Lecture 11 Machine Learning

IDS 400
Programming for Data Science in Business

Assignment

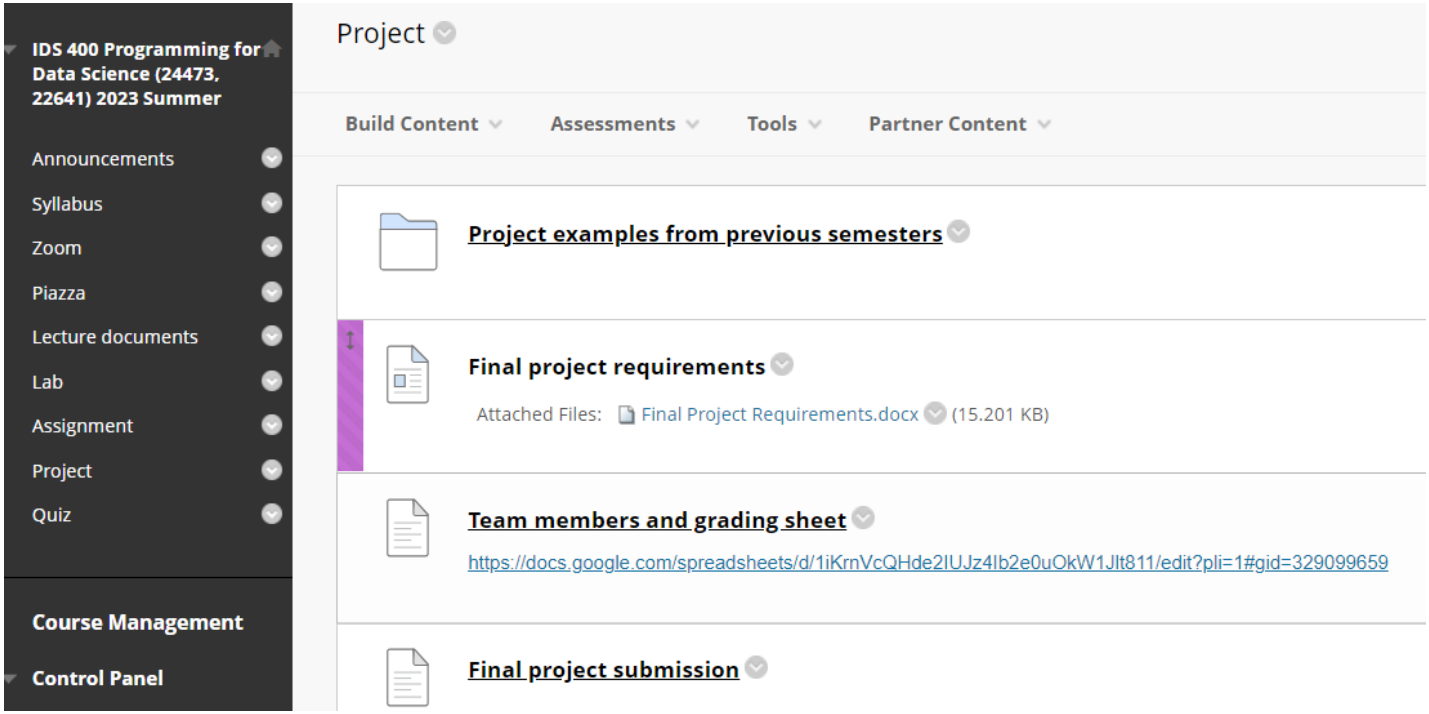
Assignment 7 is available on Blackboard.

Tentative schedule

Date	Lecture Number	Topics
08/24	Lecture 1	Introduction
08/31	Lecture 2	Basic
09/07	Lecture 3	Condition
09/14	Lecture 4	Loop
09/21	Lecture 5	String + **Quiz 1**
09/28	Lecture 6	Type
10/05	Lecture 7	Function
10/12	Lecture 8	File + **Quiz 2**
10/19	Lecture 9	Pandas
10/26	Lecture 10	Numpy
11/02	Lecture 11	Machine Learning
11/09	Lecture 12	Visualization
11/16	Lecture 13	Web Scraping & Deep Learning
11/23	***Thanksgiving***	***No lecture***
11/30	**Final presentation**	In class presentation
12/05	**Project submission due**	

Project

- Each Group need to submit their final project ***topic*** on the google sheet (link is on Piazza/Blackboard) **Before the next lecture (Nov 9th)**



Project

- Please refer to the Project Sections on blackboard for the detailed requirements of the project.

Project presentation

- Each group will have **maximum 8 minutes** to present the project,and **2 min** for **Q&A.**
- You can decide who will present the project from your group; however, all members **must** attend the final presentation.
- **Every student is asked to grade other groups' projects**, and grading is based on four aspects: ***Idea, Techniques, Execution, Presentation***. The grading sheet is available on Blackboard.
- ***Audiences are highly encouraged to ask questions***. Answering questions should also be considered in the presentation evaluation.

Project presentation

- What should be included in the presentation:
  - **Motivation**: What are the research questions and why your project is important.
  - **Data**: What dataset are you using, how did you collect the data.
  - **Data exploration:** What are the dependent and independent variables and if you find any pattern in the data (usually through graphs)
  - **The analytics component:** Explaining why you choose different models and which model performs the best (and why). If visualization is the focus of your project, you should explain what business insight you can derive from this specific type of visualization technique.
  - **Conclusion/Business insight**: What is your finding, how your project help the business improve their business decisions.
- Given the limited time of presentation, you don't need to show your python code during the presentation.

Project submission

- **Project submission:**

✓ **report (PDF or Word doc.)**

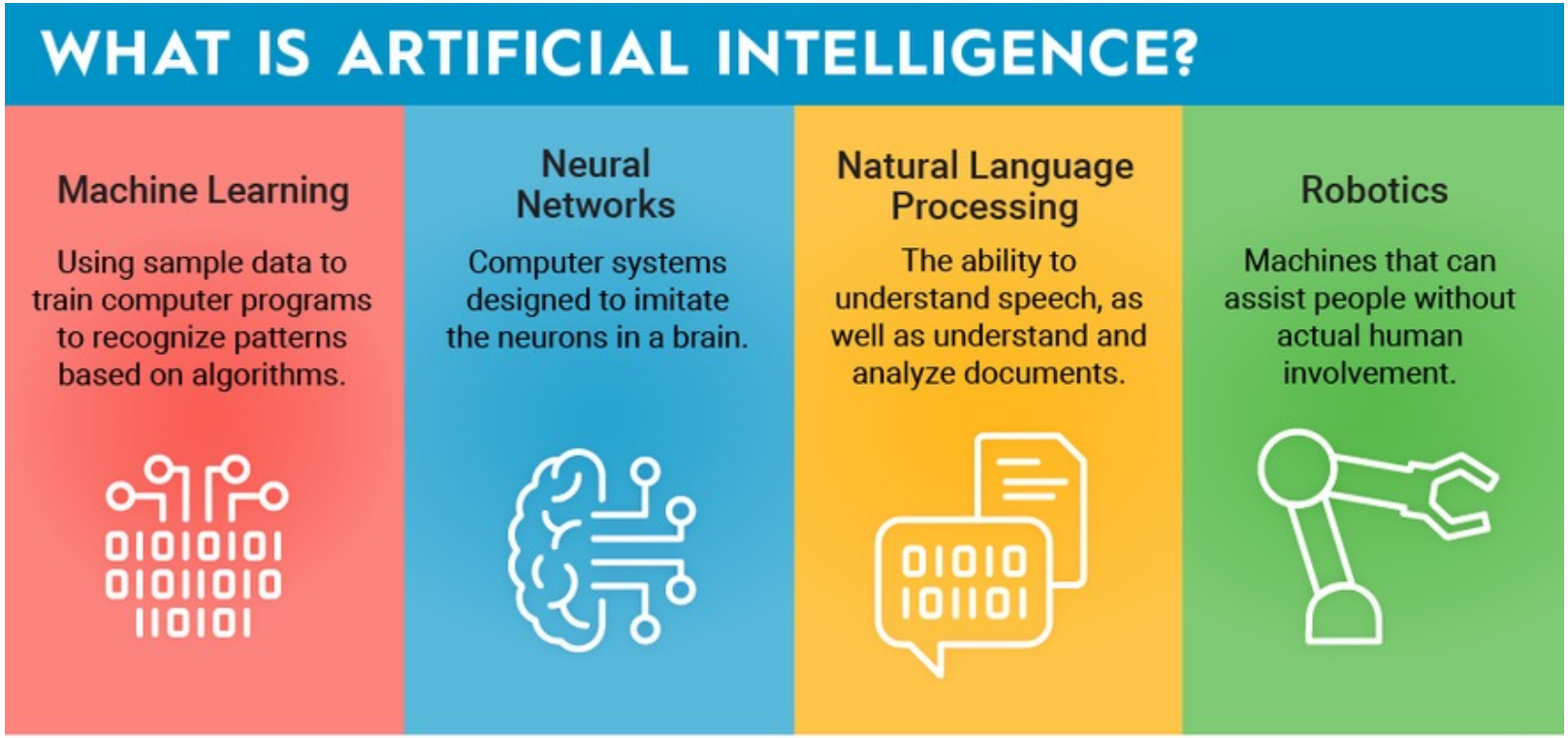
✓ **code (.ipynb files)**

✓ **presentation slides (PDF or PPT)**

- **Final project submission Due: Dec. 5th Thursday, 6:00 pm**
- Notice that given the report is due several days after the presentation, you can keep improving your project in the report after the presentation.

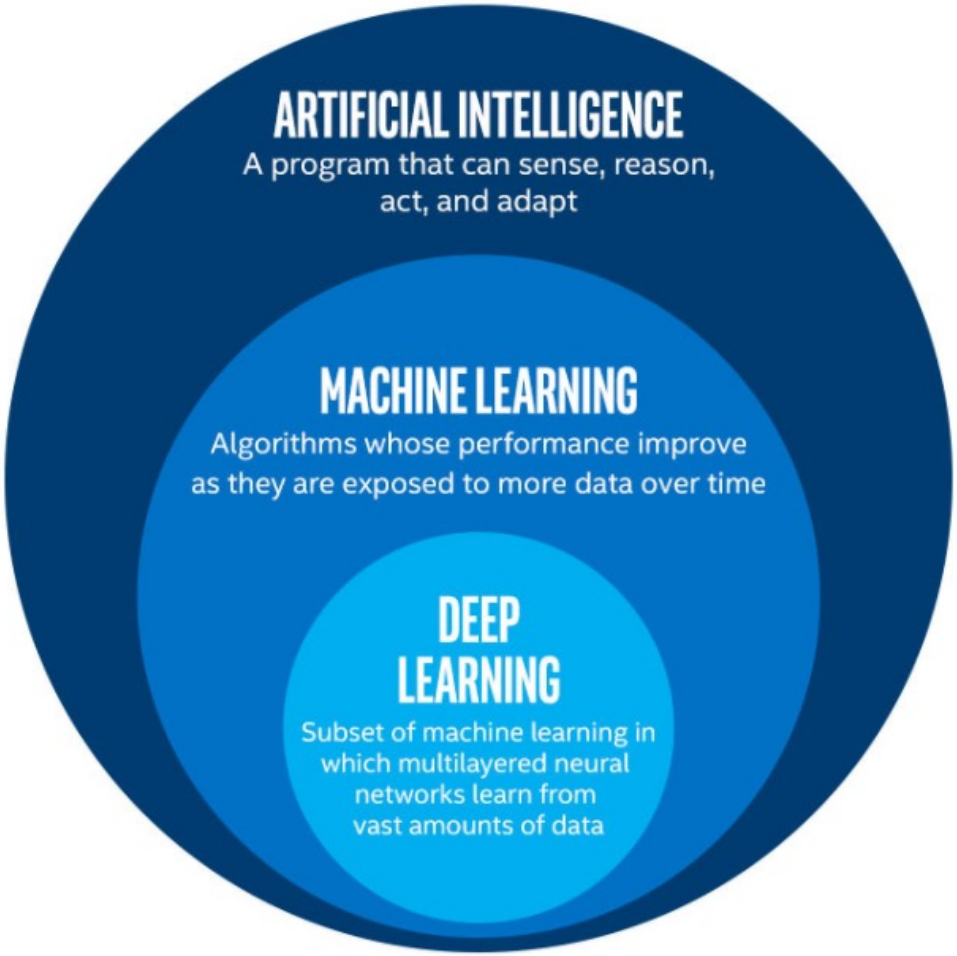
What is Artificial Intelligence?

Artificial intelligence uses machines -- generally, computers -- to mimic human intelligence



What is machine learning?

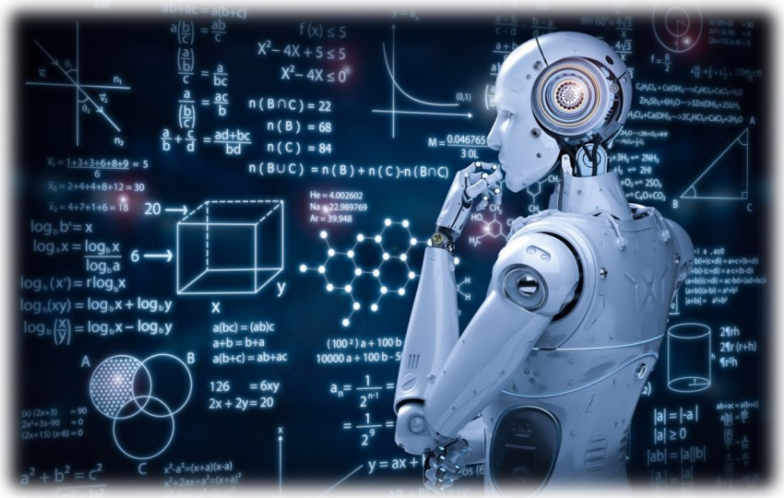
Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy.



What is machine learning?

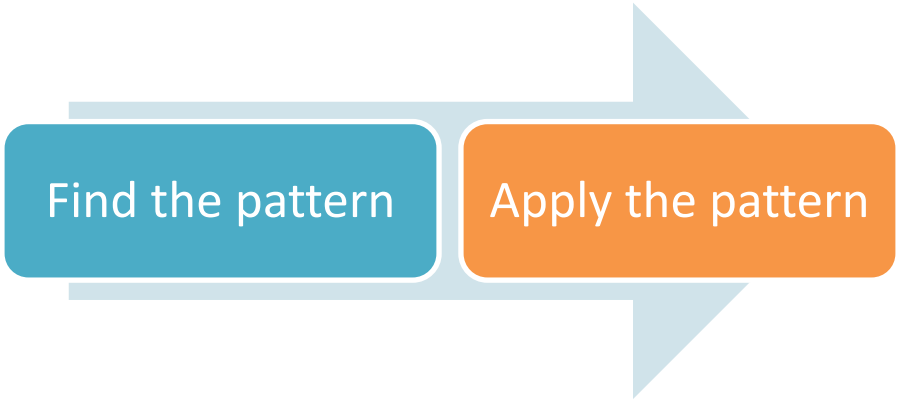
- Recommendation Systems
  - Netflix, YouTube, and Spotify
- Search Engines
  - Google, Bing and Yahoo
- Social Media feeds
  - Facebook and Twitter
- Voice Assistants
  - Siri and Alexa

The list goes on…



What is machine learning?

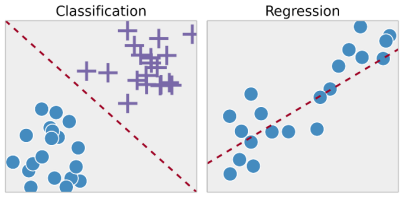
- Learn from ***data*** and make predictions
  - We hope machine will eventually improve the learning by itself without human interference.



What is machine learning?

Supervised learning

- The data is labeled to tell the machine exactly what patterns it should look for.

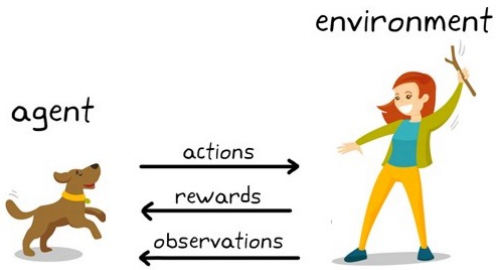


Unsupervised learning

- The machine just looks for whatever patterns it can find.

Reinforcement learning

- It learns by trial and error to achieve a clear objective.



Some common machine learning models

- Logistic Regression
- Decision Tree
- Support Vector Machine
- Random Forest
- Nearest Neighbors
- Neural Networks
- ……

Machine Learning Algorithm

- Regression
  - Supervised
  - Continuous numeric
- Classification
  - Supervised
  - Categorical
- Clustering
  - Unsupervised
  - Grouping observations

Machine Learning Algorithm

- Regression
  - Regression Model
  - Decision Tree
- Classification
  - Logistic Regression
  - Classification Tree
  - SVM
- Clustering
  - K-means
  - Hierarchical

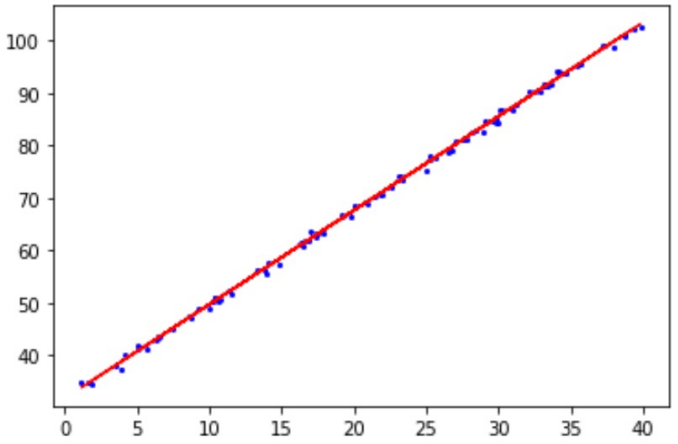
Machine learning packages

- Mlpy
- SciKit-learn
- NLTK
- PyML
- Deep learning
  - TensorFlow
  - Theano
  - PyTorch

Basic of Regression

- Example from last lecture

```
from pylab import *
from numpy import *
from scipy.stats import *

# data generation
input1 = random.randint(0,40,100)
x = input1+rand(100)
y = (input1*1.8+32)+rand(100)

# linear regression
slope,intercept,r_value,p_value,slope_std_error= stats.linregress(x,y)
# estimated y
y_modeled= x*slope+intercept

# plot true and modeled results
plot(x,y,'ob',markersize=2)
plot(x,y_modeled,'-r',linewidth=1)
show()
```



Basic of Regression

- From previous class on Numpy, we used functions like **stats.linregress**. It provides outputs such as *p-value*, *r-value*. What do they mean?

Basics of Regression

Consider a simple linear regression in which you want to find out how exercises help cholesterol level:

- There is a true relationship between exercise and cholesterol change.
- This relationship can be described as:

$$Cholesterol\ Change = \beta_0 + \beta_1 * Exercise + \varepsilon$$

- Or more generally

$$y = \beta_0 + \beta_1 x + \varepsilon$$

$\beta_1 \equiv Slope;\ \beta_0 \equiv Intercept;\ \varepsilon \equiv Error\ term$

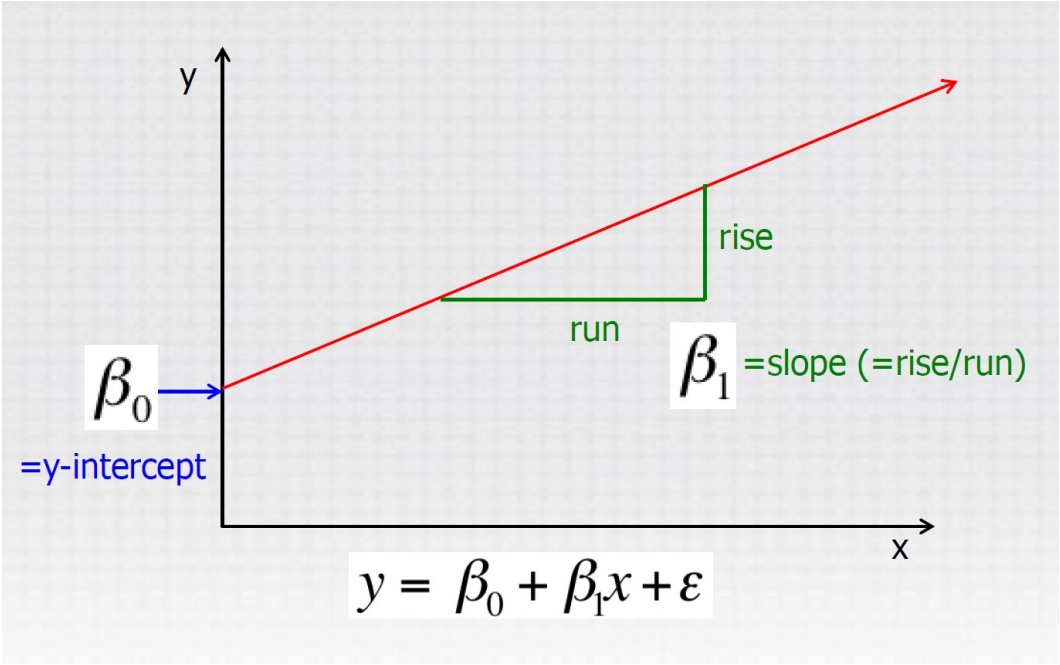
- Our estimated model is:

$$\hat{y} = b_0 + b_1 x$$

$\hat{y} \text{–} estimated\ DV;\ b \ldots regression coefficients$

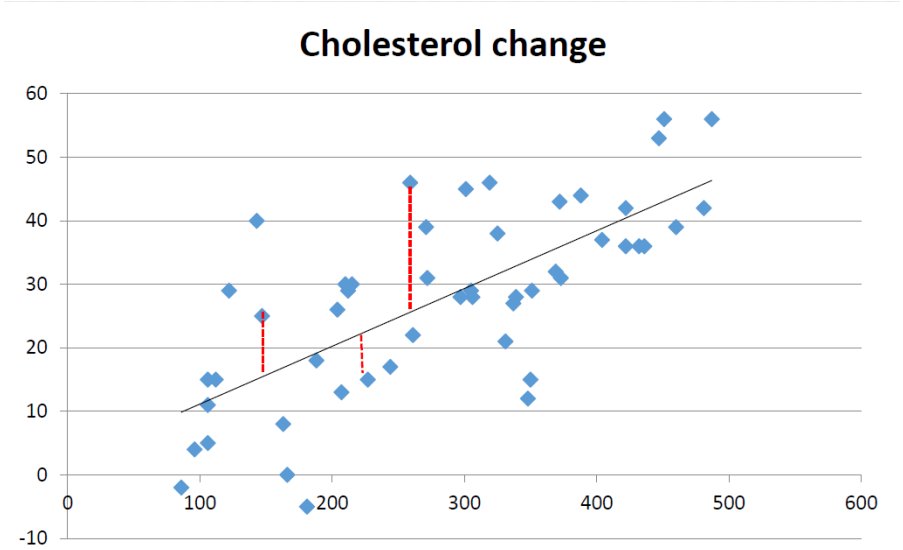
Simple Linear Regression Model

- Note that both $\beta_0$and β_1 are population parameters which are usually unknown and hence estimated from the data.

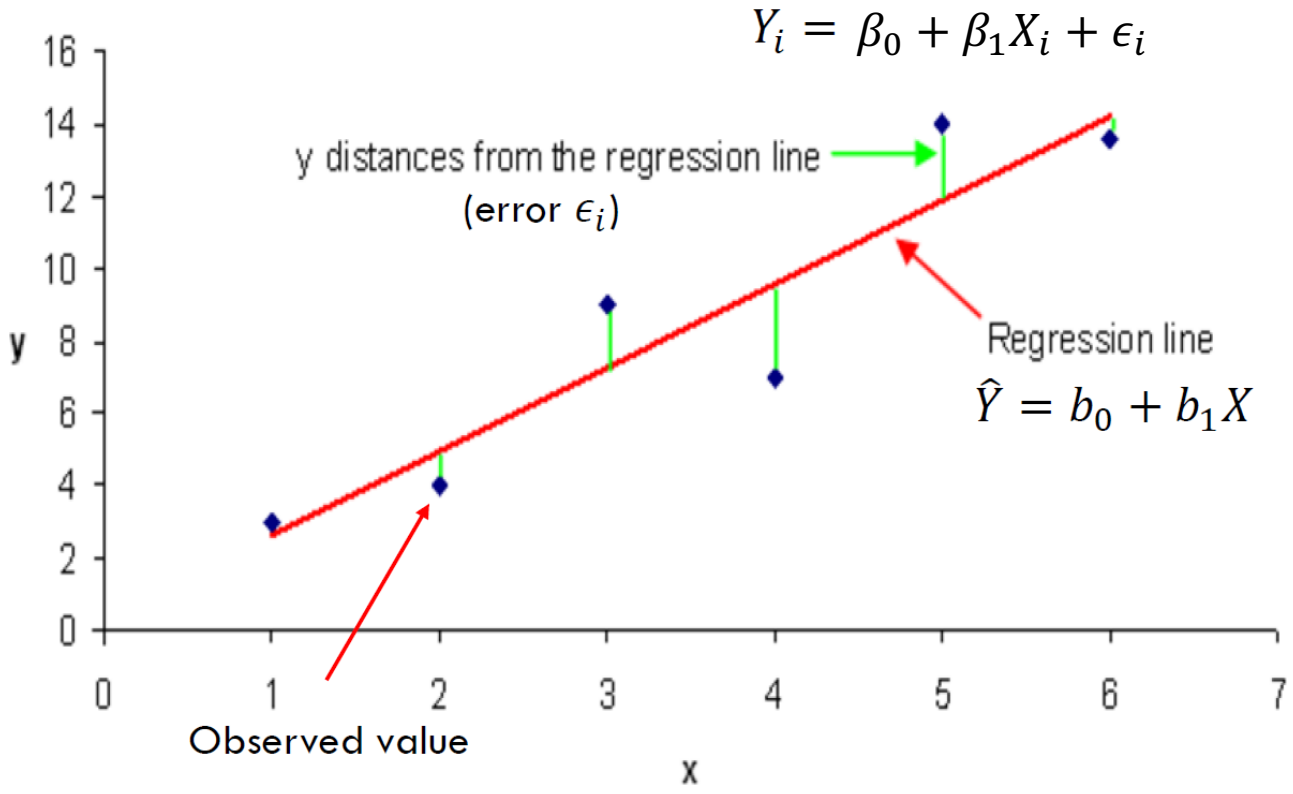


The regression line

- We try to capture the relationship between the two variables with a single line.
  - No line can hit all the points. We also don't know the ***true line***.
  - The amount we miss is called ***error (residual).***



The regression line



Basic of Regression

- ***P-value***

In regression, we have p-value for each estimated coefficients (βs). If p-value is small, the corresponding coefficient can be regarded to be significant (different from zero).

- ***R-squared***

R-squared is a goodness-of-fit measure for linear regression models. Usually, the larger the R^2, the better the regression model fits your observations.

$$r^2 = \frac{\text{Explained variation}}{\text{Total variation}} = \frac{\sum_{i=1}^{n}(y_i - \bar{y})^2 - \sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

- ***Standard error of the estimate***

The standard error of the estimate is a measure of the accuracy of predictions.

$$\sigma_{est} = \sqrt{\frac{\sum(y_i - \hat{y}_i)^2}{N}}$$

Basic of Regression

- Another way to perform linear regression using Python.

```
import numpy as np
from sklearn.linear_model import LinearRegression
X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])

# y = 1 * x_0 + 2 * x_1 + 3
y = np.dot(X, np.array([1, 2])) + 3
reg = LinearRegression().fit(X, y)
```

```
# Return the R^2 of the prediction.
reg.score(X, y)
```

```
1.0
```

```
# Estimated coefficients for the linear regression problem
reg.coef_
```

```
array([1., 2.])
```

```
# Independent term in the linear model
reg.intercept_
```

```
3.000000000000001
```

```
# Predict using the linear model.
reg.predict(np.array([[3, 5]]))
```

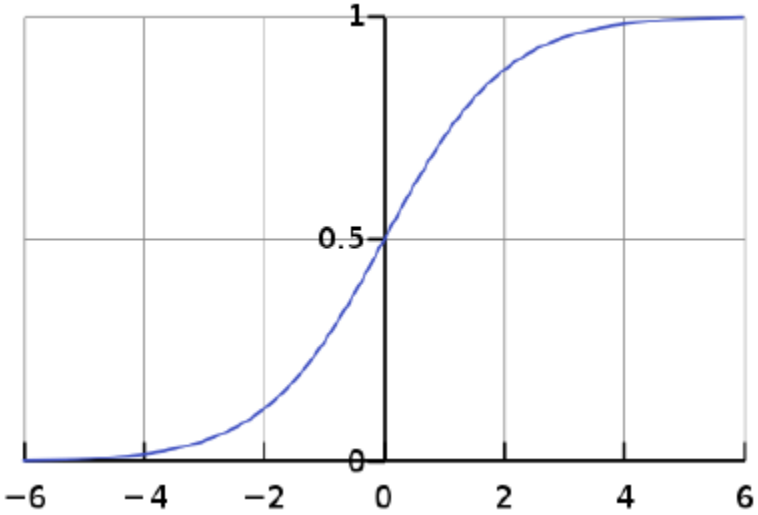
```
array([16.])
```

Logistic Regression

- Binary outcome
  - Whether a patient has certain disease
  - Propensity of purchasing
  - Probability of credit card fraud

$$Prob(D = 1) = \frac{e^{\beta x}}{1 + e^{\beta x}}$$

- Why linear model doesn't work for this?

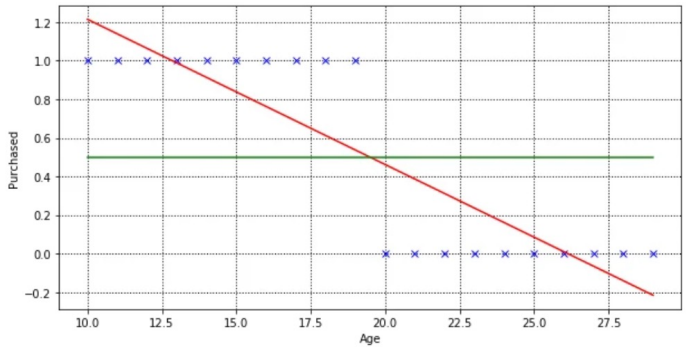


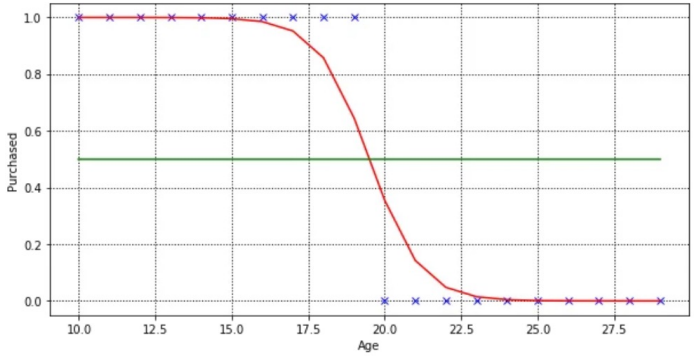
Logistic Regression

P1: In a binary classification problem, what we are interested in is the probability of an outcome occurring. Probability is ranged between 0 and 1, where the probability of something certain to happen is 1, and 0 is something unlikely to happen. But in linear regression, we are predicting an absolute number, which can range outside 0 and 1

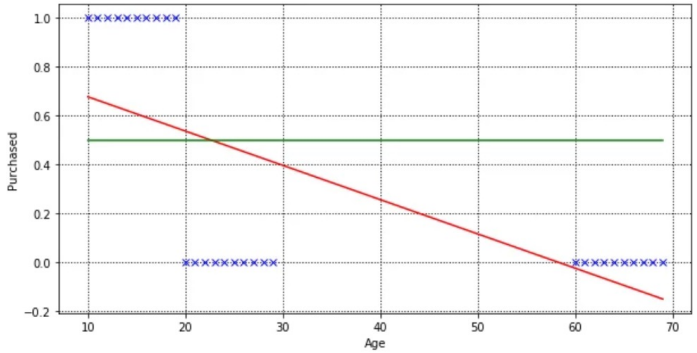
Why linear model doesn't work for this?

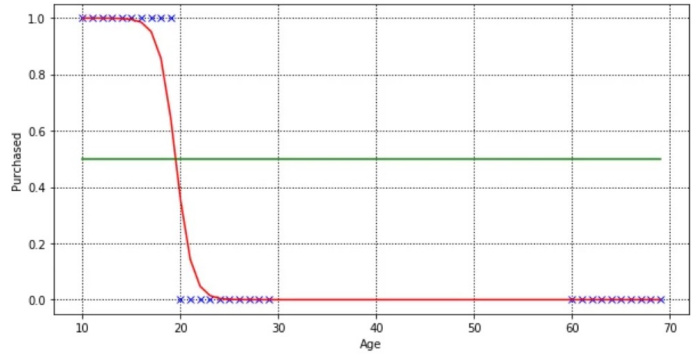
- Predicted value is continuous, not probabilistic.





- Sensitive to imbalance data





P2: Add 10 more customers age between 60 to 70, and train our linear regression model, finding the best fit line. Our linear regression model manages to fit a new line, but if you look closer, some customers (age 20 to 22) outcome are predicted wrongly. logistic regression manages to classify all data points perfectly.

Example

- Let’s use an Iris dataset as an example.
- We want to categorize the type of iris based on iris characteristics (sepal width/length and petal width/length)

Implementing Logistic Regression in Python

```
# Load libraries
import pandas
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
```

You may experience FutureWarnings

```
# import warnings filter
from warnings import simplefilter

# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)
```

Load dataset

```
# Load dataset

url= "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']

dataset = pandas.read_csv(url, names=names)
```

Data Description

```
print(dataset.shape)
```

```
(150, 5)
```

```
print(dataset.head(20))
```

```
    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
5            5.4          3.9           1.7          0.4  Iris-setosa
6            4.6          3.4           1.4          0.3  Iris-setosa
7            5.0          3.4           1.5          0.2  Iris-setosa
8            4.4          2.9           1.4          0.2  Iris-setosa
9            4.9          3.1           1.5          0.1  Iris-setosa
10           5.4          3.7           1.5          0.2  Iris-setosa
11           4.8          3.4           1.6          0.2  Iris-setosa
12           4.8          3.0           1.4          0.1  Iris-setosa
13           4.3          3.0           1.1          0.1  Iris-setosa
14           5.8          4.0           1.2          0.2  Iris-setosa
15           5.7          4.4           1.5          0.4  Iris-setosa
16           5.4          3.9           1.3          0.4  Iris-setosa
17           5.1          3.5           1.4          0.3  Iris-setosa
18           5.7          3.8           1.7          0.3  Iris-setosa
19           5.1          3.8           1.5          0.3  Iris-setosa
```

Data Description

```
print(dataset.describe())
```

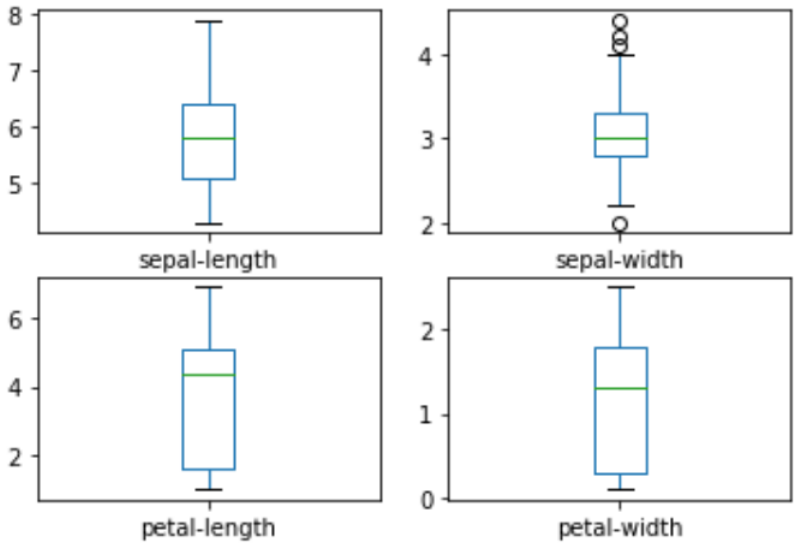
```
       sepal-length  sepal-width  petal-length  petal-width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
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
```

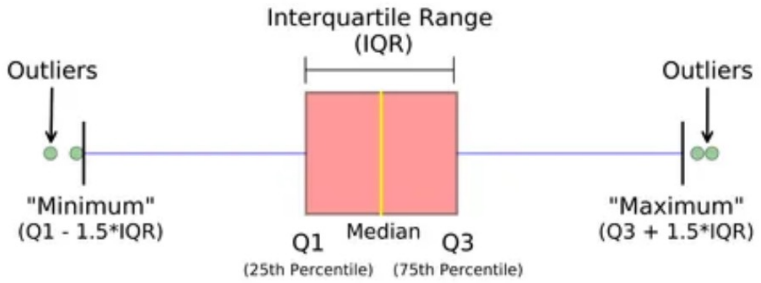
```
#number of instances for each type
print(dataset.groupby('class').size())
```

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

Visualization

```
dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
plt.show()
```

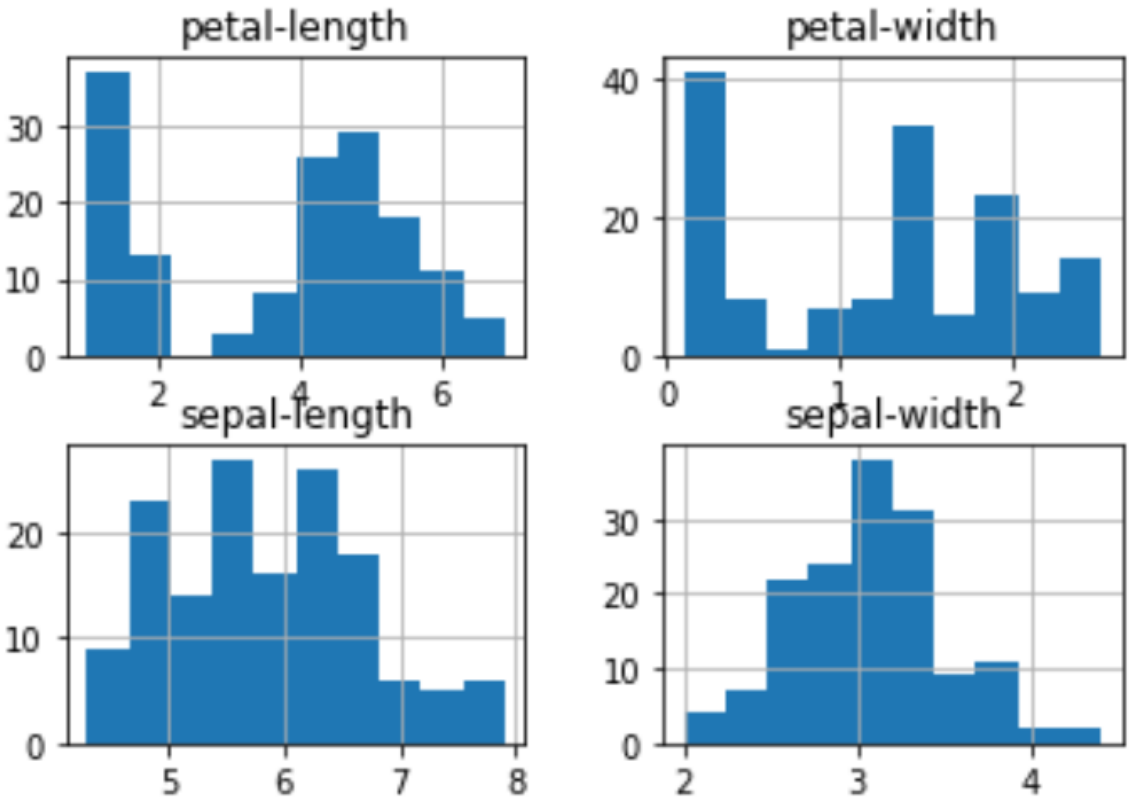




An outlier is an observation that lies an abnormal distance from other values in a random sample from a population.

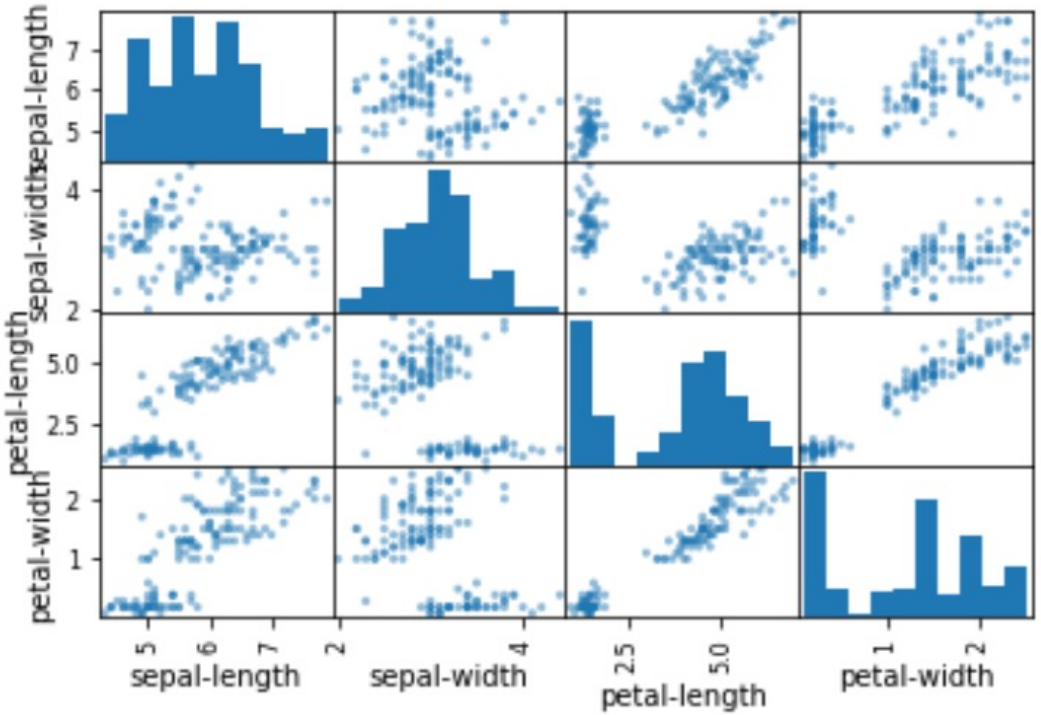
Visualization

```
#histogram
dataset.hist()
plt.show()
```



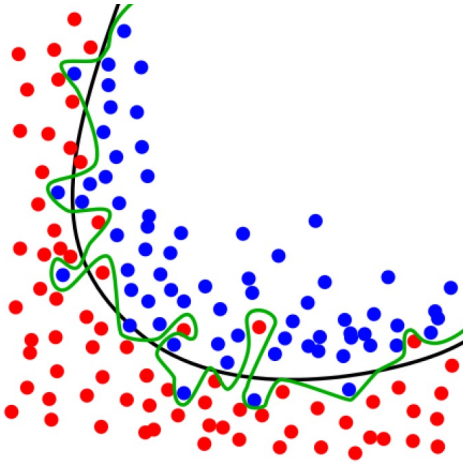
Visualization

```
#scatter plot
scatter_matrix(dataset)
plt.show()
```



Issue of overfitting

- Example:
  - The black line is the predicted classification line that separates group 1 (left) from group 2 (right).
  - But there are always noise. Some red dots on the right and some blue dots on the left



What's the problem if we train a model like the green line?

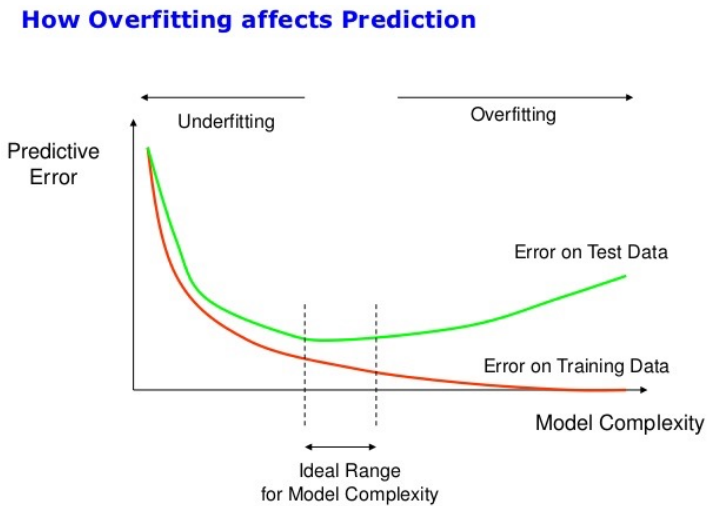
Issue of overfitting

- How do we evaluate the performance of the model?
  - We can evaluate how well the model fit the training data.
  - However, this method leads to overfitting.

- ***Overfitting***
  - The production of an analysis that corresponds too closely or exactly to a particular set of data and may therefore fail to fit additional data or predict future observations reliably.

Issue of overfitting

- Common cause of overfitting:
  - Too many parameters
  - It is fitting the noise rather than fitting the actual relationship



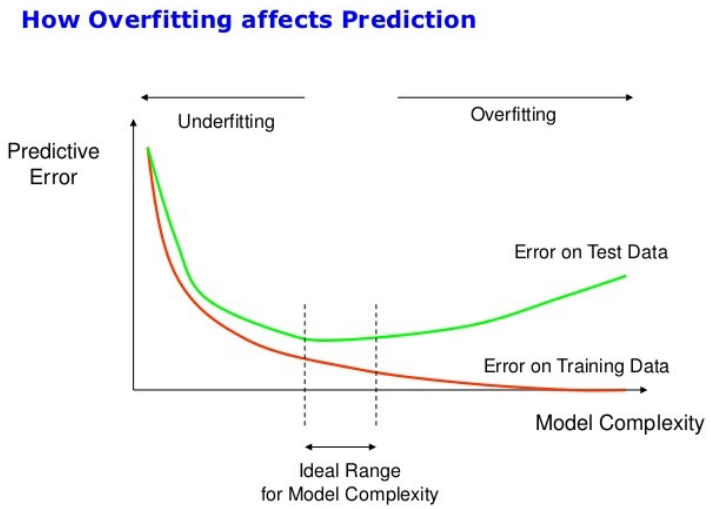
Issue of overfitting

- Common cause of overfitting:
  - Too many parameters
  - It is fitting the noise rather than fitting the actual relationship

- How to tackle Problem of Overfitting?

Cross Validation

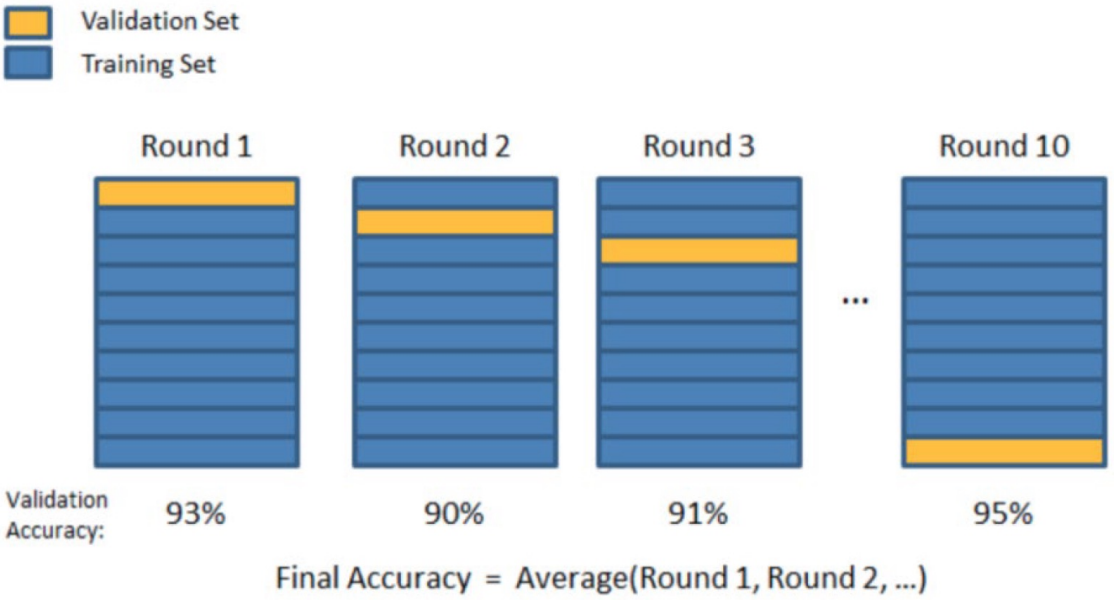
- Key challenge: we can't know how well our model will perform on new data until we actually test it.
- Solution: split our training data into training and validation set



Cross Validation

- Cross Validation is used to test the performance of a model.
- Cross Validation Steps
  1) Split your data into ***k*** equal parts, or "folds" (typically k=10).
  2) Train your model on k-1 folds (e.g. the first 9 folds).
  3) Evaluate it on the remaining "hold-out" fold (e.g. the 10th fold).
  4) Repeat steps (2) and (3) k times, each time holding out a different fold.
  5) Aggregate the performance across all ***k*** folds.

Cross Validation



Train, test, validation

Test Dataset

- The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.
- The Test dataset provides the gold standard used to evaluate the model. It is **only used once a model is completely trained** (using the train and validation sets).

Seed and random state

- ***random_state***: control the random number generator used
  - ***None (default):*** Use the global random state instance from numpy.random. Calling the function multiple times will reuse the same instance, and will produce different results.
  - ***An integer***: Use a new random number generator seeded by the given integer. Using an int will produce the same results across different calls.

```
import numpy as np
a, b = np.arange(10).reshape((5, 2)), range(5)
print(a)
print(b)
```

```
[[0 1]
 [2 3]
 [4 5]
 [6 7]
 [8 9]]
range(0, 5)
```

```
model_selection.train_test_split(a, b)
```

```
[array([[6, 7],
        [8, 9],
        [4, 5]]), array([[2, 3],
        [0, 1]]), [3, 4, 2], [1, 0]]
```

```
# Try to run it again:
model_selection.train_test_split(a, b)
```

```
[array([[2, 3],
        [8, 9],
        [0, 1]]), array([[6, 7],
        [4, 5]]), [1, 4, 0], [3, 2]]
```

```
# Then try:
model_selection.train_test_split(a, b,random_state=7)
```

```
[array([[4, 5],
        [2, 3],
        [8, 9]]), array([[0, 1],
        [6, 7]]), [2, 1, 4], [0, 3]]
```

```
model_selection.train_test_split(a, b,random_state=7)
```

```
[array([[4, 5],
        [2, 3],
        [8, 9]]), array([[0, 1],
        [6, 7]]), [2, 1, 4], [0, 3]]
```

Cross Validation

- Split train and test dataset

```
# Split-out validation dataset
array = dataset.values
X = array[:,0:4]
Y = array[:,4]
testsize= 0.20
seed = 7
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
                                                                    test_size=testsize,
                                                                    random_state=seed)
```

- Perform cross validation on train dataset

```
#estimate model using Logistic Regression
kfold= model_selection.KFold(n_splits=10, shuffle=False)
cv_results= model_selection.cross_val_score(LogisticRegression(solver='lbfgs',
                                                               max_iter=1000),
                                            X_train, Y_train, cv=kfold,
                                            scoring= 'accuracy')
#average and standard deviation of accuracies for all 10 fold models
print(cv_results.mean(),cv_results.std())
```

```
0.9833333333333332 0.03333333333333335
```

Solver is used to minimize the adjusted loss function and prevent overfitting or underfitting.

Obtaining prediction and probability

```
logistic = LogisticRegression(solver='lbfgs', max_iter=1000)
logistic.fit(X_train, Y_train).score(X_test,Y_test)
# score returns the mean accuracy on the given test data and labels.
```

```
0.8666666666666667
```

```
# predicted result
iris_y_pred=logistic.predict(X_test)
print(iris_y_pred)
```

```
['Iris-virginica' 'Iris-virginica' 'Iris-setosa' 'Iris-virginica'
 'Iris-virginica' 'Iris-setosa' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-setosa' 'Iris-versicolor' 'Iris-virginica' 'Iris-virginica'
 'Iris-setosa' 'Iris-virginica' 'Iris-setosa' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-setosa' 'Iris-setosa'
 'Iris-versicolor' 'Iris-virginica' 'Iris-versicolor' 'Iris-virginica'
 'Iris-versicolor' 'Iris-virginica' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-virginica' 'Iris-virginica']
```

```
# predicted probability
iris_y_prob=logistic.predict_proba(X_test)
print(iris_y_prob)
```

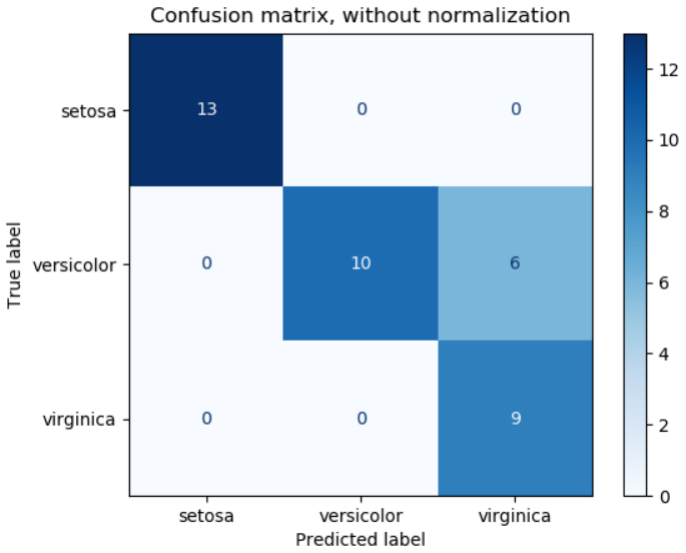
```
[[7.59943277e-04 2.46365986e-01 7.52874071e-01]
 [1.30819446e-02 7.40150487e-01 2.46767569e-01]
 [9.85960746e-01 1.40391740e-02 8.04533200e-08]
 [1.13490424e-02 7.50498899e-01 2.38152059e-01]
 [7.75899081e-03 6.64367557e-01 3.27873452e-01]
 [9.40419512e-01 5.95802953e-02 1.93006060e-07]
 [1.59111653e-03 6.00055618e-01 3.98353265e-01]
 [1.71404119e-02 9.16797815e-01 6.60617736e-02]
 [9.73345753e-01 2.66540036e-02 2.42893136e-07]
```

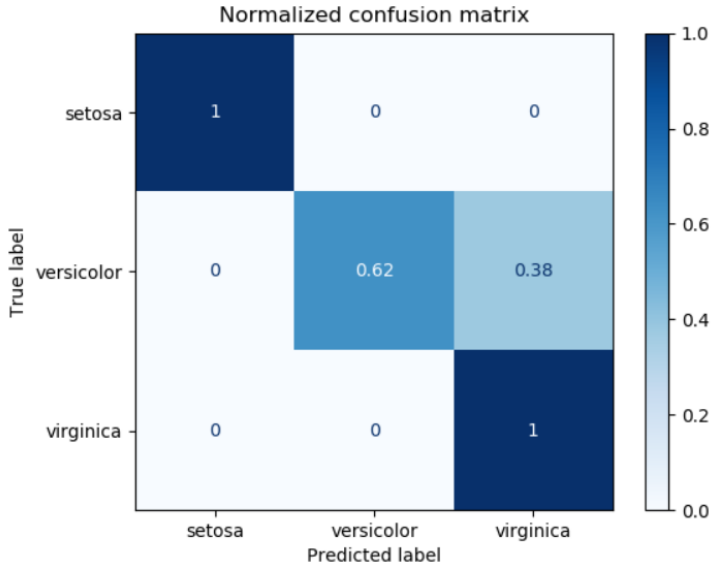
```
#performance on validation set
confusion_matrix(Y_test, iris_y_pred)
```

```
array([[ 7,  0,  0],
       [ 0, 10,  2],
       [ 0,  2,  9]])
```

Confusion matrix

- A confusion matrix is a table that is often used to describe the performance of a classification model.



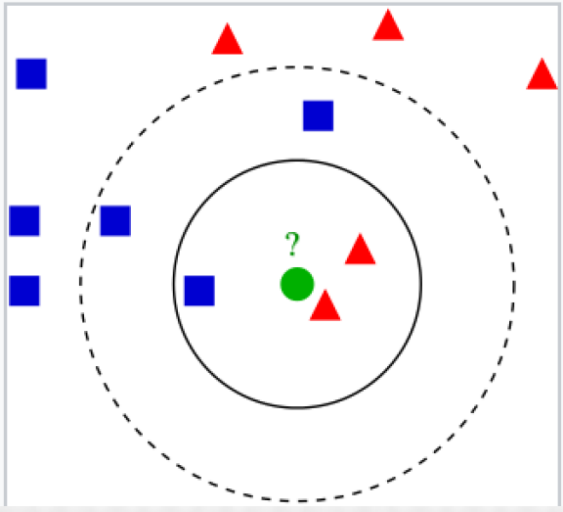


Other common machine learning algorithms

- K-nearest neighbor (KNN)
- Decision Tree
- Naïve Bayes
- Support Vector Machines

K-nearest neighbor (KNN)

- Non-parametric method
  - An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small).
- Which class the green circle should belong to?
  - K=3, examining the solid circle. Green circle belongs to red triangles class.
  - K=5, examining the dashed circle. Green circle belongs to blue square class.

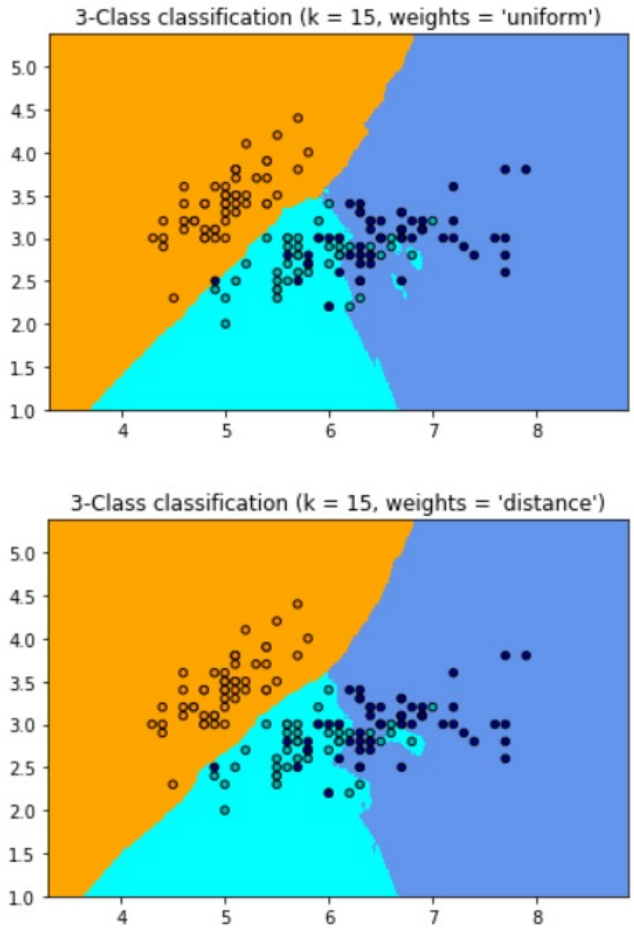


K-nearest neighbor (KNN)

- Choosing the value of K:
  - Depends on the data
  - Larger values of k reduce effect of the noise on the classification
  - Larger values of k make boundaries between classes less distinct
- KNN is sensitive to feature selection
  - Evolutionary algorithm for feature selection

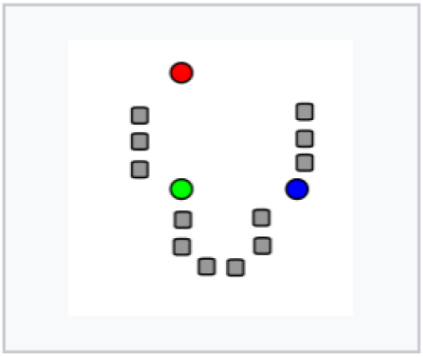
K-nearest neighbor (KNN)

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

n_neighbors = 15

# import some data to play with
iris = datasets.load_iris()

# we only take the first two features. We could avoid this ugly
# slicing by using a two-dim dataset
X = iris.data[:, :2]
y = iris.target

h = .02  # step size in the mesh

# Create color maps
cmap_light = ListedColormap(['orange', 'cyan', 'cornflowerblue'])
cmap_bold = ListedColormap(['darkorange', 'c', 'darkblue'])

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))
    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,
                edgecolor='k', s=20)
    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()
```

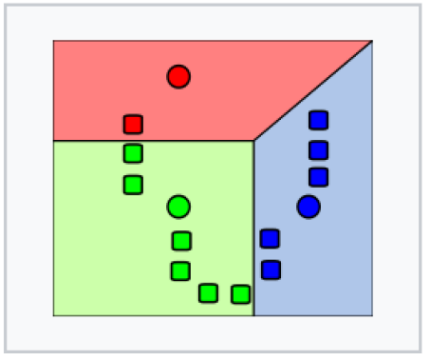


The default value, weights = 'uniform', assigns uniform weights to each neighbor. weights = 'distance' assigns weights proportional to the inverse of the distance from the query point.

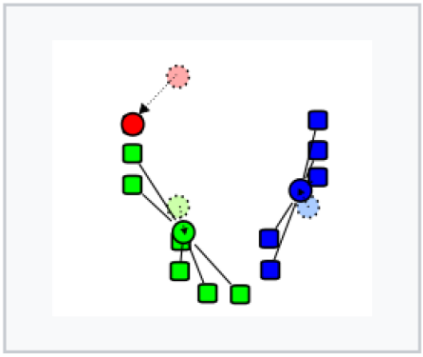
Recall K-means (last lecture)



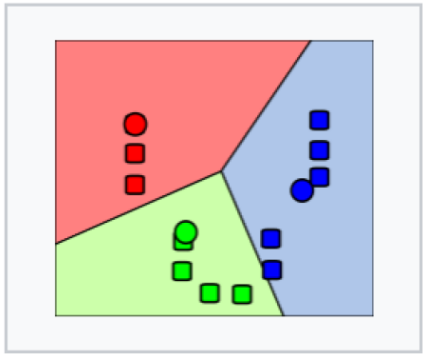
k initial "means" (in this case k=3) are randomly generated within the data domain (shown in color).



k clusters are created by associating every observation with the nearest mean.



The centroid of each of the k clusters becomes the new mean.



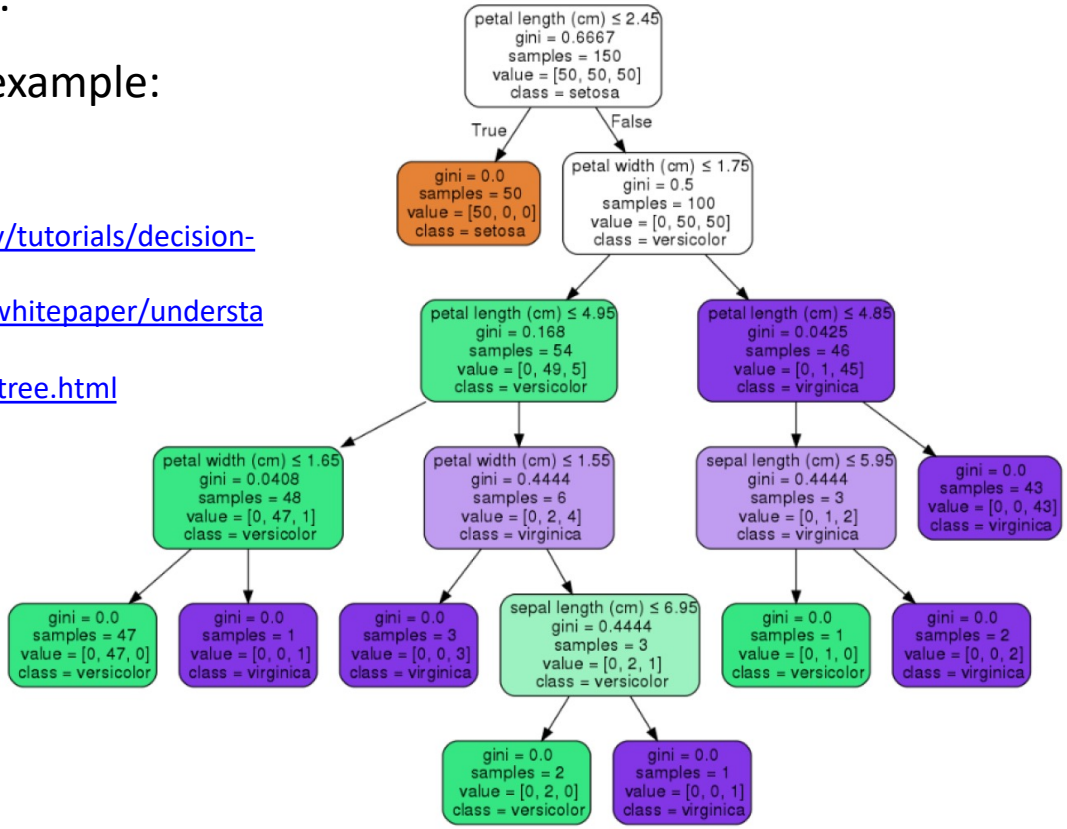
Steps 2 and 3 are repeated until convergence has been reached.

KNN vs K-means

- **K-means** is a *clustering algorithm* that tries to partition a set of points into ***K sets (clusters)*** such that the points in each cluster tend to be near each other. It is ***unsupervised*** because the points have no external classification.
- **K-nearest neighbors** is a *classification (or regression) algorithm* that in order to determine the classification of a point, combines the classification of the ***K nearest points***. It is ***supervised*** because you are trying to classify a point based on the known classification of other points.

Decision Tree

- A binary tree-structure in which every node splits into two branches based on certain variable values.
- Decision tree for our Iris example:



Tutorial:

- https://www.datacamp.com/community/tutorials/decision-tree-classification-python
- https://datascience.foundation/sciencewhitepaper/understanding-decision-trees-with-python
- https://scikit-learn.org/stable/modules/tree.html

Naïve Bayes

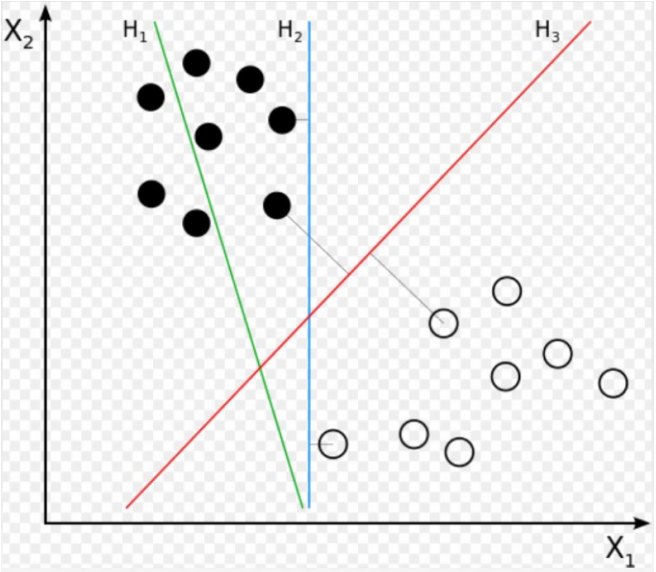
- Why “Naive”?
  - Based on the naïve assumption of conditional independence between every pair of features given the value of the class variable.
  - P(class|data) = (P(data|class) * P(class)) / P(data)
- It actually works well on spam filtering.
- Requires small amount of data and very fast.

Tutorial:

- https://scikit-learn.org/stable/modules/naive_bayes.html
- https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/
- https://www.analyticsvidhya.com/blog/2021/01/a-guide-to-the-naive-bayes-algorithm/

Support Vector Machine

- SVM produces a clear gap so that it separates two categories. This gap should be as wide as possible.
  - H1 does not separate the two classes
  - H2 separates two classes with very small margin
  - H3 separates two classes with maximum margin



Tutorial:

- https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python
- https://towardsdatascience.com/support-vector-machines-explained-with-python-examples-cb65e8172c85
- https://scikit-learn.org/stable/modules/svm.html

If you want to train multiple algorithms

```
models = []
models.append(('LR', LogisticRegression(solver='lbfgs', max_iter=1000)))
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, shuffle= False)
    cv_results= model_selection.cross_val_score(model, X_train, Y_train, cv=kfold, scoring="accuracy")
    results.append(cv_results)
    names.append(name)
    msg= "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: 0.983333 (0.033333)
KNN: 0.983333 (0.033333)
CART: 0.975000 (0.038188)
NB: 0.975000 (0.053359)
SVM: 0.983333 (0.033333)
```

Assignment Only