as much variance as possible contained in all the captured features.

<…code snippet…>



In our code set, the PCA function is invoke using the reduced Feature set. This feature set has already been reduced to the 12+ features as described in Section 6.1 of this document.

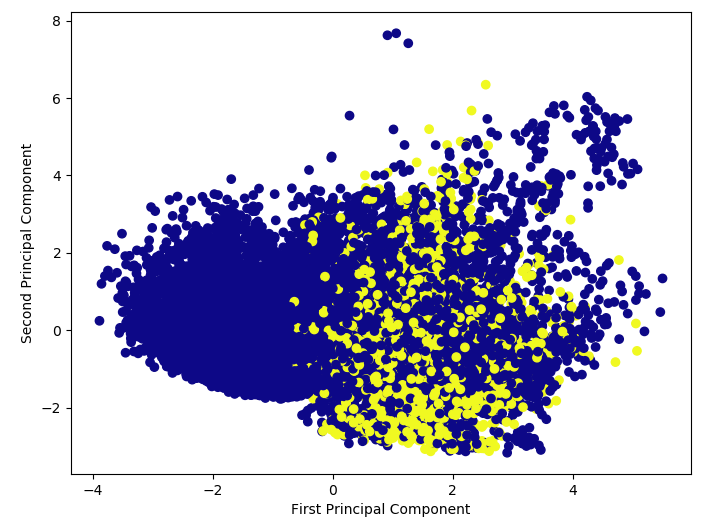
The data has also been normalised so that features with different ranges are adjusted to allow them be compared much more effectively by the Machine Learning algorithms, and by the PCA visualisation. The result is that the dataset based to the PCA Visualisation routine contains feature values that are all mostly between -1 and 1 in magnitude.

This ‘scaled’ data set are the observations of the 44 input variables for the 15, 120 forest plots in the Spruce.csv file, represented by the ‘X\_Scaled’ variable in the code snippet above.

The ‘Y’ variable contains the associated target variable for ‘Spruce/Other’, converted to a numerical value.

## PCA Implementation

Visualisation of the dataset, as output by our code:



<..text..>

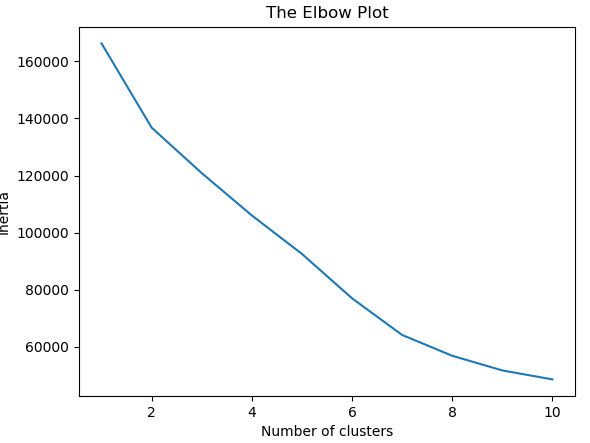
How much variance is explained by the first two principal components? Is the resultant plot a good representation of data?

**Need help here…!!**

# K-Means Clustering

## Elbow Plot Creation

The image below is our output of the Elbow Plot creation from our code analysing the Spruce tree data:



The ‘Inertia’ value on the axis represents the within cluster sun of squared errors (WCSS) for a set of K values.

In the above elbow plot we can see that inertia drops as the number of clusters increases.

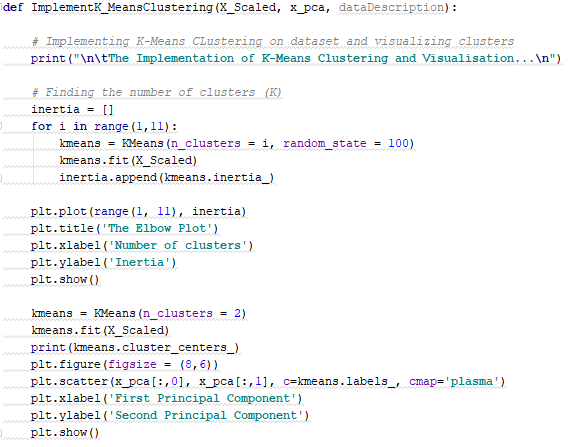
The drop in the value of inertia is less significant after number of clusters = 2.

<text>

## K-Means Implementation

We know the make-up of the Spruce dataset that there can only really be two meaningful clusters – to plant a Spruce tree **or** to plant another type of tree.

The K-Means code, including the Elbow Plot, as implemented by our group, is as follows:

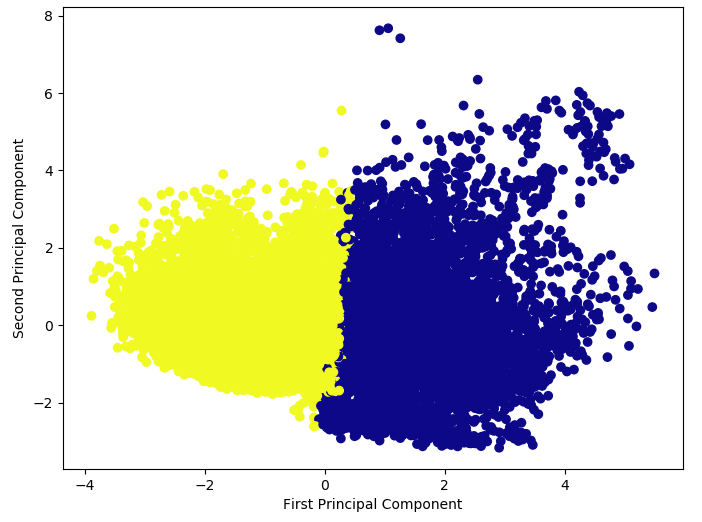


The elbow plot confirms our assumption that ‘2’ is the optimum cluster value. This would have been understood from the nature of the dataset, even without the validation of the elbow plot.

**What about ‘7’? Is the Elbow Plot correct?**

The PCA visualisation matrix is passed to the K-Means function and allows the following cluster graph to be created. The PCA matrix allows a meaningful representation of the clustering of Spruce vs Other Tree types.

Our K-Means visualisation from the code is as follows:



**Need help here…!!**

# Summary : How we used Python to Analyse the Spruce Dataset

* One..
* Two..
* Three..