K-Nearest Neighbors: Classification and Regression.

Initially KNN is a non-parametric classification method developed by Evelyn Fix and Joseph Hodges in 1951, and later expanded by Thomas Cover. It is used for classification and regression. In both the cases input consists of the K closest training examples in data set. The output depends on whether K-NN is used for classification or regression:

- In classification, the output is a class membership(discrete value). An object is classified by the majority vote that is to the class which most of its neighbors belongs.
- In regression, the output is a property value (continuous value). This value is the average of the values of K nearest neighbors.
- The neighbors are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

If the features are in different scales, then normalizing the training data can improve its accuracy.

KNN Classification Algorithm :

Given a training set X_train with labels y_train, and given a new instance x-test to be **classified**:

- Find the most similar instances (let's call them X_NN(Nearest Neighbors)) to the x_test that are in X-train.
- Get the labels y_NN for the instances in X_NN.
- Predict the labels for x_test by combining the labels y_NN e.g. simple majority vote.

k-NN regression Algorithm :

In k-NN regression, the k-NN algorithm is used for estimating continuous variables. One such algorithm uses a weighted average of the k nearest neighbors, weighted by the inverse of their distance. This algorithm works as follows:

- Compute the Euclidean distance from the query example(x-test) to the labeled examples(X_train).
- Order the labeled examples by increasing distance.
- Find a heuristically optimal number k of nearest neighbors, based on RMSE. This is done using cross validation.
- Calculate an inverse distance weighted average with the k-nearest multivariate neighbors.

Distance Metric used:

- For continuous variable → Euclidean distance.
- For discrete variables → overlap or hamming distance, Large Margin Nearest Neighbor.

Cons:

- A peculiarity of the k-NN algorithm is that it is sensitive to the local structure of the data.
- When the class distribution is skewed, that is examples of more frequent class tend to dominate the prediction.

  Solution for skewed distribution is to weight the classification, considering the distance(d) from test point to each of the k nearest neighbors. The class(value in regression) for each of the KNN is multiplied by the weight proportional to the inverse of the distance(1/d) from that point to test point (x_test), another way is by abstraction in data representation(self-organizing map).

Note :

In binary (two class) classification problems, it is helpful to choose k to be an odd number as this avoids tied votes. One popular way of choosing the empirically optimal k in this setting is via bootstrap method

Resource link : https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

Sparse data:

A variable with sparse data is one in which a relatively high percentage of the variable's cells do **not** contain actual data. Such "**empty**," or **NA**, values take up storage space in the file.

Classification margin :

Defined as the maximum width the decision boundary area can be increased before hitting a data point.

Maximum margin classifier :

The linear classifier with maximum margin is a linear SVM.

SVM : Support Vector Machine.

Sklearn.svm import SVC,

C regularised parameter

Kernelized SVM:

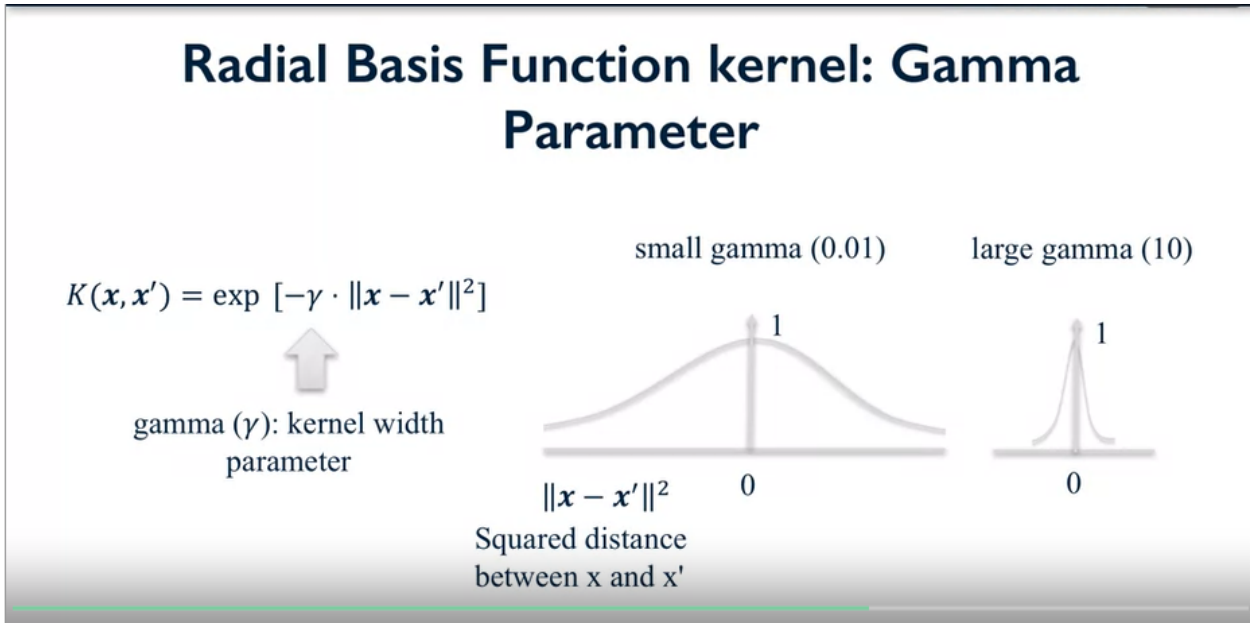
What if data is not linearly classified, then kernelized SVMs are used.

Transforms data to high dimensional data, so it is easy for linear classifiers to classify the data.

Radial Basis Functional Kernel: **RBF**

```
from sklearn.svm import SVC
from adspy_shared_utilities import plot_class_regions_for_classifier

X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2,
                                                    random_state = 0)

plot_class_regions_for_classifier(SVC().fit(X_train, y_train),
                                  X_train, y_train, None, None,
                                  'Support Vector Classifier: RBF kernel')

plot_class_regions_for_classifier(SVC(kernel = 'poly', degree = 3)
                                  .fit(X_train, y_train), X_train,
                                  y_train, None, None,
                                  'Support Vector Classifier: \
Polynomial kernel, degree = 3')
```

In RBF kernel we have a parameter gama:



Sklearn code for gama.

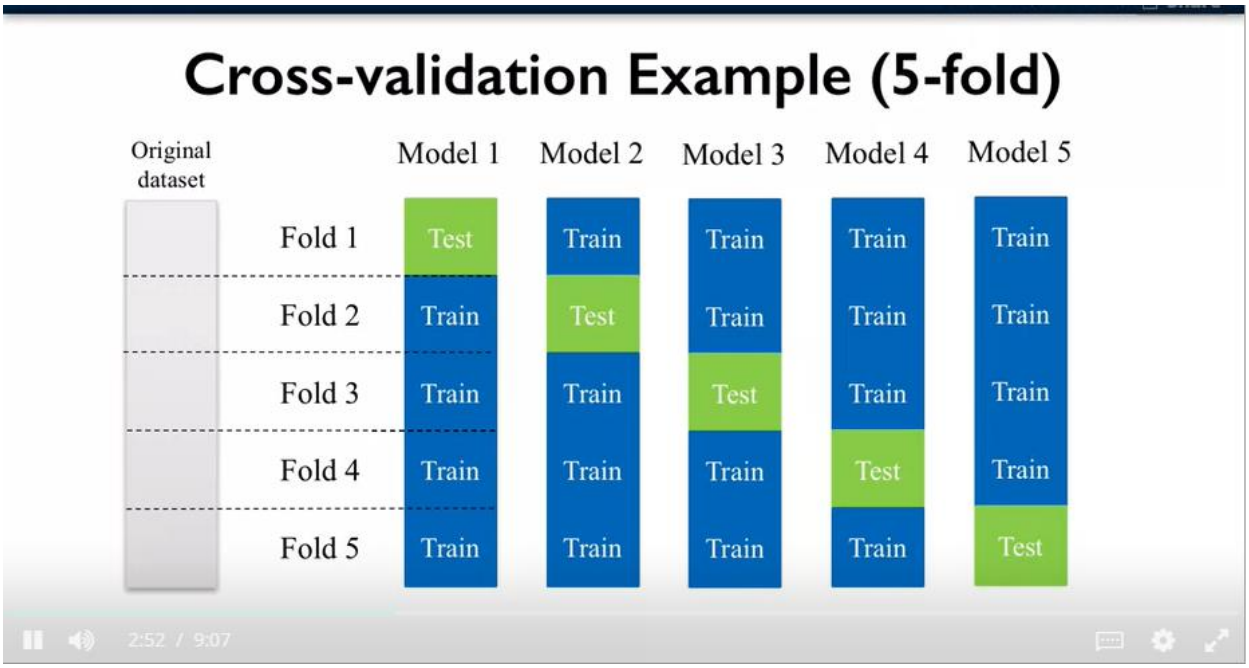
Support Vector Machine with RBF kernel: gamma parameter

```
from adspy_shared_utilities import plot_class_regions_for_classifier

X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2,
                                                    random_state = 0)
fig, subaxes = plt.subplots(1, 3, figsize=(11, 4))

for this_gamma, subplot in zip([0.01, 1.0, 10.0], subaxes):
    clf = SVC(kernel = 'rbf', gamma=this_gamma).fit(X_train, y_train)
    title = 'Support Vector Classifier: \n\
RBF kernel, gamma = {:.2f}'.format(this_gamma)
    plot_class_regions_for_classifier_subplot(clf, X_train, y_train,
                                              None, None, title, subplot)
    plt.tight_layout()
```

Inputs must be normalized to use SVM's.

Cross-Validation : Where we split data into multiple train and test datasets.

Most commonly used is → K-Fold , where we portioned data into k parts , we will have k accuracies values.



Implementation in Sklearn:

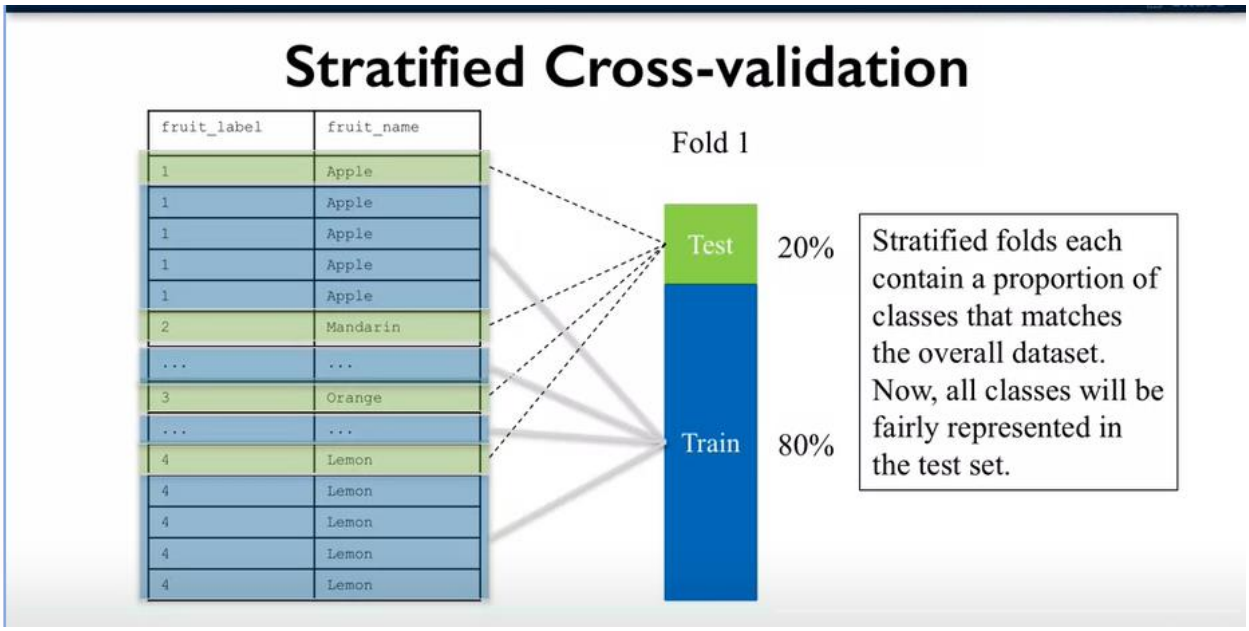
Cross-validation

Example based on k-NN classifier with fruit dataset (2 features)

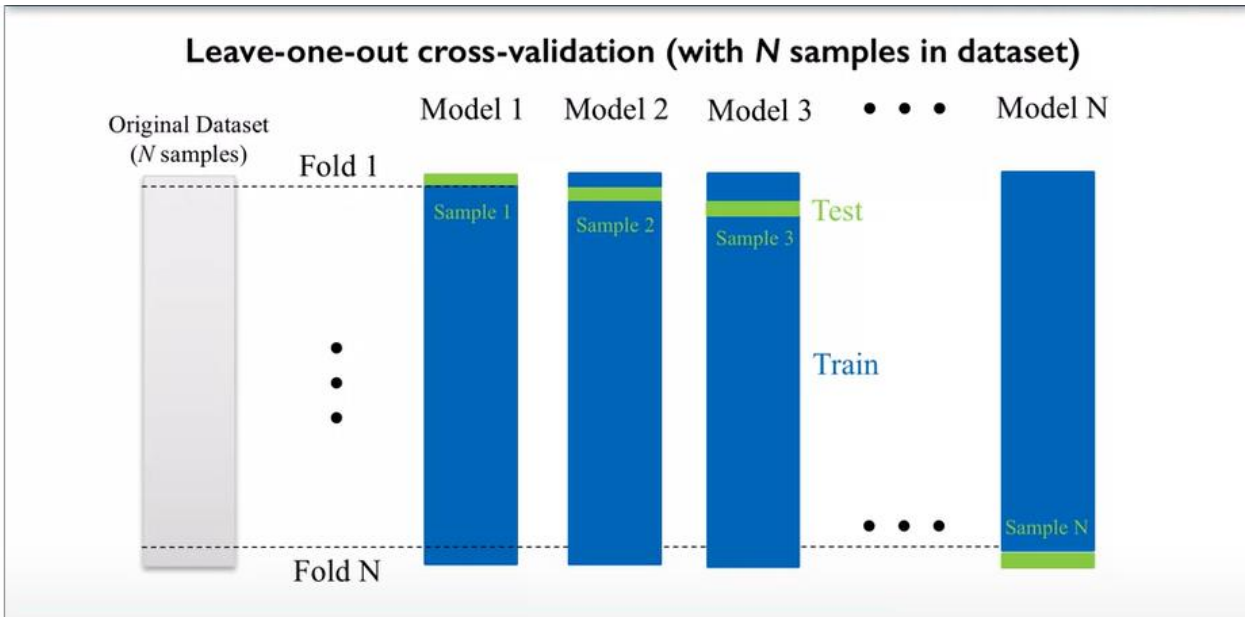
In [30]:
```
from sklearn.model_selection import cross_val_score

clf = KNeighborsClassifier(n_neighbors = 5)
X = X_fruits_2d.as_matrix()
y = y_fruits_2d.as_matrix()
cv_scores = cross_val_score(clf, X, y)

print('Cross-validation scores (3-fold):', cv_scores)
print('Mean cross-validation score (3-fold): {:.3f}'
     .format(np.mean(cv_scores)))
```

```
Cross-validation scores (3-fold): [ 0.77  0.74  0.83]
Mean cross-validation score (3-fold): 0.781
```

Stratified The proportions of classes in each fold made proportional to the classes in data set.



Leave-one-out cross validation : Each fold consists of single sample as test data.

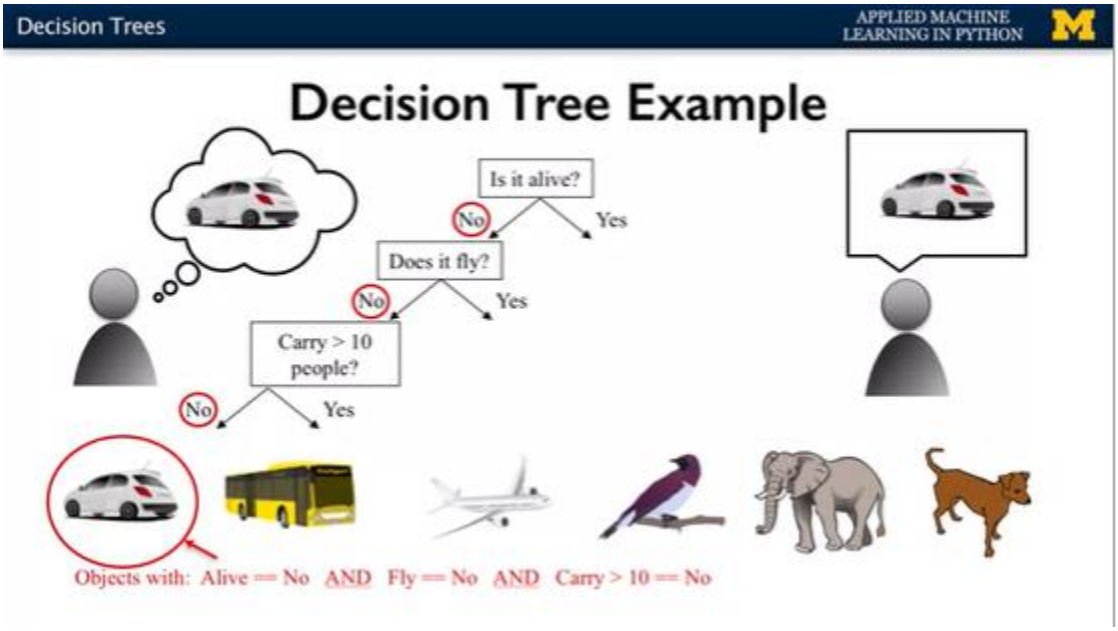


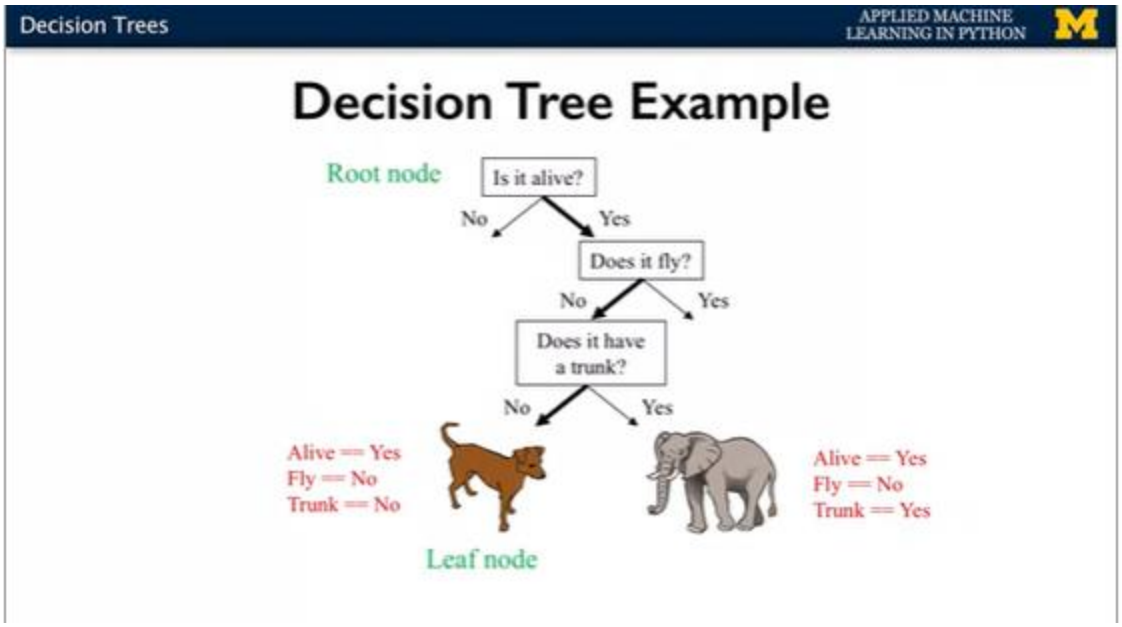
Grid Search to tune the parameters. Cross validation is only to evaluate the model.

Decision Tree Algorithm :

Used for both Regression and Classification, often used in exploratory data analysis(to find the important features in predicting output).

Uses if-then rules.

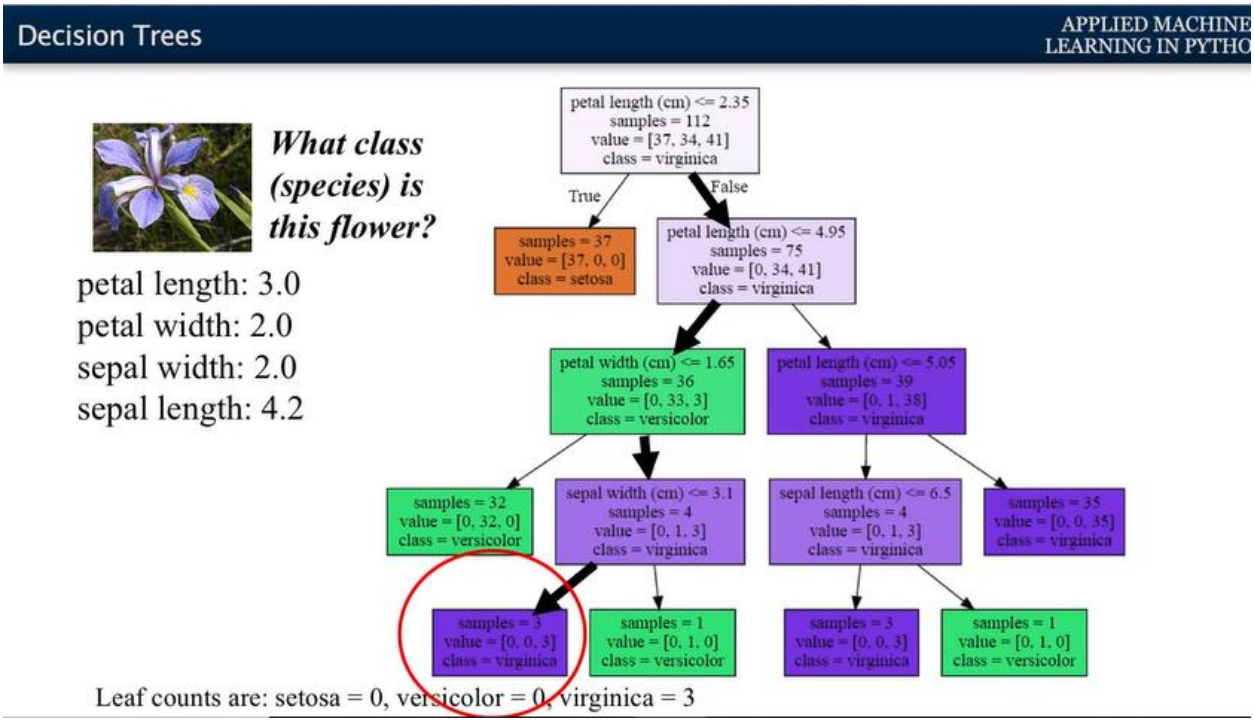




A dog is alive and doesn't fly and doesn't have trunk.

Example dataset : Iris. → to predict what species of Iris flower, we have 150 examples with 50 each for each sample.

Informativeness of splits: for best split the result should be homogeneous to yield better results.



For regression the

Implementation in sklearn:

Decision Trees

In [5]:
```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from adspy_shared_utilities import plot_decision_tree
from sklearn.model_selection import train_test_split


iris = load_iris()

X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
                                                    random_state = 3)
clf = DecisionTreeClassifier().fit(X_train, y_train)

print('Accuracy of Decision Tree classifier on training set: {:.2f}'
     .format(clf.score(X_train, y_train)))
print('Accuracy of Decision Tree classifier on test set: {:.2f}'
     .format(clf.score(X_test, y_test)))
```

```
Accuracy of Decision Tree classifier on training set: 1.00
Accuracy of Decision Tree classifier on test set: 0.97
```

It will cause overfitting:

Pre-pruning→So we need to stop the growth early before it becomes too detailed

Pruning → We allow for full detailed leaves and then we prune back to simpler form.

SK-learn only implements pre-pruning. Using max_depth or max_leaf parameter, or min samples leaf parameter.

```
clf2 = DecisionTreeClassifier(max_depth = 3).fit(X_train, y_train)

print('Accuracy of Decision Tree classifier on training set: {:.2f}'
     .format(clf2.score(X_train, y_train)))
print('Accuracy of Decision Tree classifier on test set: {:.2f}'
     .format(clf2.score(X_test, y_test)))
```

```
Accuracy of Decision Tree classifier on training set: 0.98
Accuracy of Decision Tree classifier on test set: 0.97
```

TO visualize →

```
plot_decision_tree(clf, iris.feature_names, iris.target_names)
```

Feature Importance → value between 0 to 1

```
from adspy_shared_utilities import plot_feature_importances

plt.figure(figsize=(10,4))
plot_feature_importances(clf, iris.feature_names)
plt.show()

print('Feature importances: {}'
      .format(clf.feature_importances_))
```

Figure 7

Decision Trees on a real-world dataset

```
from sklearn.tree import DecisionTreeClassifier
from adspy_shared_utilities import plot_decision_tree
from adspy_shared_utilities import plot_feature_importances

X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer,
                                                    random_state = 0)

clf = DecisionTreeClassifier(max_depth = 4, min_samples_leaf = 8,
                             random_state = 0).fit(X_train, y_train)

plot_decision_tree(clf, cancer.feature_names, cancer.target_names)
```

Decision Trees: Pros and Cons

Pros:

- Easily visualized and interpreted.
- No feature normalization or scaling typically needed.
- Work well with datasets using a mixture of feature types (continuous, categorical, binary)

Cons:

- Even after tuning, decision trees can often still overfit.
- Usually need an ensemble of trees for better generalization performance.