```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O
import matplotlib.pyplot as plt
%pylab inline

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in

from subprocess import check_output
print(check_output(["ls", "/content/Iris.csv"]).decode("utf8"))
```

```
Populating the interactive namespace from numpy and matplotlib
/content/Iris.csv
```

```
#Reading data from CSV file
df = pd.read_csv("../content/Iris.csv")


#Defining data and label
X = df.iloc[:, 1:5]
y = df.iloc[:, 5]


#Split data into training and test datasets (training will be based on 70% of data)
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
#test_size: if integer, number of examples into test dataset; if between 0.0 and 1.0, means p
print('There are {} samples in the training set and {} samples in the test set'.format(X_trai
```

```
There are 105 samples in the training set and 45 samples in the test set
```

```
#Scaling data
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score

sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

#X_train_std and X_test_std are the scaled datasets to be used in algorithms


#Applying SVC (Support Vector Classification)
from sklearn.svm import SVC

svm = SVC(kernel='rbf', random_state=0, gamma=.10, C=1.0)
svm.fit(X_train_std, y_train)
```

```
print('The accuracy of the SVM classifier on training data is {:.2f}'.format(svm.score(X_trai
print('The accuracy of the SVM classifier on test data is {:.2f}'.format(svm.score(X_test_std
```

```
The accuracy of the SVM classifier on training data is 0.97
The accuracy of the SVM classifier on test data is 0.98
```

```
#Applying Knn
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors = 7, p = 2, metric='minkowski')
knn.fit(X_train_std, y_train)

print('The accuracy of the Knn classifier on training data is {:.2f}'.format(knn.score(X_trai
print('The accuracy of the Knn classifier on test data is {:.2f}'.format(knn.score(X_test_std
```

```
The accuracy of the Knn classifier on training data is 0.97
The accuracy of the Knn classifier on test data is 0.98
```

```
#Applying XGBoost
import xgboost as xgb

xgb_clf = xgb.XGBClassifier()
xgb_clf = xgb_clf.fit(X_train_std, y_train)

print('The accuracy of the XGBoost classifier on training data is {:.2f}'.format(xgb_clf.scor
print('The accuracy of the XGBoost classifier on test data is {:.2f}'.format(xgb_clf.score(X_
```

```
The accuracy of the XGBoost classifier on training data is 1.00
The accuracy of the XGBoost classifier on test data is 0.98
```

```
#Applying Decision Tree
from sklearn import tree

#Create tree object
decision_tree = tree.DecisionTreeClassifier(criterion='gini')

#Train DT based on scaled training set
decision_tree.fit(X_train_std, y_train)

#Print performance
print('The accuracy of the Decision Tree classifier on training data is {:.2f}'.format(decisi
print('The accuracy of the Decision Tree classifier on test data is {:.2f}'.format(decision_t
```

```
The accuracy of the Decision Tree classifier on training data is 1.00
The accuracy of the Decision Tree classifier on test data is 0.98
```

```
#Applying RandomForest
from sklearn.ensemble import RandomForestClassifier

#Create Random Forest object
```

```
random_forest = RandomForestClassifier()

#Train model
random_forest.fit(X_train_std, y_train)

#Print performance
print('The accuracy of the Random Forest classifier on training data is {:.2f}'.format(random
print('The accuracy of the Random Forest classifier on test data is {:.2f}'.format(random_for
```

```
The accuracy of the Random Forest classifier on training data is 1.00
The accuracy of the Random Forest classifier on test data is 0.98
```

Conclusion: This notebook explored 5 basic machine learning algorithms. In the majority of them, the accuracy was higher in the training dataset, as expected. In the ones it was lower, the difference in not significant. The accuracy for those cases should be evaluated with a cross validation and not only with a single fold. Using cross validation will probably result in higher accuracy for all algorithms.

Visualizing KNN, SVM, and XGBoost on Iris Dataset

Here we use Python to visualize how certain machine learning algorithms classify certain data points in the Iris dataset. Let's begin by importing the Iris dataset and splitting it into features and labels. We will use only the petal length and width for this analysis.

```
# Import data and modules
import pandas as pd
import numpy as np
from sklearn import datasets
%pylab inline
pylab.rcParams['figure.figsize'] = (10, 6)

iris = datasets.load_iris()

# We'll use the petal length and width only for this analysis
X = iris.data[:, [2, 3]]
y = iris.target

# Place the iris data into a pandas dataframe
iris_df = pd.DataFrame(iris.data[:, [2, 3]], columns=iris.feature_names[2:])

# View the first 5 rows of the data
print(iris_df.head())

# Print the unique labels of the dataset
print('\n' + 'The unique labels in this data are ' + str(np.unique(y)))
```

```
Populating the interactive namespace from numpy and matplotlib
   petal length (cm)  petal width (cm)
0                1.4               0.2
1                1.4               0.2
2                1.3               0.2
3                1.5               0.2
4                1.4               0.2

The unique labels in this data are [0 1 2]
```

Next, we'll split the data into training and test datasets.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3, random_state=0)

print('There are {} samples in the training set and {} samples in the test set'.format(
X_train.shape[0], X_test.shape[0]))
print()
```

```
There are 105 samples in the training set and 45 samples in the test set
```

For many machine learning algorithms, it is important to scale the data. Let's do that now using sklearn.

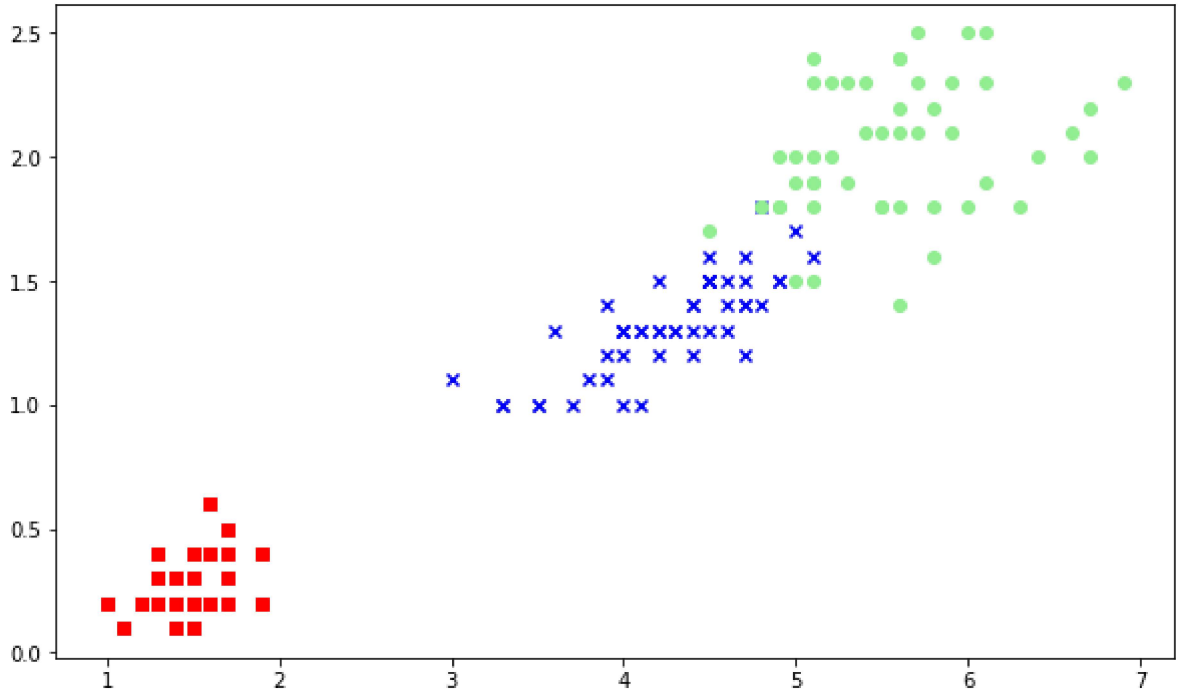
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

sc.fit(X_train)

X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

print('After standardizing our features, the first 5 rows of our data now look like this:\n')
print(pd.DataFrame(X_train_std, columns=iris_df.columns).head())
```

```
After standardizing our features, the first 5 rows of our data now look like this:

   petal length (cm)  petal width (cm)
0          -0.182950         -0.293181
1           0.930661          0.737246
2           1.042022          1.638870
3           0.652258          0.350836
4           1.097702          0.737246
```

If we plot the original data, we can see that one of the classes is linearly separable, but the other two are not.

```
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt

markers = ('s', 'x', 'o')
colors = ('red', 'blue', 'lightgreen')
cmap = ListedColormap(colors[:len(np.unique(y_test))])
for idx, cl in enumerate(np.unique(y)):
    plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                c=cmap(idx), marker=markers[idx], label=cl)
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
```



Let's try to use a Linear SVC to predict the the labels of our test data.

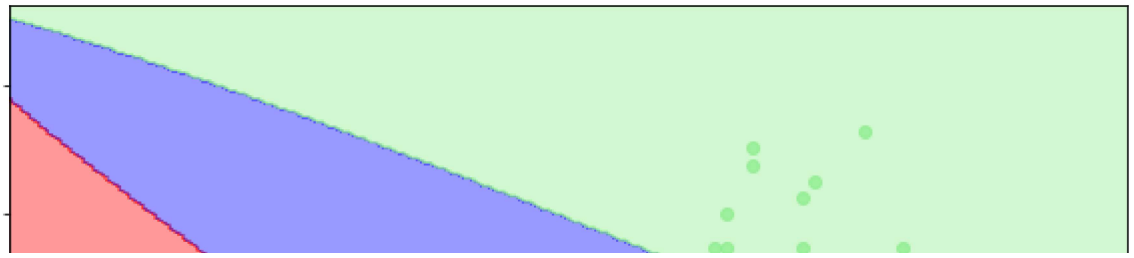
```
from sklearn.svm import SVC

svm = SVC(kernel='rbf', random_state=0, gamma=.10, C=1.0)
svm.fit(X_train_std, y_train)

print('The accuracy of the svm classifier on training data is {:.2f} out of 1'.format(svm.sco

print('The accuracy of the svm classifier on test data is {:.2f} out of 1'.format(svm.score(X
```

```
The accuracy of the svm classifier on training data is 0.95 out of 1
The accuracy of the svm classifier on test data is 0.98 out of 1
```

It looks like our classifier performs pretty well. Let's visualize how the model classified the samples in our test data.

```
import warnings


def versiontuple(v):
    return tuple(map(int, (v.split("."))))


def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02):

    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                    alpha=0.8, c=cmap(idx),
                    marker=markers[idx], label=cl)


plot_decision_regions(X_test_std, y_test, svm)
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
```



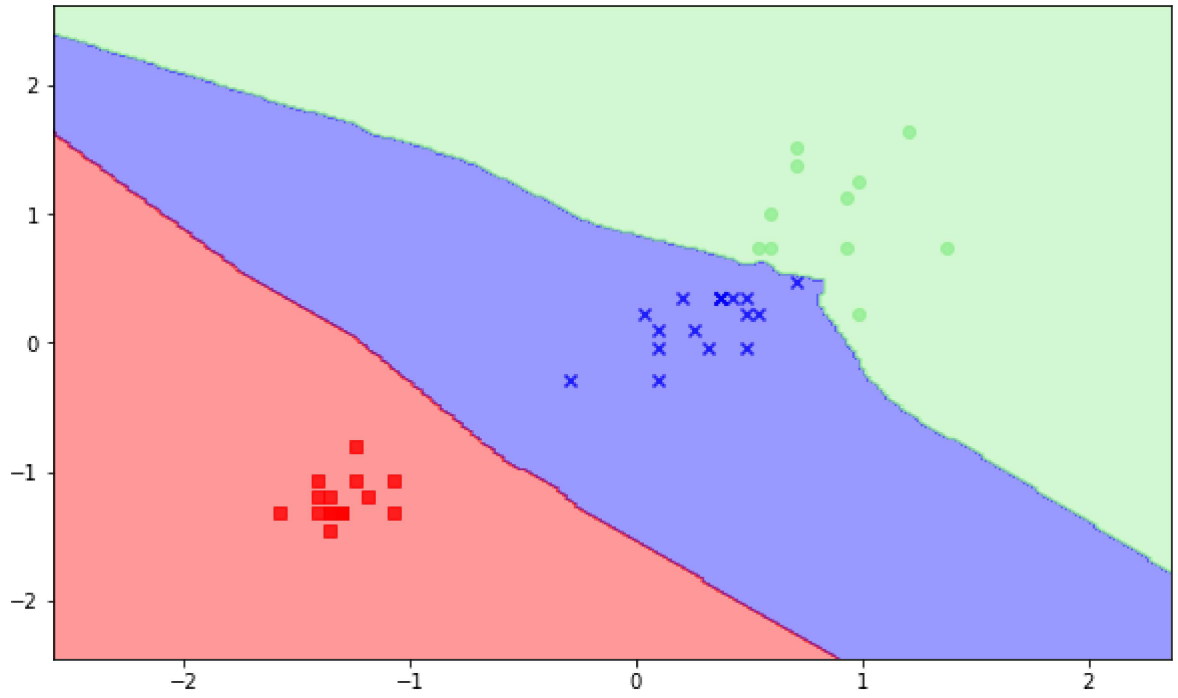
Now, let's test out a KNN classifier.


```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5, p=2, metric='minkowski')
knn.fit(X_train_std, y_train)

print('The accuracy of the knn classifier is {:.2f} out of 1 on training data'.format(knn.sco
print('The accuracy of the knn classifier is {:.2f} out of 1 on test data'.format(knn.score(X
```

```
The accuracy of the knn classifier is 0.95 out of 1 on training data
The accuracy of the knn classifier is 1.00 out of 1 on test data
```

```
plot_decision_regions(X_test_std, y_test, knn)
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
```



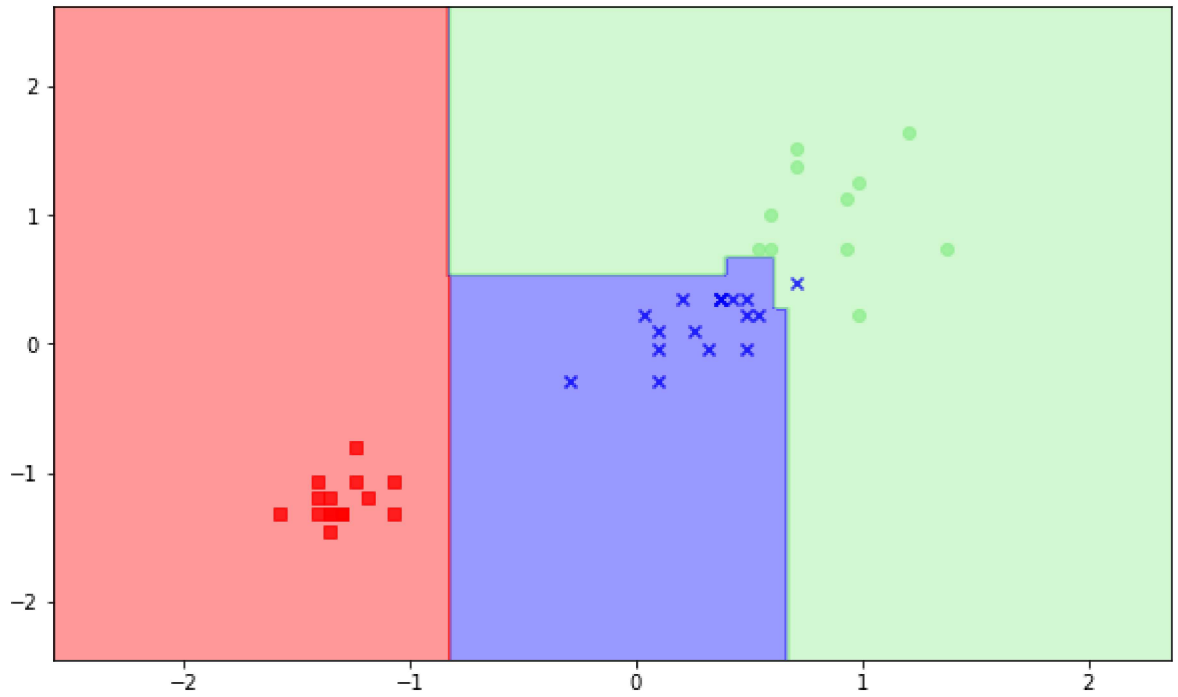
And just for fun, we'll plot an XGBoost classifier.

```
import xgboost as xgb

xgb_clf = xgb.XGBClassifier()
xgb_clf = xgb_clf.fit(X_train_std, y_train)

print('The accuracy of the xgb classifier is {:.2f} out of 1 on training data'.format(xgb_clf
print('The accuracy of the xgb classifier is {:.2f} out of 1 on test data'.format(xgb_clf.sco
```

```
The accuracy of the xgb classifier is 0.98 out of 1 on training data
The accuracy of the xgb classifier is 0.98 out of 1 on test data
```

```
plot_decision_regions(X_test_std, y_test, xgb_clf)
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
```



In all classifiers, the performance on the test data was better than the training data. At least with the parameters specified in this very simple approach, the KNN algorithm seems to have performed the best. However, this may not be the case depending on the dataset and more careful parameter tuning