

Python \ AWS \ ML Training:

Machine Learning Day 5



Authorized & published by Summitworks Technologies Inc



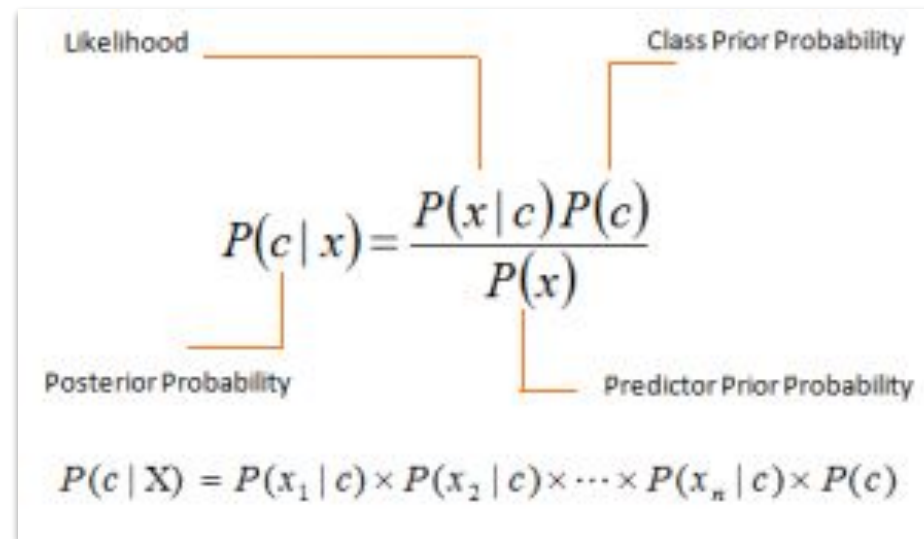
Supervised Learning 2

- Naive Bayes Classifier
 - Naive Bayes Classifier Steps
 - Building Likelihood Tables
 - Predicting the output
 - NaiveBayes() in python
 - Confusion Matrix
 - Accuracy Prediction
- Support Vector Machine (SVM)
 - Hyperplane
 - How SVM works?
 - Choosing Optimal Hyperplane
 - svm() in python
 - Kernel in svm
 - The 'C'-Values
 - The 'Gamma'-Values
 - Hyperparameter Search
 - Grid Wise Search
 - Random Search
 - SVM model building
 - Model Accuracy
 - Eager and Lazy learner

Naive Bayes Classifier

01

- Naive Bayes Classifier is a classification technique based on Bayes' Theorem with an assumption of independence among predictors
- A Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.



The diagram illustrates the components of Bayes' Theorem for a Naive Bayes Classifier. It features the general formula $P(c | x) = \frac{P(x | c)P(c)}{P(x)}$ with labels and arrows indicating the meaning of each term: 'Likelihood' points to $P(x | c)$, 'Class Prior Probability' points to $P(c)$, 'Posterior Probability' points to $P(c | x)$, and 'Predictor Prior Probability' points to $P(x)$. Below this, the expanded formula for the Naive Bayes assumption is shown: $P(c | X) = P(x_1 | c) \times P(x_2 | c) \times \dots \times P(x_n | c) \times P(c)$.

$$P(c | x) = \frac{P(x | c)P(c)}{P(x)}$$

Likelihood Class Prior Probability

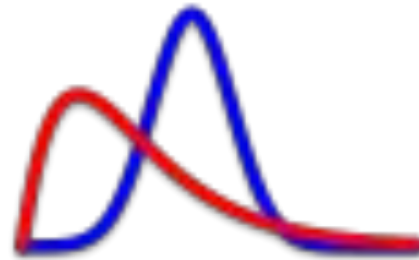
Posterior Probability Predictor Prior Probability

$$P(c | X) = P(x_1 | c) \times P(x_2 | c) \times \dots \times P(x_n | c) \times P(c)$$

Naive Bayes Classifier

01

- Bayes' theorem describes the probability of an event, based on prior knowledge of conditions that might be related to the event.
- For example, if cancer is related to age, then, using Bayes' theorem, a person's age can be used to more accurately assess the probability that they have cancer



- Bayes' theorem is stated mathematically as the following equation:
 - A and B is events and $P(B) \neq 0$
 - $P(A/B)$ is conditional probability: the likelihood of event A occurring given that B is true
 - $P(B/A)$ is conditional probability: the likelihood of event B occurring given that A is true
 - $P(A)$ is marginal probability: the probabilities of observing A independently (not taking into account B event)
 - $P(B)$ is marginal probability: the probabilities of observing B independently (not taking into account A event)

$$P(A | B) = \frac{P(B | A)P(A)}{P(B)}$$

Naive Bayes Classifier Steps

01

- Naive Bayes Classifier Steps
- Let's apply Naive Bayes Classifier for this dataset

Day	Outlook	Humidity	Wind	Play Outside
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D3	Overcast	High	Weak	Yes
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D7	Overcast	Normal	Strong	Yes
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D10	Rain	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes
D14	Rain	High	Strong	No

Create frequency tables

01

- Create a frequency table for each attribute

Outlook	Play outside	
	Yes	No
Sunny	2	3
Overcast	4	0
Rainy	3	2

Humidity	Play outside	
	Yes	No
High	3	4
Normal	6	1

Wind	Play outside	
	Yes	No
Strong	6	2
Weak	3	3

Building Likelihood Tables

01

- Based on each frequency table, we build a likelihood table:

Outlook	Play outside		
	Yes	No	
Sunny	2/9	3/5	5/14
Overcast	4/9	0/5	4/14
Rainy	3/9	2/5	5/14
	9/14	5/14	

$P(x | c) = P(\text{Sunny} | \text{Yes}) = 2/9 = 0.22$

$P(x) = P(\text{Sunny}) = 5/14 = 0.36$

$P(c) = P(\text{Yes}) = 9/14 = 0.64$

- Likelihood of Yes given Sunny is
 - $P(c | x) = P(\text{Yes} | \text{Sunny}) = P(\text{Sunny} | \text{Yes}) * P(\text{Yes}) / P(\text{Sunny}) = (0.22 \times 0.64) / 0.36 = 0.3911$
- Likelihood of No given Sunny is
 - $P(c | x) = P(\text{No} | \text{Sunny}) = P(\text{Sunny} | \text{No}) * P(\text{No}) / P(\text{Sunny}) = (0.6 \times 0.36) / 0.36 = 0.6$

Building Likelihood Tables

01

- Do it the same with other table we have

Humidity	Play outside		
	Yes	No	
High	3/9	4/5	7/14
Normal	6/9	1/5	7/14
	9/14	5/14	

$$P(\text{Yes} \mid \text{High}) = 0.33 \times 0.6 / 0.5 = 0.42$$

$$P(\text{No} \mid \text{High}) = 0.8 \times 0.36 / 0.5 = 0.58$$

Wind	Play outside		
	Yes	No	
Weak	6/9	2/5	8/14
Strong	3/9	3/5	6/14
	9/14	5/14	

$$P(\text{Yes} \mid \text{Weak}) = 0.67 \times 0.64 / 0.57 = 0.75$$

$$P(\text{No} \mid \text{Strong}) = 0.4 \times 0.36 / 0.57 = 0.25$$

- We can use all of the information above to predicting the outcome
- For example: the given condition are:
 - Outlook = Rain
 - Humidity = High
 - Wind = Weak
- Likelihood of Yes on that day is:
 - $P(\text{Outlook} = \text{Rain} | \text{Yes}) * P(\text{Humidity} = \text{High} | \text{Yes}) * P(\text{Wind} = \text{Weak} | \text{Yes}) * P(\text{Yes})$
 $= 2/9 * 3/9 * 6/9 * 9/14 = 0.0199$
- Likelihood of No on that day is:
 - $P(\text{Outlook} = \text{Rain} | \text{No}) * P(\text{Humidity} = \text{High} | \text{No}) * P(\text{Wind} = \text{Weak} | \text{No}) * P(\text{No})$
 $= 2/5 * 4/5 * 2/5 * 5/14 = 0.0166$

- Normalize these two value (divide by the total)
 - $P(\text{Yes} | X) = 0.0199 / (0.0199 + 0.0166) = 0.55$
 - $P(\text{No} | X) = 0.0199 / (0.0199 + 0.0166) = 0.45$
- The model predicts that the chance to play outside tomorrow is 55%
- The model predicts that the chance to not play outside tomorrow is 45%



- Apply Naive Bayes Classifier on following data sets
- Predict outcome for Refund = No, Married, Income = 120K

<i>Tid</i>	Refund	Marital Status	Taxable Income	Evade
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

- Predict outcome for Refund = No, Married, Income = 120K
- The results show $P(X|\text{No})P(\text{No}) > P(X|\text{Yes})P(\text{Yes})$
- Therefore $P(\text{No}|X) > P(\text{Yes}|X) \Rightarrow \text{Predicted Evade} = \text{No}$

<i>Tid</i>	Refund	Marital Status	Taxable Income	Evade
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

- Importing GaussianNB from sklearn package

```
from sklearn.naive_bayes import GaussianNB
```

- Create a model

```
gnb = GaussianNB()
```

- Training and predict the outcome

```
model = gnb.fit(X_train,Y_train)  
predictions = model.predict(X_test)
```

- A **confusion matrix** (an error matrix) is a specific table layout that allows visualization of the performance of an algorithm

	Predicted: No	Predicted: Yes
Actual: No	True Negative (TN)	False Positive (FP)
Actual: Yes	False Negative (FN)	True Positive (TP)

- **True positives (TP)**: These are cases in which we predicted yes but it is actually no
- **True negatives (TN)**: We predicted no, and they is actually no.
- **False positives (FP)**: We predicted yes, but is is not. (Also known as a "***Type I error.***")
- **False negatives (FN)**: We predicted no, but is is actually yes. (Also known as a "***Type II error.***")

- Based on these information we can compute the accuracy:
 - **Accuracy:** Overall, how often is the classifier correct?
 - $(TP+TN)/total$
 - **Precision:** When it predicts yes, how often is it correct?
 - $TP/(FP + TP)$ or $TP/(predicted\ yes)$
 - **Recall:** recall expresses the ability to find all relevant instances in a dataset
 - $TP/(TP + FN)$ or $TP/(actual\ yes)$

Practice on Confusion Matrix

01

- Compute accuracy, precision and recall of following confusion matrix

```
[[262 15]  
 [26 347]]
```

Practice on Confusion Matrix

01

- Compute accuracy, precision and recall of following confusion matrix
- Accuracy = 0.93
- Precision = 0.94
- Recall = 0.93

```
[[262 15]  
 [26 347]]
```

Practice on Confusion Matrix

01

- Compute precision and recall of following confusion matrix

		Predicted			
		A	B	C	
Actual	A	2	2	0	4
	B	1	2	0	3
	C	0	0	3	3
		3	4	3	Total

Practice on Confusion Matrix

01

- Compute precision and recall of following confusion matrix
- precision of A = $2/3 = 0.67$
- precision of B = $2/4 = 0.50$
- precision of C = $3/3 = 1.00$
- recall of A = $2/4 = 0.50$
- recall of B = $2/3 = 0.67$
- recall of C = $3/3 = 1.00$

		Predicted			
		A	B	C	
Actual	A	2	2	0	4
	B	1	2	0	3
	C	0	0	3	3
		3	4	3	Total

- Confusion matrix in python:
 - With 2 outcome values:

```
from sklearn.metrics import confusion_matrix

Y_predicted = [True, False, True, False, True, False, True, False]
Y_actual = [True, False, True, False, True, True, False, False]
print(confusion_matrix(Y_actual, Y_predicted))
```

```
[[3 1]
 [1 3]]
```

- With more than 2 outcome values:

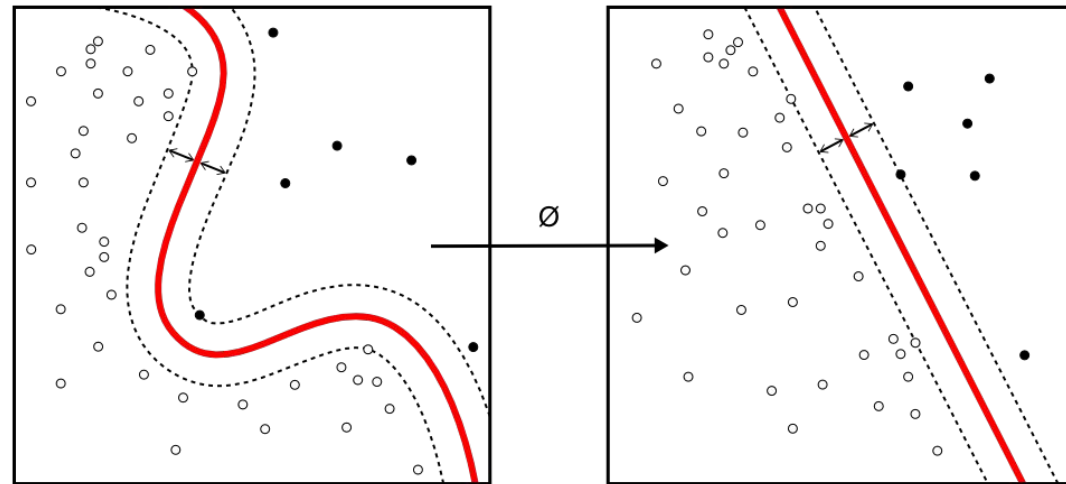
```
from sklearn.metrics import confusion_matrix
Y_actual = [2, 0, 2, 2, 0, 1]
Y_predicted = [0, 0, 2, 2, 0, 2]
print(confusion_matrix(Y_actual, Y_predicted))
```

```
[[2 0 0]
 [0 0 1]
 [1 0 2]]
```

- You can also use `accuracy_score` to compute the accuracy

```
from sklearn.metrics import accuracy_score  
print(accuracy_score(Y_actual, Y_predicted))
```

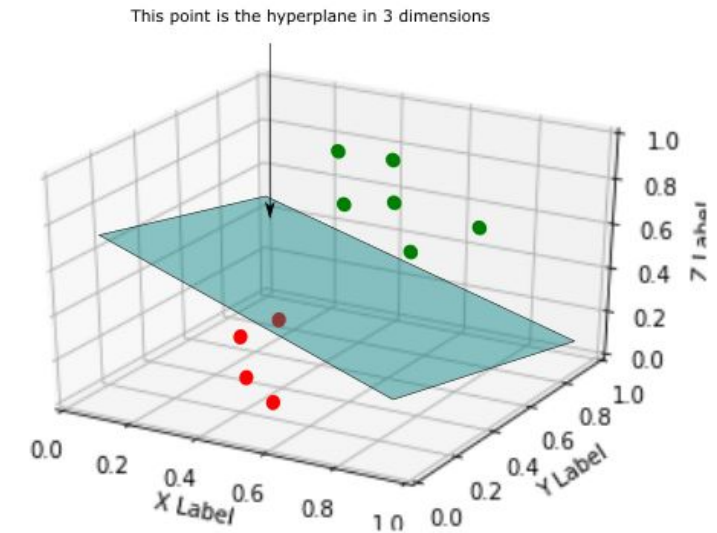
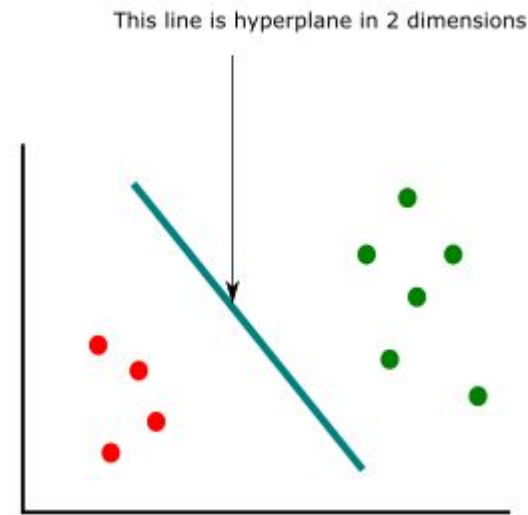
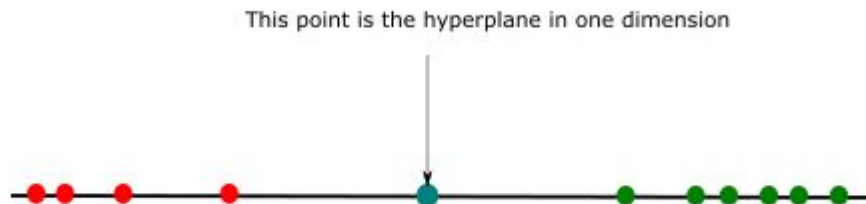
- In machine learning, support-vector machines (SVM) are supervised learning models with associated learning algorithms that analyze data used for both classification and regression analysis
- SVM tries to define a hyperplane to split data in the optimal way. For example, Suppose some given data points each belong to one of two classes, and the goal is to decide which class a new data point will be in.
- It is one of the most efficient algorithm



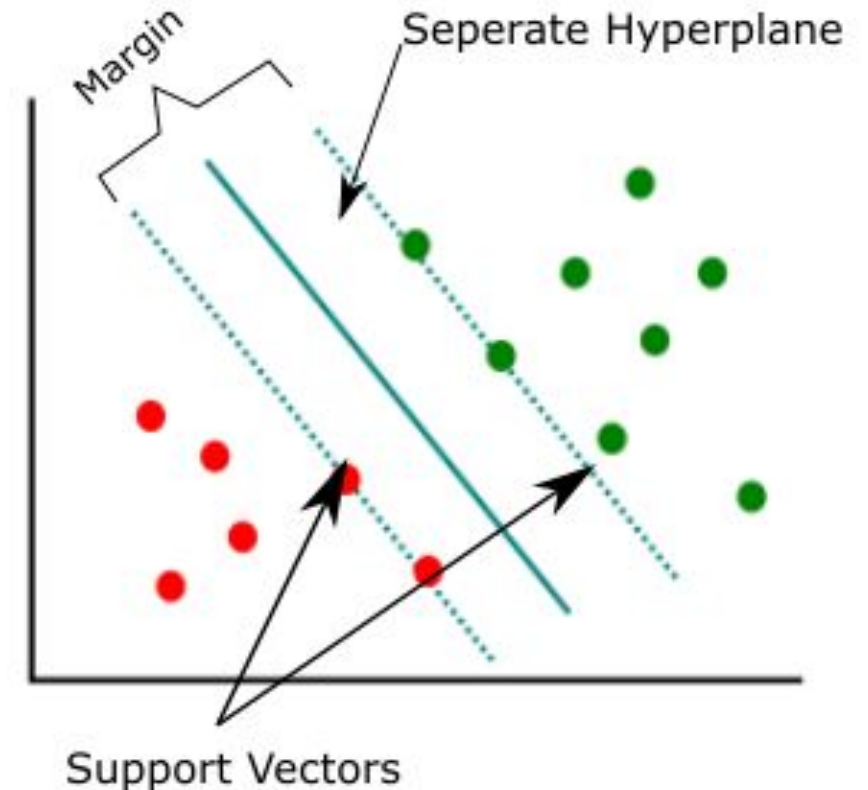
Hyperplane

01

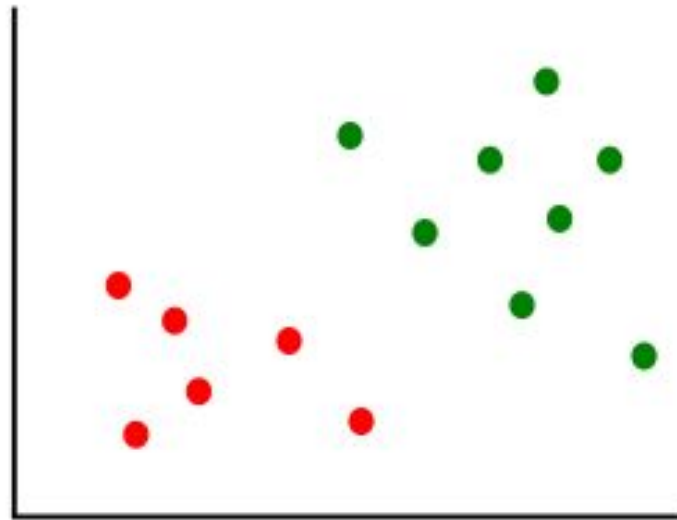
- An hyperplane is a generalization of a plane
 - In one dimension, a hyperplane is a point
 - In two dimensions, it is a line
 - In three dimensions, it is a plane
 - It is called hyperplane in more than three dimensions



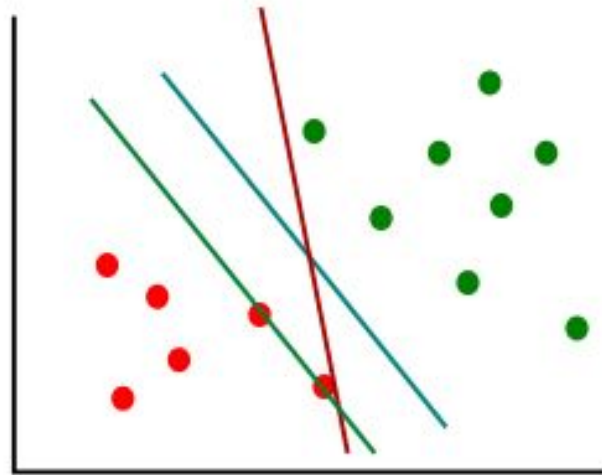
- Support vector machines: The linearly separable case
 - If the training data is linearly separable, we can select two parallel hyperplanes that separate the two classes of data, so that the distance between them is as large as possible.
 - The region bounded by these two hyperplanes is called the "**margin**", and the **maximum-margin hyperplane** is the hyperplane that lies halfway between them



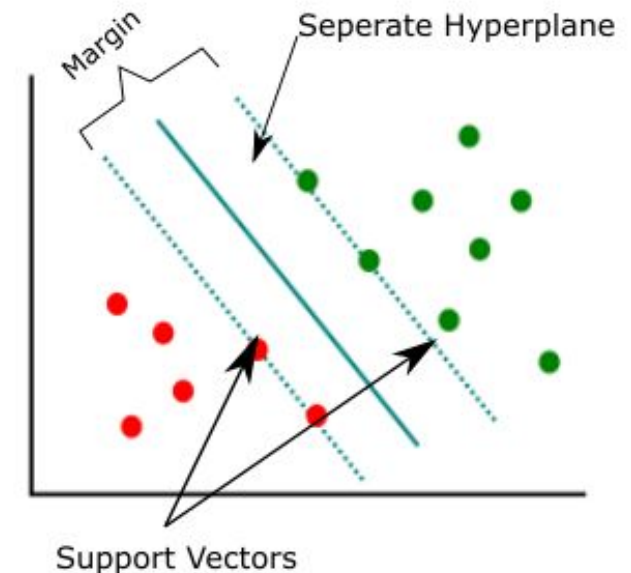
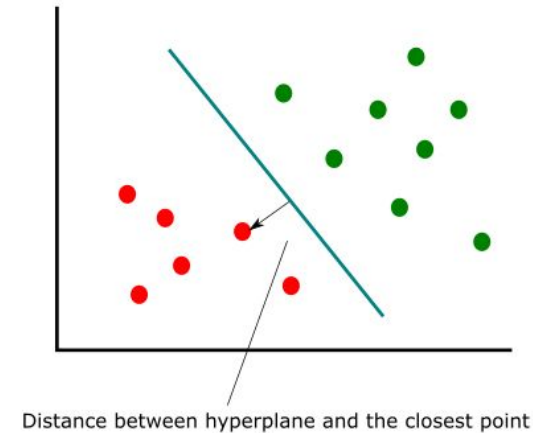
- Let's look at 2 dimensions classification problem
- The task is find the line that separates the data set to two classes



- We randomly create.draw the line that might separate data set into two classes
- The SVM helps to find the most optimal line or hyperplane



- SVM algorithm:
 - Find the points closest to the line from both the classes. (These points are called support vectors)
 - Compute the distance between the line and the support vectors. (This distance is called the margin.)
 - Find the hyperplane that maximizes the margin.
 - The hyperplane for which the margin is maximum is the optimal hyperplane.



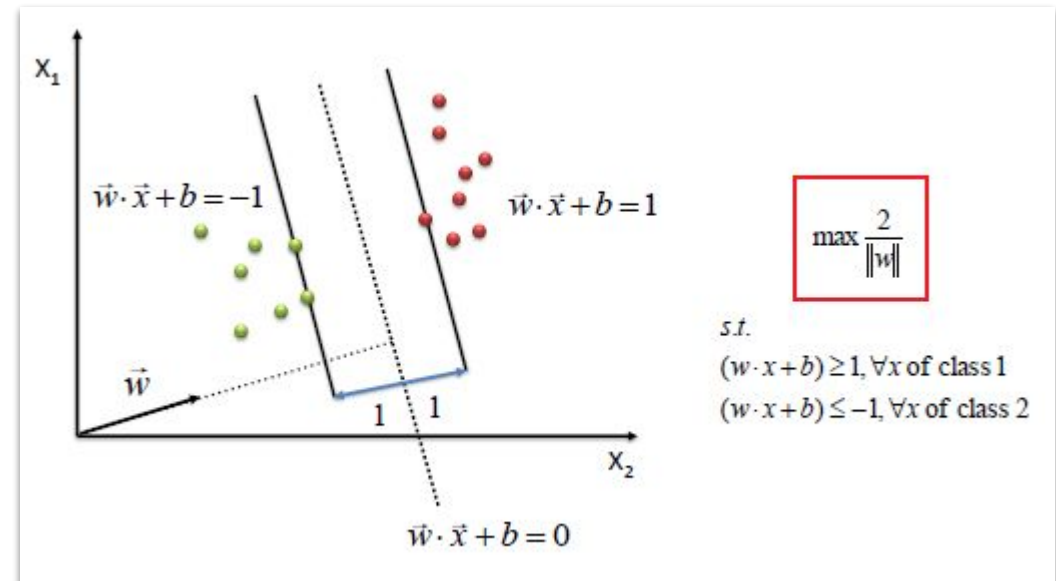
How it works (as implementation)

01

- SVM algorithm is using cost function called “Hinge Loss function”
 - The hinge loss is used for "maximum-margin" classification
 - c is the loss function, x the sample, y is the true label, $f(x)$ the predicted label

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases}$$

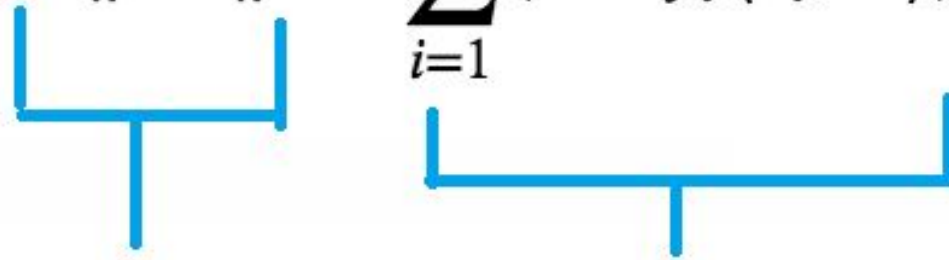
- Our purpose is finding the value of w vector
So that our cost is minimize



How it works (as implementation)

01

- Rewrite the purpose which include the regularizer.
- The regularizer balances between margin maximization and loss
- w (weight/slope) is a variable that we need to find to make this object function minimum

$$\min_w \lambda \|w\|^2 + \sum_{i=1}^n (1 - y_i \langle x_i, w \rangle).$$


Regularizer

Hinge Loss function

- We need to use sklearn package for svm

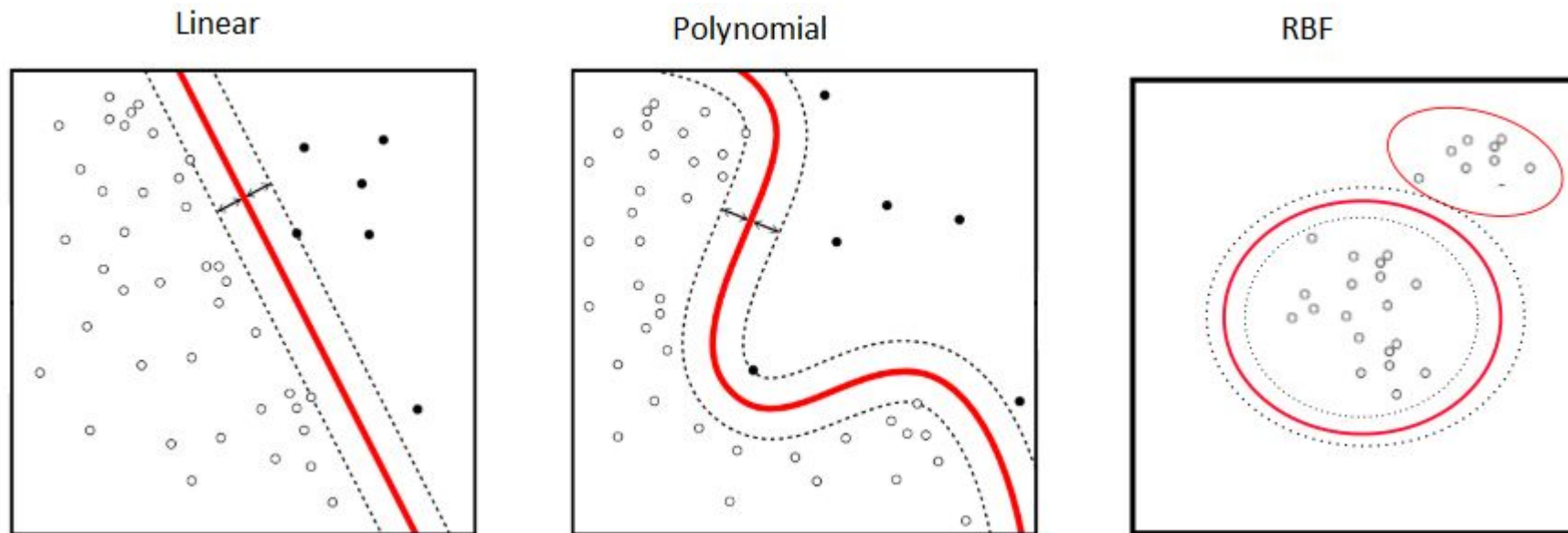
```
from sklearn import svm
```

- Create a svm mode

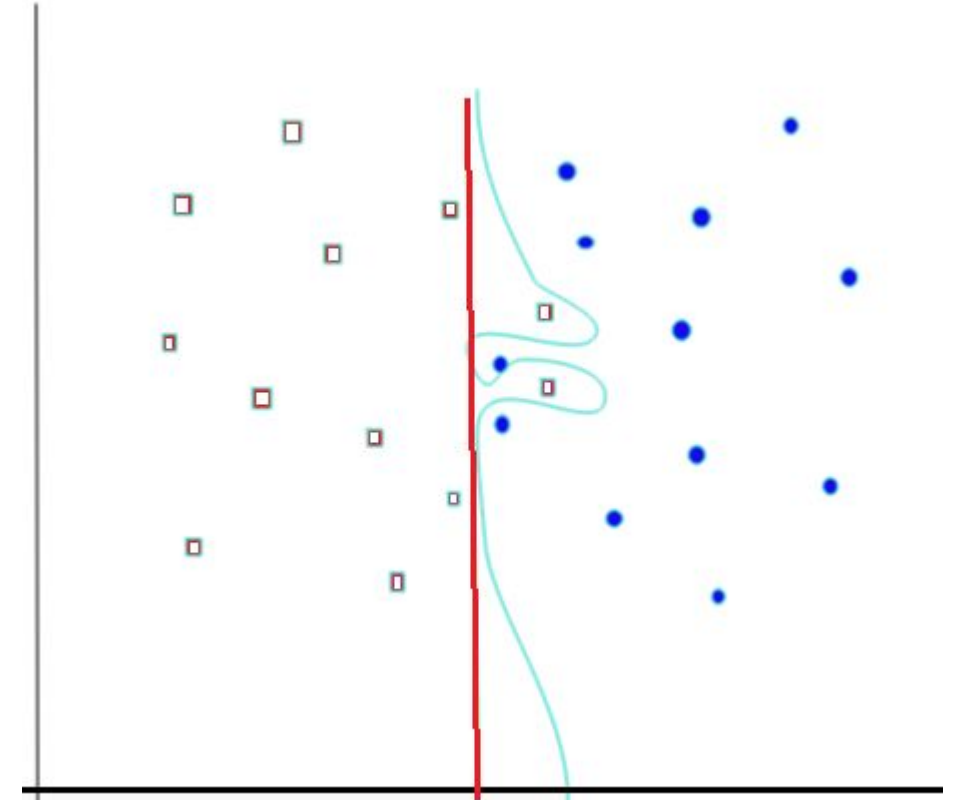
```
model = svm.svc(kernel='linear',c=1,gamma=1)
```

- The svm model has following parameter:
 - Kernels
 - 'C'-Value (supports non-linear)
 - 'Gamma'-value (supports non-linear)

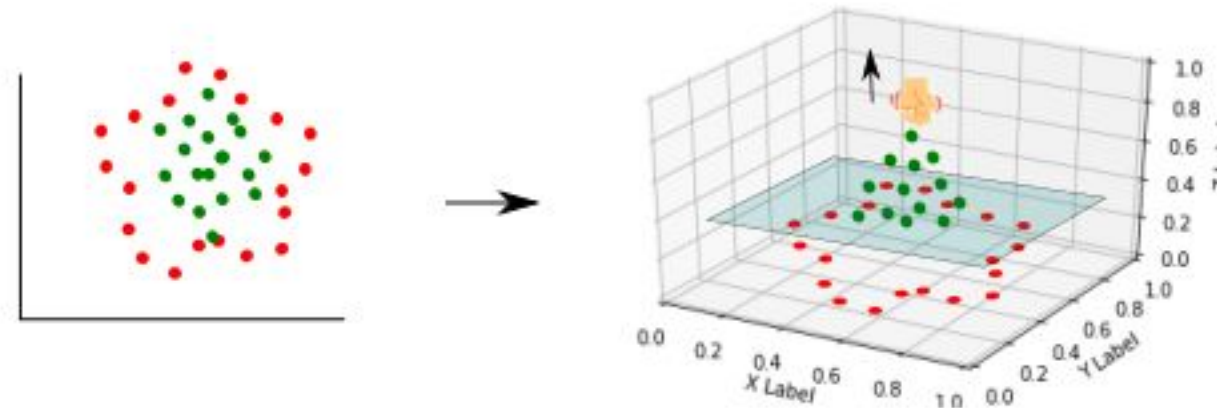
- There are three types of kernel available in svm
 - Linear kernel: using when data is linearly separable
 - Polynomial: using when data is non linearly separable and can be classified using curve
 - Radial basis function (RBF):using when data is non linearly separable but can be classified using curve



- The 'C' value determines the width of the margin
 - Larger the 'C' value smaller is the margin
 - 'C' value affects the misclassification error
 - Recommended 'C' value is from 2^{-10} to 2^{10}
 - It controls the trade off between **smooth decision boundary** and **classifying training points correctly**. A large value of c means you will get more training points correctly.
 - For example, a straight line decision boundary which is simple but it costs a few points being misclassified



- Gamma is the parameter of a Gaussian Kernel (non-linear classification)
- The data set is not linearly separable in 2 dimensions we can transform them to higher dimension where they can be linearly separable
- Gamma control the shape of the peak where you raise the points
- The small gamma make a pointed bump, a larger gamma give a broader bump
- The small gamma give you low bias and high variance, the large gamma give you high bias and low variance

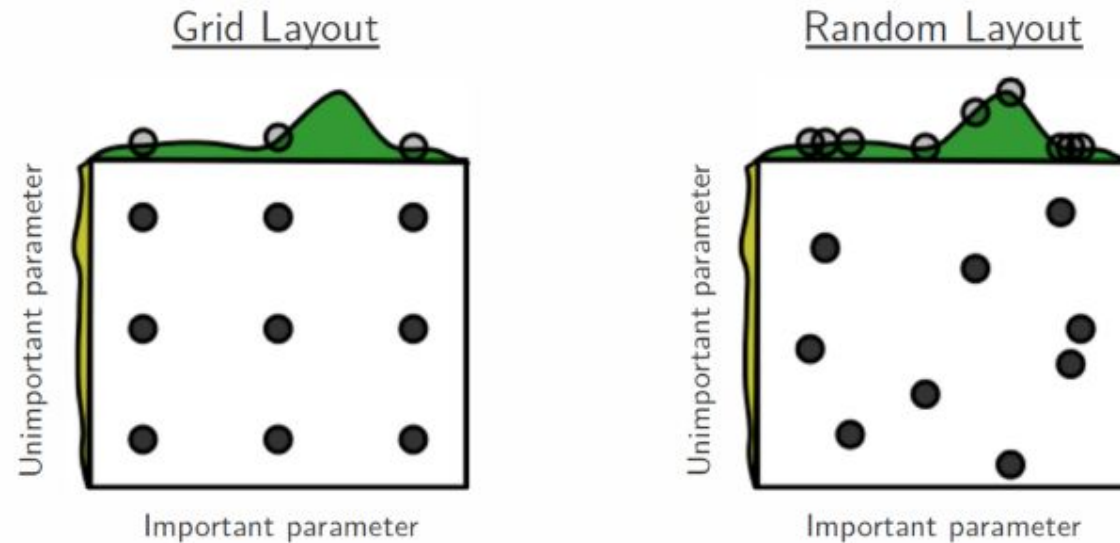


- We can configure our model using the hyperparameters (which express “higher-level” properties of the model such as its complexity or how fast it should learn)
 - For example:
 - number of tree in random forest
 - number of layers in neural network
 - Learning rate for logistic regression
 - C_values and gamma_values of SVM
- Choosing the best hyperparameters manually is tedious task
- We can use hyperparameter search to do this task

Hyperparameter Search

01

- There are two methods for optimizing hyperparameters:
 - Grid wise search
 - Random search



- The grid wise search the parameter values are equally distributed within the parameter range specified by the user.
- For example, if user give range from -5 to 5, the grid search values as: -4,-3,-2,-1,1,2,3,4
- An example of Grid search with Python:

```
C_values = [0.01,0.03,0.1,0.3,1,3,10,30,100]
gamma_values = [0.01,0.03,0.1,0.3,1,3,10,30,100]

best_score = 0
best_params = {'C-value': None, 'gamma-value': None}

for C_value in C_values:
    for gamma_value in gamma_values:
        model = svm.SVC(C=C_value,gamma = gamma_value)
        model.fit(x,y)
        score = svc.score(X_val,Y_val)

        if score > best_score:
            best_score = score
            best_params['C-value'] = C
            best_params['gamma-value'] = gamma
```

- The random search the parameter values are randomly chosen within the parameter range specified by the user.
- For example, if user give range from -10 to 10, the grid search values as: -9,9,0,3,7,-8,5,-3,9
- An example of Random search with Python:

```
best_score = 0
best_params = {'C-value': None, 'gamma-value': None}

for i in range(10):
    model = svm.SVC(C=random(0,9),gamma = random(0,3))
    model.fit(x,y)
    score = svc.score(X_val,Y_val)

    if score > best_score:
        best_score = score
        best_params['C-value'] = C
        best_params['gamma-value'] = gamma
```

- SVM model building using Python

```
from sklearn.metrics import accuracy_score
from sklearn import svm

model = svm.SVC(kernel = 'linear')
model.fit(X_train,Y_train)
predictions = model.predict(X_test)
```


- Using metrics and `accuracy_score` to checking model accuracy

```
print("Accuracy score:", accuracy_score(Y_test, predictions))  
  
#Accuracy score: 0.962341414111
```

Eager vs Lazy Learner

01

Eager Learner	Lazy Learner
Generalized model from training set as soon as it receives the training set	Training dataset is stored in system to build the model
It's fast as it has pre-trained algorithm	It's slow as it calculates based on the current data set instead of coming up with an algorithm based on historic data
Example: Decision Tree	Example: K-nearest neighbor

Q & A