Machine Learning with Scikit learn

Jayanti Prasad

Inter-University Centre for Astronomy & Astrophysics (IUCAA)
PUNE - 411007

June 22, 2018

Plan of the Talk

- Introduction
- Scikit-learn
- (1) Linear Regression
- (2) Logistic Regression
- (3) Nearest Neighbors

Machine Learing : Definitions

Arthur Samuel

A field of study that gives computers the ability to learn without being explicitly programmed.

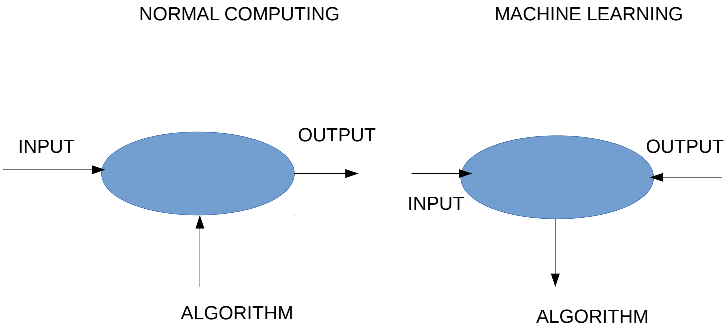
Tom M. Mitchell

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with the experience E.
Here we need to define :

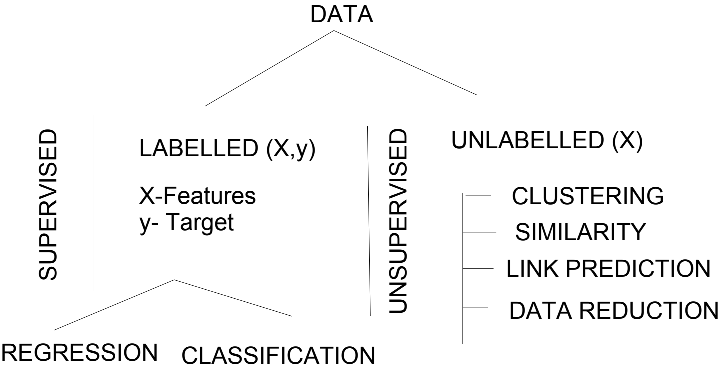
- Task (T), either one or more
- Experience (E)
- Performance (P)

Machine Learning

Training Machines to see patterns in the data and learn from that - Ability to learn in a changing environment.



Machine Learning



Layman's Machine Learning (Supervised) Recipes

1. Split the data into two parts: Training/Learning and Testing.

$$(X, y) = (X_{\text{Train}}, y_{\text{Train}}) + (X_{\text{Test}}, y_{\text{Test}}) \tag{1}$$

2. Find a model $f(X, \theta) : X \longrightarrow y$ from the training data. This need some mismatch function to be minimized.

$$\chi^2(\theta) = \sum_{i=1}^{N} (y_i - f(X_i, \theta))^2 \tag{2}$$

3. The model could be predictive (make predictions in the future), or descriptive (gain knowledge from data), or both.
4. Apply the model on testing data and find the accuracy.
5. Once satisfied apply model for some unknown feature X and get the target y for that.

Machine Learning: Frameworks

Caffe

Caffe2

moa

H_2O.ai

TensorFlow™

K

NumPy

PYTORCH

scikit learn

APACHE Spark™

Theano

torch

Scitkit-learn : What it is ?

- Scikit-learn is a free machine learning library for the Python programming language.
- The scikit-learn project started as scikits.learn, a Google Summer of Code project by David Cournapeau.
- Its name stems from the notion that it is a "SciKit" (SciPy Toolkit), a separately-developed and distributed third-party extension to SciPy.
- Scikit-learn is largely written in Python, with some core algorithms written in Cython to achieve performance.

Scikit-learn : Why to learn ?

- If you have problem with a lot of data and you want to draw insight from that using some open source python library.
- If you want to understand common problems of Machine Learning such as regression, classification, clusters ...
- If you want to develop new techniques or compare different techniques.

Scikit-learn : What it has ?

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, ... — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, ... — Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, ... — Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization. — Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics. — Examples

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction. — Examples

Reference : http://scikit-learn.org/stable/index.html

Scikit-learn : Installation

- Scikit-learn requires:
  - Python (>= 2.7 or >= 3.3),
  - NumPy (>= 1.8.2),
  - SciPy (>= 0.13.3).
- One can install it from the source by cloning the git repo:
  git clone https://github.com/scikit-learn/scikit-learn.git
- pip install -U scikit-learn
- conda install scikit-learn

Scikit-learn : The package/s

- cluster
- compose
- covariance
- cross_decomposition
- cross_validation
- datasets
- decomposition
- discriminant_analysis
- ensemble
- externals
- feature_extraction
- feature_selection
- gaussian_process
- linear_model
- manifold
- metrics
- mixture

scikit-learn packages

- model_selection
- neighbors
- neural_network
- preprocessing
- random_projection
- semi_supervised
- svm
- tests
- tree
- utils

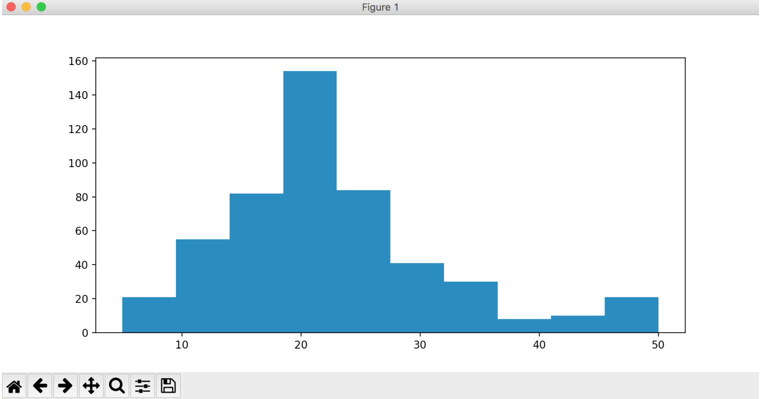
Scikit-learn : Test datasets

```
>>> import sklearn.datasets as datasets
>>> dir(datasets)
['__all__', '__builtins__', '__doc__', '__file__', '__name__', '__package__', '__path__', '_svmlight_format', 'ba
r_data_home', 'covtype', 'dump_svmlight_file', 'fetch_20newsgroups', 'fetch_20newsgroups_vectorized', 'fetch_cali
, 'fetch_kddcup99', 'fetch_lfw_pairs', 'fetch_lfw_people', 'fetch_mldata', 'fetch_olivetti_faces', 'fetch_rcv1',
get_data_home', 'kddcup99', 'lfw', 'load_boston', 'load_breast_cancer', 'load_diabetes', 'load_digits', 'load_fil
', 'load_mlcomp', 'load_sample_image', 'load_sample_images', 'load_svmlight_file', 'load_svmlight_files', 'load_w
lobs', 'make_checkerboard', 'make_circles', 'make_classification', 'make_friedman1', 'make_friedman2', 'make_frie
', 'make_hastie_10_2', 'make_low_rank_matrix', 'make_moons', 'make_multilabel_classification', 'make_regression',
ded_signal', 'make_sparse_spd_matrix', 'make_sparse_uncorrelated', 'make_spd_matrix', 'make_swiss_roll', 'mlcomp'
olivetti_faces', 'rcv1', 'samples_generator', 'species_distributions', 'svmlight_format', 'twenty_newsgroups']
>>> boston=datasets.load_boston()
>>> dir(boston)
['DESCR', 'data', 'feature_names', 'filename', 'target']
>>> print(boston['data'].shape)
(506, 13)
>>> print(boston['data'].size)
6578
>>> print(boston['data'].dtype)
float64
>>> print(boston['feature_names'])
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
>>> print(boston['target'].shape)
(506,)
>>> print(boston['target'].size)
506
>>> print(boston['target'].dtype)
float64
>>>
```

Test datasets

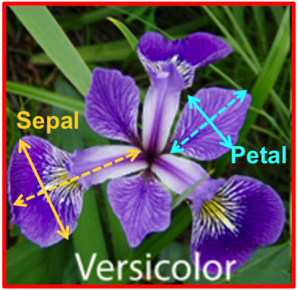
```
--------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the targ

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/
```

Datasets




```
>>>
>>>
>>> plt.hist(boston['target'])
(array([ 21.,  55.,  82., 154.,  84.,  41.,  30.,   8.,  10.,  21.]), array([ 5. ,  9.5, 14. , 18.5, 23. , 27.5, 32. , 36.5,
41. , 45.5, 50. ]), <a list of 10 Patch objects>)
>>> plt.show()
```

Test datasets : Iris Flower species



Test datasets : Iris Flower species

```
>>> import sklearn.datasets as datasets
>>> iris=datasets.load_iris()
>>> iris.keys()
['target_names', 'data', 'target', 'DESCR', 'feature_names']
>>> iris['feature_names']
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
>>> iris['target_names']
array(['setosa', 'versicolor', 'virginica'], dtype='|S10')
>>> iris['data'].size
600
>>> iris['data'].shape
(150, 4)
>>> iris['data'].dtype
dtype('float64')
>>> iris['target'].size
150
>>> iris['target'].shape
(150,)
>>> iris['target'].dtype
dtype('int64')
>>> iris['target'][1]
0
>>> iris['target'][11]
0
>>> iris['target'][111]
2
>>> iris['data'][1,:]
array([4.9, 3. , 1.4, 0.2])
>>> iris['data'][12,:]
array([4.8, 3. , 1.4, 0.1])
```

Linear Regression : Theory

1. The data is fitted with a linear model :

$$y_i = \sum_{j=1}^{N} w_i x_i = w_0 + w_1 x_i \text{ for one dimensional case.} \quad (3)$$

2. Minimize the mismatch (wrt w_0 and w_1) for training data :

$$\chi^2(w, X) = |y_i - (w_0 + w_1 x_i)|^2 \quad (4)$$

3. Apply the model on testing data and see the accuracy.
4. Once satisfied apply the model for unknown data.

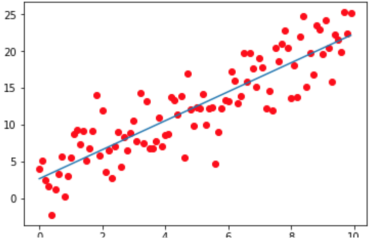
Linear Regression : Examples [linear-regression-general.ipynb]

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import sklearn.linear_model
from sklearn.metrics import r2_score, mean_squared_error

#set some parameters
w0,w1,c=2.1,2.0,3.0

# create data
x=np.arange(0.0,10.0,0.1)

# create Gaussian noise
n=np.random.standard_normal(len(x))

#create data
y = w0+ w1*x +c * n

# now get the model
model = sklearn.linear_model.LinearRegression()

# fit the data
x=x.reshape(-1,1)
model.fit(x,y)

# check the accuracy of the model
y_predict = model.predict(x)

```

Linear Regression: Examples [linear-regression-general.ipynb]

```
#print the coefficients
print("input: intercept=%9.6f, coefficient=%9.6f, noise=%9.6f" %(w0,w1,c))
print("output: intercept=%9.6f, coefficient=%9.6f" %(model.intercept_,model.coef_))

print("r2 score = %9.6f, mean squarred error=%9.6f"
      %(r2_score(y,y_predict),mean_squared_error(y,y_predict)))

y1=model.intercept_ + model.coef_ * x
plt.plot(x,y,'ro')
plt.plot(x,y1)
plt.show()


```

```
input: intercept= 2.100000, coefficient= 2.000000, noise= 3.000000
output: intercept= 2.672812, coefficient= 1.968090
r2 score =  0.775411, mean squarred error= 9.348059
```



Linear Regression : Ridge (Regularization)

- Ordinary Least Square (OLS) can over-fit the data and in order to avoid that we add an extra term in the mismatch function.

$$Q(w, X) = |y_i - (w_0 + w_1 x_i)|^2 + \lambda R(w, X), \tag{5}$$

  where λ is the regularization parameter and R is regularization function.

- There are many choices for R and one of those is the what is called L_2 norm.

$$Q(w, X) = |y_i - (w_0 + w_1 x_i)|^2 + \lambda ||w||_2^2 \tag{6}$$

- There are many ways to set the regularization parameter λ and one of those is generalized Cross-Validation.

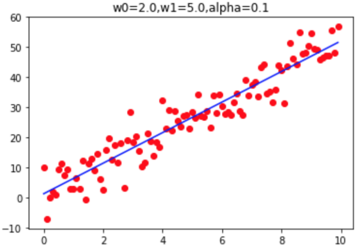
Linear Regression : Example [linear-regression-ridge.ipynb]

```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt
import sklearn.linear_model
from  sklearn.metrics  import r2_score, mean_squared_error

# Set some parameters
w0,w1,c=2.0,5.0,5.0

# create data
x=np.arange(0,10.0,0.1)
er=np.random.standard_normal(len(x))
y=w0 + w1 *x + c * er

# get the model
a=0.1
model=sklearn.linear_model.Ridge(alpha=a)

# fit the model
x=x.reshape(-1,1)
reg=model.fit(x,y)

# see how good the model is
y_predict=model.predict(x)

```

Linear Regression [linear-regression-ridge.ipynb]

```
#print the output
print("input: intercept=%9.6f, coefficient=%9.6f, noise=%9.6f" %(w0,w1,c))
print("output: intercept=%9.6f, coefficient=%9.6f" %(model.intercept_,model.coef_))
print("r2 score = %9.6f, mean squared error=%9.6f"
      %(r2_score(y,y_predict),mean_squared_error(y,y_predict)))
title="w0="+str(w0)+",w1="+str(w1)+",alpha="+str(a)
plt.scatter(x,y,color='red')
plt.plot(x,y_predict,'b-')
plt.title(title)
plt.show()

```

```
input: intercept= 2.000000, coefficient= 5.000000, noise= 5.000000
output: intercept= 1.354194, coefficient= 5.059978
r2 score =  0.901346, mean squared error=23.356040
```



Logistic Regression

- It is used when the 'target' is binary - '0' and '1' or 'yes' and 'no'.
- If the probability of '1' is 'p' then the 'odd' is defined as $p/(1-p)$ and a function called 'logit' is defined as :

$$l = \ln\left(\frac{p}{1-p}\right) \tag{7}$$

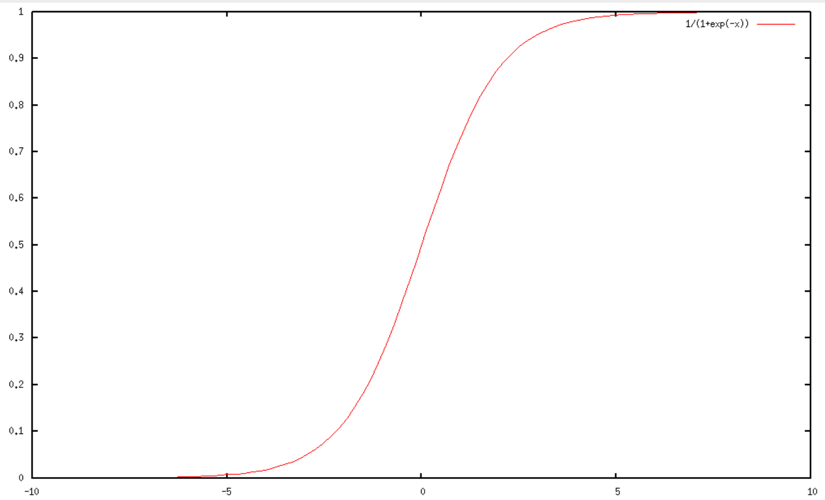
- We try to compute p by fitting a line to the 'logit' function:

$$\ln\left(\frac{p}{1-p}\right) = w_0 + w_1 * x, \tag{8}$$

which gives:

$$p = \frac{1}{1 + e^{-(w_0 + w_1 * x)}} \tag{9}$$

Logistic Regression



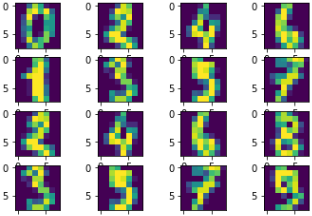
Logistic Regression : MNIST

[18]:

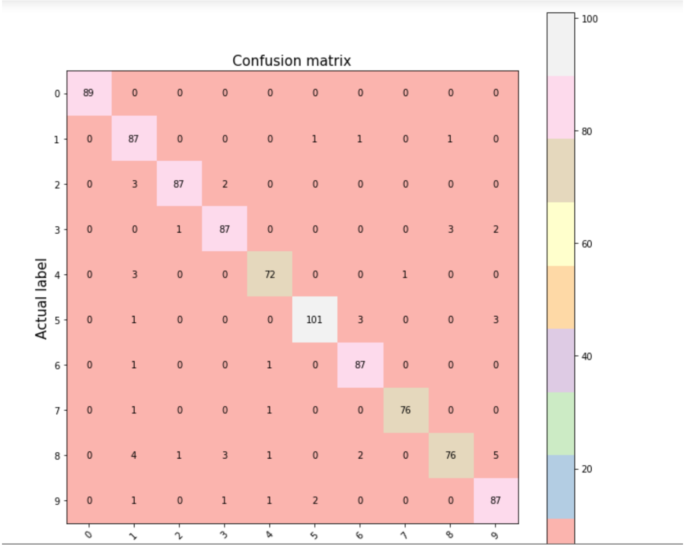
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

#load the image data
digits = load_digits()

# let us show some images
for i in range(0,4):
   for j in range(0,4):
    plt.subplot(4,4,j+4*i+1)
    plt.imshow(digits['images'][i+2*j,:,:])

plt.show()


X,y=digits['data'], digits['target']
#split the data
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
# get the model
model = LogisticRegression()
# fit the model
model.fit(x_train, y_train)
# make prediction for the test set
predictions = model.predict(x_test)
# check the accuracy
score = model.score(x_test, y_test)
print(score)
# see the confusion matrix
cm = metrics.confusion_matrix(y_test, predictions)
```

Logistic Regression : MNIST

```
plt.figure(figsize=(9,9))
plt.imshow(cm, interpolation='nearest', cmap='Pastel1')
plt.title('Confusion matrix', size = 15)
plt.colorbar()
tick_marks = np.arange(10)
plt.xticks(tick_marks, ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"], rotation=45, size = 10)
plt.yticks(tick_marks, ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"], size = 10)
plt.tight_layout()
plt.ylabel('Actual label', size = 15)
plt.xlabel('Predicted label', size = 15)
width, height = cm.shape

for x in xrange(width):
   for y in xrange(height):
      plt.annotate(str(cm[x][y]), xy=(y, x),horizontalalignment='center',verticalalignment='center')

plt.show()

```



Logistic Regression: MNIST



Logistic Regression: Iris

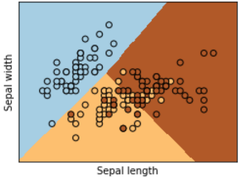
Example 4: Logistic Regression (Iris)

In [9]:

```python
%matplotlib inline

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

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

# get the model
model=linear_model.LogisticRegression(C=1e5)

#fit the model
fit=model.fit(X,y)

#now we will create a mesh and make prediction for mesh points
h=0.02
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

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

Logistic Regression: Iris

```
# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

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

plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())

plt.show()
```



Nearest neighbors : Theory

- Nearest Neighbors is one of the import techniques of ML which can be used for supervised as well as un-supervised learning.
- sklearn.neighbors package from scikit-learn can be used for this purpose.
- The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label from these.
- The number of samples can be a user-defined constant (k-nearest neighbor learning), or vary based on the local density of points (radius-based neighbor learning).
- The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice,

Nearest neighbors : Scikit-learn

- scikit-learn implements two different nearest neighbors classifiers:
  - KNeighborsClassifier: learning based on the k nearest neighbors of each query point.
  - RadiusNeighborsClassifier: learning based on the number of neighbors within a fixed radius r of each training point.
- Weights to neighbours can be uniform or non-uniform.

Nearest Neighbors : Regression

- NN based regression can be used for the purpose when 'labels' are continuous in place of discrete.
- The 'label' for the query point can be found from the mean of the labels of the nearest points.
- NN based regression can be used for finding some missing/blurred area in an image.

Nearest neighbors

```
>>> from sklearn.neighbors import NearestNeighbors
>>> x=np.random.rand(10)
>>> x=x.reshape(-1,1)
>>> nbrs = NearestNeighbors(n_neighbors=2, algorithm='ball_tree').fit(x)
>>> distances, indices = nbrs.kneighbors(x)
>>> x
array([[0.42143913],
       [0.85333869],
       [0.01510071],
       [0.43042917],
       [0.39932002],
       [0.87046633],
       [0.79568511],
       [0.69421828],
       [0.1628455 ],
       [0.31897889]])
>>> indices
array([[0, 3],
       [1, 5],
       [2, 8],
       [3, 0],
       [4, 0],
       [5, 1],
       [6, 1],
       [7, 6],
       [8, 2],
       [9, 4]])
>>> nbrs.kneighbors_graph(x).toarray()
array([[1., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 1., 0.],
       [1., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 1., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 1.]])
>>>
```

Iris with NN: [knn-neighbours-iris.ipynb]

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

iris = load_iris()

X_train, X_test, y_train, y_test = train_test_split(iris['data'], iris['target'],random_state=0)

knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)

X_new = np.array([[5, 2.9, 1, 0.2]])

prediction = knn.predict(X_new)

print("new features=",X_new)
print("new type=",iris['target_names'][prediction])


y_pred = knn.predict(X_test)
print("acuracy=",np.mean(y_pred == y_test))

```

```
('new features=', array([[5. , 2.9, 1. , 0.2]]))
('new type=', array(['setosa'], dtype='|S10'))
('acuracy=', 0.9736842105263158)
```

Iris classification with Nearest Neighbors

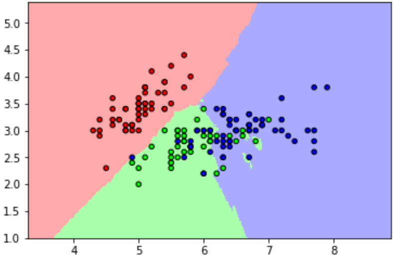
- We can consider only two features of the iris flowers.
- Can make prediction for all the points on the two dimensional grid on the basis of nearest neighbors.
- The result can be shown by a plot with distinct regions for different type of flowers.

Iris KNN : [knn-iris.ipynb]

```
# Firstly let us plot 2-d projects of iris-data

iris = datasets.load_iris()

n_neighbors = 15
h = .02

# get the two-dimensional data

X = iris.data[:, :2]
y = iris.target

# now chose the model
clf = neighbors.KNeighborsClassifier(n_neighbors, weights='uniform')

# fit the model
clf.fit(X, y)

# now we need prediction for the grid so create that
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))

#make the prediction for the whole grid
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
```

Iris KNN : [knn-iris.ipynb]

```
# 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,edgecolor='k', s=20)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.show()
```



Thank You !

References

1. http://scikit-learn.org/stable/documentation.html
2. Guillermo Moncecchi and Raul Garreta, *Learning Scikit-learn: Machine Learning in Python*
3. Trent Hauck, *Scikit-Learn Cookbook*