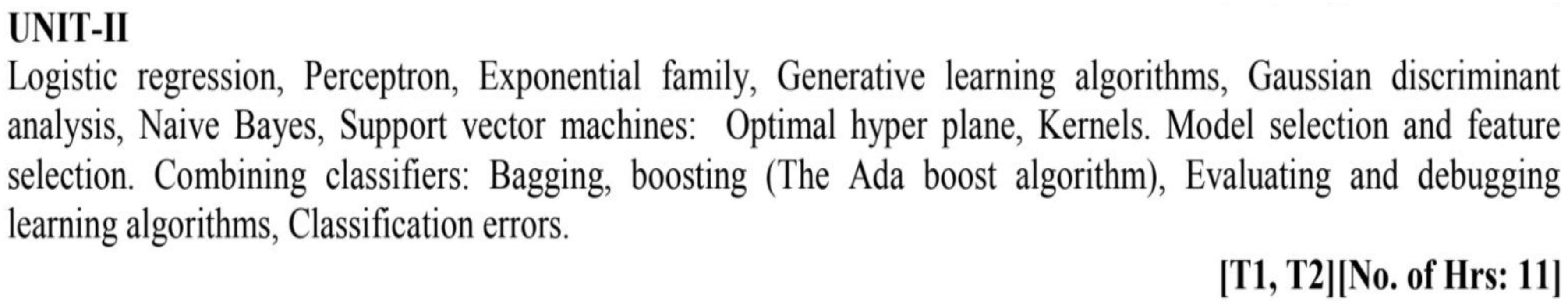
Unit 2

Ms. Hina Firdaus

To study



UNIT-II

Logistic regression, Perceptron, Exponential family, Generative learning algorithms, Gaussian discriminant analysis, Naive Bayes, Support vector machines: Optimal hyper plane, Kernels. Model selection and feature selection. Combining classifiers: Bagging, boosting (The Ada boost algorithm), Evaluating and debugging learning algorithms, Classification errors.

[T1, T2][No. of Hrs: 11]

Introduction

- Logistic Regression was used in the biological sciences in early twentieth century
- Logistic Regression is used when the dependent variable(target) is categorical.

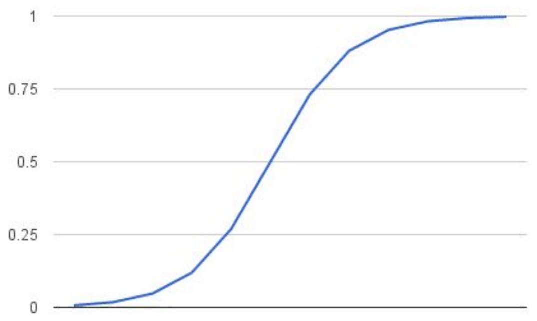
For example,

- To predict whether an email is spam (1) or (0)
- Whether the tumor is malignant (1) or not (0)
-

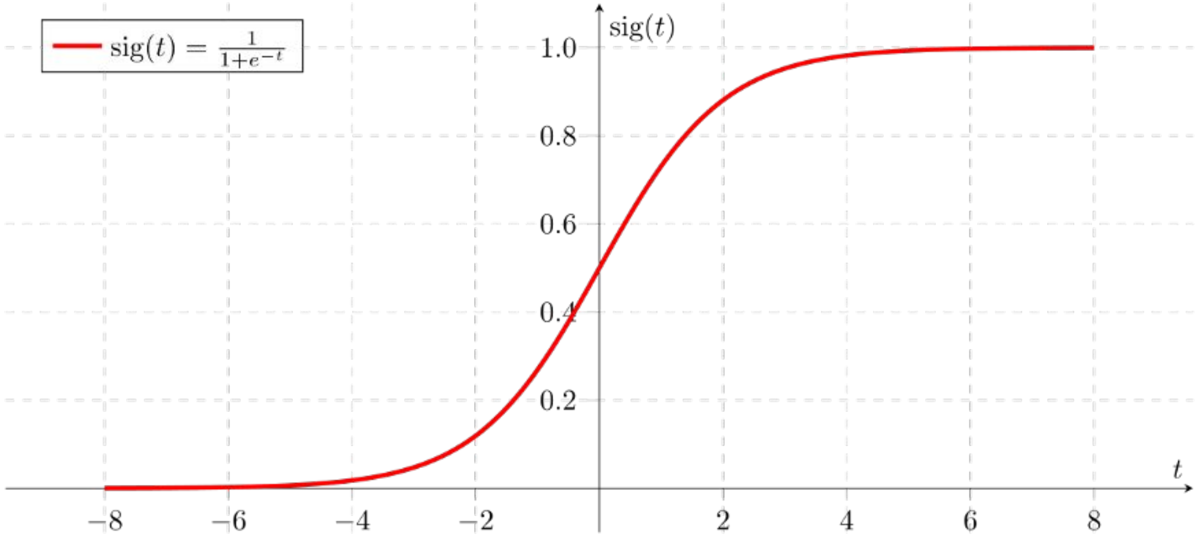
Logistic Function

- Logistic regression is named for the function used at the core of the method, the logistic function.
- The logistic function, also called the sigmoid function
- It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

1 / (1 + e^-value)



Logistic function



Model

Output = 0 or 1

Hypothesis => $Z = WX + B$

$h\Theta(x)$ = sigmoid (Z)

Representation Used for Logistic Regression

Below is an example logistic regression equation:

$$y = e^{(b_0 + b_1*x)} / (1 + e^{(b_0 + b_1*x)})$$

Where,

- ❖ y is the predicted output,
- ❖ b_0 is the bias or intercept term
- ❖ b_1 is the coefficient for the single input value (x).
- ❖ Each column in your input data has an associated b coefficient (a constant real value) that must be learned from your training data.

Types of Logistic Regression

1. Binary Logistic Regression

   The categorical response has only two 2 possible outcomes. Example: Spam or Not

2. Multinomial Logistic Regression

   Three or more categories without ordering. Example: Predicting which food is preferred more (Veg, Non-Veg, Vegan)

3. Ordinal Logistic Regression

   Three or more categories with ordering. Example: Movie rating from 1 to 5

Making Predictions with Logistic Regression

- we have a model that can predict whether a person is male or female based on their height (completely fictitious). Given a height of 150cm is the person male or female.
- We have learned the coefficients of $b0 = -100$ and $b1 = 0.6$.
- Using the equation above we can calculate the probability of male given a height of 150cm or more formally P(male|height=150)

$$y = e^{\wedge}(b0 + b1*X) / (1 + e^{\wedge}(b0 + b1*X))$$

$$y = \exp(-100 + 0.6*150) / (1 + EXP(-100 + 0.6*X))$$

$$y = 0.0000453978687$$

- Or a probability of near zero that the person is a male.

Cont...

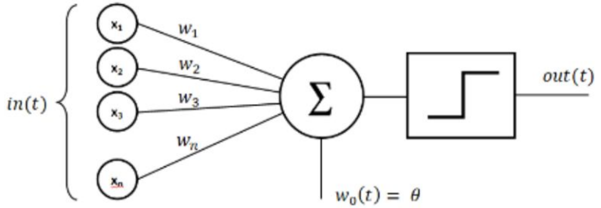
- Because this is classification and we want a crisp answer, we can snap the probabilities to a binary class value

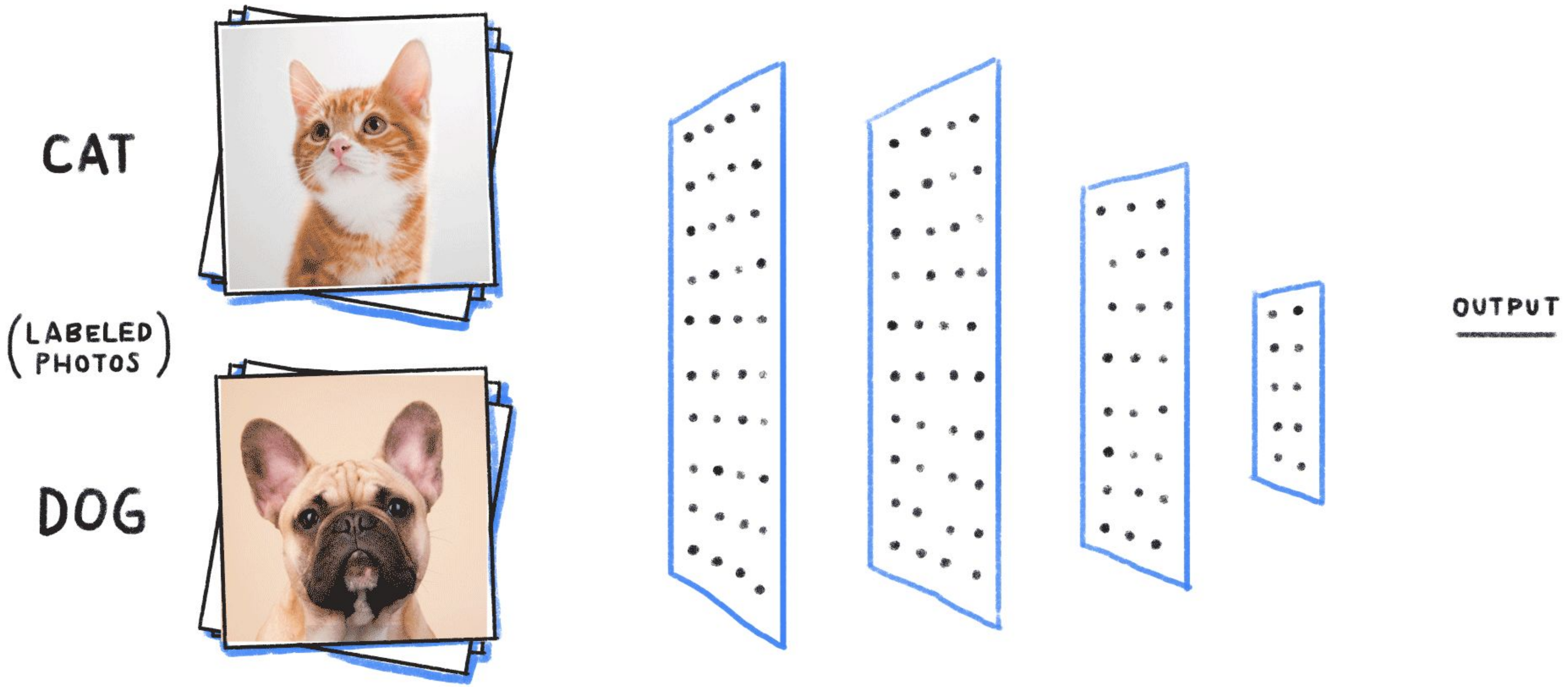
for example:

  - 0 if p(male) < 0.5
  - 1 if p(male) >= 0.5

WHAT PERCEPTRON SOUNDS LIKE

wHat pERceptRoNS ArE



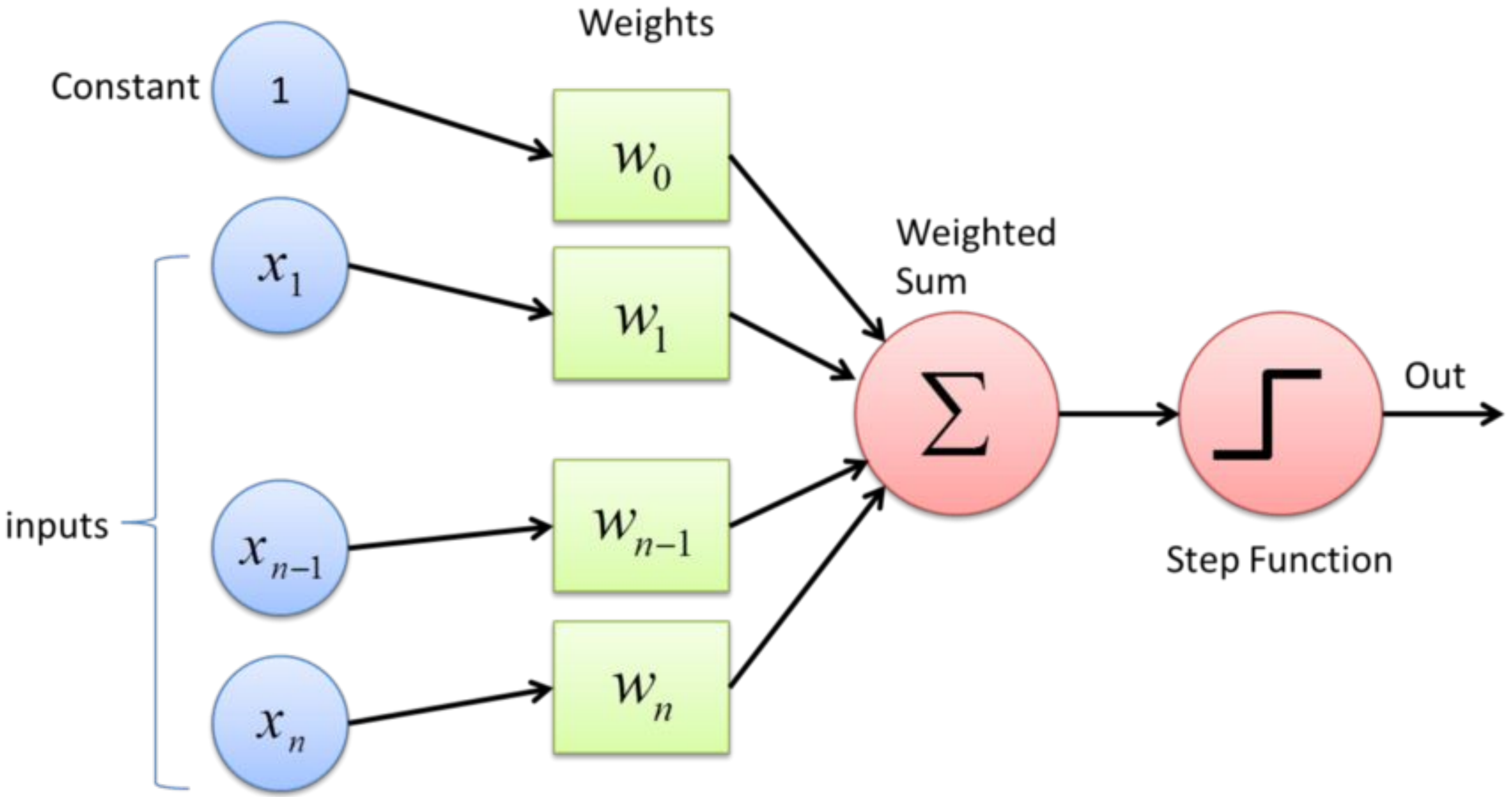

CAT
(LABELED PHOTOS)
DOG
OUTPUT

Perceptron

- The Perceptron is a linear machine learning algorithm for binary classification tasks.
- It is a type of neural network model, perhaps the simplest type of neural network model.
- It consists of a single node or neuron that takes a row of data as input and predicts a class label.
- This is achieved by calculating the weighted sum of the inputs and a bias (set to 1).
- The weighted sum of the input of the model is called the activation.
    - Activation = Weights * Inputs + Bias
- If the activation is above 0.0, the model will output 1.0; otherwise, it will output 0.0.
    - Predict 1: If Activation > 0.0
    - Predict 0: If Activation <= 0.0

Cont...

- The Perceptron is a linear classification algorithm
- This means that it learns a decision boundary that separates two classes using a line (called a hyperplane) in the feature space.
- it is appropriate for those problems where the classes can be separated well by a line or linear model, referred to as linearly separable.

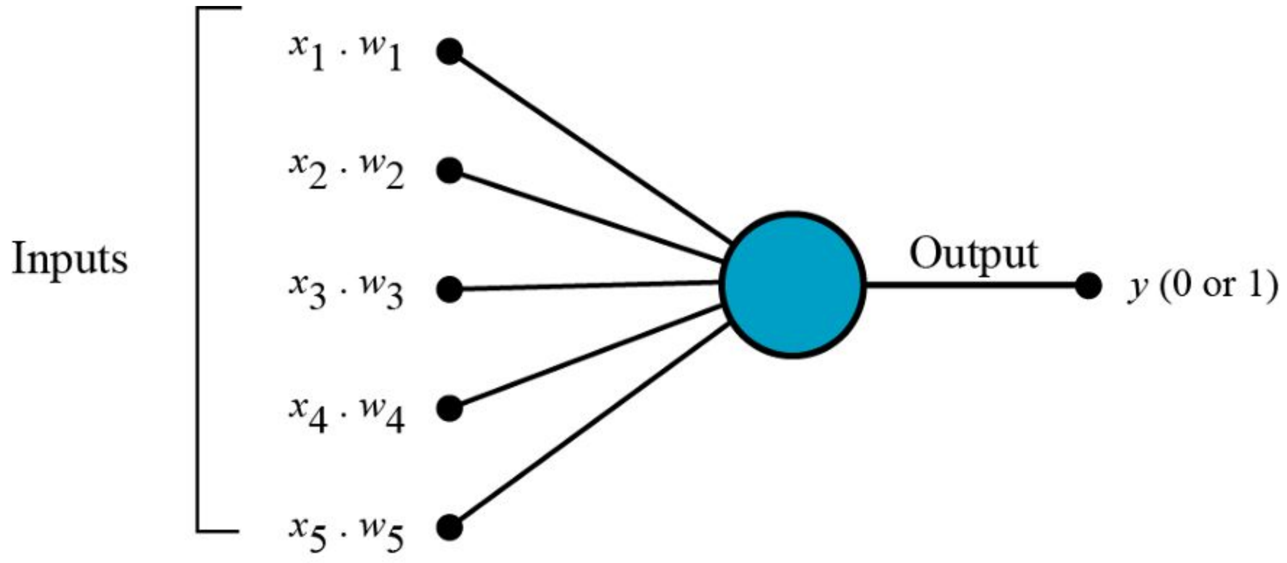


Weights
Constant
1
w_0
x_1
w_1
Weighted Sum
Σ
Out
inputs
x_{n-1}
w_{n-1}
Step Function
x_n
w_n

But how does it work?

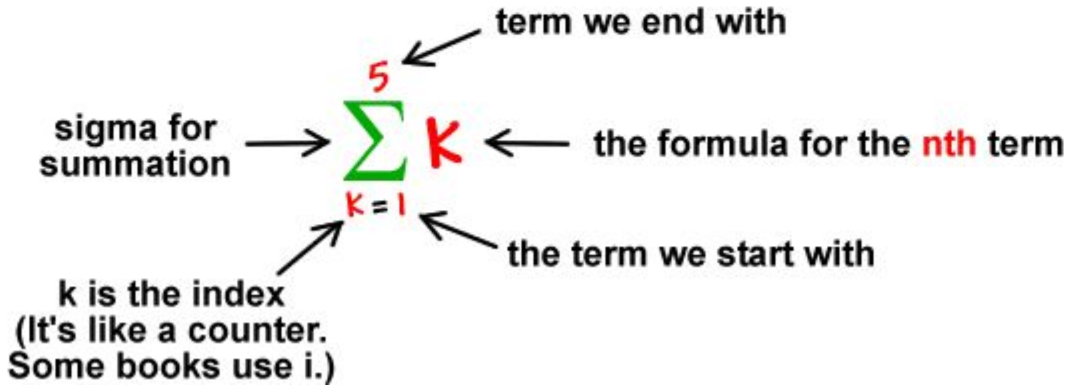
The perceptron works on these simple steps

a. All the inputs x are multiplied with their weights w. Let’s call it k.



Cont...

b. Add all the multiplied values and call them Weighted Sum.

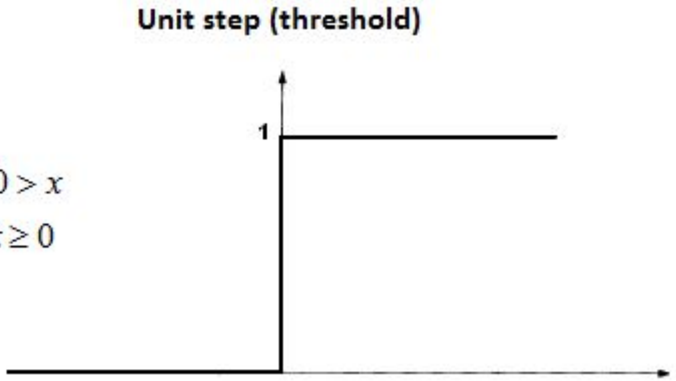

term we end with
5
sigma for summation
$\sum$ k
the formula for the nth term
k = 1
the term we start with
k is the index (It's like a counter. Some books use i.)

Cont...

c. Apply that weighted sum to the correct Activation Function.

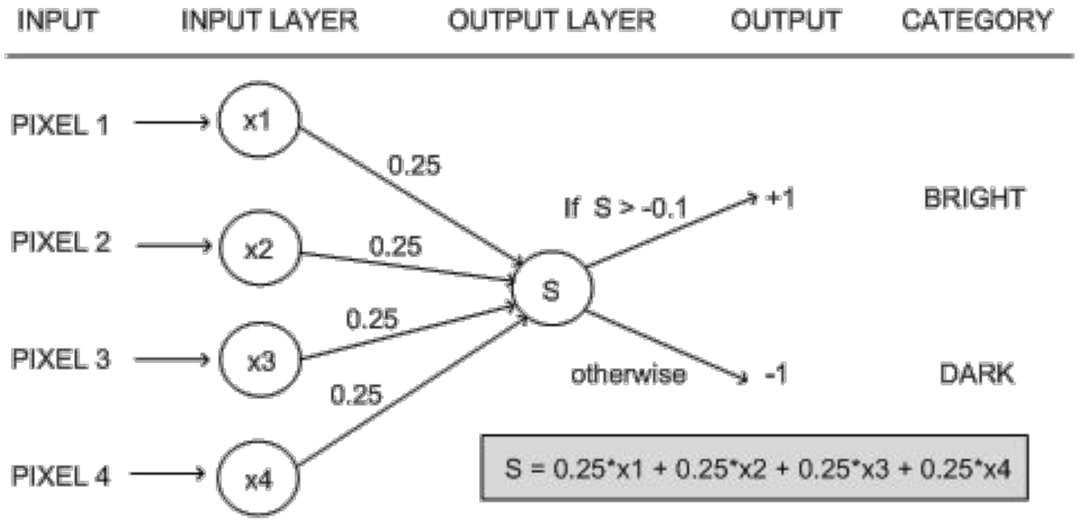
For Example: Unit Step Activation Function.

$$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$



Why do we need Weights and Bias?

- Weights shows the strength of the particular node.
- A bias value allows you to shift the activation function curve up or down.

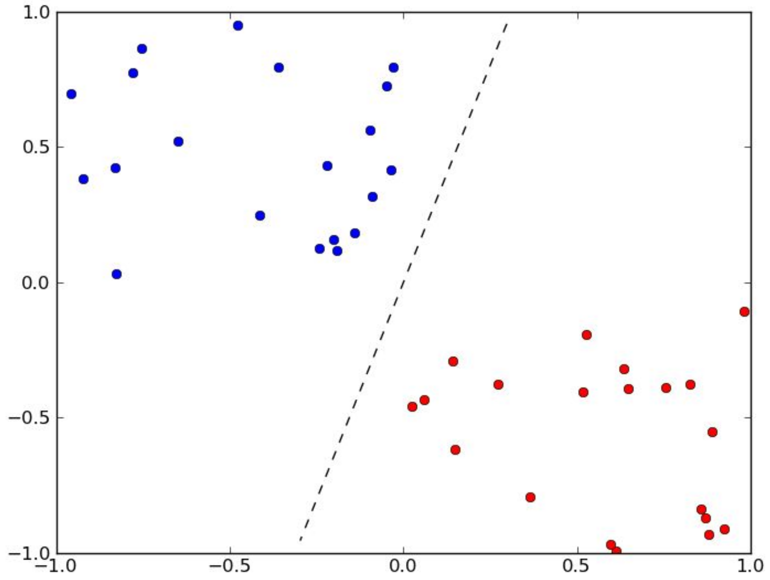


Why do we need Activation Function?

the activation functions are used to map the input between the required values like (0, 1) or (-1, 1).

Where we use Perceptron?

Perceptron is usually used to classify the data into two parts. Therefore, it is also known as a Linear Binary Classifier.



Exponential family

- Exponential family includes the Gaussian, binomial, multinomial, Poisson, Gamma and many others distributions.
- a distribution belongs to exponential family if it can be transformed into the general form:

$$p(x|\eta) = h(x)exp(\eta^T T(x) - A(\eta))$$

where,

η is canonical parameter

T(x) is sufficient statistic

A(η) is cumulant function

Gaussian discriminant analysis model

- Gaussian Discriminant Analysis is a Generative Learning Algorithm and in order to capture the distribution of each class
- it tries to fit a Gaussian Distribution to every class of the data separately.
- classification problem in which the input features are continuous random variable, we can use GDA
- assume p(x|y) is distributed according to a multivariate normal distribution and p(y) is distributed according to Bernoulli. So the model is

$$p(y) = \phi^{y}(1-\phi)^{(1-y)}$$

$$p(x|y=0) = \frac{1}{(2\pi)^{n/2}|\sum|^{\frac{1}{2}}} exp(-\frac{1}{2}(x-\mu_0)^T \sum^{-1}(x-\mu_0))$$

$$p(x|y=1) = \frac{1}{(2\pi)^{n/2}|\sum|^{\frac{1}{2}}} exp(-\frac{1}{2}(x-\mu_1)^T \sum^{-1}(x-\mu_1))$$

Here the parameters of the model are ϕ, μ_0, μ_1 and $\sum$.
And n is the dimension of the density function

Note : While there are two separate mean vectors μ_0 and μ_1(each for one class), but the covariance matrix $\sum$ is common for all the classes.

Cont...

we need to define the log likelihood function L and then by maximising L with respect to model parameters, find the maximum likelihood parameters

$$\ell(\phi, \mu_0, \mu_1, \sum) = \log \prod_{i=1}^{m} p(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \sum)$$

$$= \log \prod_{i=1}^{m} p(x^{(i)} \mid y^{(i)}; \phi, \mu_0, \mu_1, \sum) p(y^{(i)}; \phi)$$

Lets make log-likelihood function generic w.r.t classes and for a class k where $k \in \{0,1\}$, log-likelihood $\ell(\phi, \mu_k, \sum)$

$$= \log \prod_{i=1}^{m} p(x \mid y; \phi, \mu_k, \sum) p(y; \phi)$$

$$= \log \prod_{i=1}^{m} \left(\frac{1}{(2\pi)^{n/2} |\sum|^{\frac{1}{2}}} exp(-\frac{1}{2}(x-\mu_k)^T \sum^{-1} (x-\mu_k))\right) . \phi^y (1-\phi)^{(1-y)}$$

$$= \sum_{i=1}^{m} [-\frac{n}{2} \log 2\pi - \frac{1}{2} \log |\sum| - \frac{1}{2}(x-\mu)^T \sum^{-1} (x-\mu) + y \log \phi + (1-y) \log(1-\phi)]$$

—> Eq(1)

Now we need to take partial derivatives w.r.t to each parameter and equate it to zero to find the maximum likelihood of that parameter

1. For ϕ

$\frac{\partial}{\partial\phi}\ell(\phi, \mu_k, \sum) = 0$ From Eq(1)

$$\frac{\partial}{\partial\phi}\ell(\phi, \mu_k, \sum) = \sum_{i=1}^{m}\{-\frac{n}{2} * 0 - \frac{1}{2} * 0 - \frac{1}{2} * 0 + \frac{y^{(i)}}{\phi} - \frac{(1-y^{(i)})}{(1-\phi)}\} = 0$$

$$=> \sum_{i=0}^{m}\{y^{(i)}(1-\phi) - \phi(1-y^{(i)})\} = 0$$

$$=> \sum_{i=1}^{m}\{y^{(i)} - y^{(i)}\phi - \phi + y^{(i)}\phi\} = 0$$

$$=> \sum_{i=1}^{m}\{y^{(i)} - \phi\} = 0$$

$$=> \sum_{i=1}^{m} y^{(i)} - m\phi = 0 \qquad \because \sum_{i=1}^{m}\phi = \phi\sum_{i=1}^{m}1 = \phi m$$

$=> \phi = \frac{1}{m}\sum_{i=1}^{m} y^{(i)}$ $\because y \in \{0,1\}$for binary classification we can write it as

$=> \phi = \frac{1}{m}\sum_{i=1}^{m} 1\{y^{(i)} = 1\}$ —> Eq(2)

where
1{} is function which return either 0 or 1 depending up the condition inside for e.g. 1{true} = 1 and 1{false} = 0

2. For μ_0, μ_1

$\frac{\partial}{\partial \mu_k} \ell(\phi, \mu_k, \sum) = 0$ From Eq(1)

$$=> \sum_{i=1}^{m} \{-\frac{n}{2} * 0 - \frac{1}{2} * 0 - \frac{1}{2} \frac{\partial}{\partial \mu_k}[(x^{(i)} - \mu_k)^T \sum^{(-1)} (x^{(i)} - \mu_k)] + 0 + 0\} = 0$$

$\because \frac{\partial \alpha^T A x}{\partial \alpha} = 2\alpha^T A$, as A is symmetric and doesn't depend on x. Putting

$\alpha = (x_k^{(i)} - \mu_k)$, so by chain rule $\frac{\partial \ell}{\partial \mu_k} = \frac{\partial \ell}{\partial \alpha} \frac{\partial \alpha}{\partial \mu_k}$, and

$\frac{\partial \alpha}{\partial \mu_k} = \begin{cases} -1 & if \quad i \,|\, y^{(i)} = k \\ 0 & otherwise \end{cases}$ which can be written as

$= 1\{y^{(i)} = k\}$ —> Eq(3) in concise notation function

$$=> -\frac{1}{2} * 2 \sum_{i=1}^{m} \{(x^{(i)} - \mu_k)^T \sum^{-1} \frac{\partial \alpha}{\partial \mu_k}\} = 0$$

$=> \sum_{i=1}^{m} x_k^{(i)} \frac{\partial \alpha}{\partial \mu_k} - \sum_{i=1}^{m} \mu_k \frac{\partial \alpha}{\partial \mu_k} = 0$ putting value of $\frac{\partial \alpha}{\partial \mu_k}$ from Eq(3)

$=> \mu_k = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = k\} x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = k\}}$ —> Eq(4) so for μ_0, μ_1 put k=0,1 respectively

---

$$=> \mu_0 = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 0\} x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = 0\}} \quad \& \quad \mu_1 = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 1\} x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}}$$

3. For $\sum$

$$\frac{\partial}{\partial \sum} \ell(\phi, \mu_k, \sum) = 0$$ From Eq(1)

$$=> \sum_{i=1}^{m} \{-\frac{n}{2} * 0 - \frac{1}{2} * \frac{\partial \log(|\sum|)}{\partial \sum} - \frac{1}{2} \frac{\partial}{\partial \sum} [(x^{(i)} - \mu_k)^T \sum^{(-1)} (x^{(i)} - \mu_k)] + 0 + 0\}$$
$= 0$

$$=> \sum_{i=1}^{m} \{-\frac{1}{2} \sum^{-T} - \frac{1}{2} [- \sum^{-T} (x^{(i)} - \mu_k)(x^{(i)} - \mu_k)^T \sum^{-T}]\} = 0$$

$$\because \frac{\partial \log |X|}{\partial X} = X^{-T} \text{ and } \frac{\partial a^T X^{-1} b}{\partial X} = - X^{-T} a b^T X^{-T}$$

Taking out $-\frac{1}{2} \sum^{-T}$ common from above equation

$$=> \sum_{i=1}^{m} \{1 - \sum^{-T} (x^{(i)} - \mu_k)(x^{(i)} - \mu_k)^T\} = 0$$

$$=> m - \sum_{i=1}^{m} \sum^{-T} (x^{(i)} - \mu_k)(x^{(i)} - \mu_k)^T = 0$$

$$=> m \sum = \sum_{i=1}^{m} (x^{(i)} - \mu_k)(x^{(i)} - \mu_k)^T$$

$$=> \sum = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu_k)(x^{(i)} - \mu_k)^T$$ —> Eq(5) where $k = 1\{y^{(i)} = 1\}$

So Eq(2) , Eq(4) and Eq(5) defines all the maximum likelihood parameters of GDA as below

$$\phi = \frac{1}{m}\sum_{i=1}^{m} 1\{y^{(i)} = 1\}$$

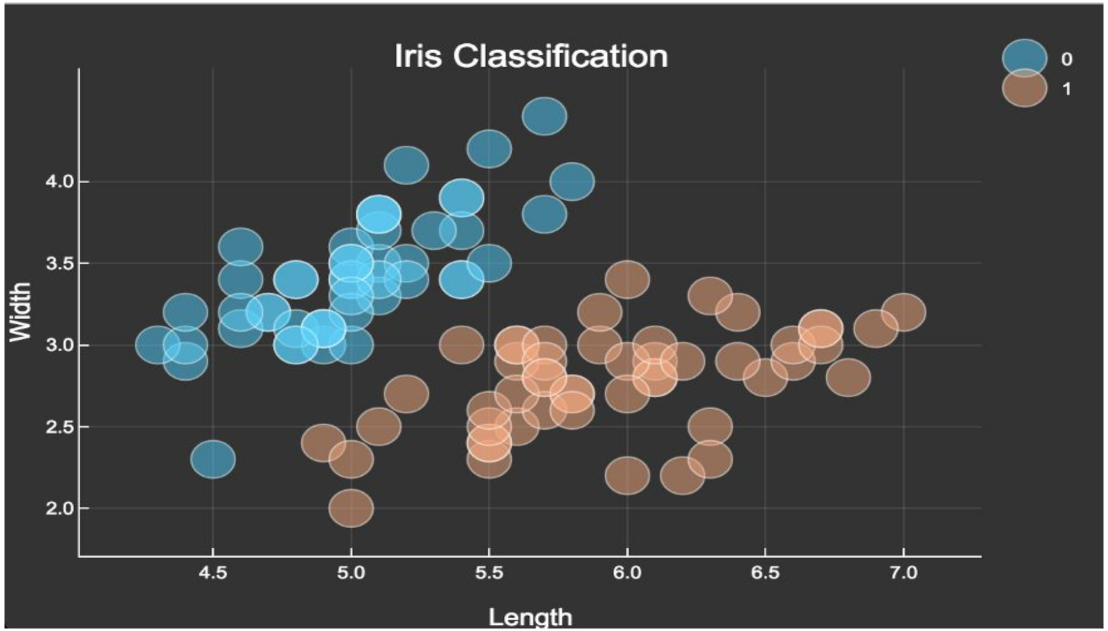
$$\mu_0 = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 0\}x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = 0\}}$$

$$\mu_1 = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}}$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m} (x^{(i)} - \mu_k)(x^{(i)} - \mu_k)^T \quad \text{where } k = 1\{y^{(i)} = 1\}$$

Example

a custom iris dataset (reduced features to fit it in to two dimensions and removed the 3rd class from the data set to make it for binary classification)for this example.The data plot looks as



cont...

For the dataset computed model parameters are as below

$$\phi = 0.50505050505051$$

$$\mu_0 = \begin{pmatrix} 0.0505463 \\ 0.0345083 \end{pmatrix} \qquad \mu_1 = \begin{pmatrix} 0.0599596 \\ 0.0279798 \end{pmatrix} \qquad \Sigma = \begin{pmatrix} 29.7764 & 16.5206 \\ 16.5206 & 9.57842 \end{pmatrix}$$

Conclusion

- GDA makes an assumption about the probability distribution of the $p(x|y=k)$ where k is one of the classes
- GDA, if the initial assumptions about distribution of $p(x|y=k)$ (Gaussian)and $p(y)$(Bernoulli) are true, then $p(y|x)$ can be expressed as Sigmoid
- Which means GDA makes more specific assumptions about the data set then Logistic Regression and if those assumptions are true then it works better than LR

What is Naive Bayes algorithm?

- It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors
- Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.
- For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter
- Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.
- Naive Bayes model is easy to build and particularly useful for very large data sets
- Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.
- Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:

$$\underbrace{P(c\mid x)}_{\text{Posterior Probability}}=\frac{\overbrace{P(x\mid c)}^{\text{Likelihood}}\overbrace{P(c)}^{\text{Class Prior Probability}}}{\underbrace{P(x)}_{\text{Predictor Prior Probability}}}$$

$$P(c\mid X)=P(x_1\mid c)\times P(x_2\mid c)\times\cdots\times P(x_n\mid c)\times P(c)$$

Cont...

Above,

- $P(c|x)$ is the posterior probability of class (c, target) given predictor (x, attributes).
- $P(c)$ is the prior probability of class.
- $P(x|c)$ is the likelihood which is the probability of predictor given class.
- $P(x)$ is the prior probability of predictor.

Problem: Players will play if weather is sunny. Is this statement is correct?

- We can solve it using above discussed method of posterior probability.
- P(Yes | Sunny) = P(Sunny | Yes) * P(Yes) / P (Sunny)
- Here we have P (Sunny |Yes) = 3/9 = 0.33, P(Sunny) = 5/14 = 0.36, P(Yes)= 9/14 = 0.64
- Now, P (Yes | Sunny) = 0.33 * 0.64 / 0.36 = 0.60, which has higher probability.
- Naive Bayes uses a similar method to predict the probability of different class based on various attributes. This algorithm is mostly used in text classification and with problems having multiple classes.

What are the Pros and Cons of Naive Bayes?

Pros:

- It is easy and fast to predict class of test data set. It also perform well in multi class prediction
- When assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and you need less training data.
- It perform well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell curve, which is a strong assumption).

Cont...

Cons:

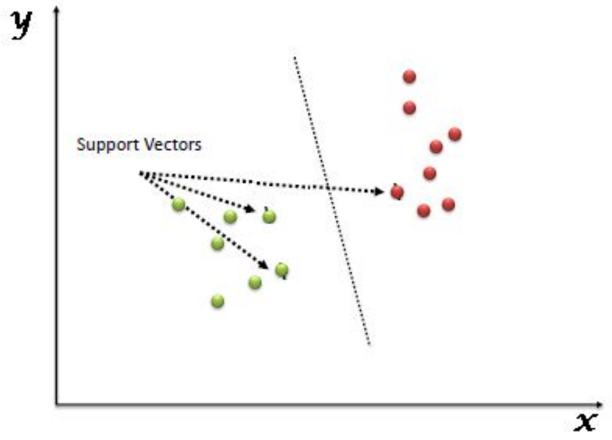
- If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as “Zero Frequency”. To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.
- On the other side naive Bayes is also known as a bad estimator, so the probability outputs from predict_proba are not to be taken too seriously.
- Another limitation of Naive Bayes is the assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.

Applications of Naive Bayes Algorithms

- **Real time Prediction:** Naive Bayes is an eager learning classifier and it is sure fast. Thus, it could be used for making predictions in real time.
- **Multi class Prediction:** This algorithm is also well known for multi class prediction feature. Here we can predict the probability of multiple classes of target variable.
- **Text classification/ Spam Filtering/ Sentiment Analysis:** Naive Bayes classifiers mostly used in text classification (due to better result in multi class problems and independence rule) have higher success rate as compared to other algorithms. As a result, it is widely used in Spam filtering (identify spam e-mail) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments)
- **Recommendation System:** Naive Bayes Classifier and Collaborative Filtering together builds a Recommendation System that uses machine learning and data mining techniques to filter unseen information and predict whether a user would like a given resource or not

What is Support Vector Machine?

- “Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges.
- In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate.
- Then, we perform classification by finding the hyper-plane that differentiates the two classes very well

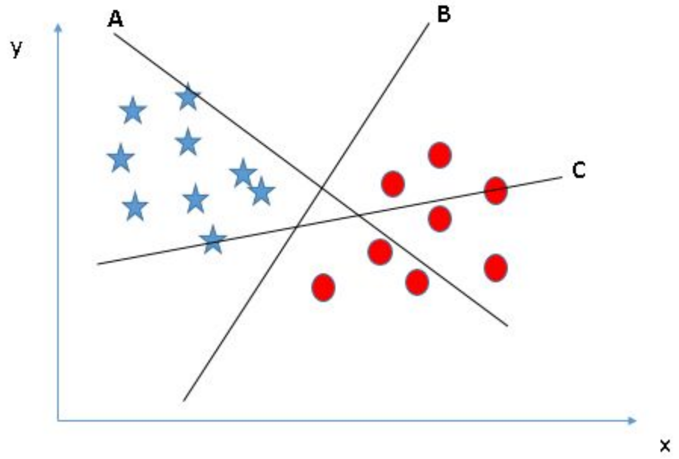


Support Vectors are simply the co-ordinates of individual observation.

The SVM classifier is a frontier which best segregates the two classes (hyper-plane/ line).

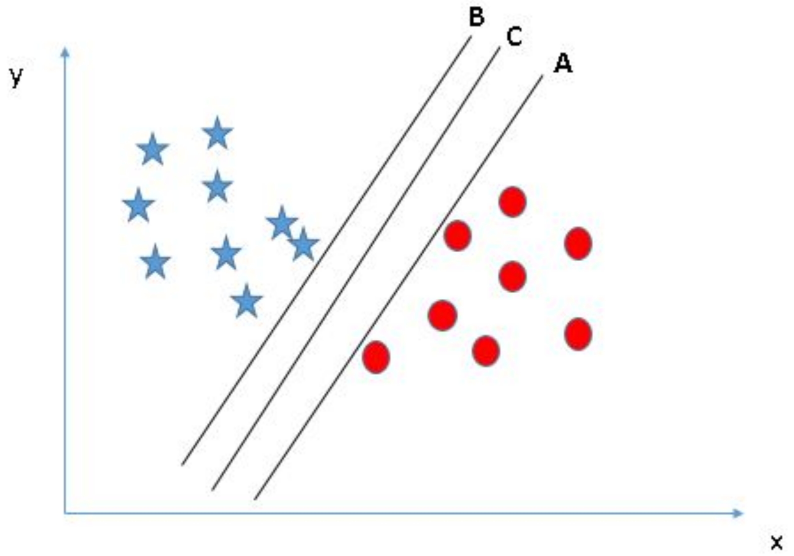
How does it work?

- Identify the right hyper-plane (Scenario-1): Here, we have three hyper-planes (A, B and C). Now, identify the right hyper-plane to classify star and circle.
- You need to remember a thumb rule to identify the right hyper-plane: "Select the hyper-plane which segregates the two classes better".
- In this scenario, hyper-plane "B" has excellently performed this job.



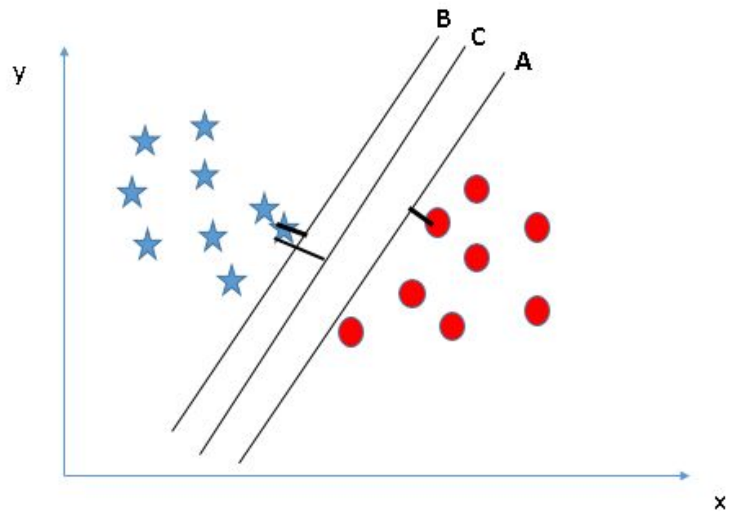
cont...

- Identify the right hyper-plane (Scenario-2): Here, we have three hyper-planes (A, B and C) and all are segregating the classes well. Now, How can we identify the right hyper-plane?
- Here, maximizing the distances between nearest data point (either class) and hyper-plane will help us to decide the right hyper-plane.
- This distance is called as Margin.

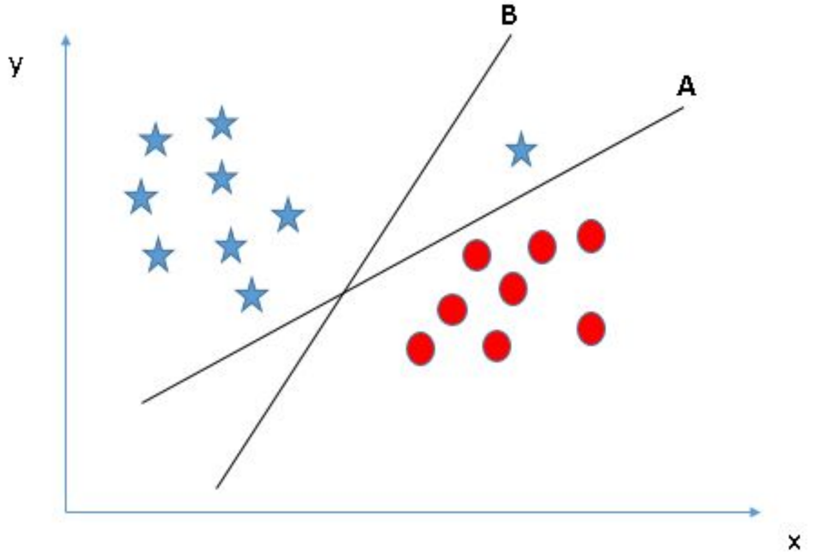


cont...

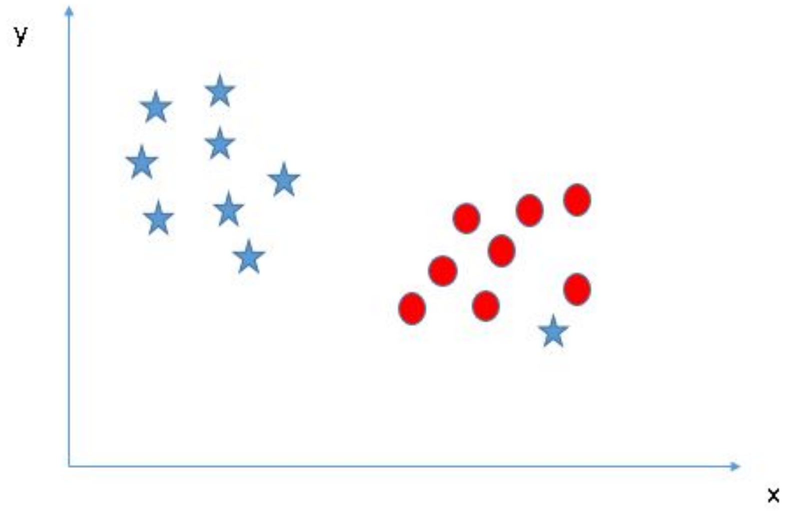
- Above, you can see that the margin for hyper-plane C is high as compared to both A and B.
- Hence, we name the right hyper-plane as C.
- Another lightning reason for selecting the hyper-plane with higher margin is robustness.
- If we select a hyper-plane having low margin then there is high chance of miss-classification.

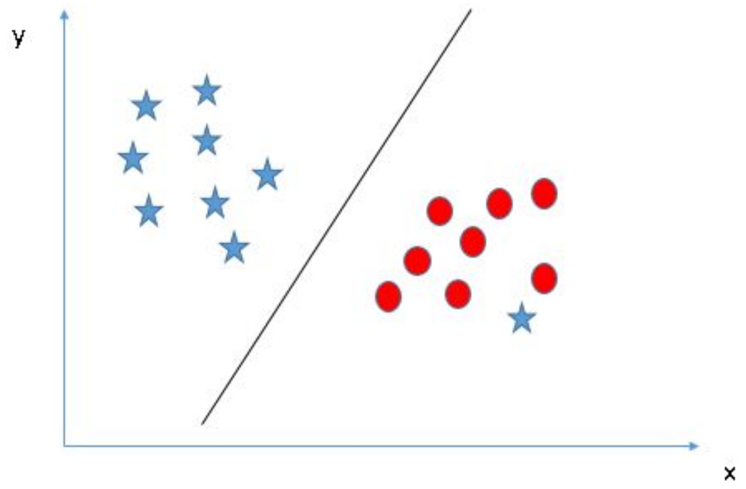


- Identify the right hyper-plane (Scenario-3):Hint: Use the rules as discussed in previous section to identify the right hyper-plane
- Some of you may have selected the hyper-plane B as it has higher margin compared to A.
- But, here is the catch, SVM selects the hyper-plane which classifies the classes accurately prior to maximizing margin.
- Here, hyper-plane B has a classification error and A has classified all correctly.
- Therefore, the right hyper-plane is A.

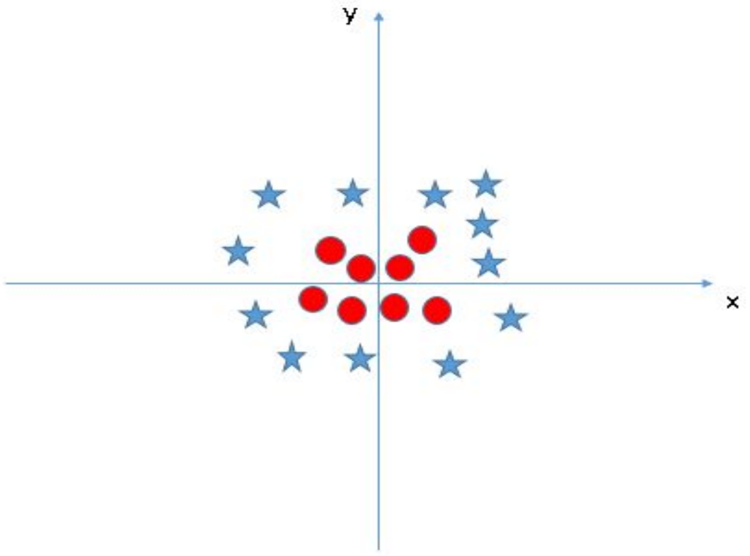


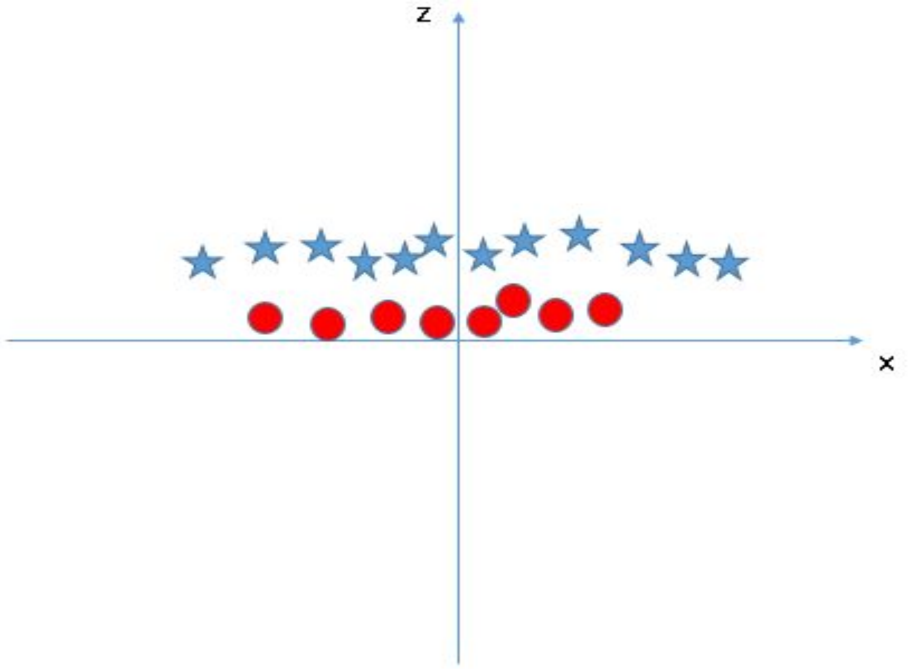
- **Can we classify two classes (Scenario-4)?:** Below, I am unable to segregate the two classes using a straight line, as one of the stars lies in the territory of other(circle) class as an outlier.
- As I have already mentioned, one star at other end is like an outlier for star class.
- The SVM algorithm has a feature to ignore outliers and find the hyper-plane that has the maximum margin.
- Hence, we can say, SVM classification is robust to outliers.





- Find the hyper-plane to segregate to classes (Scenario-5): In the scenario below, we can't have linear hyper-plane between the two classes, so how does SVM classify these two classes?
- Till now, we have only looked at the linear hyper-plane.
- SVM can solve this problem. Easily! It solves this problem by introducing additional feature.
- Here, we will add a new feature z=x^2+y^2.
- Now, let's plot the data points on axis x and z:





cont...

n above plot, points to consider are:

- All values for z would be positive always because z is the squared sum of both x and y
- In the original plot, red circles appear close to the origin of x and y axes, leading to lower value of z and star relatively away from the origin result to higher value of z.

But, another burning question which arises is, should we need to add this feature manually to have a hyper-plane

No, the SVM algorithm has a technique called the kernel trick.

- The SVM kernel is a function that takes low dimensional input space and transforms it to a higher dimensional space i.e. it converts not separable problem to separable problem
- It is mostly useful in non-linear separation problem.
- it does some extremely complex data transformations, then finds out the process to separate the data based on the labels or outputs you’ve defined.

What Is Model Selection

- Model selection is the process of selecting one final machine learning model from among a collection of candidate machine learning models for a training dataset.
- Model selection is a process that can be applied:
    - different types of models (e.g. logistic regression, SVM, KNN, etc.)
    - across models of the same type configured with different model hyperparameters (e.g. different kernels in an SVM).
- For example, we may have a dataset for which we are interested in developing a classification or regression predictive model. We do not know beforehand as to which model will perform best on this problem, as it is unknowable. Therefore, we fit and evaluate a suite of different models on the problem.

What do we care about when choosing a final model?

a “good enough” model may refer to many things and is specific to your project, such as:

- A model that meets the requirements and constraints of project stakeholders.
- A model that is sufficiently skillful given the time and resources available.
- A model that is skillful as compared to naive models.
- A model that is skillful relative to other tested models.
- A model that is skillful relative to the state-of-the-art.

model selection as the process of selecting among model development pipelines.

- Each pipeline may take in the same raw training dataset and outputs
- a model that can be evaluated in the same manner but may require different or overlapping computational steps, such as:
  - Data filtering.
  - Data transformation.
  - Feature selection.
  - Feature engineering.
  - And more…

The closer you look at the challenge of model selection, the more nuance you will discover.

Model Selection Techniques

- **Probabilistic Measures:** Choose a model via in-sample error and complexity.
- **Resampling Methods:** Choose a model via estimated out-of-sample error.

Probabilistic Measures

- Probabilistic measures involve analytically scoring a candidate model using both its performance on the training dataset and the complexity of the model.
- Four commonly used probabilistic model selection measures include:
    - Akaike Information Criterion (AIC).
    - Bayesian Information Criterion (BIC).
    - Minimum Description Length (MDL).
    - Structural Risk Minimization (SRM).

Resampling Methods

- Resampling methods seek to estimate the performance of a model (or more precisely, the model development process) on out-of-sample data.

Three common resampling model selection methods include:

- Random train/test splits.
- Cross-Validation (k-fold, LOOCV, etc.).
- Bootstrap.

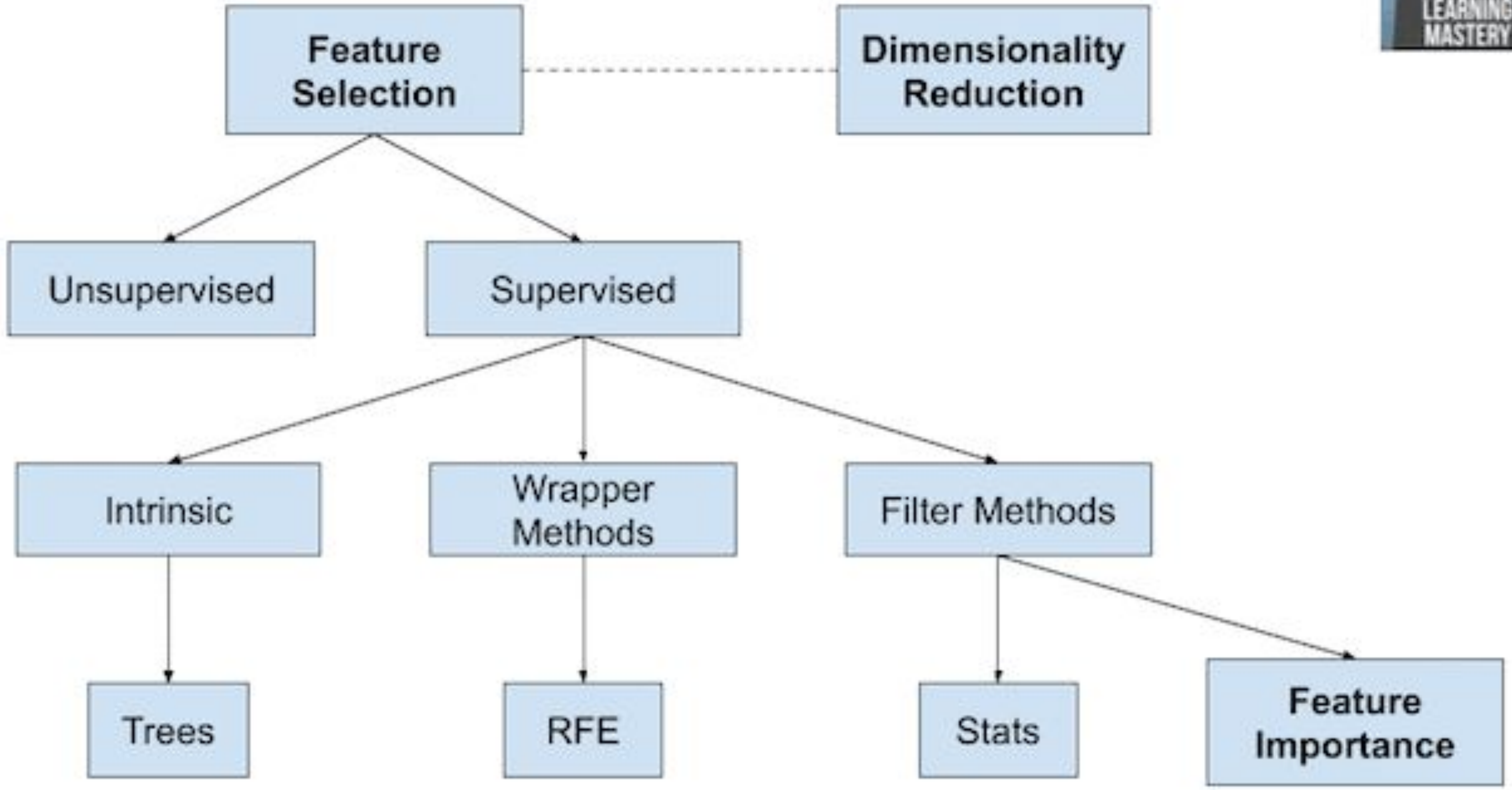
Feature Selection Methods

- Feature Selection is the process where you automatically or manually select those features which contribute most to your prediction variable or output in which you are interested in.
- Feature selection methods are intended to reduce the number of input variables to those that are believed to be most useful to a model in order to predict the target variable.
- How to select features and what are Benefits of performing feature selection before modeling your data?
    - **Reduces Overfitting:** Less redundant data means less opportunity to make decisions based on noise.
    - **Improves Accuracy:** Less misleading data means modeling accuracy improves.
    - **Reduces Training Time:** fewer data points reduce algorithm complexity and algorithms train faster.

Summary

- Feature Selection: Select a subset of input features from the dataset.
    - Unsupervised: Do not use the target variable (e.g. remove redundant variables).
        - Correlation
    - Supervised: Use the target variable (e.g. remove irrelevant variables).
        - Wrapper: Search for well-performing subsets of features.
            - RFE
        - Filter: Select subsets of features based on their relationship with the target.
            - Statistical Methods
            - Feature Importance Methods
        - Intrinsic: Algorithms that perform automatic feature selection during training.
            - Decision Trees
- Dimensionality Reduction: Project input data into a lower-dimensional feature space.

Overview of Feature Selection Techniques

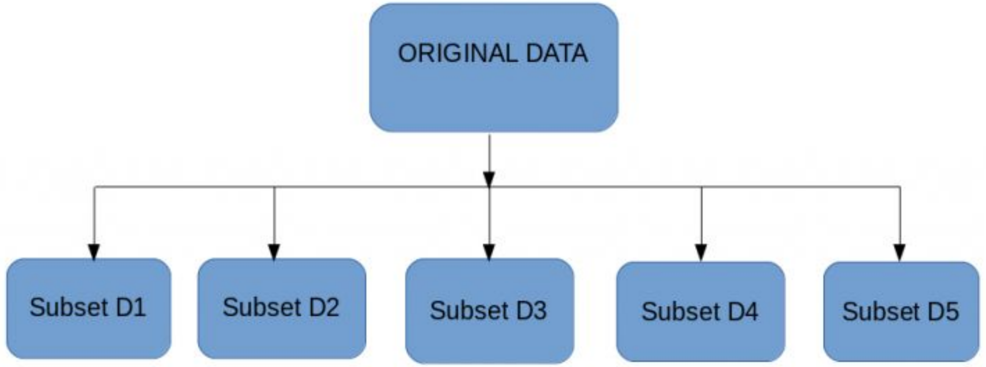


Ensemble method

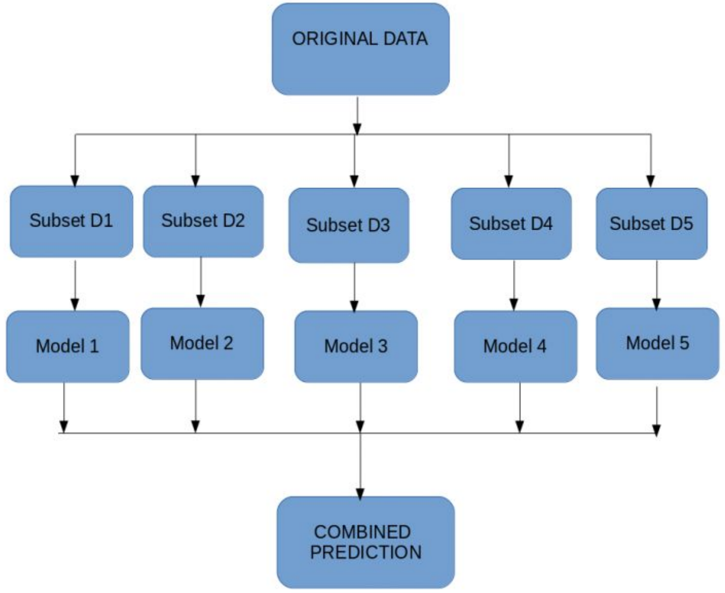
- Ensemble models in machine learning operate on a similar idea.
- They combine the decisions from multiple models to improve the overall performance.
- Ensemble learning is a machine learning paradigm where multiple models (often called “weak learners”) are trained to solve the same problem and combined to get better results.
- The main hypothesis is that when weak models are correctly combined we can obtain more accurate and/or robust models.

Bagging

- that often considers homogeneous weak learners, learns them independently from each other in parallel and combines them following some kind of deterministic averaging process
- Bagging technique uses these subsets (bags) to get a fair idea of the distribution (complete set). The size of subsets created for bagging may be less than the original set.
-



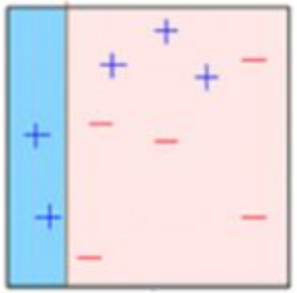
1. Multiple subsets are created from the original dataset, selecting observations with replacement.
2. A base model (weak model) is created on each of these subsets.
3. The models run in parallel and are independent of each other.
4. The final predictions are determined by combining the predictions from all the models



Boosting

- Boosting is a sequential process, where each subsequent model attempts to correct the errors of the previous model.
- The succeeding models are dependent on the previous model.
- Let's understand the way boosting works in the below steps:

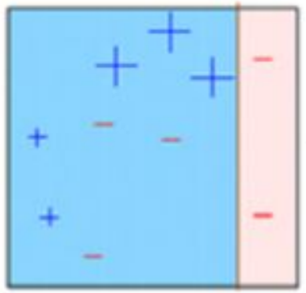
1. A subset is created from the original dataset.
2. Initially, all data points are given equal weights.
3. A base model is created on this subset.
4. This model is used to make predictions on the whole da



5. Errors are calculated using the actual values and predicted values.

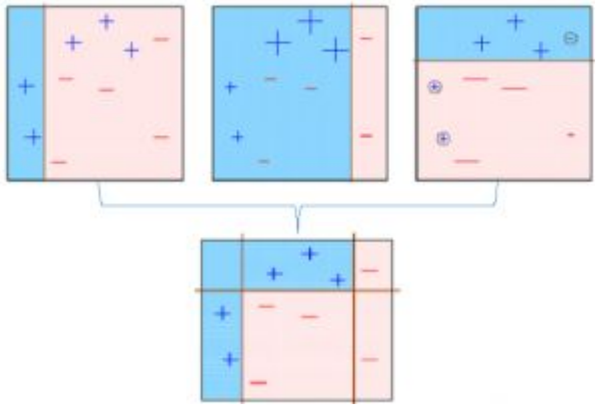
6. The observations which are incorrectly predicted, are given higher weights.
(Here, the three misclassified blue-plus points will be given higher weights)

7. Another model is created and predictions are made on the dataset.
(This model tries to correct the errors from the previous model)



8. Similarly, multiple models are created, each correcting the errors of the previous model.

9. The final model (strong learner) is the weighted mean of all the models (weak learners)



S.NO	Bagging	Boosting
1.	Simplest way of combining predictions that belong to the same type.	A way of combining predictions that belong to the different types.
2.	Aim to decrease variance, not bias.	Aim to decrease bias, not variance.
3.	Each model receives equal weight.	Models are weighted according to their performance.
4.	Each model is built independently.	New models are influenced by performance of previously built models.
5.	Different training data subsets are randomly drawn with replacement from the entire training dataset.	Every new subsets contains the elements that were misclassified by previous models.
6.	Bagging tries to solve over-fitting problem.	Boosting tries to reduce bias.
7.	If the classifier is unstable (high variance), then apply bagging.	If the classifier is stable and simple (high bias) the apply boosting.
8.	Random forest.	Gradient boosting.

AdaBoost algorithm

- AdaBoost is best used to boost the performance of decision trees on binary classification problems.
- AdaBoost can be used to boost the performance of any machine learning algorithm.
- It is best used with weak learners. These are models that achieve accuracy just above random chance on a classification problem.
- Weak models are added sequentially, trained using the weighted training data.
- The process continues until a pre-set number of weak learners have been created (a user parameter) or no further improvement can be made on the training dataset.
-

Evaluating learning algorithm

- Evaluating your machine learning algorithm is an essential part of any project
- Various methods we have:
    - Classification Accuracy
    - Logarithmic Loss
    - Confusion Matrix
    - Area under Curve
    - F1 Score
    - Mean Absolute Error
    - Mean Squared Error

Classification Accuracy

It is the ratio of number of correct predictions to the total number of input samples.

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

Logarithmic Loss

- Logarithmic Loss or Log Loss, works by penalising the false classifications.
- It works well for multi-class classification.
- When working with Log Loss, the classifier must assign probability to each class for all the samples.
- Suppose, there are N samples belonging to M classes, then the Log Loss is calculated as below :

$$LogarithmicLoss = \frac{-1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} * \log(p_{ij})$$

where,

y_ij, indicates whether sample i belongs to class j or not

p_ij, indicates the probability of sample i belonging to class j

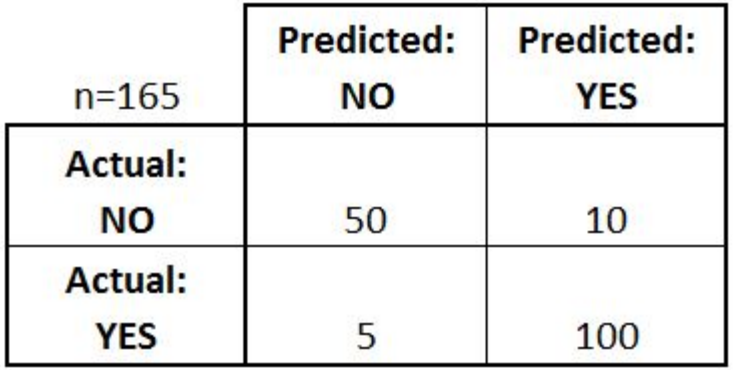
Log Loss has no upper bound and it exists on the range [0, ∞). Log Loss nearer to 0 indicates higher accuracy, whereas if the Log Loss is away from 0 then it indicates lower accuracy.

In general, minimising Log Loss gives greater accuracy for the classifier.

Confusion Matrix

There are 4 important terms :

- True Positives : The cases in which we predicted YES and the actual output was also YES.
- True Negatives : The cases in which we predicted NO and the actual output was NO.
- False Positives : The cases in which we predicted YES and the actual output was NO.
- False Negatives : The cases in which we predicted NO and the actual output was YES.



n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

Area Under Curve

- It is used for binary classification problem. AUC of a classifier is equal to the probability that the classifier will rank a randomly chosen positive example higher than a randomly chosen negative example. Before defining AUC, let us understand two basic terms :
- True Positive Rate (Sensitivity) : True Positive Rate is defined as TP/ (FN+TP).

$$TruePositiveRate = \frac{TruePositive}{FalseNegative + TruePositive}$$

- True Negative Rate (Specificity) : True Negative Rate is defined as TN / (FP+TN).

$$TrueNegativeRate = \frac{TrueNegative}{TrueNegative + FalsePositive}$$

False Positive Rate : False Positive Rate is defined as *FP / (FP+TN)*.

$$FalsePositiveRate = \frac{FalsePositive}{TrueNegative + FalsePositive}$$

F1 Score

- F1 Score is used to measure a test's accuracy
- F1 Score is the Harmonic Mean between precision and recall.
- The range for F1 Score is [0, 1].
- It tells you how precise your classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances).

$$F1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$$

Mean Absolute Error

- Mean Absolute Error is the average of the difference between the Original Values and the Predicted Values.
- It gives us the measure of how far the predictions were from the actual output.

$$MeanAbsoluteError = \frac{1}{N}\sum_{j=1}^{N}|y_j - \hat{y}_j|$$

Debugging Learning Algorithms

- A notoriously hard problem in general
  - Note that code for ML algorithms is not procedural but data-driven
- What to do when our model (say logistic regression) isn't doing well (i.e., giving an acceptable level of test accuracy) but you are confident that your implementation is otherwise correct?
  - Use more training examples to train the model?
  - Use a smaller number of features?
  - Introduce new features (can be combinations of existing features)?
  - Try tuning the regularization parameter?
  - Run (the iterative) optimizer longer, i.e., for more iterations?
  - Change the optimization algorithm (e.g., GD to SGD or Newton..)?
  - Give up and switch to a different model (e.g., SVM)?
- How to know what might be going wrong and how to debug?

Bias-Variance Decomposition

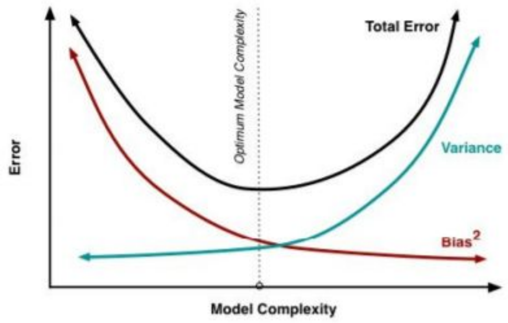
- For some model $y = f(\boldsymbol{x}) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma^2)$, given its estimate $\hat{f}$ learned by a "learner" using a finite training set, the following decomposition holds

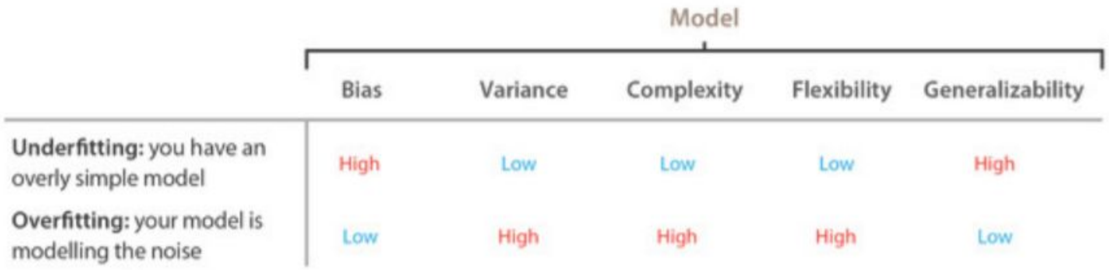
$$\mathbb{E}[(y - \hat{f}(\boldsymbol{x}))^2] = \text{Bias}[\hat{f}(\boldsymbol{x})]^2 + \text{Var}[\hat{f}(\boldsymbol{x})] + \sigma^2$$

- Note: The above expectation is over all choices of training sets
- $\text{Bias}[\hat{f}(\boldsymbol{x})] = \mathbb{E}[\hat{f}(\boldsymbol{x}) - f(\boldsymbol{x})]$:Error due to wrong (perhaps too simple) model
- $\text{Var}[\hat{f}(\boldsymbol{x})] = \mathbb{E}[\hat{f}(\boldsymbol{x})^2] - \mathbb{E}[\hat{f}(\boldsymbol{x})]^2$:Learner's sensitivity to choice of training set
- The proof (note that $\mathbb{E}[y] = f(\boldsymbol{x})$):

$$\begin{aligned}
\mathrm{E}\big[(y - \hat{f})^2\big] &= \mathrm{E}[y^2 + \hat{f}^2 - 2y\hat{f}] \\
&= \mathrm{E}[y^2] + \mathrm{E}[\hat{f}^2] - \mathrm{E}[2y\hat{f}] \\
&= \mathrm{Var}[y] + \mathrm{E}[y]^2 + \mathrm{Var}[\hat{f}] + \mathrm{E}[\hat{f}]^2 - 2f\mathrm{E}[\hat{f}] \\
&= \mathrm{Var}[y] + \mathrm{Var}[\hat{f}] + (f - \mathrm{E}[\hat{f}])^2 \\
&= \mathrm{Var}[y] + \mathrm{Var}[\hat{f}] + \mathrm{E}[f - \hat{f}]^2 \\
&= \sigma^2 + \mathrm{Var}[\hat{f}] + \mathrm{Bias}[\hat{f}]^2
\end{aligned}$$

- Simple models have high bias and small variance, complex models have small bias and high varianc



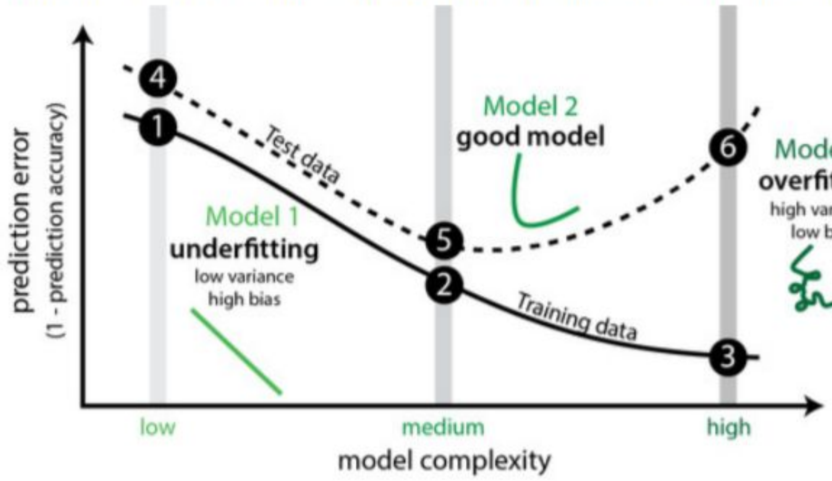


	Model				
	Bias	Variance	Complexity	Flexibility	Generalizability
Underfitting: you have an overly simple model	High	Low	Low	Low	High
Overfitting: your model is modelling the noise	Low	High	High	High	Low

- If you modified a model to reduce its bias (e.g., by increasing the model's complexity), you are likely to increase its variance, and vice-versa (if both increase then you might be doing it wrong!)

High Bias or High Variance?

- The bad performance (low accuracy on test data) could be due either
  - High Bias (Underfitting)
  - High Variance (Overfitting)
- Looking at the training and test error can tell which of the two is the case



- High Bias: Both training and test errors are large
- High Variance: Small training error, large test error (and huge gap)

Some Guidelines for Debugging Learning Algorithms

- Adding more training examples won't usually bring the bias down. If your model has a high bias, try making the model richer (e.g., adding more features or using a more sophisticated model).
- Using more training data can help bring the variance down. If your model has a high variance, try adding more training examples or make model simpler (e.g., use fewer features or regularize more)
- Suppose you have learned two models $\boldsymbol{w}_{LR}$ and $\boldsymbol{w}_{SVM}$ (LR and SVM, respectively) using the same training data, and SVM gives higher test accuracy. How do I know why LR does worse and what could I improve it?
  - Is it because the optimizer for LR didn't do a good job at finding the optima?
  - Is my model choice (choosing LR over SVM) wrong for this data set?
  - Looking at the value of the LR loss function $\mathcal{L}$ can give some insights
  - If $\mathcal{L}(\boldsymbol{w}_{SVM}) < \mathcal{L}(\boldsymbol{w}_{LR})$ then improving the LR optimizer might help
  - If $\mathcal{L}(\boldsymbol{w}_{LR}) < \mathcal{L}(\boldsymbol{w}_{SVM})$ then LR isn't a good model for this problem

Classification Error

The classification error E_i of an individual program i depends on the number of samples incorrectly classified (false positives + false negatives) and is evaluated by the formula:

$$E_i = \frac{f}{n} \cdot 100$$

where f is the number of sample cases incorrectly classified, and n is the total number of sample cases.