Python Basic Machine Learning

Olarik Surinta, PhD.

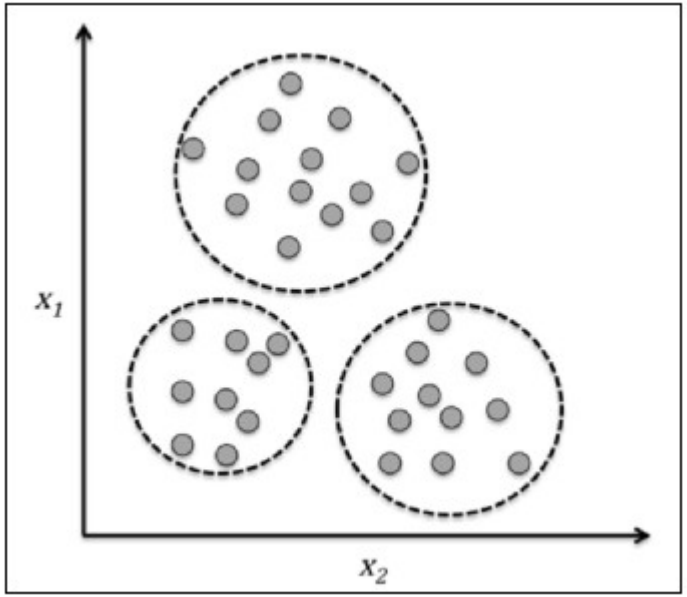
Outline

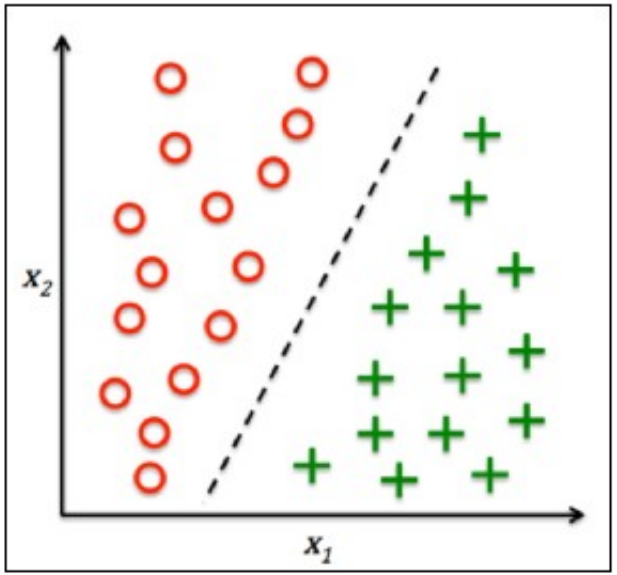
- Types of machine learning
- Classic Iris flower dataset
- A roadmap for building machine learning systems
- Data visualization
- K-Nearest neighbors (KNN) classification

The three different types of machine learning

- Unsupervised learning - *clustering*
- Supervised learning - *classification*
- Reinforcement learning

Unsupervised vs supervised learning

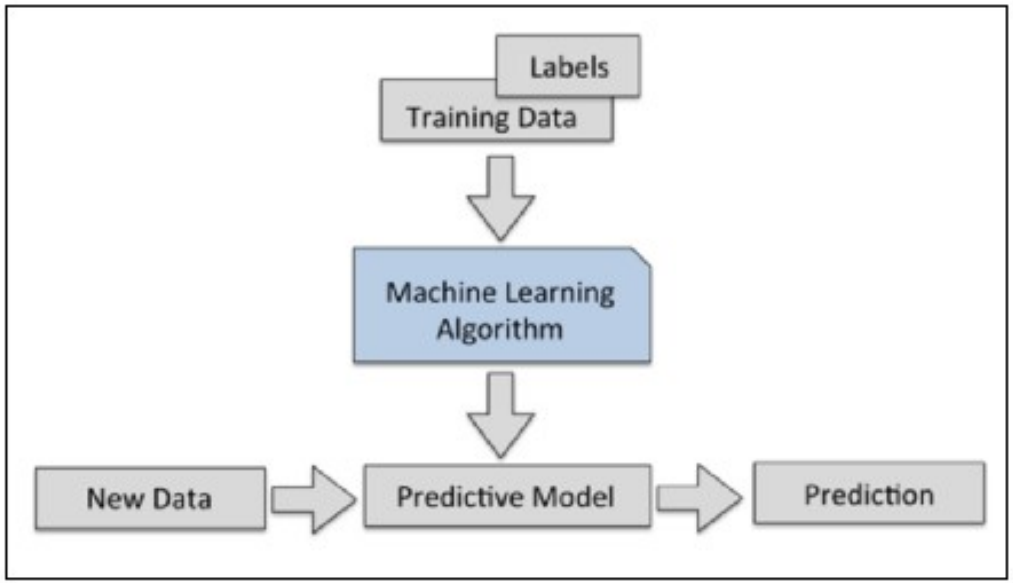




Supervised learning

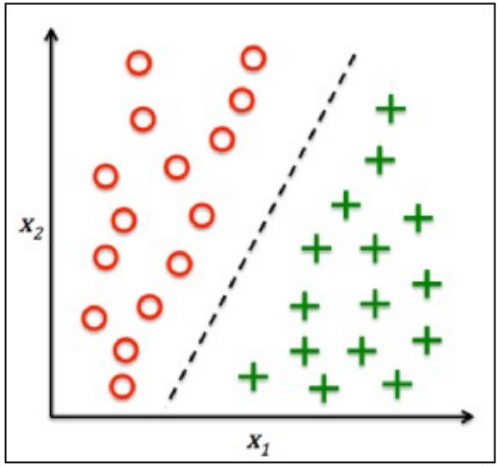
- The main goal in supervised learning is to learn a model from labeled training data that allows us to make predictions about *unseen data*.
- The term *supervised* refers to a set of samples where the desired output signals (*labels*) are already known.

Supervised learning



Binary classification task

- The following figure illustrates the concept of a binary classification task given 30 training samples:
  - 15 training samples are labeled as negative class (circles) and 15 training samples are labeled as positive class (plus signs)
- In this scenario, our dataset is two-dimensional, which means that each sample has two values associated with it: x_1 and x_2



Binary classification task

- We can use a supervised machine learning algorithm to learn a rule – the decision boundary represented as a black dashed line – that can separate those two classes and classify new data into each of those two categories given its x_1 and x_2 values.

Unsupervised learning

- Clustering is an exploratory data analysis technique that allows us to organize a pile of information into meaningful subgroups (clusters) without having any prior knowledge of their group memberships.
- Each cluster that may arise during the analysis defines a group of objects that share a certain degree of similarity but are more dissimilar to objects in other clusters, which is why clustering is also called “unsupervised classification.”
- Clustering is a great technique for structuring information and deriving meaningful relationships among data.

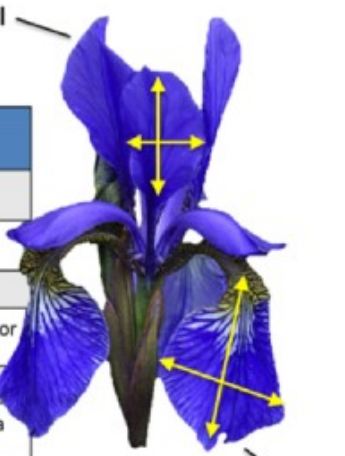
Dataset – Iris flower dataset

- The Iris dataset is a classic example in the field of machine learning.
- The Iris dataset contains the measurements of 150 iris flowers from three different species: Setosa, Versicolor, and Viriginica.
- Each flower sample represents one row in dataset, and the flower measurements in centimeters are stored as columns, which we also call the features of the dataset.

The Iris flower dataset

Samples
(instances, observations)

	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica



Features
(attributes, measurements, dimensions)

Class labels
(targets)

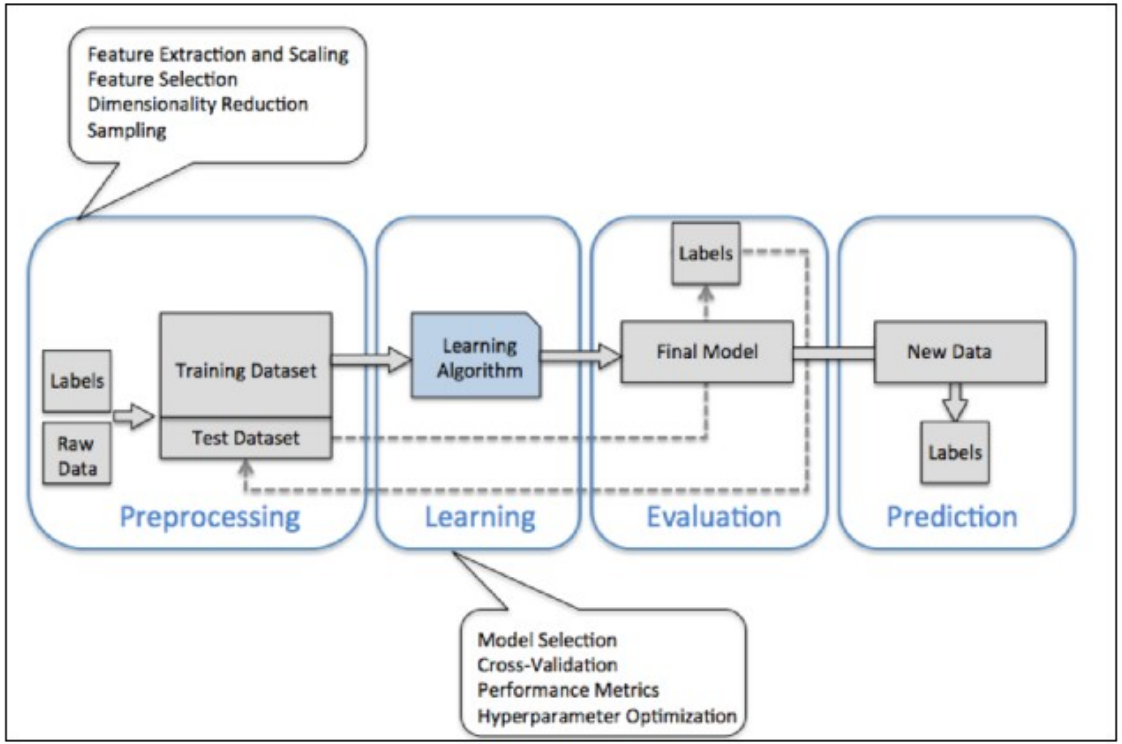
The Iris flower dataset

- The Iris dataset, consisting of 150 samples and 4 features.

$$\begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & x_4^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & x_4^{(2)} \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{(150)} & x_2^{(150)} & x_3^{(150)} & x_4^{(150)} \end{bmatrix}$$

$$\boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ \dots \\ y^{(150)} \end{bmatrix} \left(y \in \{\text{Setosa, Versicolor, Virginica}\} \right)$$

A roadmap for building machine learning systems



Visualization Iris dataset

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets
from sklearn.decomposition import PCA
```

Visualization Iris dataset

```
iris = datasets.load_iris()
X = iris.data[:, :2]  # we only take the first two features.
Y = iris.target

x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5

plt.figure(2, figsize=(8, 6))
plt.clf()
```

Visualization Iris dataset

```
# Plot the training points
plt.scatter(X[:, 0], X[:, 1], c=Y, cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
```

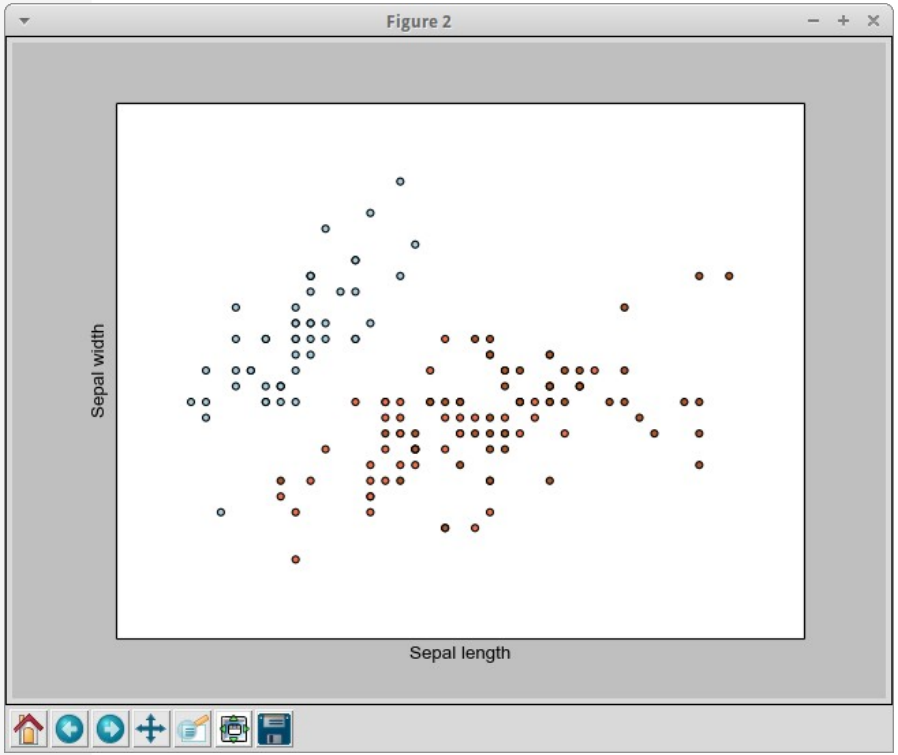
Visualization Iris dataset

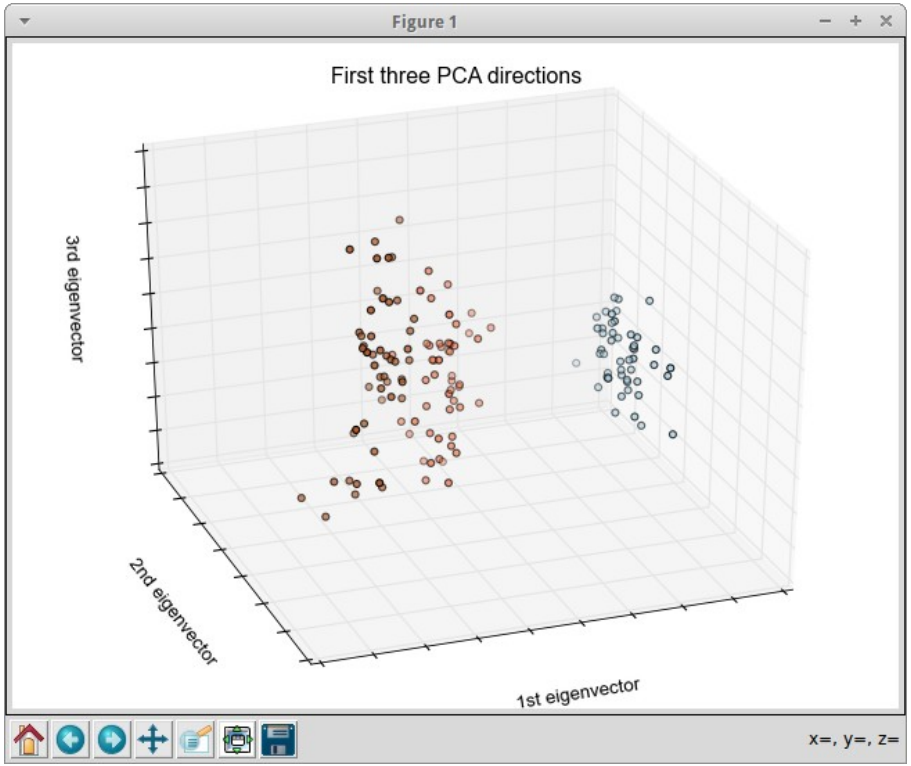
```
# To getter a better understanding of interaction of the dimensions

# plot the first three PCA dimensions

fig = plt.figure(1, figsize=(8, 6))

ax = Axes3D(fig, elev=-150, azim=110)

X_reduced = PCA(n_components=3).fit_transform(iris.data)
```

Visualization Iris dataset

```
ax.scatter(X_reduced[:, 0], X_reduced[:, 1], X_reduced[:, 2], c=Y,cmap=plt.cm.Paired)
ax.set_title("First three PCA directions")
ax.set_xlabel("1st eigenvector")
ax.w_xaxis.set_ticklabels([])
ax.set_ylabel("2nd eigenvector")
ax.w_yaxis.set_ticklabels([])
ax.set_zlabel("3rd eigenvector")
ax.w_zaxis.set_ticklabels([])


plt.show()
```

Visualization Iris dataset





K-Nearest neighbors (KNN) classification

```
import numpy as np
from sklearn import datasets

iris = datasets.load_iris()  # load iris dataset
iris_X = iris.data       # data, x
iris_y = iris.target     # label, target, y

print "number of class: ", len(np.unique(iris_y))
print "number of features: ", iris_X.shape[1]
print "number of instances: ", iris_X.shape[0]
```

number of class: 3
number of instances: 150
number of features: 4

KNN - Cont

```
# split iris data in train and  test data
# A random permutation, to split the data randomly
np.random.seed(0)
indices = np.random.permutation(len(iris_X))    # random
iris_X_train = iris_X[indices[:-10]] # select 1-140 instances
iris_y_train = iris_y[indices[:-10]]
iris_X_test  = iris_X[indices[-10:]] # select 141-150 instances
iris_y_test  = iris_y[indices[-10:]]
```

KNN - Cont

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

KNN - Cont

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
knn.fit(iris_X_train, iris_y_train)

print "Predict Y: ", knn.predict(iris_X_test)
print "Actual Y: ", iris_y_test
```

Tuning parameters

```
>>> knn = KNeighborsClassifier(n_neighbors=3, weights="distance")

>>> knn.fit(iris_X_train, iris_y_train)

KNeighborsClassifier(algorithm='auto', leaf_size=30,
           metric='minkowski', metric_params=None,
           n_jobs=1, n_neighbors=3, p=2, weights='distance')
```

Tuning parameters

```
>>> print "Predict Y", knn.predict(iris_X_test)
Predict Y [1 2 1 0 0 0 2 1 2 0]
>>> print "Actual Y: ", iris_y_test
Actual Y:  [1 1 1 0 0 0 2 1 2 0]
```

Tuning parameters

```
>>> knn = KNeighborsClassifier(n_neighbors=9, weights="uniform")
>>> knn.fit(iris_X_train, iris_y_train)
KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=9, p=2,
          weights='uniform')
>>> print "Predict Y: ", knn.predict(iris_X_test)
Predict Y: [1 1 1 0 0 0 2 1 2 0]
```

Example: 2 – iris-knn.py

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets
```

Example: 2 – iris-knn.py

```
n_neighbors = 9

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2]  # we only take the first two features. We could
                      # avoid this ugly slicing by using a two-dim dataset
y = iris.target

h = .02  # step size in the mesh

# Create color maps
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
```

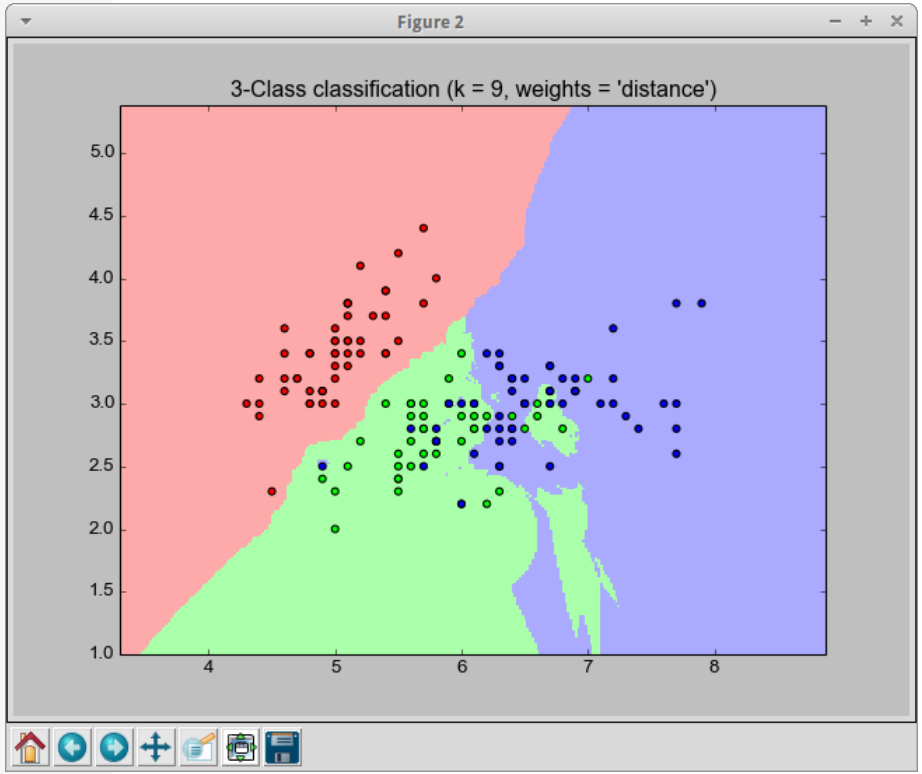
Example: 2 – iris-knn.py

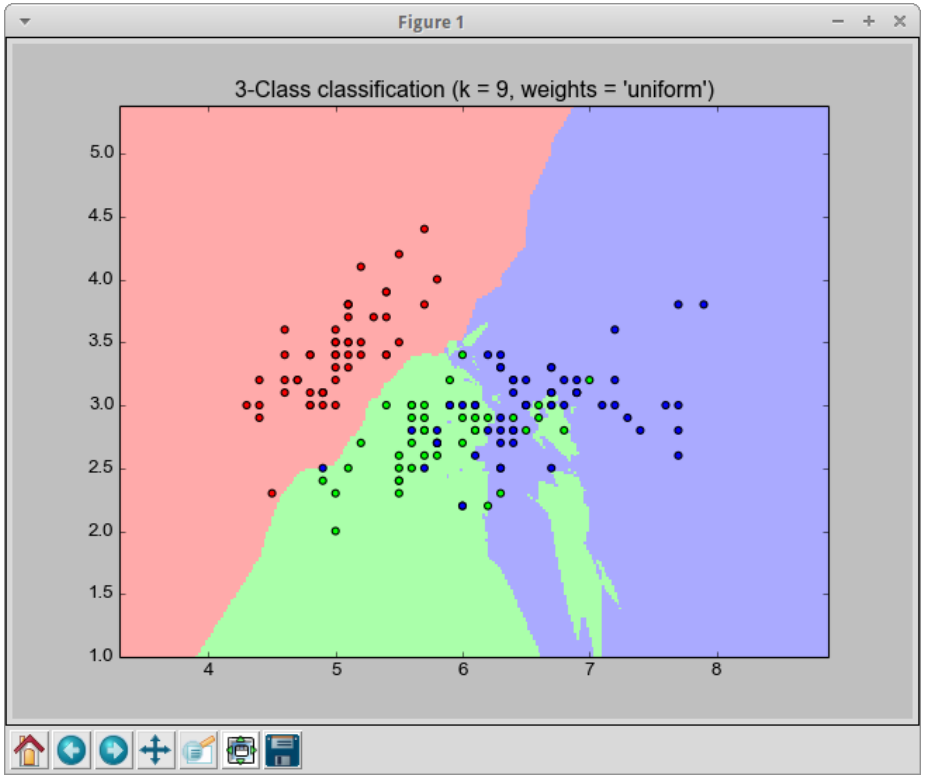
```
for weights in ['uniform', 'distance']:
    # we create an instance of Neighbours Classifier and fit the data.
    clf = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)
    clf.fit(X, y)

    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
```

Example: 2 – iris-knn.py

```
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title("3-Class classification (k = %i, weights = '%s')"
              % (n_neighbors, weights))

plt.show()
```

Example: 2 - $ python iris-knn.py





References

- R. Sebastian (2015), Python machine learning – Unlock deeper insights into machine learning with this vital guide to cutting-edge predictive analytics
- http://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html
- http://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html#sphx-glr-auto-examples-neighbors-plot-classification-py

References

- http://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html#sphx-glr-auto-examples-neighbors-plot-classification-py
- http://scikit-learn.org/stable/tutorial/statistical_inference/supervised_learning.html
- http://scikit-learn.org/stable/modules/neighbors.html
- http://machinelearningmastery.com/machine-learning-in-python-step-by-step/