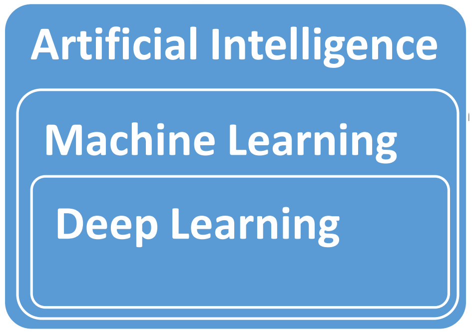
Python-Machine Learning

using Scikit-Learn package



Agenda

- Introduction (SciKit-Learn Toolkit)
- History, contributors
- Data representation in Machine Learning
- Supervised learning example
- Classification model
- Machine Learning Project

  using Iris dataset

Machine learning is a branch in computer science that studies the design of algorithms that can learn.

History

- Scikit-learn was original authored by an data scientist

  David Cournapeau in 2007

- Google Summer of Code Project was started in 2007 by David Cournapeau. Later that year, Matthieu Brucher started work on this project as part of his thesis.
- In 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort and Vincent Michel of INRIA took leadership of the project and made the first public release, February the 1st 2010.
- Since then, several releases have appeared following a ~3 month cycle, and a thriving international community has been leading the development.
- Of the various scikits, scikit-learn as well as scikit-image were described as "well-maintained and popular" in November 2012

Introduction

- Machine learning library written in **Python**
- **Simple and efficient**, for both experts and non-experts
- Classical, **well-established machine learning algorithms**
- **BSD 3 license**
- characterized by a clean, uniform, and streamlined API

- **Community driven development**
- 20~ core developers (mostly researchers)
- 500+ occasional contributors
- **All working publicly together** on GitHub
- Emphasis on **keeping the project maintainable**
  - Style consistency
  - Unit-test coverage
  - Documentation and examples
  - Code review

Pandas NumPy Scikit-Learn workflow

- Start with CSV
- Convert to Pandas DataFrame
- Slice and dice in Pandas
- Convert to NumPy array to feed to Scikit-Learn

Additional web resource :

- **UCI Machine Learning Dataset Repository** The University of California at Irvine (UCI) maintains an online repository of machine learning datasets (at the time of writing, they are listing 233 datasets). The repository is available online: http://archive.ics.uci.edu/ml/
- https://github.com/rasbt/pattern_classification/blob/master/resources/machine_learning_ebooks.md

Data Representation in Scikit-Learn

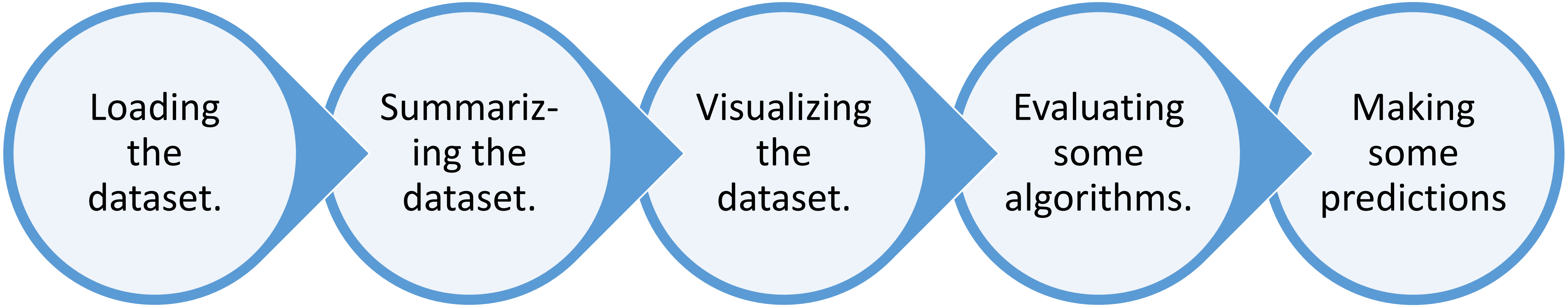
- Machine learning is about creating models from data

- The best way to think about data within Scikit-Learn is in terms of tables of data.

- Data as table : A basic table is a two-dimensional grid of data, in which the rows represent individual elements of the dataset, and the columns represent quantities related to each of these elements.
  - E.g. the Iris dataset, famously analyzed by Ronald Fisher in 1936.
  - This can be downloaded in dataset in the form of a Pandas DataFrame.

Layman’s view of Machine Learning

- Loading the dataset.
- Summarizing the dataset.
- Visualizing the dataset.
- Evaluating some algorithms.
- Making some predictions.



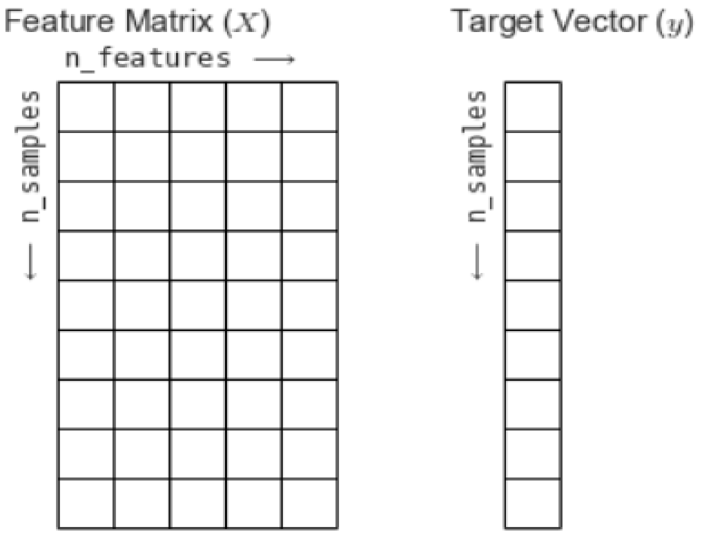
Basics of the Scikit-Learn estimator API

1. Choose a class of model by importing the appropriate estimator class from Scikit-Learn.
2. Choose model hyperparameters by instantiating this class with desired values.
3. Arrange data into a features matrix and target vector
4. Fit the model to your data by calling the fit() method

   of the model instance.
5. Apply the model to new data:
   - For supervised learning, often we predict labels for unknown data using the predict() method.
   - For unsupervised learning, we often transform or infer properties of the data using the transform() or predict() method

Basics of the Scikit-Learn estimator API

1. Choose a class of model
2. Choose model hyperparameters
3. Arrange data into a features matrix and target vector
4. Fit the model to your data
5. Apply model to new data



Supervised learning example:Simple linear regression

1. Choose a class of model. - In Scikit-Learn, every class of model is represented by a Python class.

    from sklearn.linear_model import LinearRegression

- once the model class is selected, hyperparameters are *selected*.

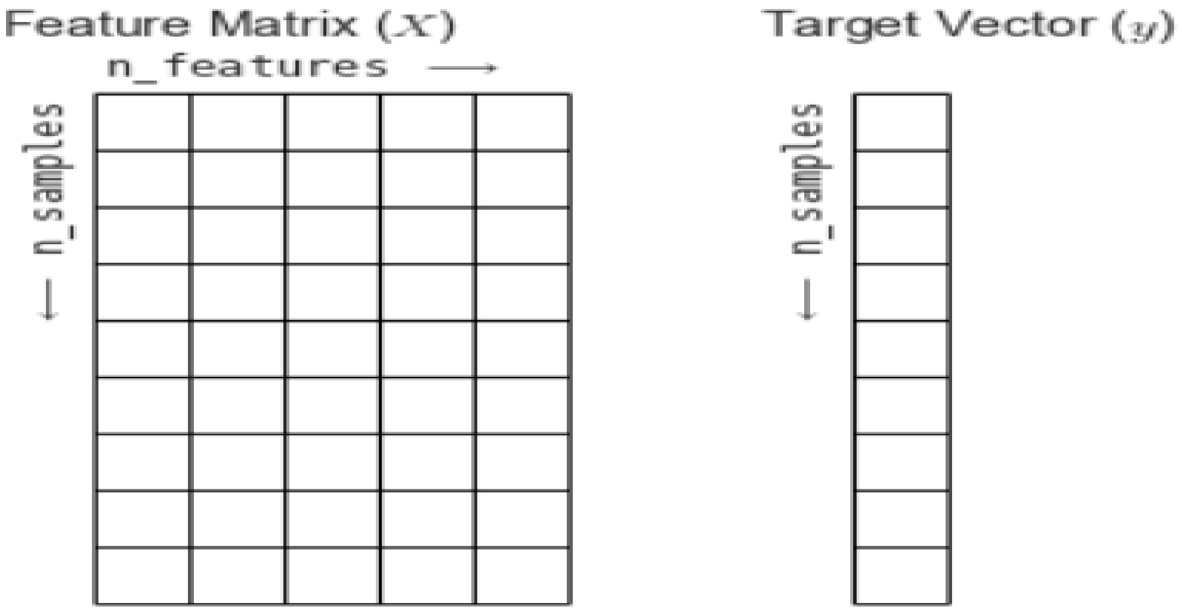
2. Choose model hyperparameters. *An important point is that a class of model is not the same as an instance of a model*.
    - hyperparameters are parameters that must be set before the model is fit to data
    - In Scikit-Learn, hyperparameters are chosen by passing values at model instantiation.
    - model = LinearRegression(fit_intercept=True)

Supervised learning example: Simple linear regression

3. <u>Arrange data into a features matrix and target vector</u>.
   - Make two-dimensional features matrix (X) and
     a one-dimensional target array (Y)
   - Make the data x into a matrix of size [n_samples, n_features].

     X = x[:, 0:4]

     Y=y[:,4]



Supervised learning example: Simple linear regression

4. Fit the model to your data.
   - apply model to data using fit() method

     model.fit(X , y)

   Final: LinearRegression()

   - fit() command causes a number of model-dependent internal computations to take place, and the results of these computations are stored in model specific attributes
   - In Scikit-Learn, by convention all model parameters that were learned during the fit() process have trailing underscores

```
In[10]: model.coef_

Out[10]: array([ 1.9776566])

In[11]: model.intercept_

Out[11]: -0.9033107255311635
```

Supervised learning example: Simple linear regression

4. Fit the model to your data.(contd..)

- The two parameters represent the slope and intercept of the simple linear fit to the data. In our data definition, its very close to the input slope of 2 and intercept of –1
- In general, Scikit-Learn does not provide tools to draw conclusions from internal model parameters themselves: interpreting model parameters is much more a *statistical modeling* question than a *machine learning* question.
- Machine learning rather focuses on what the model predicts.

Supervised learning example: Simple linear regression

5. Predict labels for unknown data.
   - Once the model is trained, the main task of supervised machine learning is to evaluate it based on what it says about new data that was not part of the training set.
   - In Scikit-Learn, the predict() method is used.

     yfit = model.predict (Xfit)

First Machine Learning Project using Iris dataset

Hello world program of machine learning

"classification of iris flowers"

Iris virginica

Iris setosa

Iris versicolor

Question

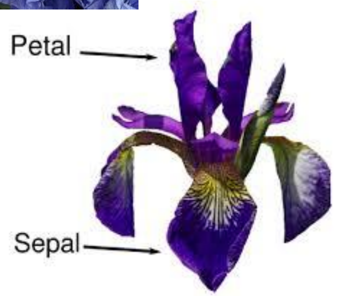
- After looking at new flower in the field, could we make a good prediction about its species from its measurements?

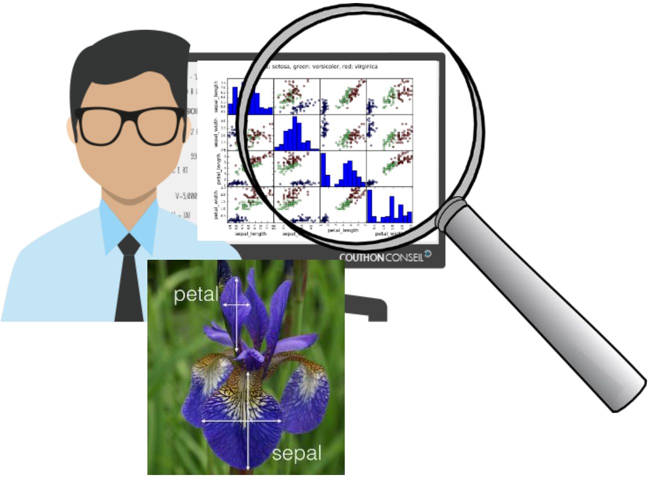
Python

Iris virginica

Iris setosa

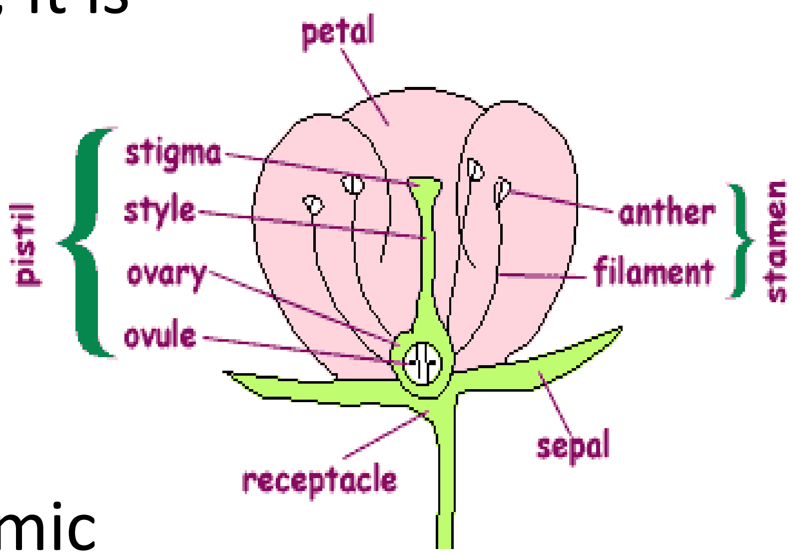
Iris versicolor

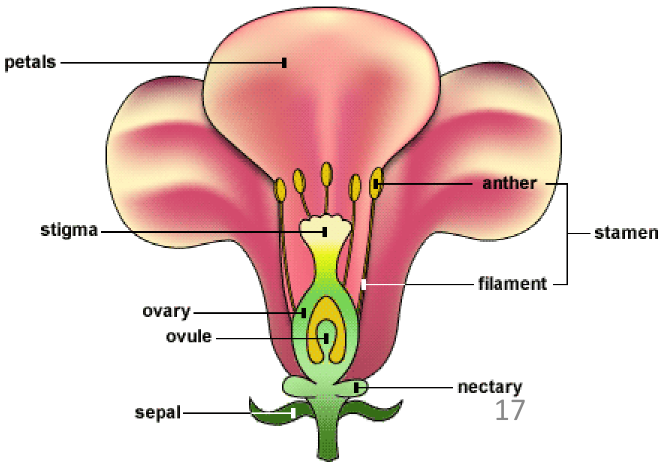




Iris dataset

- The Iris dataset is a classic dataset from the 1930s; it is one of the first modern examples of statistical classification.
- The setting is that of Iris flowers, of which there are multiple species that can be identified by their morphology.
- Today, the species would be defined by their genomic signatures, but in the 1930s, DNA had not even been identified as the carrier of genetic information.
- The following four attributes of each plant were measured:
    - Sepal length , Sepal width, Petal length, Petal width





Iris dataset

- Generally, any measurement from our data is features.
- This is the supervised learning or classification problem; given labeled examples, we can design a rule that will eventually be applied to other examples.
- Other modern application examples of Pattern classification : Optical Character Recognition (OCR) in the post office, spam filtering in our email clients(spam messages vs “ham” {= not-spam} messages), barcode scanners in the supermarket, etc

Hello World of Machine Learning with Iris

- The best small project to start with on a new tool is the classification of iris flowers. *why iris dataset*
  - Attributes are numeric so you have to figure out how to load and handle data.
  - It is a classification problem, allowing to practice with perhaps an easier type of supervised learning algorithm.
  - It is a *multi-class classification problem* (multi-nominal) that may require some specialized handling.
  - It only has 4 attributes and 150 rows, meaning it is small and easily fits into memory (and a screen or A4 page).
  - All of the numeric attributes are in the same units and the same scale, not requiring any special scaling or transforms to get started.

Iris Dataset

- Iris dataset contains 150 observations of iris flowers.
- Has four columns of measurements of the flowers in centimeters.
- The fifth column is the species of the flower observed.
- All observed flowers belong to one of three species

```
import pandas as pd
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
a = pd.read_csv('data/Iris.csv', names=names, header = None)
irisDataframe = pd.DataFrame(a)
iris = irisDataframe.values
```

```
#dataset dimension - rows x columns
irisDataframe.shape
```

```
(150, 5)
```

```
#peep into the dataset
irisDataframe.head()
```

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Inputs from : machinelearningmastery, google, kaggle,etc

Summarize dataset

- Take statistical summary using describe().
- Grouping the rows/records based on class of flower, using

  irisDataframe.groupby('class').size()

```
#Statistical Summary of the dataset
irisDataframe.describe()
```

	sepal-length	sepal-width	petal-length	petal-width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```
#check number of instances/rows in each class
#classwise distribution of rows
irisDataframe.groupby('class').size()
```

```
class
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

Create a Validation Dataset

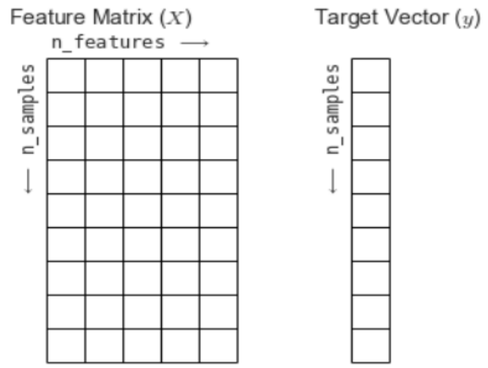
Split the loaded dataset into two:

- 80% of which we will use to train our models and
- 20% that we will hold back as a validation dataset.

training data in the

- *X_train* and *Y_train* for preparing models and
- *X_validation* and *Y_validation* sets

Arranging data into a features matrix and target vector



ning in Python

```python
# Split-out validation dataset
from sklearn import model_selection
irisarray = irisDataframe.values
X = irisarray[:,0:4]
Y = irisarray[:,4]
validation_size = 0.20
seed = 7
X_train, X_validation, Y_train, Y_validation =
    model_selection.train_test_split(X, Y, test_size=validation_size, random_state=seed)
```

```python
print ("irisDataframe.shape=",irisDataframe.shape, " irisDataframe.size =", irisDataframe.size)
print ("X_train.shape      =",X_train.shape, " X_train.size =", X_train.size)
print ("X_validation.shape =",X_validation.shape, "  X_validation.size =", X_validation.size)
print ("Y_train.shape      =",Y_train.shape, "   Y_train.size =", Y_train.size)
print ("Y_validation.shape =",Y_validation.shape, "    Y_validation.size =", Y_validation.size)
```

```
irisDataframe.shape= (150, 5)  irisDataframe.size = 750
X_train.shape      = (120, 4)  X_train.size = 480
X_validation.shape = (30, 4)   X_validation.size = 120
Y_train.shape      = (120,)    Y_train.size = 120
Y_validation.shape = (30,)     Y_validation.size = 30
```

Test Harness

- use 10-fold cross validation to estimate accuracy.
- This will split the dataset into 10 parts, train on 9 and test on 1 and repeat for all combinations of train-test splits.
- use '*accuracy*' metric to evaluate models.
  - This is a ratio of the number of correctly predicted instances in divided by the total number of instances in the dataset multiplied by 100 to give a percentage (e.g. 95% accurate).

Evaluate 6 different algorithms:

- Logistic Regression (LR)
- Linear Discriminant Analysis (LDA)
- K-Nearest Neighbours (KNN).
- Classification and Regression Trees (CART).
- Gaussian Naive Bayes (NB).
- Support Vector Machines (SVM).

Its good mix of simple linear

(LR and LDA), nonlinear

(KNN, CART, NB and SVM) algorithms.

To ensures the results are directly comparable, reset the random number seed before each run to ensure that the evaluation of each algorithm is performed using exactly the same data splits.

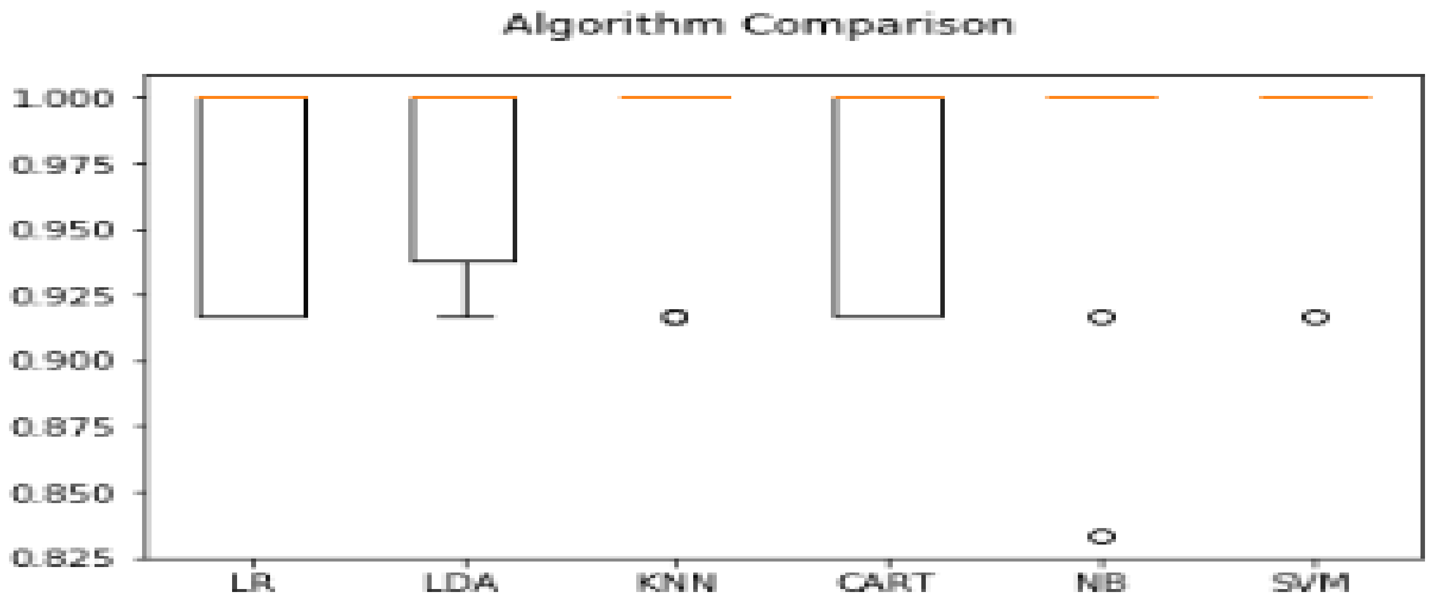
```
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
# Test options and evaluation metric
seed = 7
scoring = 'accuracy'
# Spot Check Algorithms
models = []
models.append(('LR',  LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
# evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=seed)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: 0.966667 (0.040825)
LDA: 0.975000 (0.038188)
KNN: 0.983333 (0.033333)
CART: 0.966667 (0.040825)
NB: 0.975000 (0.053359)
SVM: 0.991667 (0.025000)
```

Compare algorithms

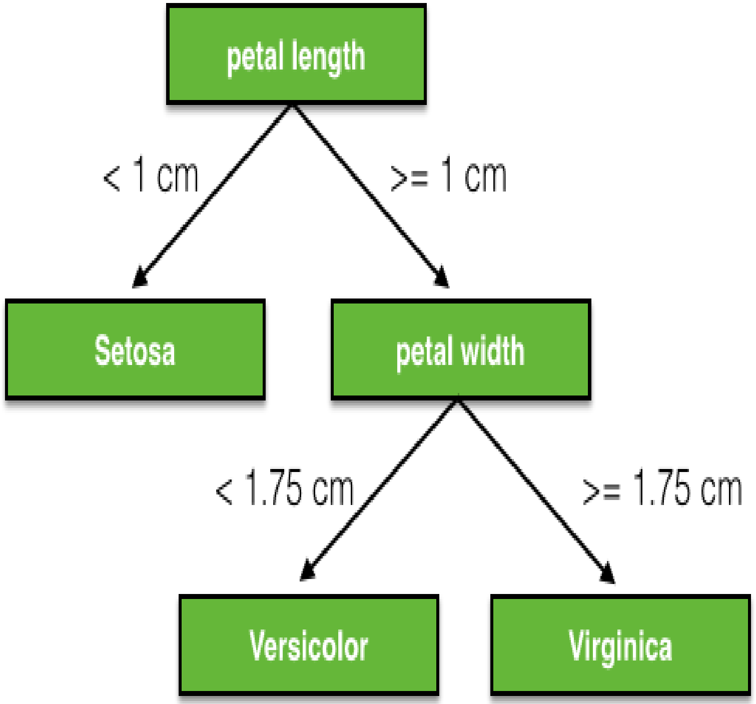
```
LR: 0.966667 (0.040825)
LDA: 0.975000 (0.038188)
KNN: 0.983333 (0.033333)
CART: 0.966667 (0.040825)
NB: 0.975000 (0.053359)
SVM: 0.991667 (0.025000)
```

```python
# Compare Algorithms
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



Fit the model

Fit the model to your data




```python
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
# Make predictions on validation dataset
knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)
predictions = knn.predict(X_validation)
print("Accuracy Score :", accuracy_score(Y_validation, predictions))
print("Confusion Matrix : \n",confusion_matrix(Y_validation, predictions))
print("Classification Report :\n",classification_report(Y_validation, predictions))
```

```
Accuracy Score : 0.9
Confusion Matrix :
 [[ 7  0  0]
 [ 0 11  1]
 [ 0  2  9]]
Classification Report :
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       0.85      0.92      0.88        12
 Iris-virginica       0.90      0.82      0.86        11

    avg / total       0.90      0.90      0.90        30
```

Loss Functions

- We might have different loss functions. It might be that one type of error is much more costly than another. In a medical setting, false negatives and false positives are not equivalent.
- A **false negative** (when the result of a test comes back negative, but that is false) might lead to the patient not receiving treatment for a serious disease.
- A **false positive** (when the test comes back positive even though the patient does not actually have that disease) might lead to additional tests for confirmation purposes or unnecessary treatment (which can still have costs, including side effects from the treatment).
- *With spam filtering, we may face the same problem; incorrectly deleting a non-spam e-mail can be very dangerous for the user, while letting a spam e-mail through is just a minor annoyance.*

```
Optimization terminated successfully.
         Current function value: 0.555865
         Iterations 7
                          Results: Logit
=====================================================================
Model:              Logit            No. Iterations:   7.0000
Dependent Variable: y                Pseudo R-squared: 0.198
Date:               2018-09-10 12:38 AIC:              56879.2425
No. Observations:   51134            BIC:              57020.7178
Df Model:           15               Log-Likelihood:   -28424.
Df Residuals:       51118            LL-Null:          -35443.
Converged:          1.0000           Scale:            1.0000
---------------------------------------------------------------------
                      Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
---------------------------------------------------------------------
euribor3m            -0.4488   0.0074 -60.6837 0.0000 -0.4633 -0.4343
job_blue-collar      -0.2060   0.0278  -7.4032 0.0000 -0.2605 -0.1515
job_housemaid        -0.2784   0.0762  -3.6519 0.0003 -0.4278 -0.1290
marital_unknown       0.7619   0.2244   3.3956 0.0007  0.3221  1.2017
education_illiterate  1.3080   0.4346   3.0096 0.0026  0.4562  2.1598
month_apr             1.2863   0.0380  33.8180 0.0000  1.2118  1.3609
month_aug             1.3959   0.0411  33.9688 0.0000  1.3153  1.4764
month_dec             1.8084   0.1441  12.5483 0.0000  1.5259  2.0908
month_jul             1.6747   0.0424  39.5076 0.0000  1.5916  1.7578
month_jun             1.5574   0.0408  38.1351 0.0000  1.4773  1.6374
month_mar             2.8215   0.0908  31.0891 0.0000  2.6437  2.9994
month_may             0.5848   0.0304  19.2166 0.0000  0.5251  0.6444
month_nov             1.2725   0.0445  28.5720 0.0000  1.1852  1.3598
month_oct             2.7279   0.0816  33.4350 0.0000  2.5680  2.8878
poutcome_failure     -0.2797   0.0351  -7.9753 0.0000 -0.3485 -0.2110
```