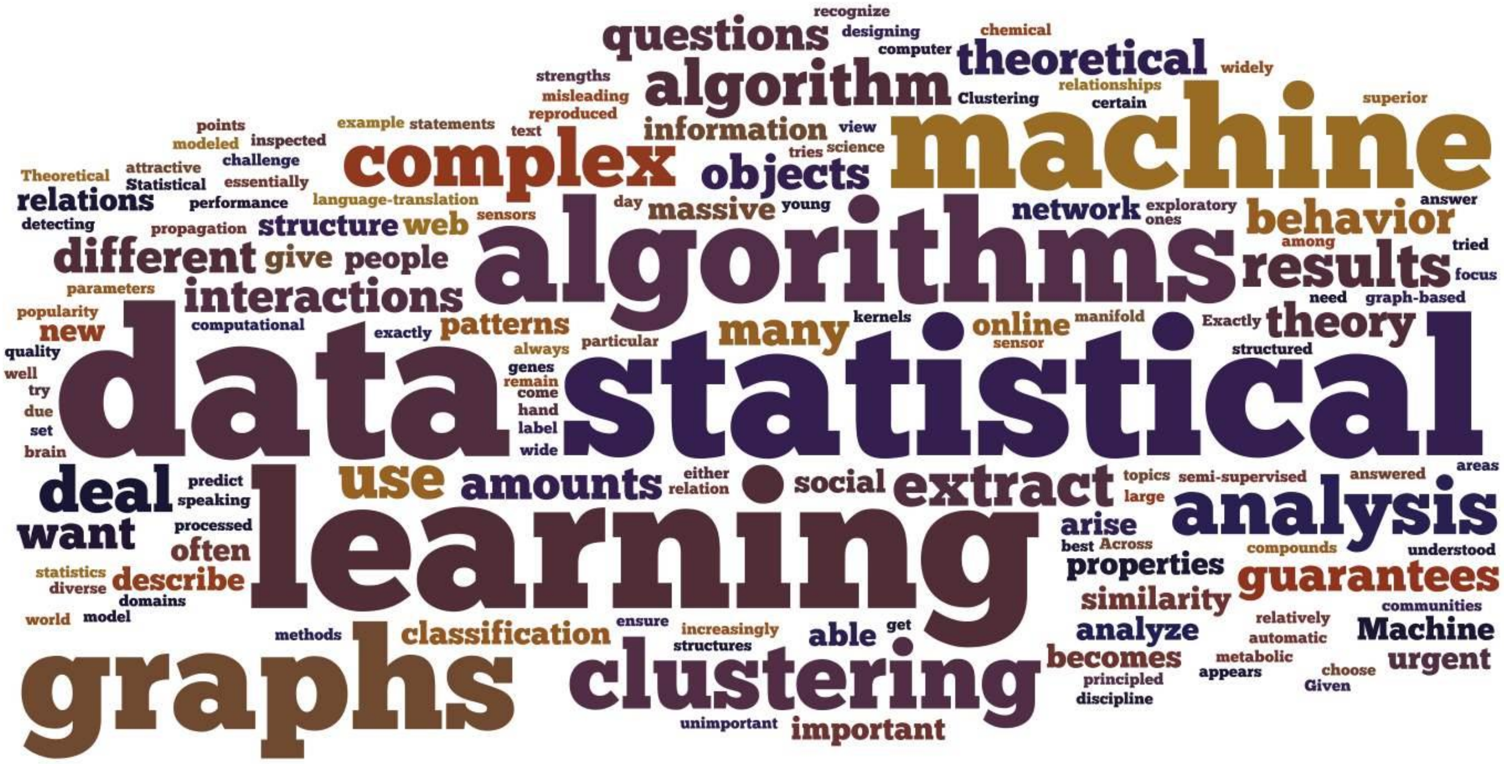


# Machine Learning using scikit-learn

Sarath Chandar A P

IBM Research India

# Scikit-learn

- Open source ML library for python.
- Contains various classification, regression, clustering algorithms.
- Interoperate with NumPy and SciPy.

# General structure of a learning program

1. Import necessary modules.
2. Setup train/test data.
3. Create a learning algorithm object.
4. Fit the training data to the algorithm (training)
5. Predict the results for the test data (testing)
6. Evaluate the model based on the performance.

# A simple perceptron

```
import numpy as np
from sklearn.linear_model import Perceptron

X = np.asarray([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
 Y = np.asarray([1, 1, 1, 2, 2, 2])

clf = Perceptron()
clf.fit(X, Y)

print(clf.predict([[-0.8, -1]]))
print(clf.decision_function([[-0.8, -1]]))
```

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.metrics import *

iris = load_iris()
X = iris.data
Y = iris.target
xtrain,xtest,ytrain,ytest = train_test_split(X,Y,random_state=0)

clf = Perceptron()
clf.fit(xtrain,ytrain)

pred = clf.predict(xtest)
cm = confusion_matrix(ytest,pred)
print cm
print accuracy_score(ytest,pred)
print precision_score(ytest,pred)
print recall_score(ytest,pred)
print precision_score(ytest,pred,average=None)
print recall_score(ytest,pred,average=None)
```

# Let us try Naïve Bayes !

```
from sklearn.naive_bayes import GaussianNB

clf = GaussianNB()
```

# K- Nearest Neighbors

```
from sklearn.neighbors import KNeighborsClassifier

clf = KNeighborsClassifier(n_neighbors=3)
```

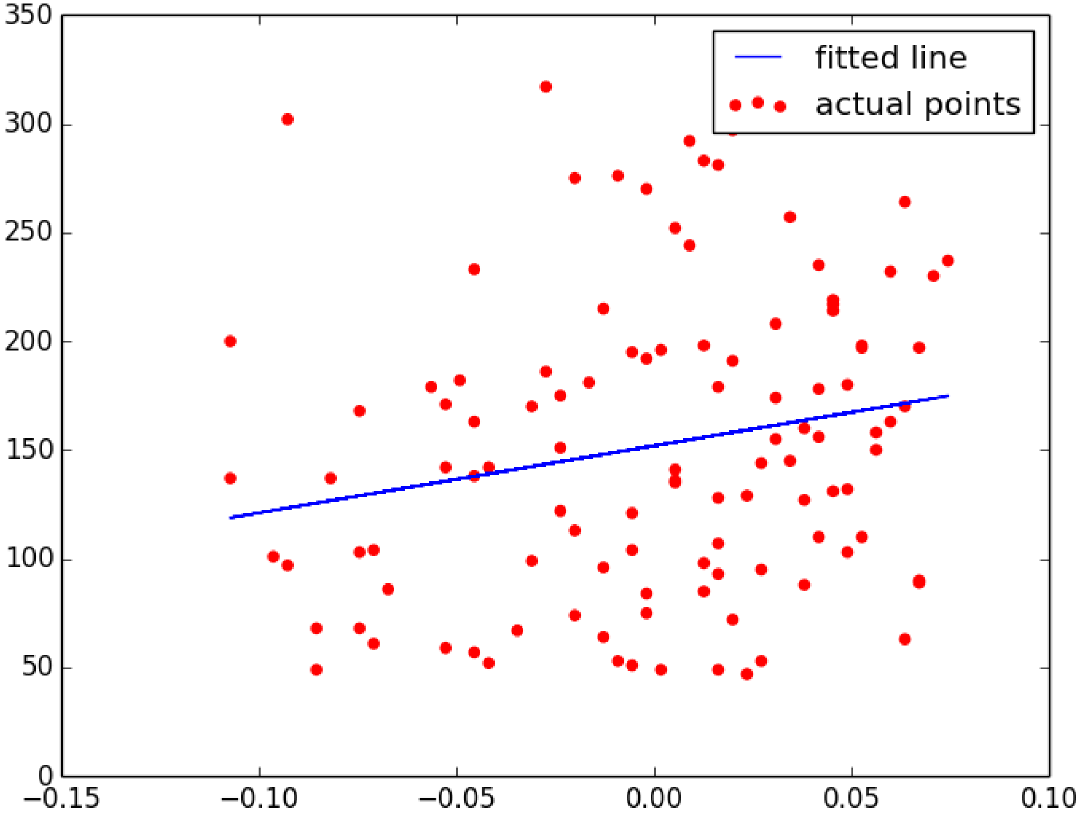
# Decision Trees

```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier()
```

# Regression

```
import numpy as np
import numpy
import matplotlib.pyplot as plt
from sklearn import datasets, linear_model
from sklearn.cross_validation import train_test_split

diabetes = datasets.load_diabetes()
x = diabetes.data[:,0]
x = x.reshape((x.shape[0],1))
y = diabetes.target
xtrain,xtest,ytrain,ytest = train_test_split(x,y,random_state=0)

lin_reg = linear_model.LinearRegression()
lin_reg.fit(xtrain, ytrain)
```

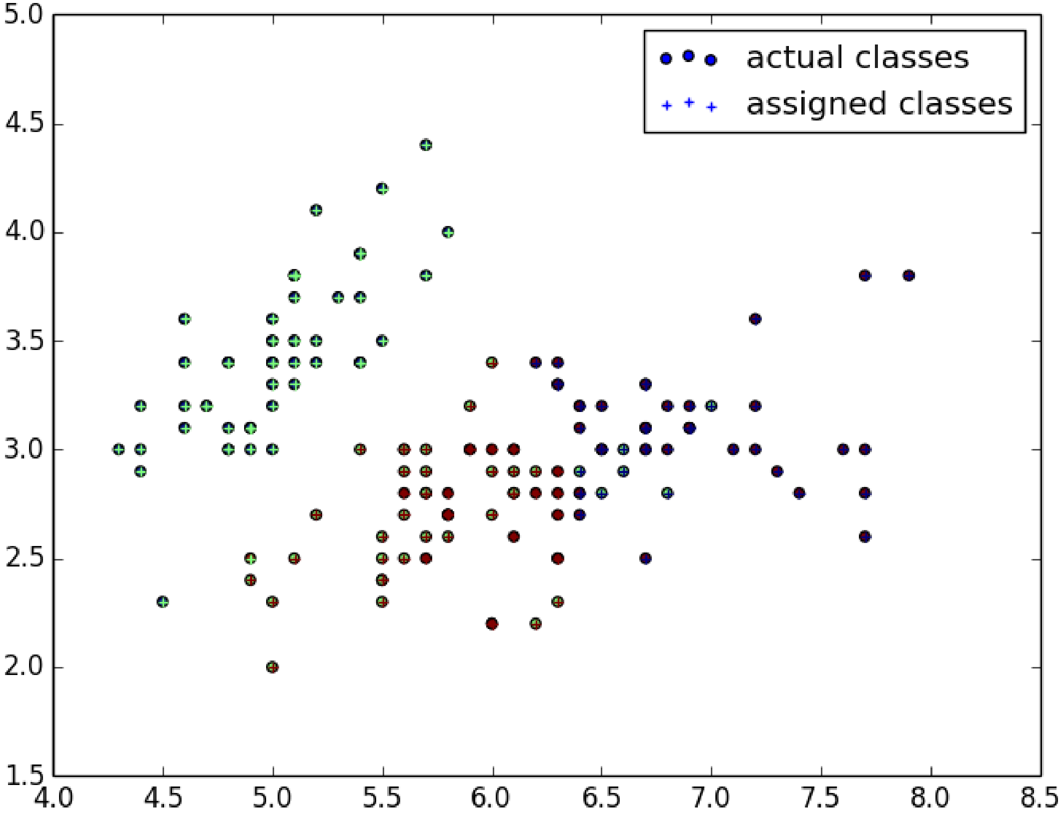
```
predicted_y = lin_reg.predict(xtest)
mse = np.mean((predicted_y - ytest) ** 2)
print mse

plt.figure()
plt.scatter(xtest, ytest, color = 'red', label = 'actual points')
plt.plot(xtest, predicted_y, color = 'blue', label = 'fitted line')
plt.legend(loc = 'upper right')
plt.show()
```


350
300
250
200
150
100
50
0
−0.15
−0.10
−0.05
0.00
0.05
0.10
fitted line
actual points

# Clustering - kmeans

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets, cluster, metrics

iris = datasets.load_iris()
labels = iris.target
iris_x = iris.data[:]
iris_x = iris_x[:, 0:2]

kmeans = cluster.KMeans(n_clusters = 3)
kmeans.fit(iris_x)
```

```
plt.figure()

plt.scatter(iris_x[:, 0], iris_x[:, 1], c = labels, marker = 'o', label =
'actual classes‘)

plt.scatter(iris_x[:, 0], iris_x[:, 1], c = kmeans.labels_ , marker = '+',
label = 'assigned classes')

plt.legend(loc = 'upper right')
plt.show()
```


actual classes
assigned classes
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5