Machine Learning Package:

Package Used: Scikit Learn

I have used **Scikit Learn** for implementing the given task chosen because of the following reason:

- It's easy to use even for beginners – and a great choice for simpler data analysis tasks.
- Can understand how the data is being analysed and users have great control over data to get different outcomes.
- Also, it will be good start for me to work on ML tasks with python coding involved in it.

Features [a]:

- **Consistency:** All objects share a common interface drawn from a limited set of methods, with consistent documentation.
- **Inspection:** All specified parameter values are exposed as public attributes.
- **Limited object hierarchy:** Only algorithms are represented by Python classes; datasets are represented in standard formats (NumPy arrays, Pandas DataFrames, SciPy sparse matrices) and parameter names use standard Python strings.
- **Composition:** Many machine learning tasks can be expressed as sequences of more fundamental algorithms, and Scikit-Learn makes use of this wherever possible.
- **Sensible defaults:** When models require user-specified parameters, the library defines an appropriate default value

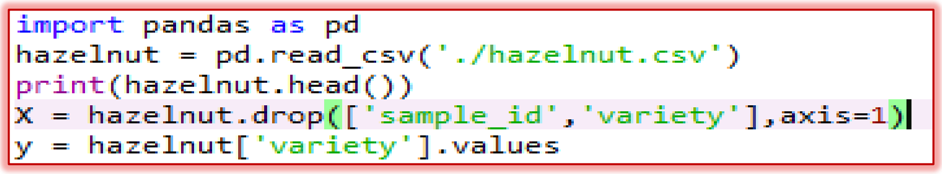
Data Preparation:

Data preparation before feeding data into algorithm:

1. Copy hazelnut.txt data into **hazelnut.csv. Transpose the data** (to correctly arrange rows and columns for the headers provided)
2. **Add headers** to each column in the order as specified in the assignment1.pdf document.

Data preparation after feeding data into algorithm:

1. **Read csv** file and store it in a data frame (using pandas).
2. **Drop 'sample_id'** (since it contains only unique ID and doesn't contribute in deciding the target) and **'variety'** (since it the target) column, and store the remaining data into X (now X contains attributes)
3. Load the values of column **'variety' into 'y'** (y contains the target data)
4. Code Implementation:



```
import pandas as pd
hazelnut = pd.read_csv('./hazelnut.csv')
print(hazelnut.head())
X = hazelnut.drop(['sample_id','variety'],axis=1)
y = hazelnut['variety'].values
```

References: [a]: https://jakevdp.github.io/PythonDataScienceHandbook/05.02-introducing-scikit-learn.html

Classification Algorithms:

Algorithm 1: K Nearest Neighbors (k-NN) [1]

- The principle behind nearest neighbor method is to find a predefined number of training samples closest in distance to the new point and predict the label from these.
- k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally, and all computation is deferred until classification.
- The **input consists of the k closest training examples** in the feature space.
- The **output is a class membership**.
- An object is **classified by a plurality vote of its neighbors**, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small).
- If k = 1, then the object is simply assigned to the class of that single nearest neighbor.
- Choosing the **optimal value for K is best done by first inspecting the data**.
- In general, a large K value is more precise as it reduces the overall noise but there is no guarantee.
- **Cross-validation is another way to retrospectively determine a good K** value by using an independent dataset to validate the K value.
- Historically, the optimal K for most datasets has been between 3-21.

Algorithm 2: Support Vector Machines (SVM) [2][3]

- The objective of the support vector machine algorithm is to **find a hyperplane** in an N-dimensional space(N - the number of features) that distinctly classifies the data points.
- An SVM model is a **representation of the examples as points in space**, mapped so that the examples of the separate categories are **divided by a clear gap that is as wide as possible**.
- A good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the **larger the margin, the lower the generalization error of the classifier**.
- New examples are then mapped into that same space and predicted to belong to a category based on the side of the gap on which they fall.
- An SVM is a discriminative classifier formally defined by a separating hyperplane.
- In other words, given labelled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples.
- In **two-dimensional space this hyperplane is a line** dividing a plane in two parts where in each class lay in either side.

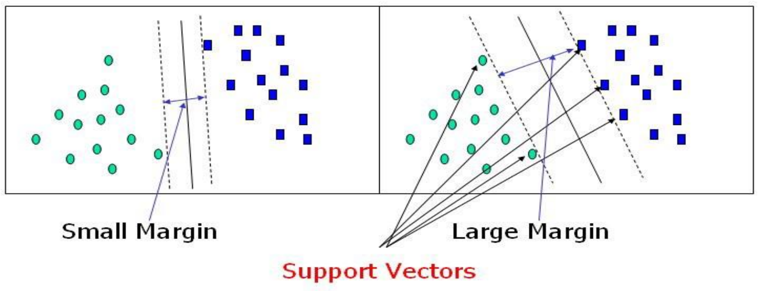


image source:
https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

References:
[1]: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

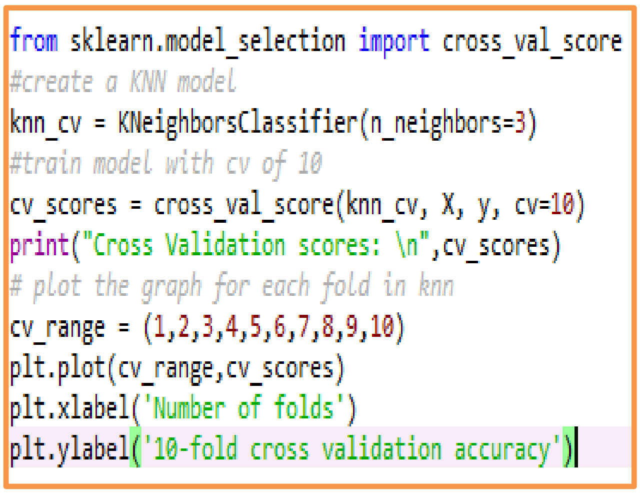
[2]: https://en.wikipedia.org/wiki/Support-vector_machine

[3]: https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72

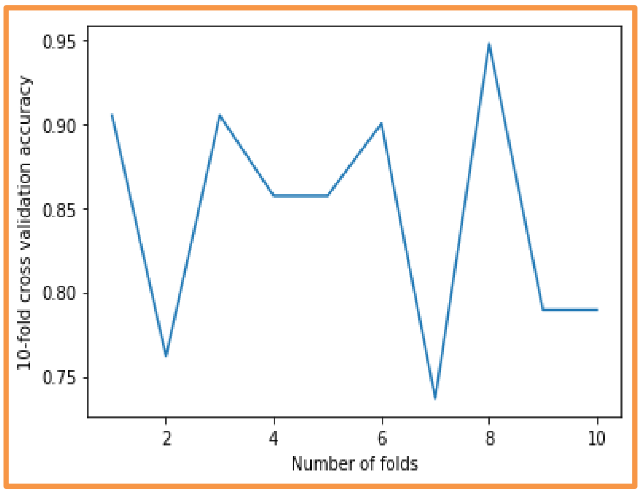
10-fold cross-validation:

Algorithm 1: K Nearest Neighbors (k-NN)

- To perform 10-fold cross validation, the value of k is taken as 3
- For each value fold in 10-fold cross validation, and the prediction accuracy is stored in list.
- Below code shows the implementation of 10-fold cross validation and the corresponding graph.
- Implementation takes the complete data prepared in previous step.



```
from sklearn.model_selection import cross_val_score
#create a KNN model
knn_cv = KNeighborsClassifier(n_neighbors=3)
#train model with cv of 10
cv_scores = cross_val_score(knn_cv, X, y, cv=10)
print("Cross Validation scores: \n",cv_scores)
# plot the graph for each fold in knn
cv_range = (1,2,3,4,5,6,7,8,9,10)
plt.plot(cv_range,cv_scores)
plt.xlabel('Number of folds')
plt.ylabel('10-fold cross validation accuracy')
```



10-fold Cross Validation score for k=3:

```
print("Cross Validation scores: \n",cv_scores)
```

```
Cross Validation scores:
 [0.9047619  0.76190476 0.9047619  0.85714286 0.85714286 0.9
 0.73684211 0.94736842 0.78947368 0.78947368]
```

By taking the mean of 10-fold Cross Validation score, accuracy of the model can be measured as below:

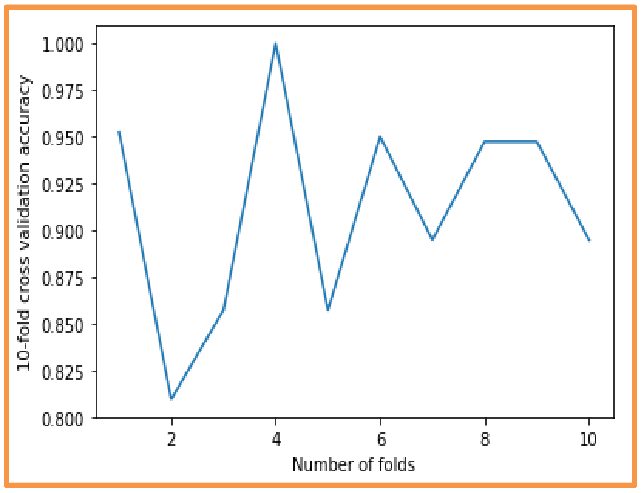
```
mean = format(np.mean(cv_scores))
mean = float(mean)*int(100)
print("Accuracy of the model after 10-fold cross validation: %0.2f"%mean,"%")
```

```
Accuracy of the model after 10-fold cross validation: 84.49 %
```

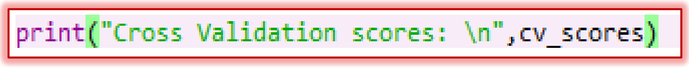
Algorithm 2: Support Vector Machines (SVM)

- To perform 10-fold cross validation, **linear kernel** is used.
- A penalty parameter is introduced for the error term
- Below code shows the implementation of 10-fold cross validation and the corresponding graph.
- Implementation takes the complete data prepared in previous step.

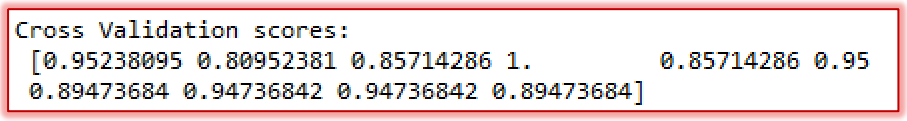
```
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
# select kernel as linear with penalty parameter C
clf = SVC(kernel='linear', C=1)
# train model with cv of 10
cv_scores = cross_val_score(clf, X, y, cv=10)
# plot the graph for each fold in knn
cv_range = (1,2,3,4,5,6,7,8,9,10)
plt.plot(cv_range,cv_scores)
plt.xlabel('Number of folds')
plt.ylabel('10-fold cross validation accuracy')
```



10-fold Cross Validation score:

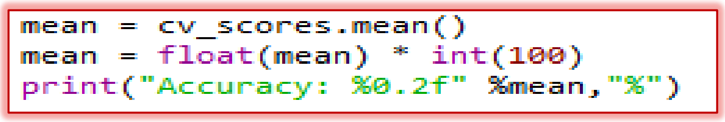


```
print("Cross Validation scores: \n",cv_scores)
```

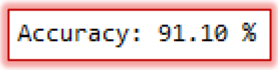


```
Cross Validation scores:
 [0.95238095 0.80952381 0.85714286 1.         0.85714286 0.95
 0.89473684 0.94736842 0.94736842 0.89473684]
```

By taking the mean of 10-fold Cross Validation score, accuracy of the model can be measured as below:



```
mean = cv_scores.mean()
mean = float(mean) * int(100)
print("Accuracy: %0.2f" %mean,"%")
```



```
Accuracy: 91.10 %
```

Conclusion:

*The results and observations show that **SVMs are a more reliable classifier** than k-NN.*

Algorithm	Training data	Accuracy
k-NN	201	84..45
SVM	201	91.10

- *KNN is less computationally intensive than SVM.*
- *Since, k-NN is easy to implement, the classification of Multi-class data should be done with k-NN.*
- *If data is homogenous to look at, one might be able to classify better by putting in a kernel into the SVM.*
- *For most practical problems, KNN is a bad choice because it scales badly, if there are a million labelled examples, it would take a long time (linear to the number of examples) to find K nearest neighbours*