Title: Relationship Between Features and Accuracy of a machine learning algorithm.

Research Question: To what extent can the features of a dataset alter the prediction accuracy of instance based and decision tree machine learning algorithms.

Contents

Introduction

The path to this extended essay started when I first heard about machine learning from the YouTube channel Siraj Raval[1]. I always believed machine learning would be integrated into future systems to revolutionize the user experience. The recent computer science case study, based on autonomous cars and deep learning, made me want to explore machine learning even further. Thus I chose it as an extended essay topic to give myself the opportunity to explore something that I loved in great depth. Machine learning as defined by Herbert Simon: "Learning is any process by which a system improves performance from experience."[2] This definition can be said to be the entire idea behind machine learning. In other words, machine learning is a technology which consists of a group of algorithms which can "learn" from sets of data and use the "knowledge" it has gained to make a prediction. A Machine learning algorithm "learns" in different ways, which are supervised and unsupervised learning. The algorithms that will be used in the exploration, K-nearest neighbors and Decision trees, are supervised learning based algorithms. When the algorithm learns, it is given a data set and is to make predictions on a certain problem. To do this, the algorithm must generalize the data given to it; hence it must create a mapping function which maps the inputs to the outputs (answers) of the dataset based on the relationship between the inputs and the outputs. This function is then tested on a dataset containing similar fields but different values. The test is to see how well the algorithm does, and this result is given in the form of an accuracy percentage. The mapping function is considered "knowledge," and that knowledge is developed through experience or in this case through multiple test-runs which continuously improve the mapping function. After learning this much, the thing that interested me the most was the mapping function which the algorithms created and how the accuracy score was produced. Thus I took to investigating the datasets which were fed to the algorithms and decided to look at the features of the datasets and how they contributed to the accuracy score. Carrying that thought, I learned that the features played a big role in the accuracy of the algorithm. I then decided to investigate the relationship between the features used and the accuracy score of a machine learning algorithm.

Supervised Learning

To reach the purpose of this experiment a few concept ideas must be covered. The understanding of the algorithms that are to be used in this experiment is very important. The algorithms that are to be used are trained using the supervised learning method. Supervised learning is learning which takes place under guidance, an example of this would be where a student's learning process is supervised by the teacher[3] thus correcting the student if he/she goes wrong. It can then be said that the predictions made by the algorithm on the training set are checked and corrected against the output of the training set. Therefore supervised learning is where both the input variable and the corresponding output variable must be included in the dataset. When the model can make accurate predictions on unseen data after the learning process,

[1] "Siraj Raval." *YouTube*, YouTube, www.youtube.com/channel/UCWN3xxRkmTPmbKwht9FuE5A.
[2] Eaton, Eric. *Introduction to Machine Learning*. N.p.: Www.seas.upenn.edu/~cis519, n.d. PDF.
[3] "Supervised and Unsupervised Machine Learning Algorithms." Machine Learning Mastery. N.p., 22 Sept. 2016. Web. 10 Oct. 2017.

we can say that the model can generalize from the training data to the test data. Generalization is the creation of an accurate mapping function between the input values and the output values. Thus the goal is to build a model that can generalize well.

Instance-Based Algorithms

Instance-based learning is a group of machine learning algorithms which compares new instances with the training instances used, which are entirely stored in memory.[4] This feature allows it to avoid generalization. This is the same as a child applying its previously memorized knowledge of shapes of puzzle pieces to an entirely new puzzle and being able to complete it. Instance-based learning creates a general assumption from the training data; therefore the complexity of the assumption is directly proportional to the data set size. Thus if we were to use a dataset that has a large number of features or attributes, then the complexity of the model would also be large.[5]

K-Nearest Neighbors

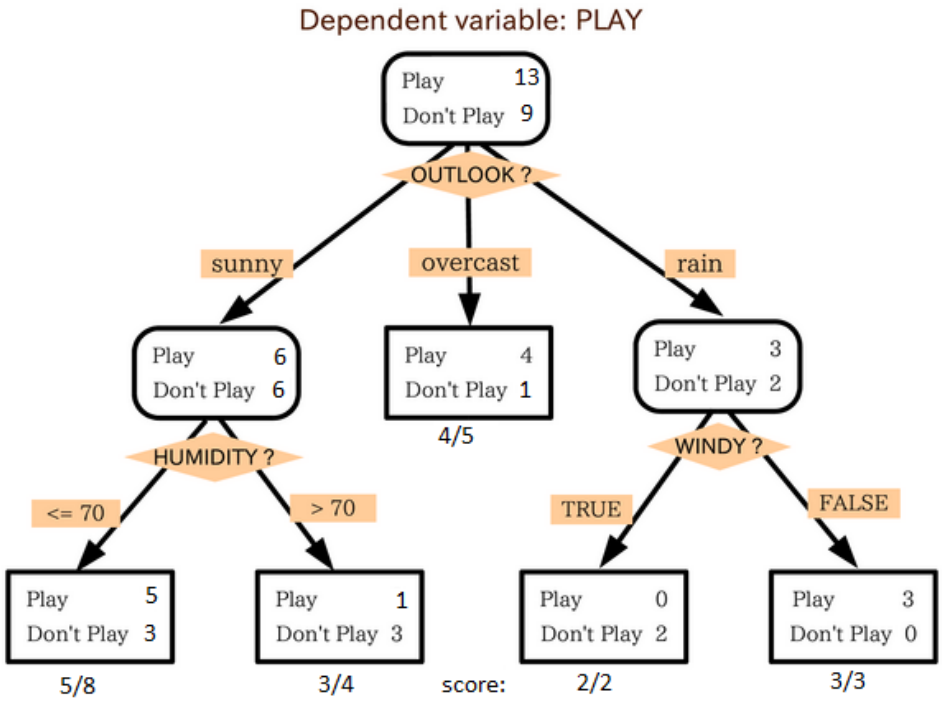
K-Nearest Neighbors is a machine learning algorithm which is a subset of the Instance-based algorithms. KNN is an algorithm that uses the entire training data set as the model by storing the entire data set; this, therefore, requires no form of generalization of the data. KNN can be used for a classification problem or a regression problem.[6]The question that arises now is that how does it make predictions? Say test data point (n) is taken; the algorithm will search through the entire stored data set for the closest matching instances called K. It would then summarize the output it scanned for, from all of the K numbers of values. Here K is considered to be a variable to hold the amount of closest instances to (n). The instances that are the closest to the instance (n) are found by measuring the distance between the stored data and the new instance. This measure of distance is called the Minkowski Distance.[7] After doing this much research, I soon realized the K value for each data set would be different and therefore affect the accuracy score. The second thing I realized is that each feature worked with a value of priority and changing this would reduce the control of the experiment. Therefore I chose to keep these values the same throughout the experiment.

Decision Tree Algorithms

The goal of decision tree learning is to accurately predict the value of the test variable using many other variables which are involved in the decision-making process. The model that will be used in this experiment is CART or classification and regression tree algorithm. Unlike the instance-based learning, decision trees must generalize with the training data to accurately make predictions and cannot store the entire set into memory.

---

[4] "RUSSELL, STUART NORVIG PETER. ARTIFICIAL INTELLIGENCE: a modern approach. PEARSON, 2018.
[5] "RUSSELL, STUART NORVIG PETER. ARTIFICIAL INTELLIGENCE: a modern approach. PEARSON, 2018.
[6] Srivastava, Tavish, Pranav Dar, and Pranjal Srivastava. "Introduction to KNN, K-Nearest Neighbors : Simplified." *Analytics Vidhya*. N.p., 16 Apr. 2015. Web. 10 Oct. 2017.
[7] "K-Nearest Neighbors for Machine Learning." *Machine Learning Mastery*. N.p., 21 Sept. 2016. Web. 10 Oct. 2017.



[8]

Figure 1

As it can be seen from the example above, in figure one, there is a decision tree with the situation of Should we "Play" Here the situation is the root node which is then split into two decision nodes and one leaf node each with their own probability. The two decision nodes, "sunny," "overcast," and "rain" respectively are then further split into two leaf nodes. A leaf node is a node which cannot split into any further nodes. The decision tree works based on the input variables, each input variable being a decision node and having a certain probability for each node depending on the target variable. Here the target variable is Play which is being run through a tree with different decision and leaf nodes which each have their own probability. If the values were: Play: Sunny =1, Humidity =1and >70, one being present and zero being not present, then by looking at the tree the decision that would be predicted would be not to play.

Classification and Regression Trees

A classification and regression tree or CART is a type of decision tree that can either be used for classification or regression tasks. Like the example above, the CART algorithm also takes up the shape of a binary tree, with the tree having nodes and each node having 2 or no other subordinate nodes. As stated above, the CART also works based on the input variables, each input variable being a decision node and having a specific probability for each node depending on the target variable. To build a binary tree, the process of splitting must happen, this is done through separating the input values from the dataset through recursive splitting. Recursive splitting is the process where all the input values are lined up, and all the split points are tested using the Gini cost function in this case.

[8] Analytics Vidhya Decision Tree Example. Digital image. *Https://www.analyticsvidhya.com*. N.p., n.d. Web. 10 Sept. 2017.

The Gini cost function[9] is used as it indicates the purity of the nodes, node purity is linked to how different the training data given to each node is. [10]This is what differentiates between Classification trees and regression trees. The split points are then chosen based on the ones with the lowest cost; this continues until nodes contain a minimum number of training examples of a maximum tree depth is reached. [11] This can be seen in the image below:

Features and attribute selection

Features and attributes can be considered as columns headers in a dataset, nodes in a decision tree, neighbors in KNN or the quantifiable properties which can change the complexity, learning, and accuracy of a model.[12] Features in a machine learning algorithm are required for learning to take place as they help differentiate the data. The purpose of this research is to find out how features really affect accuracy as there are multiple criteria; such as: using a large number of features, using a few features, using features that were picked through algorithms or using features picked out by humans. However, the complexity of the model's mapping function depends upon the number of features used or the size of the dataset. Making a model too complex is something that is avoided as is can lead to overfitting. Overfitting happens when a model is fit too closely to the training data and thus works exceptionally for that dataset but performs poorly for any other data used.[13] Thus if the data is noisy, data that cannot be understood by the machine[14], the abnormalities in the data lower the accuracy of the model.

Now that we have established that the bigger the data set, the more complex the model and the more complex a model is, the more risk is run for overfitting to occur. The question I asked myself now is that the best way that to reduce complexity, but yet have a model which generalizes sufficiently well would be to reduce the number of features of the data set right? Well not quite, as in every model the features are given some form of weight or importance. The feature weight is a quantity can decide how much a feature influences the model.[15] If an arbitrary dataset of fruits were chosen, in which there were two features which were called color and taste, and we were given a choice to choose between the most important feature. The logical choice would be color rather than taste as color is more relevant in a data set about fruits where the task is to distinguish. Thus, the color would be given more priority or weight over taste. Thus concluding that some features are more influential than others, we can now go into testing how the features and accuracy of a model are related, by observing how they compare when all are given equal importance to conserve control in the experiment.

9 Refer to the appendix for the formula

10 "How To Implement The Decision Tree Algorithm From Scratch In Python." Machine Learning Mastery. N.p., 11 Aug. 2017. Web. 5 Dec. 2017.

11 "How To Implement The Decision Tree Algorithm From Scratch In Python." Machine Learning Mastery. N.p., 11 Aug. 2017. Web. 5 Dec. 2017.

12 Bishop, Christopher M. Pattern recognition and machine learning. Springer, 2013.

13 MÃ¼ller, Andreas Christian., and Sarah Guido. Introduction to Machine Learning with Python: A Guide for Data Scientists. Sebastopol: O'Reilly, 2017. Print.

14 "What Is Noisy Data? - Definition from WhatIs.com." SearchBusinessAnalytics. N.p., n.d. Web. 4 Jan. 2018.

15 KNN Classifiers 1.1.1. Feature Weighting. N.p., n.d. Web. 4 Jan. 2018.

Methodology:

This experiment will be carried out by gathering 3 datasets from the UCL Machine Learning Repository[16], these data sets are the "Car Evaluation[17]," "Wine[18]," and "Iris[19]" datasets. The algorithms which are used are KNN and the Decision tree algorithms which will be imported from Scikit-learn's inbuilt library. The datasets will then be imported, and a certain amount of tests will be conducted using each dataset. The algorithm will be trained by changing the amount of the features every time, to see if there is a change in the accuracy value. The tests will be conducted in a manner that:

1. The first run will be using the entire dataset
2. The second run will be using half the data set
3. The third run will be using 1 more than half the dataset
4. The fourth run will be using 1 less than half the dataset

If the dataset contains more than 5 features, then 4 further tests would be run on the data set by selecting values which look the most relevant in comparison to the target value. The accuracy score will be recorded and analyzed.

Breakdown of code used

The code can be looked at using the instance that uses all the features from the dataset to obtain an overview.

```
import sklearn
import scipy as sp
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import mglearn
```

Figure 2: Importing all the necessary libraries

In this piece of code, for all trails, the necessary libraries are imported. Sklearn to import the classifier, Scipy and numpy are imported to process scientific problems. Matplotlib is imported to visualize the data in the form of graphs, but due to certain unforeseen problems, it wasn't used.

Code for KNN for Car Evaluation dataset using all features

[16] *Efi, Dheeru Dua and Karra Taniskidou. "{UCI} Machine Learning Repository." {UCI} Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences, 2017, archive.ics.uci.edu/ml.*
[17] *Efi, Dheeru Dua and Karra Taniskidou. "{UCI} Machine Learning Repository." {UCI} Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences, 2017, archive.ics.uci.edu/ml/datasets/Car+Evaluation.*
18 Efi, Dheeru Dua and Karra Taniskidou. "{UCI} Machine Learning Repository." {UCI} Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences, 2017, archive.ics.uci.edu/ml/datasets/Iris.
19 Efi, Dheeru Dua and Karra Taniskidou. "{UCI} Machine Learning Repository." {UCI} Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences, 2017, archive.ics.uci.edu/ml/datasets/Wine.

```
#importing car_data set for trail 1 using the whole dataset
car_data = pd.read_csv('D:\COS EE Datasets\Cars Dataset\car.data', header = None, index_col = False, names =['buying','maint','doors','persons','lug_boot', 'safety','target'])
#data = data[['buying','doors','persons','safety']]
# Editing the data using pandas dataframe.replace function
car_data.buying.replace(('vhigh','high','med','low'),(1,2,3,4), inplace=True)
car_data.maint.replace(('vhigh','high','med','low'),(1,2,3,4), inplace=True)
car_data.doors.replace(('2','3','4','5more'),(1,2,3,4), inplace=True)
car_data.persons.replace(('2','4','more'),(1,2,3), inplace=True)
car_data.lug_boot.replace(('small','med','big'),(1,2,3), inplace=True)
car_data.safety.replace(('low','med','high'),(1,2,3), inplace=True)
car_data.target.replace(('unacc','acc','good','vgood'),(1,2,3,4), inplace=True
)
display(car_data.head())
print("Shape of current cars dataset:{}".format(car_data.shape))
```

Figure 3

In figure 3, I have imported the data set using an inbuilt function from the pandas library called pd.read_csv(), which takes the parameters of the location of the data file, if header values in the data file are present, if the columns are already indexed, and finally the names of the column headers. Once the file had been correctly imported, I had to convert these dataset values from strings to integers as the k-nearest-neighbor algorithm cannot use string values. This was done using the replace function from the pandas data frame library. Once that was done I printed the first 5 values of the dataset and the shape of the data. This can be seen below in the figure below.

	buying	maint	doors	persons	lug_boot	safety	target
0	1	1	1	1	1	1	1
1	1	1	1	1	1	2	1
2	1	1	1	1	1	3	1
3	1	1	1	1	2	1	1
4	1	1	1	1	2	2	1

```
Shape of current cars dataset:(1728, 7)
```

Figure 4

```
from sklearn.model_selection import train_test_split
dataset = car_data.values
#assigning columns to
data1 = car_data.iloc[:,0:6]
#print(data1)
#assignning column target data
target = np.asarray(dataset[:,6], dtype="S6")
#print(target)
X_train, X_test, y_train, y_test = train_test_split(
    data1, target,test_size =0.2,random_state =0)
```

Figure 5

In figure 5 above I have imported the function called train_test_split from the scikit learn library. This function splits the dataset into training data and test data using a 75 to 25 split respectively. The function takes in the features, the target feature, the test size if needed, and the random_state.
The random_state is used to keep the split constant thus not changing the split data set every time the method is called. Line 3 shows the features variable called data1 being assigned the car_data columns 1 to 6, or all the features except the target feature. Line 6 shows that the target feature is being assigned to the variable target. The train test split function is then carried out.

```
print("X_train shape: {}".format(X_train.shape))
print("y_train shape: {}".format(y_train.shape))
```

```
X_train shape: (1382, 6)
y_train shape: (1382,)
```

```
print("X_test shape: {}".format(X_test.shape))
print("y_test shape: {}".format(y_test.shape))
```

```
X_test shape: (346, 6)
y_test shape: (346,)
```

Figure 6: Printing the shape for the training and test data

```
#Building the model: KNN using an arbitary number of neighbors
#importing KNN classifier
from sklearn.neighbors import KNeighborsClassifier
#instantiating the class for use
knn = KNeighborsClassifier()
```

```
#Training the classifier
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
```

Figure 7: Building the KNN model and training the model

In the image above, the KNeigborsClassifier is imported from the sklearn library and is then trained using the training values, X_train and Y_train. The weight is kept uniform, and the number of neighbors is set to 5, the default value, in order to keep the focus on the features rather than the controllable values associated with them.

```
prediction = knn.predict(X_test)
```

```
score = knn.score(X_test,y_test)
```

```
print(score)
```

```
0.924855491329
```

Figure 8: Predicting using test value and getting an accuracy score

In figure 8 I have used the inbuilt method called predict on the test data, and I used the score method to predict the accuracy of the model.
We can see it has an accuracy of approximately 92%. It must be stated that on the Car Evaluation Dataset has 6 features and 1 target feature thus 8 tests were run.

Code for KNN for Iris dataset using all features

This code is similar to the previous code with the only variation in the dataset as the "Iris" dataset was used. It must be stated that since Iris only contained 4 features, 4 tests were run using it.

```
iris_data = pd.read_csv('D:\COS EE Datasets\Iris Dataset\iris.data', header =
None,index_col = False, names =['sepal length','sepal width','petal length','p
etal width','target'])
iris_data.target.replace(('Iris-setosa','Iris-versicolor','Iris-virginica'),(0
,1,2), inplace = True)
display(iris_data.head())
print("Shape of current dataset:{}".format(iris_data.shape))
```

	sepal length	sepal width	petal length	petal width	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
Shape of current dataset:(150, 5)
```

```
from sklearn.model_selection import train_test_split
#using all the features
data = iris_data.iloc[:,0:4]
print(data.head())
target = iris_data.iloc[:,4]
print(target.head())
```

```
   sepal length  sepal width  petal length  petal width
0           5.1          3.5           1.4          0.2
1           4.9          3.0           1.4          0.2
2           4.7          3.2           1.3          0.2
3           4.6          3.1           1.5          0.2
4           5.0          3.6           1.4          0.2
0    0
1    0
2    0
3    0
4    0
Name: target, dtype: int64
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data,target, random_state
= 0)
```

```
print("X_train shape: {}".format(X_train.shape))
print("y_train shape: {}".format(y_train.shape))
print("X_test shape: {}".format(X_test.shape))
print("y_test shape: {}".format(y_test.shape))
```

```
X_train shape: (112, 4)
y_train shape: (112,)
X_test shape: (38, 4)
y_test shape: (38,)
```

```
#instantiating the model using the default number of k neigbors
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
```

```
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
```

```
prediction = knn.predict(X_test)
score =knn.score(X_test,y_test)
print(score)
```

```
0.973684210526
```

Code for KNN for Wine dataset using all features

This code is similar to the previous code with the only variation in the dataset as the "Wine" dataset was used. It must be stated that the Wine dataset contained 13 features; thus 8 tests were run using it.

```
wine_data = pd.read_csv(('D:\COS EE Datasets\Wine Dataset\wine.data'), header
= None, index_col = False, names =['target',
'Malic acid','Ash','Akcalinity','Magnesium','Total phenols','Flavanoids','Nonf
lavanoid phenols','Proanthocyanins','Color Intensity','Hue','OD280/OD315 of di
luted wines','Proline'])
#no need to replace as all values are numeric values and can be interpreted by
 the classifier
display(wine_data.head())
print("The shape of the current data set is: {}".format(wine_data.shape))
```

	target	Malic acid	Ash	Akcalinity	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proant
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39

```
The shape of the current data set is: (178, 13)
```

```
from sklearn.model_selection import train_test_split
#using the all features
data = wine_data.iloc[:,1:13]
print(data.head())
target = wine_data.iloc[:,0]
print(target.head())
```

```
   Malic acid   Ash  Akcalinity  Magnesium  Total phenols  Flavanoids  \
0       14.23  1.71        2.43       15.6            127        2.80
1       13.20  1.78        2.14       11.2            100        2.65
2       13.16  2.36        2.67       18.6            101        2.80
3       14.37  1.95        2.50       16.8            113        3.85
4       13.24  2.59        2.87       21.0            118        2.80

   Nonflavanoid phenols  Proanthocyanins  Color Intensity   Hue  \
0                  3.06             0.28             2.29  5.64
1                  2.76             0.26             1.28  4.38
2                  3.24             0.30             2.81  5.68
3                  3.49             0.24             2.18  7.80
4                  2.69             0.39             1.82  4.32

   OD280/OD315 of diluted wines  Proline
0                          1.04     3.92
1                          1.05     3.40
2                          1.03     3.17
3                          0.86     3.45
4                          1.04     2.93
0   1
1   1
2   1
3   1
4   1
Name: target, dtype: int64
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data,target, random_state
= 0)
```

```
print("X_train shape: {}".format(X_train.shape))
print("y_train shape: {}".format(y_train.shape))
print("X_test shape: {}".format(X_test.shape))
print("y_test shape: {}".format(y_test.shape))
```

```
X_train shape: (133, 12)
y_train shape: (133,)
X_test shape: (45, 12)
y_test shape: (45,)
```

```
#instantiating the model using the default number of k neigbors
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
```

```
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
```

```
prediction = knn.predict(X_test)
score =knn.score(X_test,y_test)
print(score)
```

```
0.866666666667
```

Code for Decision Tree for Car Evaluation dataset using all features

```
#importing car_data set for trail 1 using the whole dataset
car_data = pd.read_csv('D:\COS EE Datasets\Cars Dataset\car.data', header = No
ne, index_col = False, names =['buying','maint','doors','persons','lug_boot',
'safety','target'])
#data = data[['buying','doors','persons','safety']]
# Editing the data using pandas dataframe.replace function
car_data.buying.replace(('vhigh','high','med','low'),(1,2,3,4), inplace=True)
car_data.maint.replace(('vhigh','high','med','low'),(1,2,3,4), inplace=True)
car_data.doors.replace(('2','3','4','5more'),(1,2,3,4), inplace=True)
car_data.persons.replace(('2','4','more'),(1,2,3), inplace=True)
car_data.lug_boot.replace(('small','med','big'),(1,2,3), inplace=True)
car_data.safety.replace(('low','med','high'),(1,2,3), inplace=True)
car_data.target.replace(('unacc','acc','good','vgood'),(1,2,3,4), inplace=True
)
display(car_data.head())
print("Shape of current cars dataset:{}".format(car_data.shape))
```

In the figure above, I have imported the data set using an inbuilt function from the pandas library called pd.read_csv(), which takes the parameters of the location of the data file, if header values in the data file are present, if the columns are already indexed, and finally the names of the column headers. Once the file had been correctly imported and read into memory, I had to convert the code from string to integers as the Decision tree algorithm cannot use string values. This was done using the replace function from the pandas data frame library. Once that was done I printed the first 5 values of the dataset and the shape of the data. This can be seen below in the figure below.

	buying	maint	doors	persons	lug_boot	safety	target
0	1	1	1	1	1	1	1
1	1	1	1	1	1	2	1
2	1	1	1	1	1	3	1
3	1	1	1	1	2	1	1
4	1	1	1	1	2	2	1

```
Shape of current cars dataset:(1728, 7)
```

```
from sklearn.model_selection import train_test_split
dataset = car_data.values
#assigning columns to
data1 = car_data.iloc[:,0:6]
#print(data1)
#assignning column target data
target = np.asarray(dataset[:,6], dtype="S6")
#print(target)
X_train, X_test, y_train, y_test = train_test_split(
    data1, target,test_size =0.2,random_state =0)
```

In the figure above I have imported the function called train_test_split from the scikit learn library. This function splits the dataset into training data and test data. The function takes in the features, the target feature, the test size if needed, and the random_state. The random_state is used to keep the split constant thus not changing the split data set every time the method is called. Line 3 shows the features variable called data1 being assigned the car_data columns 1 to 6 thus all the features except the target feature. Then in line 6 shows that the target feature is being assigned to the variable target. The train test split function is then carried out.

```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(random_state =0)
```

```
tree.fit(X_train, y_train)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=0, splitter='best')
```

```
In [16]: print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))

Accuracy on test set: 0.972
```

Above I have used the inbuilt method called predict on the test data, and I used the score method to predict the accuracy of the model. We can see it has an accuracy of approximately 97%. As in the DT case, it must be stated that all datasets will be tested the same amount of times as in the KNN.

Code for Decision Tree for Iris dataset using all features

This code is similar to the previous code with the only variation in the dataset as the "Iris" dataset was used.

```
iris_data = pd.read_csv('D:\COS EE Datasets\Iris Dataset\iris.data', header = Non
iris_data.target.replace(('Iris-setosa','Iris-versicolor','Iris-virginica'),(0,1,
display(iris_data.head())
print("Shape of current dataset:{}".format(iris_data.shape))
```

	sepal length	sepal width	petal length	petal width	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
Shape of current dataset:(150, 5)
```

```
from sklearn.model_selection import train_test_split
#all of the features
data = iris_data.iloc[:,0:4]
print(data.head())
target = iris_data.iloc[:,4]
print(target.head())
X_train, X_test, y_train, y_test = train_test_split(
 data, target, random_state = 0)
```

```
   sepal length  sepal width  petal length  petal width
0           5.1          3.5           1.4          0.2
1           4.9          3.0           1.4          0.2
2           4.7          3.2           1.3          0.2
3           4.6          3.1           1.5          0.2
4           5.0          3.6           1.4          0.2
0    0
1    0
2    0
3    0
4    0
Name: target, dtype: int64
```

```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(random_state =0)
```

```
tree.fit(X_train, y_train)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=0, splitter='best')
```

```
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on test set: 0.974
```

Code for Decision Tree for Wine dataset using all features

This code is similar to the previous code with the only variation in the dataset as the "Wine" dataset was used.

```
wine_data = pd.read_csv(('D:\COS EE Datasets\Wine Dataset\wine.data'), header
= None, index_col = False, names =['target',
'Malic acid','Ash','Akcalinity','Magnesium','Total phenols','Flavanoids','Nonf
lavanoid phenols','Proanthocyanins','Color Intensity','Hue','OD280/OD315 of di
luted wines','Proline'])
#no need to replace as all values are numeric values and can be interpreted by
 the classifier
display(wine_data.head())
print("The shape of the current data set is: {}".format(wine_data.shape))
```

	target	Malic acid	Ash	Akcalinity	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proant
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39

```
The shape of the current data set is: (178, 13)
```

```
from sklearn.model_selection import train_test_split
#using the all features
data = wine_data.iloc[:,1:13]
print(data.head())
target = wine_data.iloc[:,0]
print(target.head())
```

```
   Malic acid   Ash  Akcalinity  Magnesium  Total phenols  Flavanoids  \
0       14.23  1.71        2.43       15.6            127        2.80
1       13.20  1.78        2.14       11.2            100        2.65
2       13.16  2.36        2.67       18.6            101        2.80
3       14.37  1.95        2.50       16.8            113        3.85
4       13.24  2.59        2.87       21.0            118        2.80

   Nonflavanoid phenols  Proanthocyanins  Color Intensity   Hue  \
0                  3.06             0.28             2.29  5.64
1                  2.76             0.26             1.28  4.38
2                  3.24             0.30             2.81  5.68
3                  3.49             0.24             2.18  7.80
4                  2.69             0.39             1.82  4.32

   OD280/OD315 of diluted wines  Proline
0                          1.04     3.92
1                          1.05     3.40
2                          1.03     3.17
3                          0.86     3.45
4                          1.04     2.93
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data,target, random_state
= 0)
```

```
print("X_train shape: {}".format(X_train.shape))
print("y_train shape: {}".format(y_train.shape))
print("X_test shape: {}".format(X_test.shape))
print("y_test shape: {}".format(y_test.shape))
```

```
X_train shape: (133, 12)
y_train shape: (133,)
X_test shape: (45, 12)
y_test shape: (45,)
```

```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(random_state =0)
```

```
tree.fit(X_train, y_train)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=0, splitter='best')
```

```
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on test set: 0.933
```

In the appendix, I have attached the code snips for the various tests conducted for the Car evaluation, Iris and Wine datasets. I have simply changed the number of features that were assigned to the data1 variable, and I have kept the target variable the same as the previous code as it shouldn't be changed. Besides the change in assignment of features to data1, the rest is the same. The change in the data1 variable value changes the accuracy value thus showing signs that the algorithm is fully functional and is predicting on the amount of features given to it.

Sample code illustrating the variation in the number of features selected for KNN Car dataset is shown below:

```
from sklearn.model_selection import train_test_split
dataset = car_data.values
#taking approximately half of the dataset
data1 = car_data.iloc[:,3:6]
print(data1.head())
#assignning column target data
target = np.asarray(dataset[:,6], dtype="S6")
print(target)
X_train, X_test, y_train, y_test = train_test_split(
    data1, target,test_size =0.2,random_state =0)
```

```
   persons  lug_boot  safety
0        1         1       1
1        1         1       2
2        1         1       3
3        1         2       1
4        1         2       2
[b'1' b'1' b'1' ..., b'1' b'3' b'4']
```

```
prediction = knn.predict(X_test)

score = knn.score(X_test,y_test)

print(score)
```

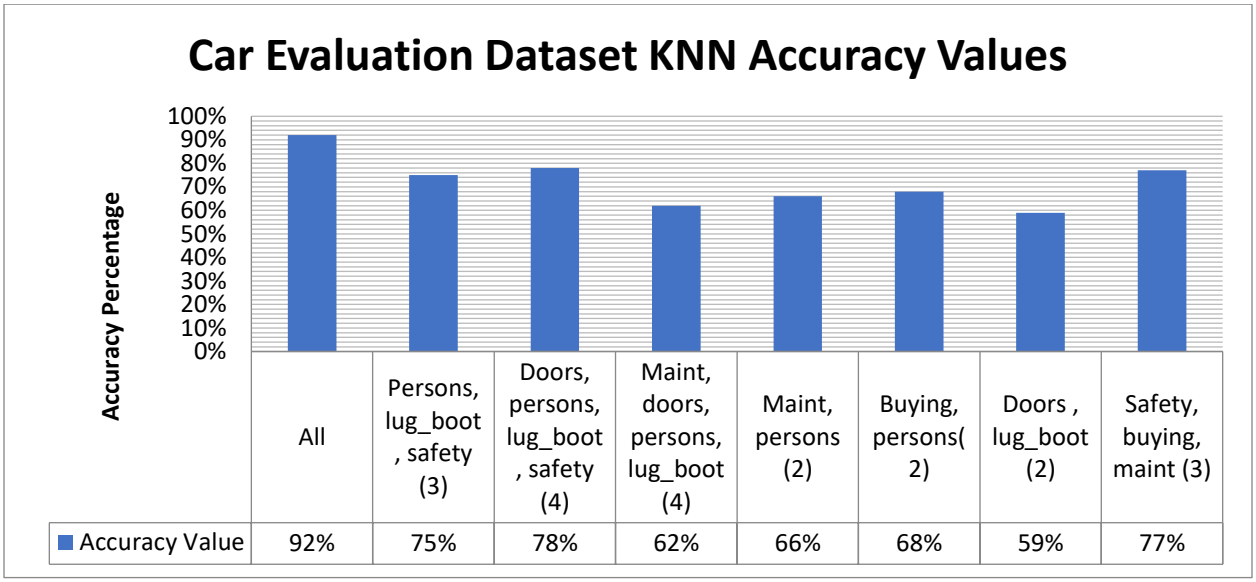
```
0.754335260116
```

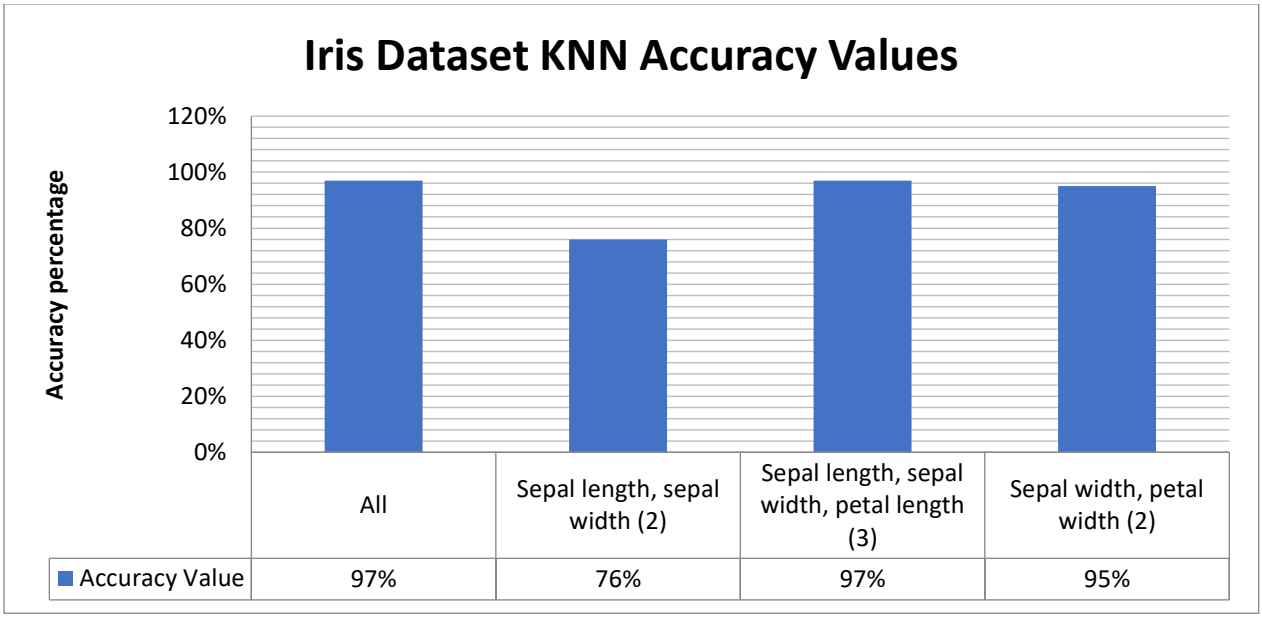
***It must be noted that empty datasets were not used as they will only produce errors within the code and that the result (if any) would be meaningless.**

Results and graph

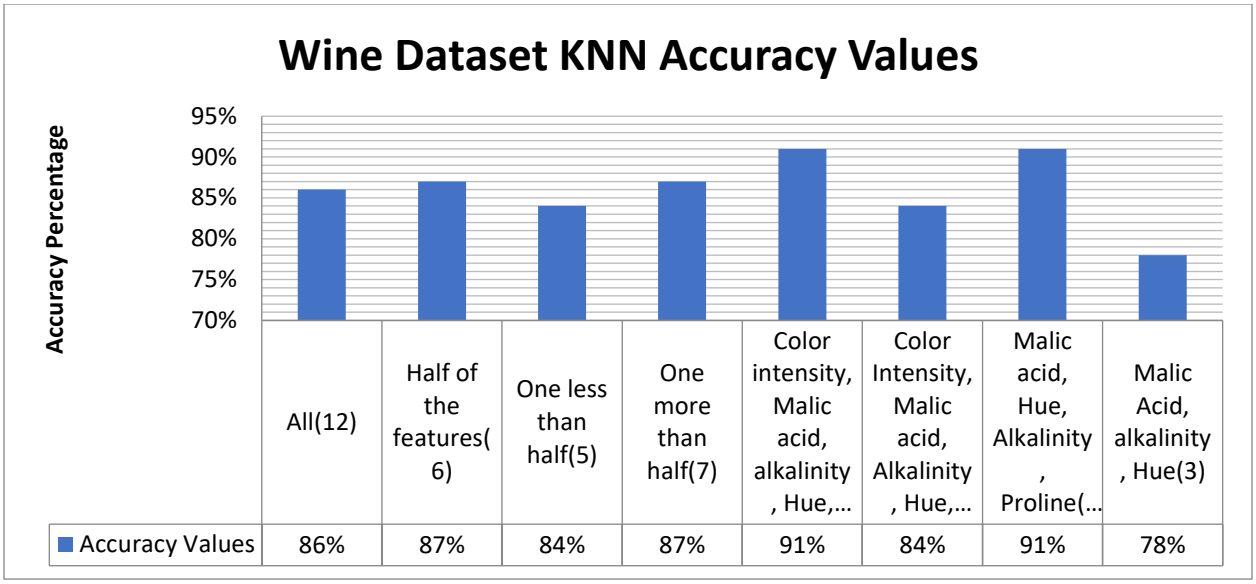
The observations made from the tests run on every dataset will be shown using charts, the tabulated accuracy values will be added in the appendix.[20]

K-Nearest Neighbor feature vs. accuracy tables

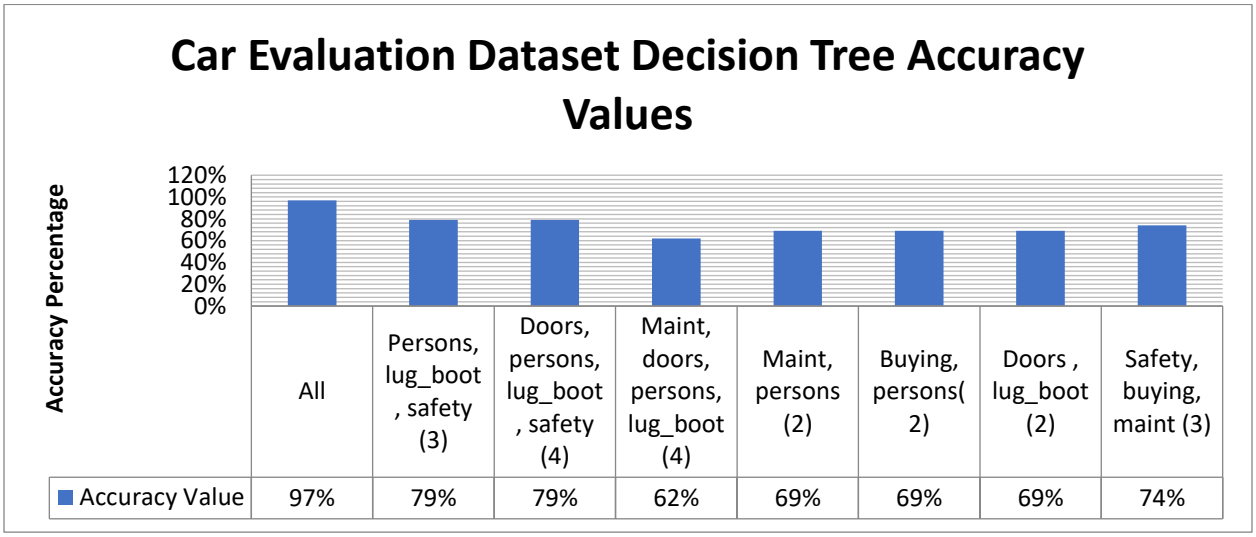


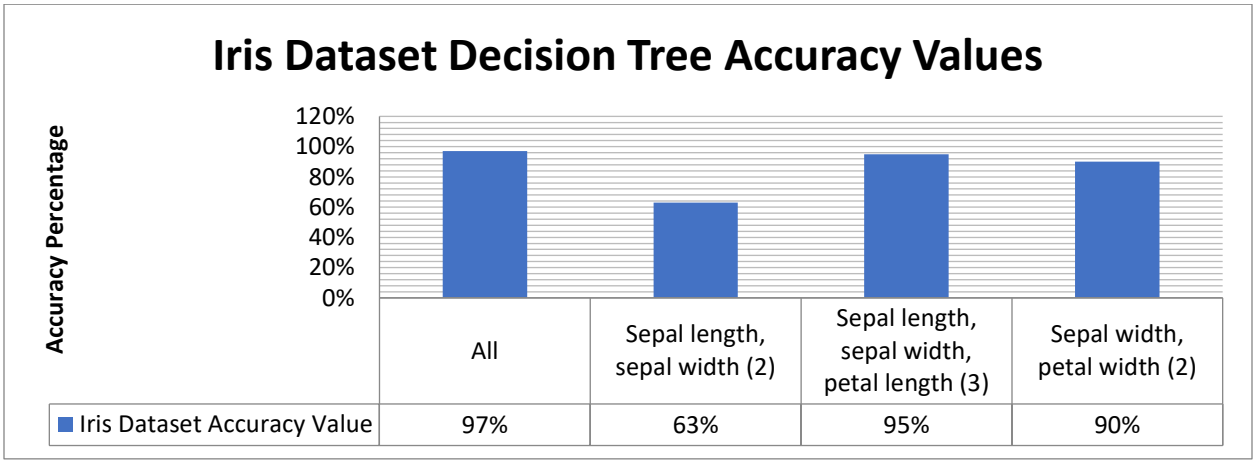


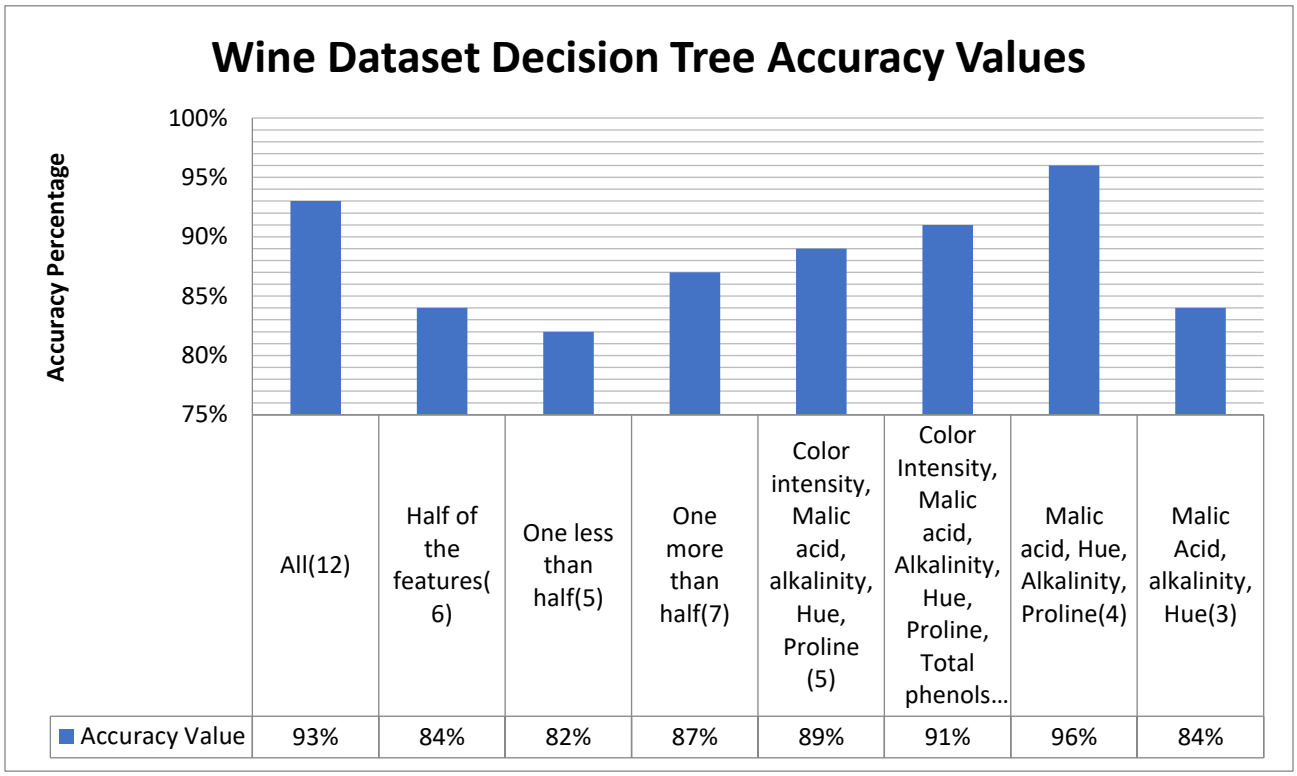
[20] See Appendix for tabulated values.



Decision tree feature vs. accuracy tables






Wine Dataset Decision Tree Accuracy Values
Accuracy Percentage
100%
95%
90%
85%
80%
75%
All(12)
Half of the features(6)
One less than half(5)
One more than half(7)
Color intensity, Malic acid, alkalinity, Hue, Proline (5)
Color Intensity, Malic acid, Alkalinity, Hue, Proline, Total phenols…
Malic acid, Hue, Alkalinity, Proline(4)
Malic Acid, alkalinity, Hue(3)
Accuracy Value
93%
84%
82%
87%
89%
91%
96%
84%

Analysis of results

Referring to the observations for both the algorithms in the previous section, we can make a few simple statements, these are that a few features provide a better accuracy score, we can get a higher accuracy score if more features are used, and in certain cases an arbitrary number of features in between the highest and lowest gives a high value. As stated before the aim of this experiment was to explore the accuracy and feature relationship as it may show how inefficient feature selections hinder algorithm accuracy of the algorithm. Thus we can now say that these statements are true as it all depends on a few factors, such as the type features used, the number of features used and the type of algorithm used.

Firstly the type of features must be analyzed; as in many situations the algorithms would contain some sort of weighting which would help differentiate them, however, I have kept the weight/priority uniform to keep some control in the experiment and give focus primarily to the number of features used and the feature used. The feature used, plays a big role within accurate predictions as they tie into how the algorithm uses the feature.

In the case of KNN it looks for the features which are the closest to the point that is being tested upon; thus it assumes the features closest to the test point has to similar to the test point and hence classifies based on an average of the closest point's value. This is very vulnerable to datasets which have lots of features as it may increase the chances of some features being very spread out thus increasing the distance between them. This can be seen the wine dataset where all 12 features were used. However, using the wine dataset as an example, when testing with certain features, these features create different spreads/clusters and which may reduce the distance between the test and training points therefore greatly increasing or decreasing the accuracy score. We see proof of this in the wine dataset where a score of 91% was awarded for the use of 5 and 4 features and a 86% when 12 are used. The main drawback for KNN, in terms of accuracy, would be the fact that it performs exceptionally when given a large amount of data however its performance degrades when a low amount of data is used as there isn’t a well-constructed spread.

In the case of Decision trees, which uses the gini index, once again a value which states the homogeneity of features of a node, to decide where to split, thus features that have a higher purity index are split on first, which then continues through recursive splitting. The decision tree was stopped at the default min sample value of 2 and was kept constant. Continuing with that idea, it can be said that a tree which is split properly would have a higher accuracy score. Hence it can then be said that when looking at the results, some features are able to divide the dataset in a much more efficient manner than others. Once again going back to the results used, if we look at the car evaluation dataset, it can be said that all the features were needed to efficiently split the tree, however in the wine dataset only 4 features were able to split and classify the wine better than all the features combined. However, likewise, there is a drawback of decision trees, which is the tree has a chance of overfitting to the data when the tree grows too big and may cause errors in prediction.

Thus it can be said that the number of features, does not matter, as too little or too much may cause overfitting and if features are carelessly picked, they may reduce accuracy. Rather features which are carefully selected and properly tailored to the type of model would provide the best accuracy result.

Conclusion & Limitations

The aim of this extended essay was to explore the accuracy and feature relationship as it may show how inefficient feature selections hinder algorithm accuracy. It can be concluded that KNN and Decision trees are still very dependent on the features provided to them to make an accurate prediction even though they operate in different ways. If the data were to be looked at, it can be said that when using all features, the decision tree is more accurate than the KNN algorithm, however when multiple predictions were made using different features, it can be seen that each algorithms' process gave different values, which showed that the features were the reason for the drastic changes in predictive accuracy. After testing, I concluded that features which are carefully selected and properly tailored to the type of model would provide the best accuracy result. However, the limitations that would be obstructing a perfect model would be the human error which comes into play when choosing features. This may cause obstruction as there is the probability of a human choosing an inefficient feature. But this may be overcome through the use of specific algorithms which systematically choose features which are guaranteed to pick features which increase a certain constant, either gini or the information gained, thus increasing the accuracy. Overall it can be said that the relationship between the features and accuracy of a machine learning model is not a linear one; however, features definitely do affect an algorithms accuracy and should not be overlooked.

Works Cited

“Siraj Raval.” *YouTube*, YouTube, www.youtube.com/channel/UCWN3xxRkmTPmbKwht9FuE5A.

Eaton, Eric. *Introduction to Machine Learning*. N.p.: Www.seas.upenn.edu/~cis519, n.d. PDF.

"RUSSELL, STUART NORVIG PETER. ARTIFICIAL INTELLIGENCE: a modern approach. PEARSON, 2018.

Srivastava, Tavish, Pranav Dar, and Pranjal Srivastava. "Introduction to KNN, K-Nearest Neighbors : Simplified." *Analytics Vidhya*. N.p., 16 Apr. 2015. Web. 10 Oct. 2017.

"K-Nearest Neighbors for Machine Learning." *Machine Learning Mastery*. N.p., 21 Sept. 2016. Web. 10 Oct. 2017.

Analytics Vidhya Decision Tree Example. Digital image. *Https://www.analyticsvidhya.com*. N.p., n.d. Web. 10 Sept. 2017.

"How To Implement The Decision Tree Algorithm From Scratch In Python." *Machine Learning Mastery*. N.p., 11 Aug. 2017. Web. 5 Dec. 2017.

Bishop, Christopher M. *Pattern recognition and machine learning*. Springer, 2013.

MÃ¼ller, Andreas Christian., and Sarah Guido. *Introduction to Machine Learning with Python: A Guide for Data Scientists*. Sebastopol: O'Reilly, 2017. Print.

"What Is Noisy Data? - Definition from WhatIs.com." *SearchBusinessAnalytics*. N.p., n.d. Web. 4 Jan. 2018.

KNN Classifiers 1.1.1. Feature Weighting. N.p., n.d. Web. 4 Jan. 2018.

Efi, Dheeru Dua and Karra Taniskidou. “{UCI} Machine Learning Repository.” {UCI} Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences, 2017, archive.ics.uci.edu/ml.

Efi, Dheeru Dua and Karra Taniskidou. “{UCI} Machine Learning Repository.” {UCI} Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences, 2017, archive.ics.uci.edu/ml/datasets/Car+Evaluation.

Efi, Dheeru Dua and Karra Taniskidou. “{UCI} Machine Learning Repository.” {UCI} Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences, 2017, archive.ics.uci.edu/ml/datasets/Iris.

Efi, Dheeru Dua and Karra Taniskidou. "{UCI} Machine Learning Repository." {UCI} Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences, 2017, archive.ics.uci.edu/ml/datasets/Wine.

De Freitas, Nando. *Machine Learning & Data Mining*. N.p.: University of British Columbia;, Sept. 2012. PDF.

"Supervised and Unsupervised Machine Learning Algorithms." *Machine Learning Mastery*. N.p., 22 Sept. 2016. Web. 10 Oct. 2017.

"Feature Selection to Improve Accuracy and Decrease Training Time." *Machine Learning Mastery*. N.p., 15 July 2016. Web. 10 Jan. 2018.

"Overfitting and Underfitting With Machine Learning Algorithms." *Machine Learning Mastery*. N.p., 27 Jan. 2017. Web. 18 Dec. 2017.

Team, Analytics Vidhya Content, Pranav Dar, and Pranjal Srivastava. "A Complete Tutorial on Tree Based Modeling from Scratch (in R & Python)." *Analytics Vidhya*. N.p., 01 May 2017. Web. 11 Dec. 2017.

KNN and ANN(MLP) for Car Evaluation | Kaggle. N.p., n.d. Web. 18 Dec. 2017.

"Feature Selection and Classification Accuracy Relation." *Machine Learning - Feature Selection and Classification Accuracy Relation - Data Science Stack Exchange*. N.p., n.d. Web. 10 Jan. 2018.

Appendixes

Tabulated Accuracy values

Car Evaluation Dataset KNN	
Features used	Accuracy Value
All	92%
Persons, lug_boot, safety (3)	75%
Doors, persons, lug_boot, safety (4)	78%
Maint, doors, persons, lug_boot (4)	62%
Maint, persons (2)	66%
Buying, persons(2)	68%
Doors , lug_boot (2)	59%
Safety, buying, maint (3)	77%

Iris Dataset KNN	
Features used	Accuracy Value
All	97%
Sepal length, sepal width (2)	76%
Sepal length, sepal width, petal length (3)	97%
Sepal width, petal width (2)	95%

Wine Dataset KNN	
Features used	Accuracy Value
All(12)	86%
Half of the features(6)	87%
One less than half(5)	84%
One more than half(7)	87%
Color intensity, Malic acid, alkalinity, Hue, Proline (5)	91%
Color Intensity, Malic acid, Alkalinity, Hue, Proline, Total phenols (6)	84%
Malic acid, Hue, Alkalinity, Proline(4)	91%
Malic Acid, alkalinity, Hue(3)	78%

Car Evaluation Dataset Decision Tree	
Features used	Accuracy Value
All	97%
Persons, lug_boot, safety (3)	79%
Doors, persons, lug_boot, safety (4)	79%
Maint, doors, persons, lug_boot (4)	62%
Maint, persons (2)	69%
Buying, persons(2)	69%
Doors , lug_boot (2)	69%
Safety, buying, maint (3)	74%

Iris Dataset Decision Tree	
Features used	Accuracy Value
All	97%
Sepal length, sepal width (2)	63%
Sepal length, sepal width, petal length (3)	95%
Sepal width, petal width (2)	90%

Wine Dataset Decision Tree	
Features used	Accuracy Value
All(12)	93%
Half of the features(6)	84%
One less than half(5)	82%
One more than half(7)	87%
Color intensity, Malic acid, alkalinity, Hue, Proline (5)	89%
Color Intensity, Malic acid, Alkalinity, Hue, Proline, Total phenols (6)	91%
Malic acid, Hue, Alkalinity, Proline(4)	96%
Malic Acid, alkalinity, Hue(3)	84%

KNN Code Snips for Cars Dataset

```
from sklearn.model_selection import train_test_split
dataset = car_data.values
#taking approximately half of the dataset
data1 = car_data.iloc[:,3:6]
print(data1.head())
#assignning column target data
target = np.asarray(dataset[:,6], dtype="S6")
print(target)
X_train, X_test, y_train, y_test = train_test_split(
    data1, target,test_size =0.2,random_state =0)
```

```
   persons  lug_boot  safety
0        1         1       1
1        1         1       2
2        1         1       3
3        1         2       1
4        1         2       2
[b'1' b'1' b'1' ..., b'1' b'3' b'4']
```

```
prediction = knn.predict(X_test)

score = knn.score(X_test,y_test)

print(score)
```

```
0.754335260116
```

```
from sklearn.model_selection import train_test_split
dataset = car_data.values
#taking approximately half of the dataset
data1 = car_data[['doors','persons','lug_boot','safety']]
print(data1.head())
#assignning column target data
target = np.asarray(dataset[:,6], dtype="S6")
print(target)
X_train, X_test, y_train, y_test = train_test_split(
    data1, target,test_size =0.2,random_state =0)
```

```
   doors  persons  lug_boot  safety
0      1        1         1       1
1      1        1         1       2
2      1        1         1       3
3      1        1         2       1
4      1        1         2       2
[b'1' b'1' b'1' ..., b'1' b'3' b'4']
```

```
prediction = knn.predict(X_test)

score = knn.score(X_test,y_test)

print(score)
```

```
0.780346820809
```

```
from sklearn.model_selection import train_test_split
dataset = car_data.values
#using half the dataset thus
#assigning half plus 1 to the data set
data1 = car_data[['maint','doors','persons','lug_boot']]
print(data1.head())
#assignning column target data
target = np.asarray(dataset[:,6], dtype="S6")
print(target)
X_train, X_test, y_train, y_test = train_test_split(
    data1, target,test_size =0.2)
```

```
   maint  doors  persons  lug_boot
0      1      1        1         1
1      1      1        1         1
2      1      1        1         1
3      1      1        1         2
4      1      1        1         2
[b'1' b'1' b'1' ..., b'1' b'3' b'4']
```

```
prediction = knn.predict(X_test)

score = knn.score(X_test,y_test)

print(score)
```

```
0.621387283237
```

```
from sklearn.model_selection import train_test_split
#using selected features
data1 = car_data[['maint','persons']]
print(data1.head())
#assignning column target data
target = car_data.iloc[:,6]
print(target.head())
X_train, X_test, y_train, y_test = train_test_split(
 data1, target, random_state = 0)
```

```
   maint  persons
0      1        1
1      1        1
2      1        1
3      1        1
4      1        1
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
prediction = knn.predict(X_test)

score = knn.score(X_test,y_test)

print(score)
```

```
0.666666666667
```

```
from sklearn.model_selection import train_test_split
#using selected features
data1 = car_data[['buying','persons']]
print(data1.head())
#assignning column target data
target = car_data.iloc[:,6]
print(target.head())
X_train, X_test, y_train, y_test = train_test_split(
 data1, target, random_state = 0)
```

```
   buying  persons
0       1        1
1       1        1
2       1        1
3       1        1
4       1        1
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
prediction = knn.predict(X_test)

score = knn.score(X_test,y_test)

print(score)
```

```
0.678240740741
```

```
from sklearn.model_selection import train_test_split
#using selected features
data1 = car_data[['doors','lug_boot']]
print(data1.head())
#assignning column target data
target = car_data.iloc[:,6]
print(target.head())
X_train, X_test, y_train, y_test = train_test_split(
 data1, target, random_state = 0)
```

```
   doors  lug_boot
0      1         1
1      1         1
2      1         1
3      1         2
4      1         2
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
prediction = knn.predict(X_test)

score = knn.score(X_test,y_test)

print(score)
```

```
0.585648148148
```

```
from sklearn.model_selection import train_test_split
#using selected features
data1 = car_data[['safety','buying','maint']]
print(data1.head())
#assignning column target data
target = car_data.iloc[:,6]
print(target.head())
X_train, X_test, y_train, y_test = train_test_split(
 data1, target, random_state = 0)
```

```
   safety  buying  maint
0       1       1      1
1       2       1      1
2       3       1      1
3       1       1      1
4       2       1      1
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
prediction = knn.predict(X_test)

score = knn.score(X_test,y_test)

print(score)
```

```
0.768518518519
```

KNN Code Snips for Iris Dataset

```
from sklearn.model_selection import train_test_split
#using 2 of the features
data = iris_data[['sepal length', 'sepal width']]
print(data.head())
target = iris_data.iloc[:,4]
print(target.head())
```

```
   sepal length  sepal width
0           5.1          3.5
1           4.9          3.0
2           4.7          3.2
3           4.6          3.1
4           5.0          3.6
0    0
1    0
2    0
3    0
4    0
Name: target, dtype: int64
```

```
prediction = knn.predict(X_test)
score =knn.score(X_test,y_test)
print(score)
```

```
0.763157894737
```

```
from sklearn.model_selection import train_test_split
#using 3 of the features
data = iris_data[['sepal length', 'sepal width','petal length']]
print(data.head())
target = iris_data.iloc[:,4]
print(target.head())
```

```
   sepal length  sepal width  petal length
0           5.1          3.5           1.4
1           4.9          3.0           1.4
2           4.7          3.2           1.3
3           4.6          3.1           1.5
4           5.0          3.6           1.4
0    0
1    0
2    0
3    0
4    0
Name: target, dtype: int64
```

```
prediction = knn.predict(X_test)
score =knn.score(X_test,y_test)
print(score)
```

```
0.973684210526
```

```
from sklearn.model_selection import train_test_split
#using 2 of the features
data = iris_data[['sepal width','petal width']]
print(data.head())
target = iris_data.iloc[:,4]
print(target.head())
```

```
   sepal width  petal width
0          3.5          0.2
1          3.0          0.2
2          3.2          0.2
3          3.1          0.2
4          3.6          0.2
0    0
1    0
2    0
3    0
4    0
Name: target, dtype: int64
```

```
prediction = knn.predict(X_test)
score =knn.score(X_test,y_test)
print(score)
```

```
0.947368421053
```

KNN Code Snips for Wine Dataset

```
from sklearn.model_selection import train_test_split
#using half of the features
data = wine_data.iloc[:,7:13]
print(data.head())
target = wine_data.iloc[:,0]
print(target.head())
```

```
   Nonflavanoid phenols  Proanthocyanins  Color Intensity   Hue  \
0                  3.06             0.28             2.29  5.64
1                  2.76             0.26             1.28  4.38
2                  3.24             0.30             2.81  5.68
3                  3.49             0.24             2.18  7.80
4                  2.69             0.39             1.82  4.32

   OD280/OD315 of diluted wines  Proline
0                          1.04     3.92
1                          1.05     3.40
2                          1.03     3.17
3                          0.86     3.45
4                          1.04     2.93
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
prediction = knn.predict(X_test)
score =knn.score(X_test,y_test)
print(score)
```

```
0.866666666667
```

```
from sklearn.model_selection import train_test_split
#using 1 less than half
data = wine_data.iloc[:,8:13]
print(data.head())
target = wine_data.iloc[:,0]
print(target.head())
```

```
   Proanthocyanins  Color Intensity   Hue  OD280/OD315 of diluted wines  \
0             0.28             2.29  5.64                          1.04
1             0.26             1.28  4.38                          1.05
2             0.30             2.81  5.68                          1.03
3             0.24             2.18  7.80                          0.86
4             0.39             1.82  4.32                          1.04

   Proline
0     3.92
1     3.40
2     3.17
3     3.45
4     2.93
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
prediction = knn.predict(X_test)
score =knn.score(X_test,y_test)
print(score)
```

```
0.844444444444
```

```
from sklearn.model_selection import train_test_split
#using 1 more than half
data = wine_data.iloc[:,6:13]
print(data.head())
target = wine_data.iloc[:,0]
print(target.head())
```

```
   Flavanoids  Nonflavanoid phenols  Proanthocyanins  Color Intensity   Hue
0        2.80                  3.06             0.28             2.29  5.64
1        2.65                  2.76             0.26             1.28  4.38
2        2.80                  3.24             0.30             2.81  5.68
3        3.85                  3.49             0.24             2.18  7.80
4        2.80                  2.69             0.39             1.82  4.32

   OD280/OD315 of diluted wines  Proline
0                          1.04     3.92
1                          1.05     3.40
2                          1.03     3.17
3                          0.86     3.45
4                          1.04     2.93
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
prediction = knn.predict(X_test)
score =knn.score(X_test,y_test)
print(score)
```

```
0.866666666667
```

```
from sklearn.model_selection import train_test_split
#using selected values
data = wine_data[['Color Intensity','Malic acid','Akcalinity','Hue','Proline']]
print(data.head())
target = wine_data.iloc[:,0]
print(target.head())
```

```
   Color Intensity  Malic acid  Akcalinity   Hue  Proline
0             2.29       14.23        2.43  5.64     3.92
1             1.28       13.20        2.14  4.38     3.40
2             2.81       13.16        2.67  5.68     3.17
3             2.18       14.37        2.50  7.80     3.45
4             1.82       13.24        2.87  4.32     2.93
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
prediction = knn.predict(X_test)
score =knn.score(X_test,y_test)
print(score)
```

```
0.911111111111
```

```
from sklearn.model_selection import train_test_split
#using selected values v 2
data = wine_data[['Color Intensity','Malic acid','Akcalinity','Hue','Proline','To
print(data.head())
target = wine_data.iloc[:,0]
print(target.head())
```

```
   Color Intensity  Malic acid  Akcalinity   Hue  Proline  Total phenols
0             2.29       14.23        2.43  5.64     3.92            127
1             1.28       13.20        2.14  4.38     3.40            100
2             2.81       13.16        2.67  5.68     3.17            101
3             2.18       14.37        2.50  7.80     3.45            113
4             1.82       13.24        2.87  4.32     2.93            118
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
prediction = knn.predict(X_test)
score =knn.score(X_test,y_test)
print(score)
```

```
0.844444444444
```

```
from sklearn.model_selection import train_test_split
#using selected values v 2
data = wine_data[['Malic acid','Akcalinity','Hue','Proline',]]
print(data.head())
target = wine_data.iloc[:,0]
print(target.head())
```

```
   Malic acid  Akcalinity   Hue  Proline
0       14.23        2.43  5.64     3.92
1       13.20        2.14  4.38     3.40
2       13.16        2.67  5.68     3.17
3       14.37        2.50  7.80     3.45
4       13.24        2.87  4.32     2.93
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
prediction = knn.predict(X_test)
score =knn.score(X_test,y_test)
print(score)
```

```
0.911111111111
```

```
from sklearn.model_selection import train_test_split
#using selected values v 2
data = wine_data[['Malic acid','Akcalinity','Hue',]]
print(data.head())
target = wine_data.iloc[:,0]
print(target.head())
```

```
   Malic acid  Akcalinity   Hue
0       14.23        2.43  5.64
1       13.20        2.14  4.38
2       13.16        2.67  5.68
3       14.37        2.50  7.80
4       13.24        2.87  4.32
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
prediction = knn.predict(X_test)
score =knn.score(X_test,y_test)
print(score)
```

```
0.777777777778
```

Decision Tree Code Snips for Cars Dataset

```
from sklearn.model_selection import train_test_split
dataset = car_data.values
#taking approximately half of the dataset
data1 = car_data.iloc[:,3:6]
print(data1.head())
#assignning column target data
target = np.asarray(dataset[:,6], dtype="S6")
print(target)
X_train, X_test, y_train, y_test = train_test_split(
    data1, target,test_size =0.2,random_state =0)
```

```
   persons  lug_boot  safety
0        1         1       1
1        1         1       2
2        1         1       3
3        1         2       1
4        1         2       2
[b'1' b'1' b'1' ..., b'1' b'3' b'4']
```

```
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on test set: 0.787
```

```
from sklearn.model_selection import train_test_split
dataset = car_data.values
#taking approximately half of the dataset
data1 = car_data[['doors','persons','lug_boot','safety']]
print(data1.head())
#assignning column target data
target = np.asarray(dataset[:,6], dtype="S6")
print(target)
X_train, X_test, y_train, y_test = train_test_split(
    data1, target,test_size =0.2,random_state =0)
```

```
   doors  persons  lug_boot  safety
0      1        1         1       1
1      1        1         1       2
2      1        1         1       3
3      1        1         2       1
4      1        1         2       2
[b'1' b'1' b'1' ..., b'1' b'3' b'4']
```

```
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on test set: 0.792
```

```
from sklearn.model_selection import train_test_split
dataset = car_data.values
#using half the dataset thus
#assigning half plus 1 to the data set
data1 = car_data[['maint','doors','persons','lug_boot']]
print(data1.head())
#assignning column target data
target = np.asarray(dataset[:,6], dtype="S6")
print(target)
X_train, X_test, y_train, y_test = train_test_split(
    data1, target,test_size =0.2)
```

```
   maint  doors  persons  lug_boot
0      1      1        1         1
1      1      1        1         1
2      1      1        1         1
3      1      1        1         2
4      1      1        1         2
[b'1' b'1' b'1' ..., b'1' b'3' b'4']
```

```
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

Accuracy on test set: 0.616

```
from sklearn.model_selection import train_test_split
#using selected features
data1 = car_data[['maint','persons']]
print(data1.head())
#assignning column target data
target = car_data.iloc[:,6]
print(target.head())
X_train, X_test, y_train, y_test = train_test_split(
 data1, target, random_state = 0)
```

```
   maint  persons
0      1        1
1      1        1
2      1        1
3      1        1
4      1        1
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

Accuracy on test set: 0.685

```
from sklearn.model_selection import train_test_split
#using selected features
data1 = car_data[['buying','persons']]
print(data1.head())
#assignning column target data
target = car_data.iloc[:,6]
print(target.head())
X_train, X_test, y_train, y_test = train_test_split(
 data1, target, random_state = 0)
```

```
   buying  persons
0       1        1
1       1        1
2       1        1
3       1        1
4       1        1
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

Accuracy on test set: 0.685

```
from sklearn.model_selection import train_test_split
#using selected features
data1 = car_data[['doors','lug_boot']]
print(data1.head())
#assignning column target data
target = car_data.iloc[:,6]
print(target.head())
X_train, X_test, y_train, y_test = train_test_split(
 data1, target, random_state = 0)
```

```
   doors  lug_boot
0      1         1
1      1         1
2      1         1
3      1         2
4      1         2
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on test set: 0.685
```

```
from sklearn.model_selection import train_test_split
#using selected features
data1 = car_data[['safety','buying','maint']]
print(data1.head())
#assignning column target data
target = car_data.iloc[:,6]
print(target.head())
X_train, X_test, y_train, y_test = train_test_split(
 data1, target, random_state = 0)
```

```
   safety  buying  maint
0       1       1      1
1       2       1      1
2       3       1      1
3       1       1      1
4       2       1      1
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on test set: 0.743
```

Decision Tree Code Snips for Iris Dataset

```
from sklearn.model_selection import train_test_split
#using 2 of the features
data = iris_data[['sepal length', 'sepal width']]
print(data.head())
target = iris_data.iloc[:,4]
print(target.head())
```

```
   sepal length  sepal width
0           5.1          3.5
1           4.9          3.0
2           4.7          3.2
3           4.6          3.1
4           5.0          3.6
0    0
1    0
2    0
3    0
4    0
Name: target, dtype: int64
```

```
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on test set: 0.632
```

```
from sklearn.model_selection import train_test_split
#using 3 of the features
data = iris_data[['sepal length', 'sepal width','petal length']]
print(data.head())
target = iris_data.iloc[:,4]
print(target.head())
```

```
   sepal length  sepal width  petal length
0           5.1          3.5           1.4
1           4.9          3.0           1.4
2           4.7          3.2           1.3
3           4.6          3.1           1.5
4           5.0          3.6           1.4
0    0
1    0
2    0
3    0
4    0
Name: target, dtype: int64
```

```
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on test set: 0.947
```

```
from sklearn.model_selection import train_test_split
#using 2 of the features
data = iris_data[['sepal width','petal width']]
print(data.head())
target = iris_data.iloc[:,4]
print(target.head())
```

```
   sepal width  petal width
0          3.5          0.2
1          3.0          0.2
2          3.2          0.2
3          3.1          0.2
4          3.6          0.2
0    0
1    0
2    0
3    0
4    0
Name: target, dtype: int64
```

```
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on test set: 0.895
```

Decision Tree Code Snips for Wine Dataset

```
from sklearn.model_selection import train_test_split
#using half of the features
data = wine_data.iloc[:,7:13]
print(data.head())
target = wine_data.iloc[:,0]
print(target.head())
```

```
   Nonflavanoid phenols  Proanthocyanins  Color Intensity   Hue  \
0                  3.06             0.28             2.29  5.64
1                  2.76             0.26             1.28  4.38
2                  3.24             0.30             2.81  5.68
3                  3.49             0.24             2.18  7.80
4                  2.69             0.39             1.82  4.32

   OD280/OD315 of diluted wines  Proline
0                          1.04     3.92
1                          1.05     3.40
2                          1.03     3.17
3                          0.86     3.45
4                          1.04     2.93
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on test set: 0.844
```

```
from sklearn.model_selection import train_test_split
#using 1 less than half
data = wine_data.iloc[:,8:13]
print(data.head())
target = wine_data.iloc[:,0]
print(target.head())
```

```
   Proanthocyanins  Color Intensity   Hue  OD280/OD315 of diluted wines  \
0             0.28             2.29  5.64                          1.04
1             0.26             1.28  4.38                          1.05
2             0.30             2.81  5.68                          1.03
3             0.24             2.18  7.80                          0.86
4             0.39             1.82  4.32                          1.04

   Proline
0     3.92
1     3.40
2     3.17
3     3.45
4     2.93
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on test set: 0.822
```

```
from sklearn.model_selection import train_test_split
#using 1 more than half
data = wine_data.iloc[:,6:13]
print(data.head())
target = wine_data.iloc[:,0]
print(target.head())
```

```
   Flavanoids  Nonflavanoid phenols  Proanthocyanins  Color Intensity   Hue
0        2.80                  3.06             0.28             2.29  5.64
1        2.65                  2.76             0.26             1.28  4.38
2        2.80                  3.24             0.30             2.81  5.68
3        3.85                  3.49             0.24             2.18  7.80
4        2.80                  2.69             0.39             1.82  4.32

   OD280/OD315 of diluted wines  Proline
0                          1.04     3.92
1                          1.05     3.40
2                          1.03     3.17
3                          0.86     3.45
4                          1.04     2.93
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on test set: 0.867
```

```
from sklearn.model_selection import train_test_split
#using selected values
data = wine_data[['Color Intensity','Malic acid','Akcalinity','Hue','Proline']]
print(data.head())
target = wine_data.iloc[:,0]
print(target.head())
```

```
   Color Intensity  Malic acid  Akcalinity   Hue  Proline
0             2.29       14.23        2.43  5.64     3.92
1             1.28       13.20        2.14  4.38     3.40
2             2.81       13.16        2.67  5.68     3.17
3             2.18       14.37        2.50  7.80     3.45
4             1.82       13.24        2.87  4.32     2.93
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on test set: 0.889
```

```
from sklearn.model_selection import train_test_split
#using selected values v 2
data = wine_data[['Color Intensity','Malic acid','Akcalinity','Hue','Proline','To
print(data.head())
target = wine_data.iloc[:,0]
print(target.head())
```

```
   Color Intensity  Malic acid  Akcalinity   Hue  Proline  Total phenols
0             2.29       14.23        2.43  5.64     3.92            127
1             1.28       13.20        2.14  4.38     3.40            100
2             2.81       13.16        2.67  5.68     3.17            101
3             2.18       14.37        2.50  7.80     3.45            113
4             1.82       13.24        2.87  4.32     2.93            118
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on test set: 0.911
```

```
from sklearn.model_selection import train_test_split
#using selected values v 2
data = wine_data[['Malic acid','Akcalinity','Hue','Proline',]]
print(data.head())
target = wine_data.iloc[:,0]
print(target.head())
```

```
   Malic acid  Akcalinity   Hue  Proline
0       14.23        2.43  5.64     3.92
1       13.20        2.14  4.38     3.40
2       13.16        2.67  5.68     3.17
3       14.37        2.50  7.80     3.45
4       13.24        2.87  4.32     2.93
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on test set: 0.956
```

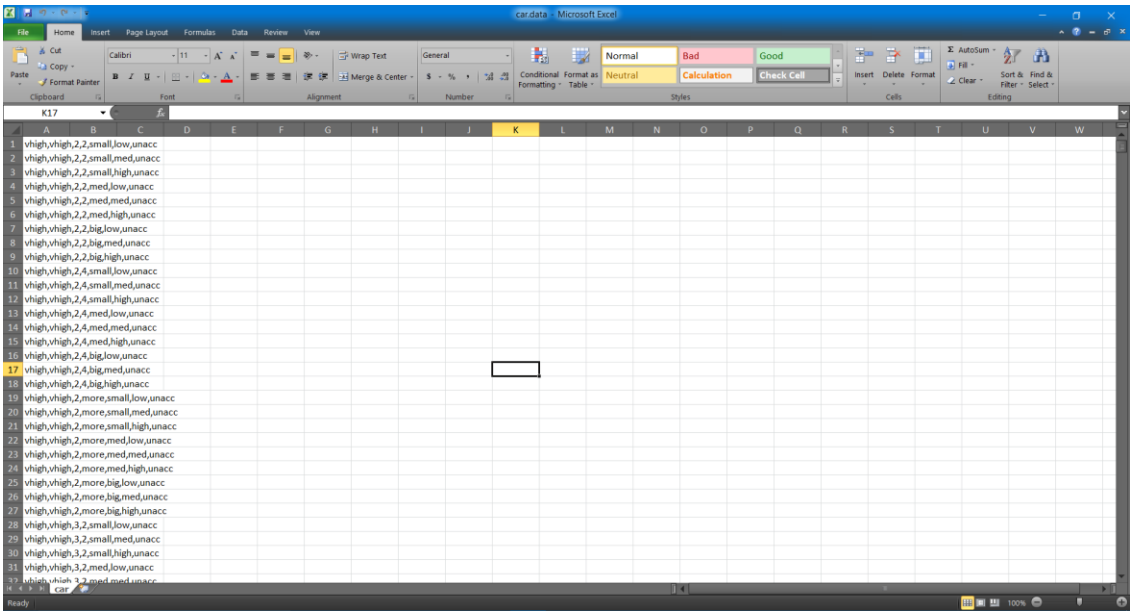
```
from sklearn.model_selection import train_test_split
#using selected values v 2
data = wine_data[['Malic acid','Akcalinity','Hue',]]
print(data.head())
target = wine_data.iloc[:,0]
print(target.head())
```

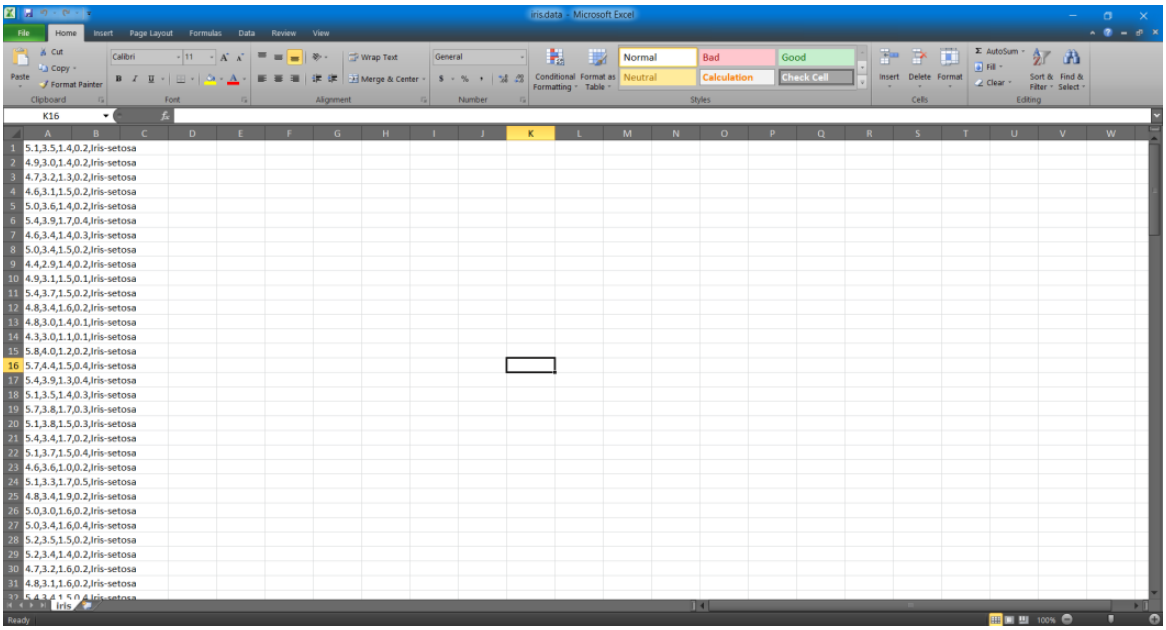
```
   Malic acid  Akcalinity   Hue
0       14.23        2.43  5.64
1       13.20        2.14  4.38
2       13.16        2.67  5.68
3       14.37        2.50  7.80
4       13.24        2.87  4.32
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

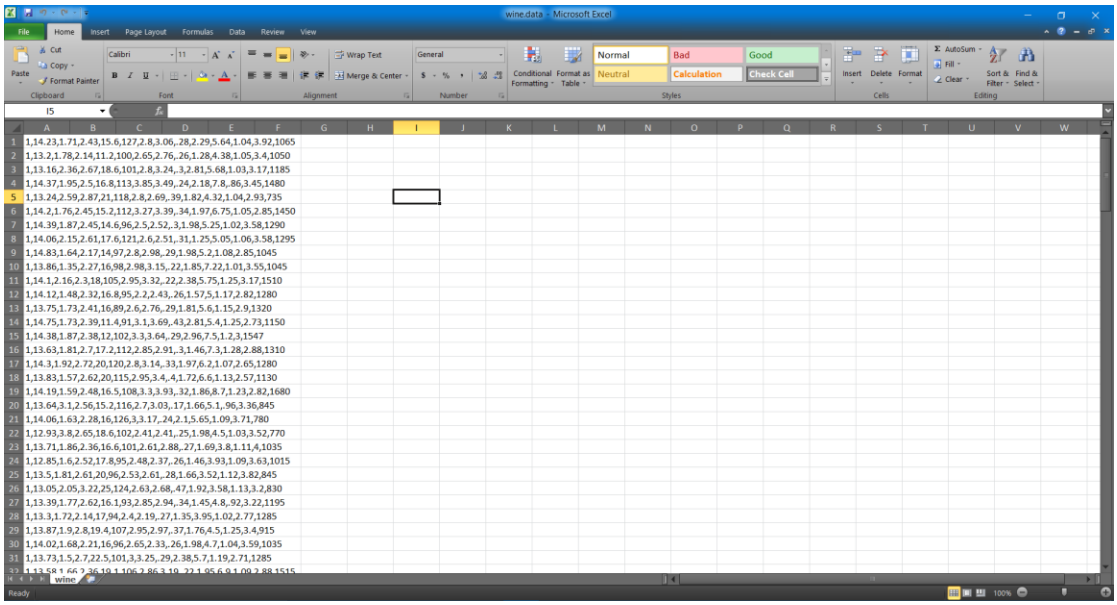
```
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on test set: 0.844
```

Datasets used in the EE:







Gini Cost function

Finding the Best Split: GINI Index

When a node p is split into k partitions (children), the quality of split is computed as:

$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

ni = number of records at child i,
n = number of records at node p