DAAN862: Analytics Programming in Python

Lesson 10: Supervised Learning with SciKit-Learn III: Advanced Models

Lesson 10: Supervised Learning with SciKit-Learn III: Advanced Models

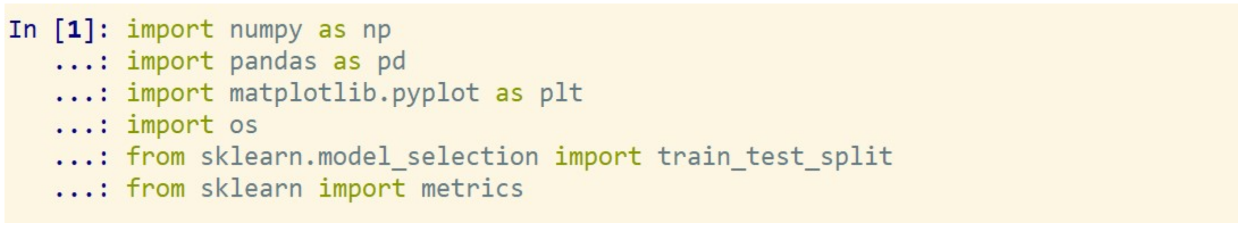
At this end of this lesson, students will use scikit-learn package to:

- Build and evaluate support vector machine models
- Build and evaluate Random Forest models
- Build and evaluate Adaboost models

By the end of this lesson, please complete all readings and assignments found in the Lesson 10 Course Schedule.

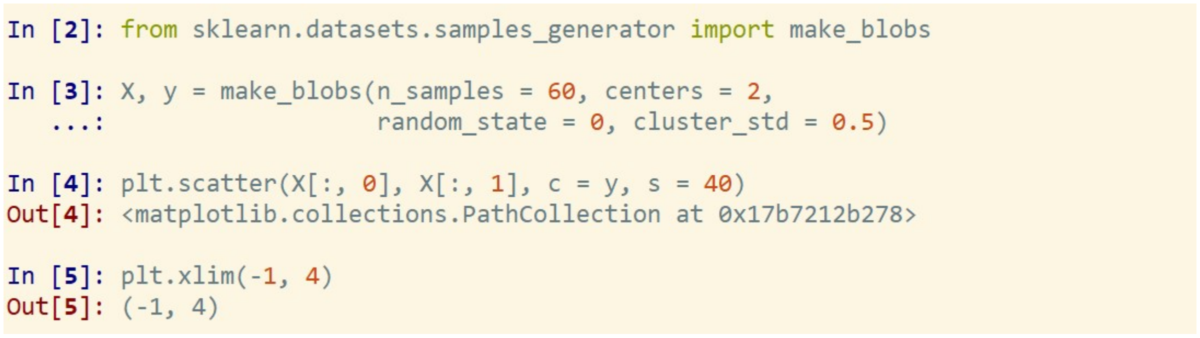
Lesson 10.1: Support Vector Machine

Before we introduce what support vector machines are, let's consider the artificial data created by make blobs in sklearn.datasets.sample generator. By using parameters n samples=60, centers=2,, and cluster std=0.5. It will use clustering method to generate two clusters with cluster standivation of 0.5 and 60 datapoints in total. Figure 10.1 A,B, and C shows you how to plot the data:



```
In [1]: import numpy as np
   ...: import pandas as pd
   ...: import matplotlib.pyplot as plt
   ...: import os
   ...: from sklearn.model_selection import train_test_split
   ...: from sklearn import metrics
```

Fig 10.1 (A) (click to enlarge)



```
In [2]: from sklearn.datasets.samples_generator import make_blobs

In [3]: X, y = make_blobs(n_samples = 60, centers = 2,
   ...:                   random_state = 0, cluster_std = 0.5)

In [4]: plt.scatter(X[:, 0], X[:, 1], c = y, s = 40)
Out[4]: <matplotlib.collections.PathCollection at 0x17b7212b278>

In [5]: plt.xlim(-1, 4)
Out[5]: (-1, 4)
```

Fig 10.1 (B) (click to enlarge)

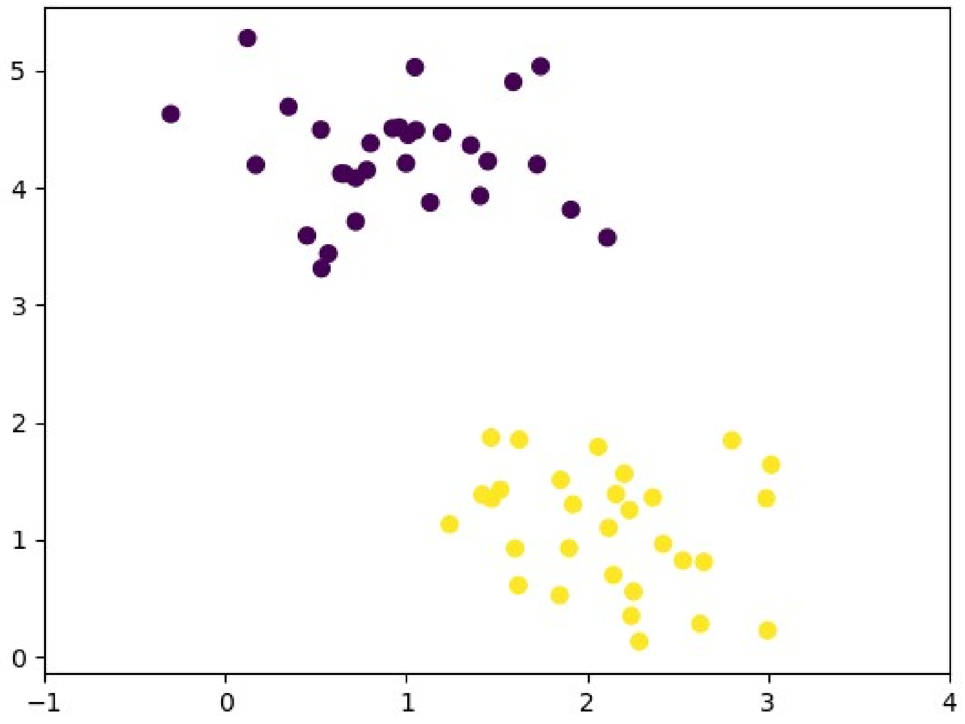
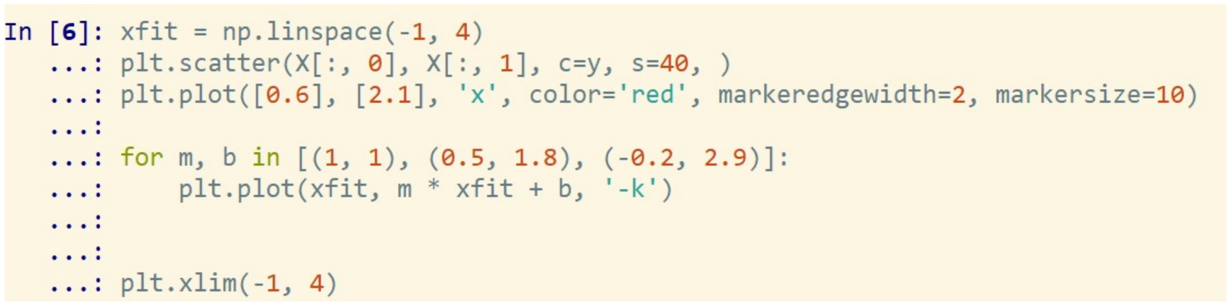


Fig 10.1 (C) (click to enlarge)

Typically, the linear classifier would draw a straight line to separate two classes. However, there is more than one possible solution for this task. For example, any black lines (as shown in Figure 10.2 B) could classify the data perfectly:



```
In [6]: xfit = np.linspace(-1, 4)
   ...: plt.scatter(X[:, 0], X[:, 1], c=y, s=40, )
   ...: plt.plot([0.6], [2.1], 'x', color='red', markeredgewidth=2, markersize=10)
   ...:
   ...: for m, b in [(1, 1), (0.5, 1.8), (-0.2, 2.9)]:
   ...:     plt.plot(xfit, m * xfit + b, '-k')
   ...:
   ...:
   ...: plt.xlim(-1, 4)
```

Fig 10.2 (A) (click to enlarge)

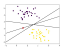
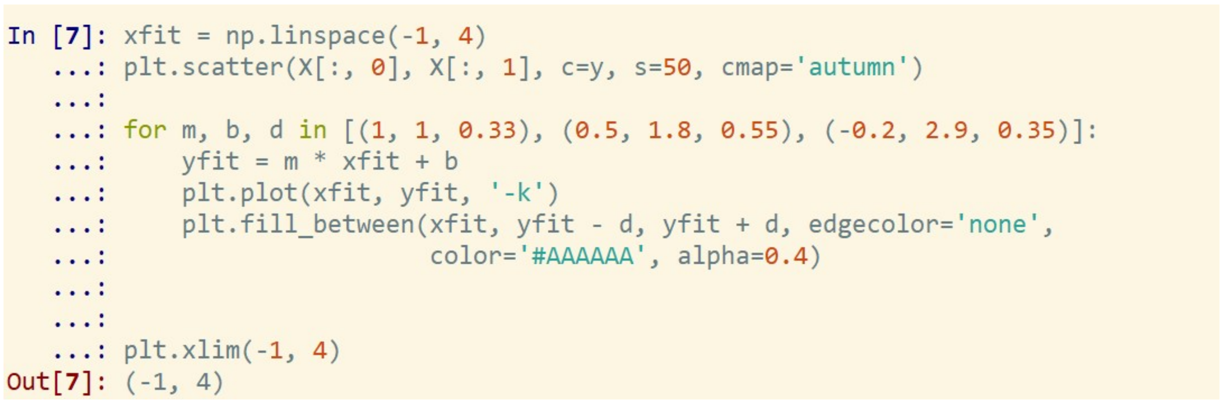


Fig 10.2 (B)

Support vector machines (SVM) provide a smart way to solve the dilemma. In addition to drawing the separation line, SVMs adds a margin of some width (as indicted by the gray area for each line shown in Figure 10.3 B) to the line. The margin should be maximized until it reaches the nearest data points. The middle line provides the largest margin, and that would be the choice for Support vector machines:



```
In [7]: xfit = np.linspace(-1, 4)
   ...: plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
   ...:
   ...: for m, b, d in [(1, 1, 0.33), (0.5, 1.8, 0.55), (-0.2, 2.9, 0.35)]:
   ...:     yfit = m * xfit + b
   ...:     plt.plot(xfit, yfit, '-k')
   ...:     plt.fill_between(xfit, yfit - d, yfit + d, edgecolor='none',
   ...:                      color='#AAAAAA', alpha=0.4)
   ...:
   ...:
   ...: plt.xlim(-1, 4)
Out[7]: (-1, 4)
```

Fig 10.3 (A) (click to enlarge)

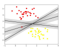


Fig 10.3 (B)

For the two dimensional data, a vector is used to separate different groups of data. For high-dimensional data it will become a hyperplane instead of a line.

What is Support Vector Machine?

Support vector machine is a supervised machine learning algorithm which tries to find an optimized hyper-plane which can maximize the margin. SVM is very powerful and flexible for both classification and regression tasks. In addition to using a linear hyperplane, you can use different kernel (https://en.wikipedia.org/wiki/Positive-definite_kernel) functions such as polynomial, radial basis function (RBF), sigmoid or predefined functions to define the hyperplane.

Transcript

In this video, I will introduce how to use Support vector machine for classification problems.

Since you haven't learned this algorithm, I will use an example to show how it works. Let’s import packages needed.

We also import make_blobs function from sklearn.datasets.samples_generator. This function is used to create artificial data. In this function, we use n_samples = 60 which means creating 60 data points. Centers = 2 means to create two clusters. Random_state = 0 and cluster_std means the cluster standard deviation for clusters.

Next, we use a scatter plot to visualize it. From the plot, we can see that we have two clusters: purple and yellow.

Next, we adding three line boundary to separate those two clusters. We can tell that both three lines do a great job in terms of accuracy. Now, the questions which one is the best?

Support vector machine introduces a concept called margin and it is trying to maximize the margin. Now let’s add the margin the three lines. The margin is defined by the closest points to the line. In this plot, we can tell that the middle line gives the largest margin, therefore it will be considered as the separation boundary for support vector machine module. Besides linear boundary, SVC also supports round and polynomial curves.

Let’s build SVC models for glass data and we try different kernel function for it. Let’s import data into Python.

Next, we import SVM from sklearn and we create learning model by using kernel = ‘linear’. Next, we can fit the model with X train, y_train.

You can check coefficients of each variable by coef_ attribute. If you check the dimension, it is 15 times 9. 9 means you have 9 variables. SVM performs one –verse one for multi-class problem. There is 6 categories in glass type. According to the combination, pick 2 from 6, you have 15 combinations, that's why there are 15 linear boundary will be built in the model.

Next, we try kernel equals RBF and gamma parameter is for kernel rbf. Here we randomly pick 0.1 for gamma. The accuracy for this model is 0.55.

The last kernel we try is polynomial, you can set up kernel equals poly. The degree parameter is particular for the polynominal kernel, and we use 2 here. The accuracy for test set is 0.63.

In reality, all parameters should be optimized to maximize the test or cross-validation accuracy.

Lesson 10.2: SVM Practice

We have introduced how SVM (https://pennstateoffice365.sharepoint.com/:w:/r/sites/DAAN862/_layouts/15/Doc.aspx?sourcedoc=%7BE89B53A3-8931-4EC1-95DC-06B237AB777A%7D&file=Lesson%2010-V3.docx&action=default&mobileredirect=true) works in Lesson 10.1. Now let's use SVM for classification problems in Python.

SVC (http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html) is the classification model in scikit-learn. The main parameters and attributes are listed in Table 10.2.1 and Table 10.2.2, respectively.

Table 10.2.1: Main Parameters of SVC Model

Parameters	Description
C	Penalty parameter C of the error term.
kernel	Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable.
degree	Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.
gamma	Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. If gamma is 'auto' then 1/n_features will be used instead.

Table 10.2.2: Main Attributes of SVC Model

Attributes	**Description**
support	Indices of support vectors.
support vectors	Support vectors.
n support	Number of support vectors for each class.
coef_	Weights assigned to the features (coefficients in the primal problem). This is only available in the case of a linear kernel
intercept_	Constants in decision function.

Figure 10.4 shows how we continue to use glass the glass dataset that was introduced in Lesson 8 as an example:

Lesson 10.4 (click to enlarge)

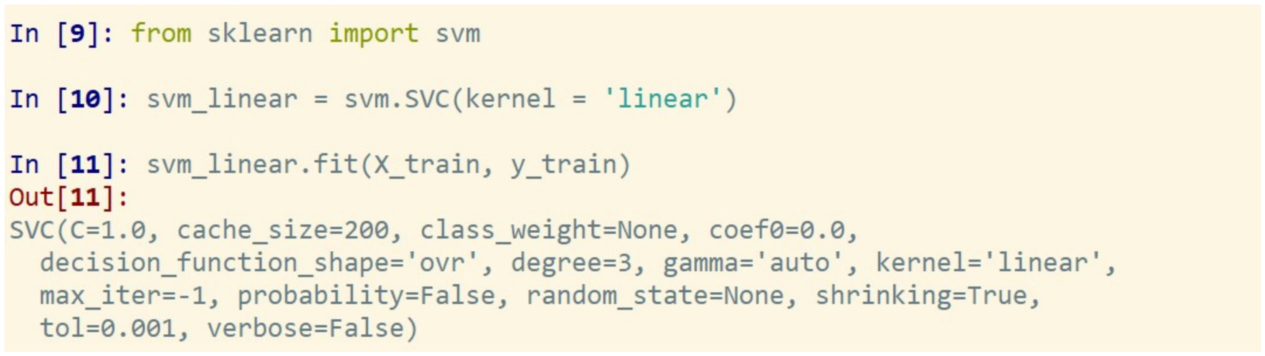
Next, you can review the results of using different kernel functions:

Kernel='linear', Kernel='rbf', and Kernel='poly':

(click tabs to learn more)

Kernel='linear':

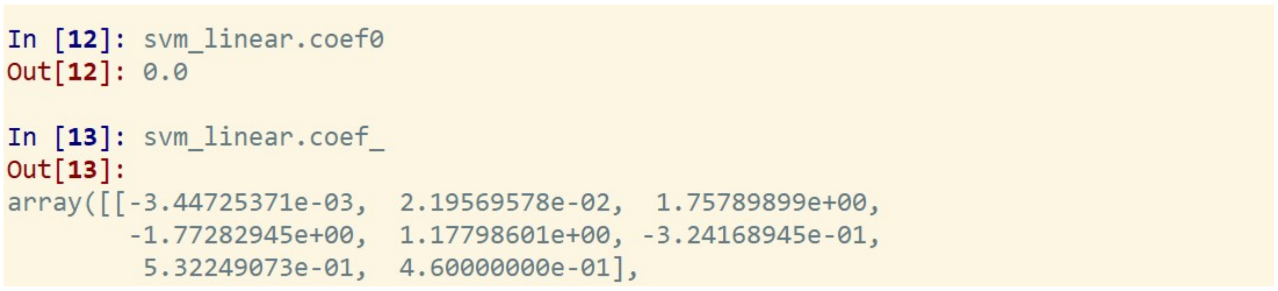
Figure 10.5 shows an example of kernel='linear':



```
In [9]: from sklearn import svm

In [10]: svm_linear = svm.SVC(kernel = 'linear')

In [11]: svm_linear.fit(X_train, y_train)
Out[11]:
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

Fig 10.5 (click to enlarge)

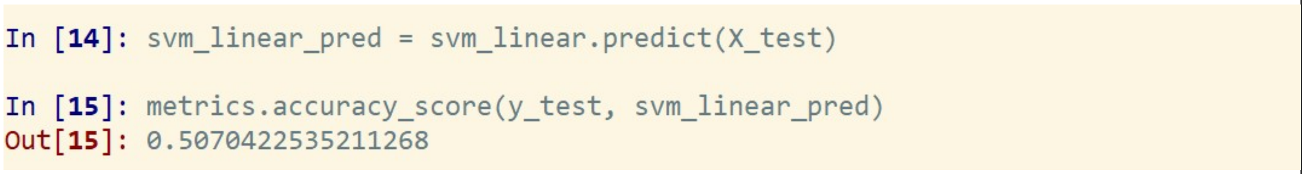
For the linear kernel, you can use *coef_* attribute to check the weights assigned to each feature as shown in Figure 10.6:



```
In [12]: svm_linear.coef0
Out[12]: 0.0

In [13]: svm_linear.coef_
Out[13]:
array([[-3.44725371e-03,  2.19569578e-02,  1.75789899e+00,
        -1.77282945e+00,  1.17798601e+00, -3.24168945e-01,
         5.32249073e-01,  4.60000000e-01],
```

Fig 10.6 (click to enlarge)

Next, Figure 10.7 shows how you can evaluate the model on the test set:

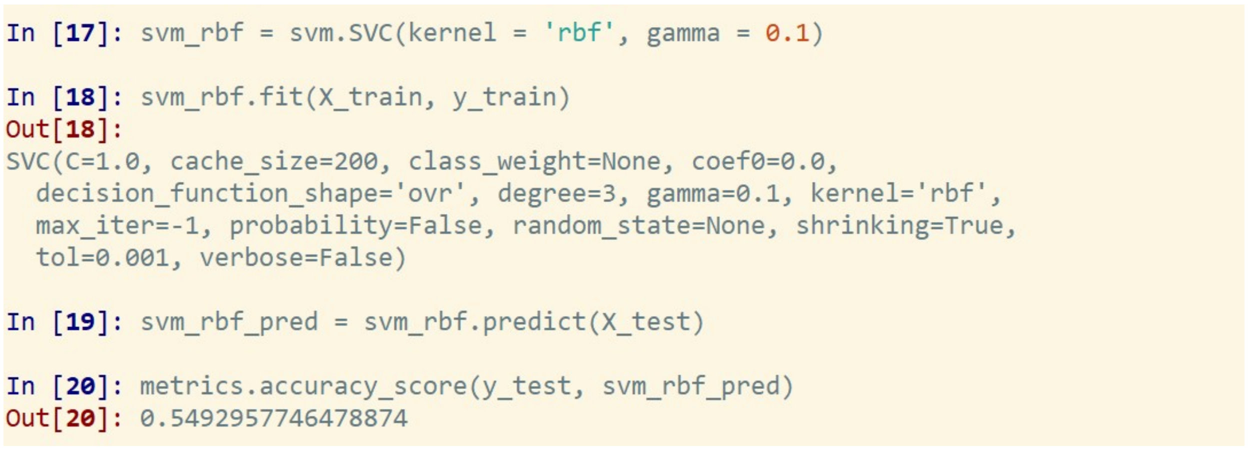


```
In [14]: svm_linear_pred = svm_linear.predict(X_test)

In [15]: metrics.accuracy_score(y_test, svm_linear_pred)
Out[15]: 0.5070422535211268
```

Fig 10.7 (click to enlarge)

Kernel='rbf':

Gamma is a special parameter for the RBF kernel, In Figure 10.8, we use gamma = 0.1:

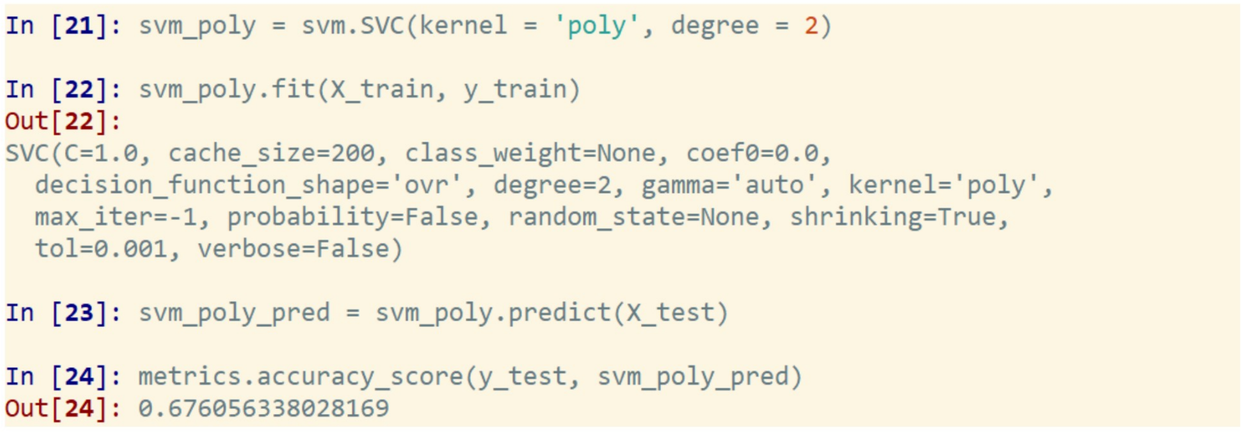


```
In [17]: svm_rbf = svm.SVC(kernel = 'rbf', gamma = 0.1)

In [18]: svm_rbf.fit(X_train, y_train)
Out[18]:
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

In [19]: svm_rbf_pred = svm_rbf.predict(X_test)

In [20]: metrics.accuracy_score(y_test, svm_rbf_pred)
Out[20]: 0.5492957746478874
```

Fig 10.8 (click to enlarge)

Kernel='poly':

Figure 10.9 shows the degree=2, which is the case of the quadratic kernel:



```
In [21]: svm_poly = svm.SVC(kernel = 'poly', degree = 2)

In [22]: svm_poly.fit(X_train, y_train)
Out[22]:
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=2, gamma='auto', kernel='poly',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

In [23]: svm_poly_pred = svm_poly.predict(X_test)

In [24]: metrics.accuracy_score(y_test, svm_poly_pred)
Out[24]: 0.676056338028169
```

Fig 10.9 (click to enlarge)

From the results, we can see that the quadratic kernel gives the best results. This is a simple demonstration of how each kernel works. You should use a grid-search to find out the best parameters.

Transcript

In this video, I will introduce how to use RandomForestClassifier.

First, let import it from sklearn.ensemble.

Building and evaluating models are exactly the same with other classifiers. The accuracy is 0.75.

You can print classification report. The model also gives feature importances, we can create a dataFame for features and their importance value.

Next, we study the relation between accuracy and numbers of estimators. The number of estimators means how many tree models in the forest.

n_estimator is a sequence from 2 to 98 with a spacing of 2.

We create an empty list to store accuracy.

For i in n_estimator, we will create a random forest classifier with n_estimator =i and scores is the result from cross-validation score function.

We append the mean score into accuracy.

Next, we could plot it. We can see that as increasing the number of estimators, the accuracy increases too. But at some point, it cannot be increased any more. For this data, n_estorimator could be chosen from 30 -40. Since you choose a larger n_estimators, the model couldn't improve but you are using more computation power.

Lesson 10.3: Ensemble Methods (Random Forests)

RandomForestClassifier (http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html) is the Random Forest model in the scikit-learn package. Aside from the fact that it builds random forests using multiple trees instead of one, RandomForestClassifier differs from decision trees in two ways:

Each tree is built from re-sampled data with replacement from the training set.

The best-split feature at each node is selected from a random subset of the features.

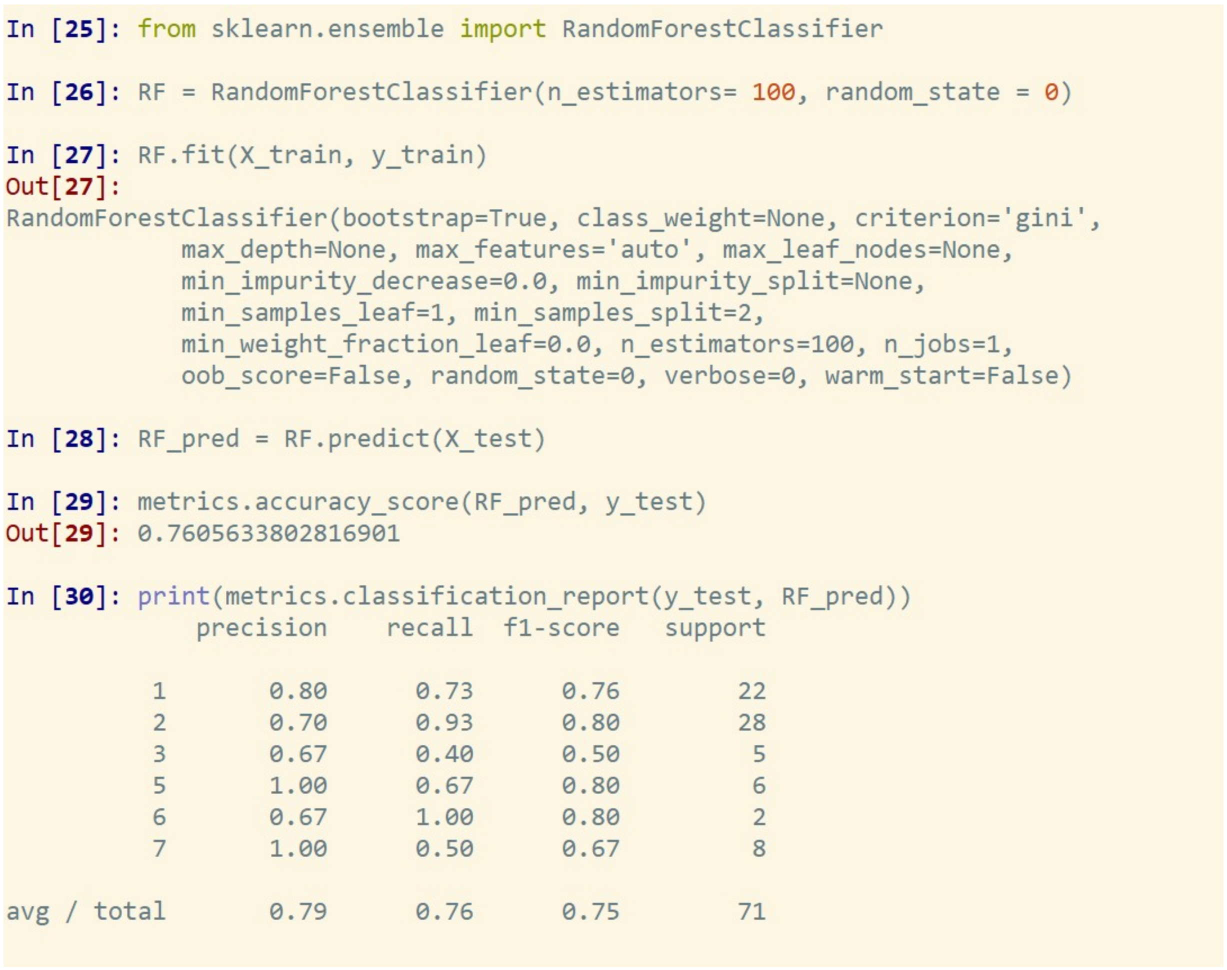
Below, table 10.3.1 lists the parameters:

Table 10.3.1: Main Parameters of RandomForestClassifier

Parameters	**Description**
n estimators	The number of trees in the forest.
criterion	The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

Parameters	Description
max features	The number of features to consider when looking for the best split.
min samples split	The minimum number of samples required to split an internal node.

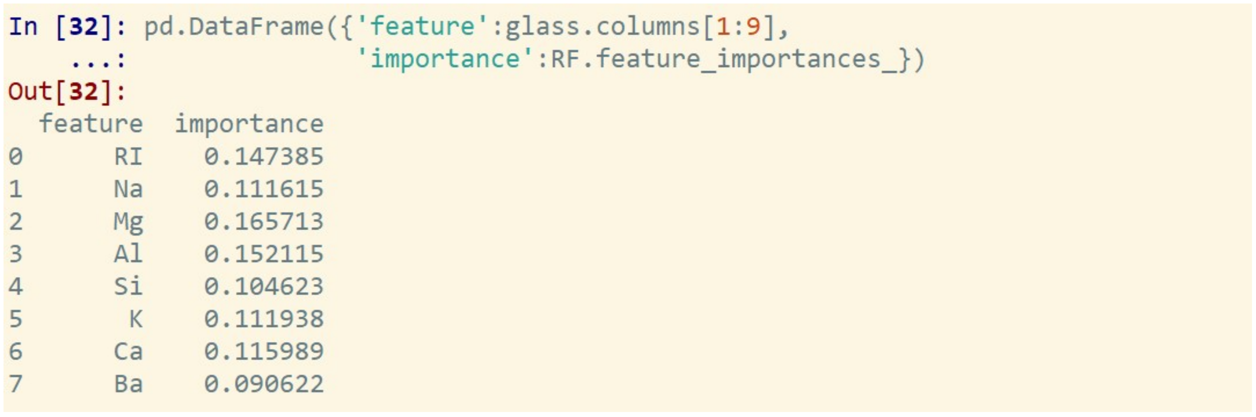
Figure 10.10 shows an example of building classification model with glass dataset using n estimators =100:



```
In [25]: from sklearn.ensemble import RandomForestClassifier

In [26]: RF = RandomForestClassifier(n_estimators= 100, random_state = 0)

In [27]: RF.fit(X_train, y_train)
Out[27]:
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
            oob_score=False, random_state=0, verbose=0, warm_start=False)

In [28]: RF_pred = RF.predict(X_test)

In [29]: metrics.accuracy_score(RF_pred, y_test)
Out[29]: 0.7605633802816901

In [30]: print(metrics.classification_report(y_test, RF_pred))
             precision    recall  f1-score   support

          1       0.80      0.73      0.76        22
          2       0.70      0.93      0.80        28
          3       0.67      0.40      0.50         5
          5       1.00      0.67      0.80         6
          6       0.67      1.00      0.80         2
          7       1.00      0.50      0.67         8

avg / total       0.79      0.76      0.75        71
```

Fig 10.10 (click to enlarge)

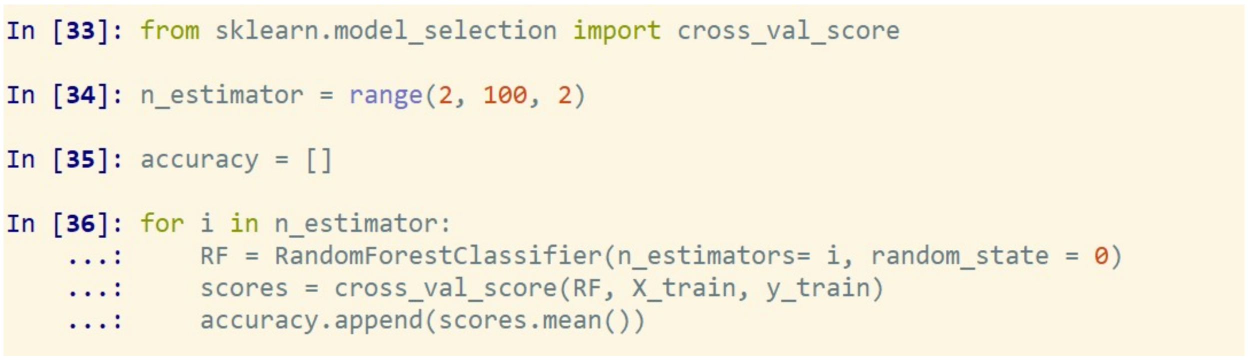
The attribute *feature_importance_* (shown in Figure 10.11) show results of the feature importance score. We use a DataFrame to show variable names and their importance scores at the same time:



```
In [32]: pd.DataFrame({'feature':glass.columns[1:9],
    ...:                'importance':RF.feature_importances_})
Out[32]:
  feature  importance
0      RI    0.147385
1      Na    0.111615
2      Mg    0.165713
3      Al    0.152115
4      Si    0.104623
5       K    0.111938
6      Ca    0.115989
7      Ba    0.090622
```

Fig 10.11 (click to enlarge)

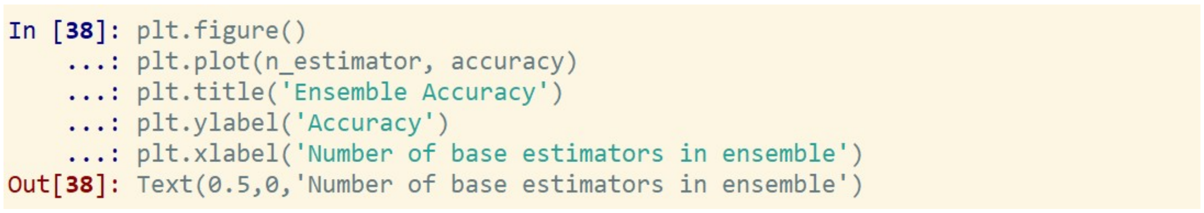
The most important parameter for Random forests is n_estimator, which is the number of trees in the forest. Let's study the relation between the accuracy and n_estimator. In the following codes, we use 2, 4, 6, … 98 as n_estimators and save the *cross_val_score* for each model n_estimators in the variable *accuracy* shown in Figure 10.12:



```
In [33]: from sklearn.model_selection import cross_val_score

In [34]: n_estimator = range(2, 100, 2)

In [35]: accuracy = []

In [36]: for i in n_estimator:
    ...:     RF = RandomForestClassifier(n_estimators= i, random_state = 0)
    ...:     scores = cross_val_score(RF, X_train, y_train)
    ...:     accuracy.append(scores.mean())
```

Fig 10.12 (click to enlarge)

Next, we will plot the accuracy vs. n estimators as shown in Figure 10.13 A&B:



```
In [38]: plt.figure()
    ...: plt.plot(n_estimator, accuracy)
    ...: plt.title('Ensemble Accuracy')
    ...: plt.ylabel('Accuracy')
    ...: plt.xlabel('Number of base estimators in ensemble')
Out[38]: Text(0.5,0,'Number of base estimators in ensemble')
```

Fig 10.13 (A) (click to enlarge)

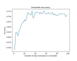


Fig 10.13 (B)

From this plot, we can see that accuracy increases with *n_estimators* and gradually reach an upper limit. The best *n_estimators* would be where the accuracy just reaches the upper limit. (as indicated by the red arrow).

Transcript

In this video, I will introduce how to use AdaBoostClassifier.

First, let import it from sklearn.ensemble.

Building and evaluating model. The test accuracy is 0.45.

For this model, we also study the relationship between accuracy and numbers of estimators. The number of estimators means how many tree models in the forest.

n_estimator is a sequence from 1 to 50 with a spacing of 1.

We create an empty list to store accuracy.

For i in n_estimator, we will create an AdaBoostclassifier with n_estimator =i and scores is the result from cross-validation score function.

We append the mean score into accuracy.

Next, we could plot it. We can see that as increasing the number of estimators, the accuracy increases too. But at some point, it cannot be increased any more. For this data, n_estorimator could be chosen from 20 -30. Since you choose a larger n_estimators, the model couldn't improve but you are using more computation power.

Lesson 10.4: Ensemble Methods (AdaBoost)

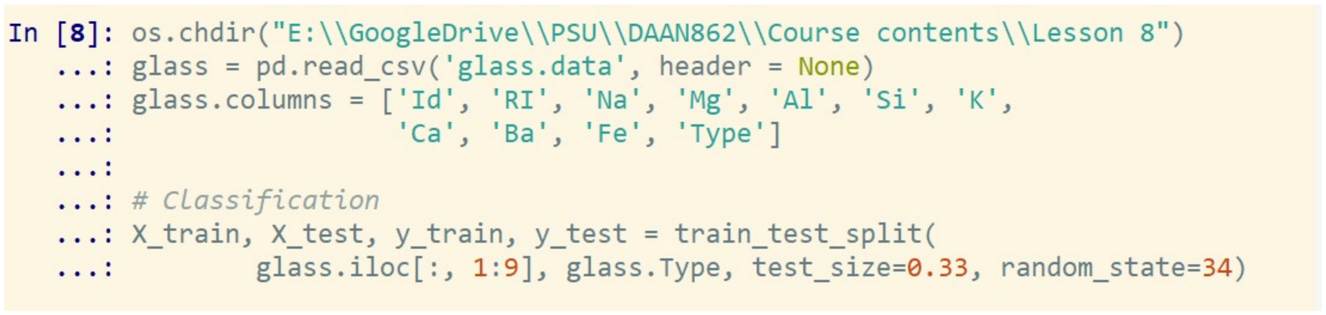
Another popular ensemble method is AdaBoost. Its main principle is to assign equal weight for each training data points, and the weight will be increased for incorrectly predicted data points and decreased for correctly predicted ones in each iteration. Therefore, the model will be forced to focus on the misclassified data points.

AdaBoostClassifier is the Adaboost model in scikit-learn. The main parameters are listed in Table 10.3.1.

Table 10.3.1: The Main Parameters of AdaBoostClassifier

Parameters	**Description**
base estimator	The base estimator from which the boosted ensemble is built. Support for sample weighting is required.
n estimators	The maximum number of estimators at which boosting is terminated.
learning rate	Learning rate shrinks the contribution of each classifier by learning_rate

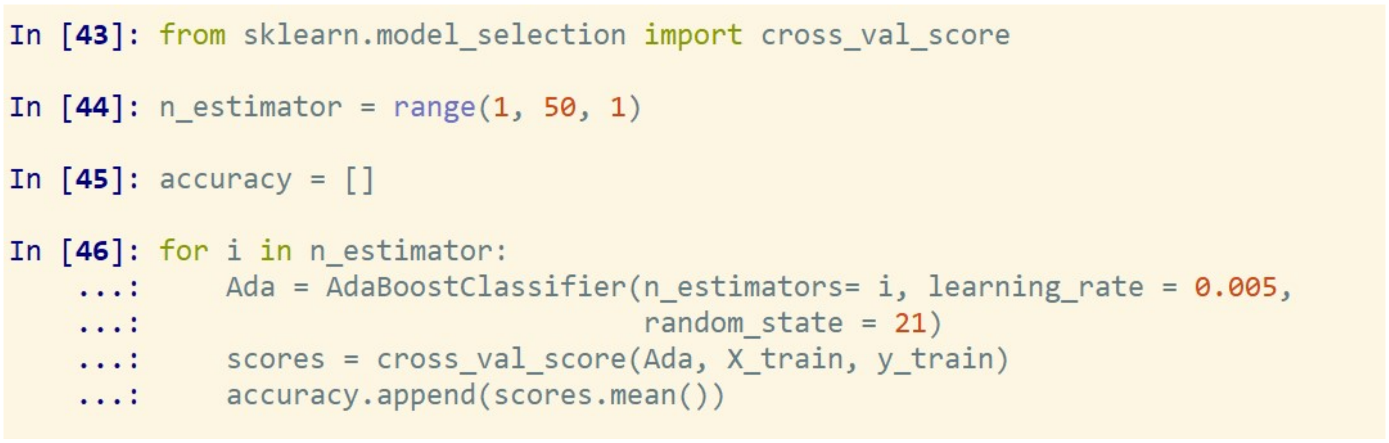
Figure 10.14 is the classification result of using AdaBoostClassifier to the glass dataset, using n estimators=20:



```
In [8]: os.chdir("E:\\GoogleDrive\\PSU\\DAAN862\\Course contents\\Lesson 8")
   ...: glass = pd.read_csv('glass.data', header = None)
   ...: glass.columns = ['Id', 'RI', 'Na', 'Mg', 'Al', 'Si', 'K',
   ...:                  'Ca', 'Ba', 'Fe', 'Type']
   ...:
   ...: # Classification
   ...: X_train, X_test, y_train, y_test = train_test_split(
   ...:         glass.iloc[:, 1:9], glass.Type, test_size=0.33, random_state=34)
```

Fig 10.14 (click to enlarge)

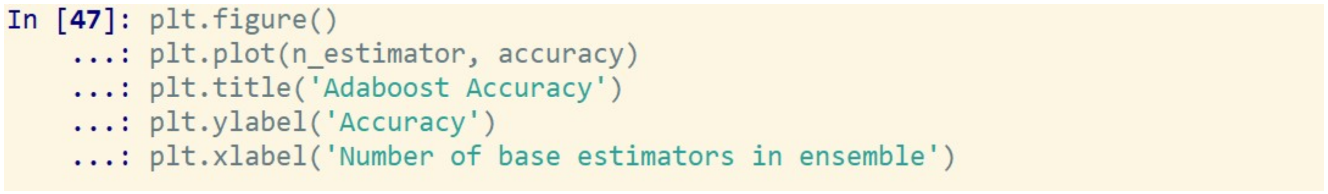
The most important parameter for *AdaBoostClassifer* is n_estimators, which is the number of trees in the forest. Let's study the relation between the accuracy and n_estimator. In the following codes, we use 1, 2, 3..., 49 as n_estimators and save the cross-validation score (shown in Figure 10.15) for each model with each n_estimators in the variable *accuracy:*



```
In [43]: from sklearn.model_selection import cross_val_score

In [44]: n_estimator = range(1, 50, 1)

In [45]: accuracy = []

In [46]: for i in n_estimator:
    ...:     Ada = AdaBoostClassifier(n_estimators= i, learning_rate = 0.005,
    ...:                              random_state = 21)
    ...:     scores = cross_val_score(Ada, X_train, y_train)
    ...:     accuracy.append(scores.mean())
```

Fig 10.15 (click to enlarge)

Now, we can plot the accuracy vs. n estimators shown in Figure 10.16 A&B:



```
In [47]: plt.figure()
    ...: plt.plot(n_estimator, accuracy)
    ...: plt.title('Adaboost Accuracy')
    ...: plt.ylabel('Accuracy')
    ...: plt.xlabel('Number of base estimators in ensemble')
```

Fig 10.16 (A) (click to enlarge)

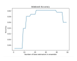


Fig 10.16 (B)

Lesson 10 References (6 of 6)

Lesson 10 References

- http://scikit-learn.org/stable/index.html (http://scikit-learn.org/stable/index.html)
- Python Data Science Handbook, Jake VanderPlas https://jakevdp.github.io/PythonDataScienceHandbook/index.html (https://jakevdp.github.io/PythonDataScienceHandbook/index.html)