Course Title : Machine Learning
Course No : C.Sc. 561
Nature of the Course : Theory + Lab

Full Marks : 45+30
Pass Marks : 22.5+15
Credit Hrs : 3

Course Description

This course provides a broad introduction to machine learning and statistical pattern recognition. Topics include: supervised learning (generative/discriminative learning, parametric/non-parametric learning, neural networks, support vector machines); unsupervised learning (clustering, dimensionality reduction, kernel methods); learning theory (bias/variance tradeoffs; VC theory; large margins); reinforcement learning and adaptive control.

Course Objective

Purpose of this course is to present different machine learning techniques and also analyze their pros and cons. In addition to this, this course also concepts on learning theory and their applications.

Prerequisites

Computer Programming, Probability Theory, Linear Algebra.

Unit 1 : Introduction - 5 hrs

The Motivation & Applications of Machine Learning, The Definition of Machine Learning, The Overview of Supervised Learning, The Overview of Learning Theory, The Overview of Unsupervised Learning, The Overview of Reinforcement Learning

Unit 2: Supervised learning - 12 hrs

Application of Supervised Learning, Linear Regression, Gradient Descent, Batch Gradient Descent, Stochastic Gradient Descent (Incremental Descent), Matrix Derivative Notation for Deriving Normal Equations, Derivation of Normal Equations, The Concept of Underfitting and Overfitting, The Concept of Parametric Algorithms and Non-parametric Algorithms, Locally Weighted Regression, The Probabilistic Interpretation of Linear Regression, The motivation of Logistic Regression, Supervised learning setup, Least Mean squares, Logistic Regression, Perceptron Learning Algorithm, Discriminative Algorithms, Generative Algorithms, Gaussian Discriminant Analysis (GDA) GDA and Logistic Regression, Naive Bayes, Laplace Smoothing, Intuitions about Support Vector Machine (SVM), Notation for SVM, Functional and Geometric Margins, Optimal Margin Classifier, Lagrange Duality, Karush-Kuhn-Tucker (KKT) Conditions, SVM Dual, The Concept of Kernels, kernels, Mercer's Theorem, Non-linear Decision Boundaries and Soft Margin SVM, Coordinate Ascent Algorithm, The sequential Minimization Optimization (SMO) Algorithm, Applications of SVM.

Unit 3: Learning theory - 9 hrs

Bias/variance Tradeoff, Empirical Risk Minimization (ERM), The Union Bound, Hoeffding Inequality, Uniform Convergence- The case of Finite H, Sample complexity Bound, Error Bound, Uniform Convergence Theorem & Corollary, Uniform Convergence- The case of Infinite H, The Concept of 'Shatter' and VC Dimension, SVM Example, Model Selection, Cross Validation, Feature Selection, Bayesian Statistics and Regularization, Online Learning,

Advice for Applying Machine Learning Algorithms, Debugging / fixing Learning Algorithms, Diagnostics for Bias & Variance, Optimization Algorithm Diagnostics, Diagnostic Example Error Analysis, Getting started on a Learning Problem

Unit 4 : Unsupervised Learning - 9hrs

The Concept of Unsupervised Learning, K-means Clustering Algorithm, K-means Algorithm, Mixtures of Gaussians and the EM Algorithm, Jensen's Inequality, The EM Algorithm, Mixture of Gaussian, Mixture of Naive Bayes - Text clustering (EM Application), Factor Analysis, Restrictions on a Covariance Matrix, The Factor Analysis Model, EM for Factor Analysis, The Factor Analysis Model, EM for Factor Analysis, Principal Component Analysis (PCA), PCA as a Dimensionality Reduction Algorithm, Applications of PCA, Face Recognition by using PCA, Latent Semantic Indexing (LSI), Singular Value Decomposition (SVD) Implementation, Independent Component Analysis (ICA), The Application of ICA, Cumulative Distribution Function (CDF), ICA Algorithm, The Applications of ICA

Unit 5 : Reinforcement learning and control - 10hrs

Applications of Reinforcement Learning, Markov Decision Process (MDP), Defining Value & Policy Functions, Value Function, Optimal Value Function, Value Iteration, Policy Iteration, Generalization to Continuous States, Discretization & Curse of Dimensionality, Models / Simulators, Fitted Value Iteration, Finding Optimal Policy, State - action Rewards, Finite Horizon MDPs, The Concept of Dynamical Systems, Examples of Dynamical Models, Linear Quadratic Regulation (LQR), Linearizing a Non-Linear Model, Computing Rewards, Riccati Equation, Advice for Applying Machine Learning,

Debugging Reinforcement Learning (RL) Algorithm, Linear Quadratic Regularization (LQR), Differential Dynamic Programming (DDP), Kalman Filter & Linear Quadratic Gaussian (LQG), Predict | update steps of Kalman Filter, Linear Quadratic Gaussian (LQG) Partially Observable MDPs (POMDPs), Policy search, Reinforce Algorithm, Pegasus Algorithm, Pegasus Policy search, Applications of Reinforcement Learning

Text Books

1. Christopher Bishop, Pattern Recognition and Machine Learning Springer, 2006
2. Richard Duda, Peter Hart and David Stork, Pattern Classification, 2nd ed. John Wiley & sons, 2001.

References

1. Tom Mitchell, Machine Learning. McGraw-Hill, 1997.
2. Richard Sutton and Andrew Barto, Reinforcement Learning : An introduction. MIT Press, 1998.

classmate
Date 2019/02/20
Page

Unit 1 : Introduction

What is Machine Learning?

- Machine learning is an application of AI that enables systems to learn and improve from experience without being explicitly programmed.
- Machine learning focuses on developing computer programs that can access data and use it to learn for themselves.
- The machine learning process begins with observations or data, such as examples, direct experience or instruction.
- It looks for patterns in data so it can later make inferences based on the examples provided.
- The primary aim of ML is to allow computers to learn autonomously without human intervention or assistance and adjust actions accordingly.

How Machine Learning Works?

- Learning system of a machine learning algorithm into three main parts :

- A Decision Process : In general, machine learning algorithms are used to make a prediction or classification Based on some input data, which can be labelled or unlabelled, an algorithm will produce an estimate about a pattern in the data.

- An Error Function : An error function serves to evaluate the prediction of the model. If there are known examples, an error function can make a comparison to assess the accuracy of the model.

- An Model Optimization Process: If the model can fit better to the data points in the training set, then parameters are adjusted to reduce the discrepancy between the known example and the model estimate. The algorithm will repeat this evaluate and optimize process, until a threshold of accuracy has been met.

Applications of Machine Learning

- We are using machine learning in our daily life even without knowing it such as Google Maps, Google assistant, Alexa, etc. Below are some most trending real-world applications of Machine Learning:

1. **Image Recognition:**
   It is used to identify objects, persons, places, digital images, etc. The popular use case of image recognition and face detection is, Automatic friend tagging suggestion.

2. **Speech Recognition:**
   Speech recognition is a process of converting voice instructions into text, and it is also known as speech to text, or Computer speech recognition. Google assistant, Siri, and Alexa are using speech recognition technology to follow the voice instructions.

3. **Traffic Prediction:**
   If we want to visit a new place, we take help of Google Maps, which shows us the correct path with the shortest route and predicts the traffic conditions.

4. Product Recommendations:

Machine learning is widely used by various e-commerce and entertainment companies such as Amazon, Netflix, etc. for product recommendation to the user.

5. Self-driving Cars:

Machine learning plays a significant role in self-driving cars. Tesla, the most popular car manufacturing company is working on self-driving car.

6. Email Spam and Malware Filtering:

Whenever we receive a new email, it is filtered automatically as important, normal, and spam. The technology behind this is Machine Learning.

7. Online Fraud Detection:

Machine learning is making our online transaction safe and secure by detecting fraud transaction. Outlier detection approach is primarily used in fraud detection.

8. Stock Market Trading:

Machine learning is widely used in stock market trading. Time series prediction ML models are used for the prediction of stock market trends.

9. Medical Diagnosis:

In medical science, machine learning is used for disease diagnoses.

10. Machine Translation:

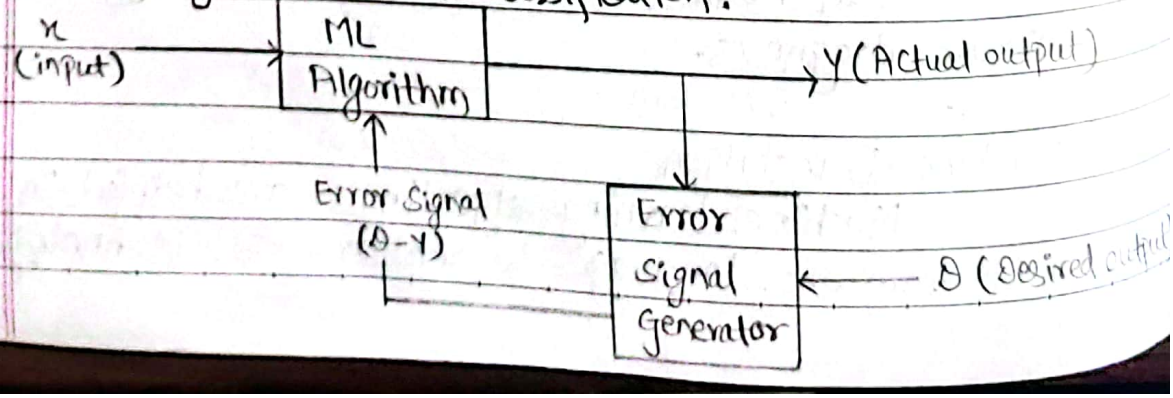
Machine learning algorithms are helpful in translating text from one language to another. The technology behind

the automatic translation is a sequence to sequence learning algorithm.

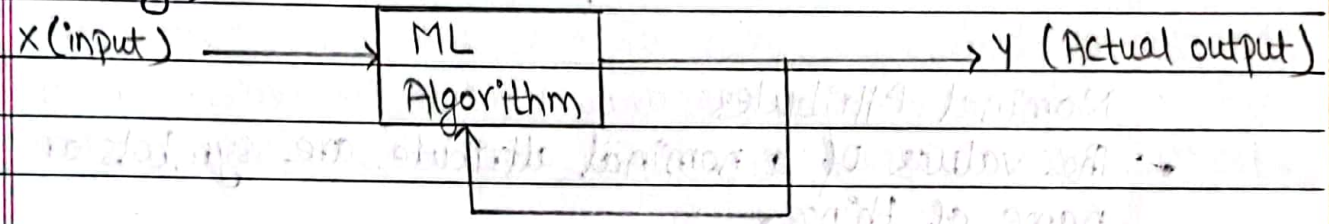
Machine Learning Methods

- Machine Learning Methods fall into three primary categories:
  - Supervised
  - Unsupervised, and
  - Reinforcement

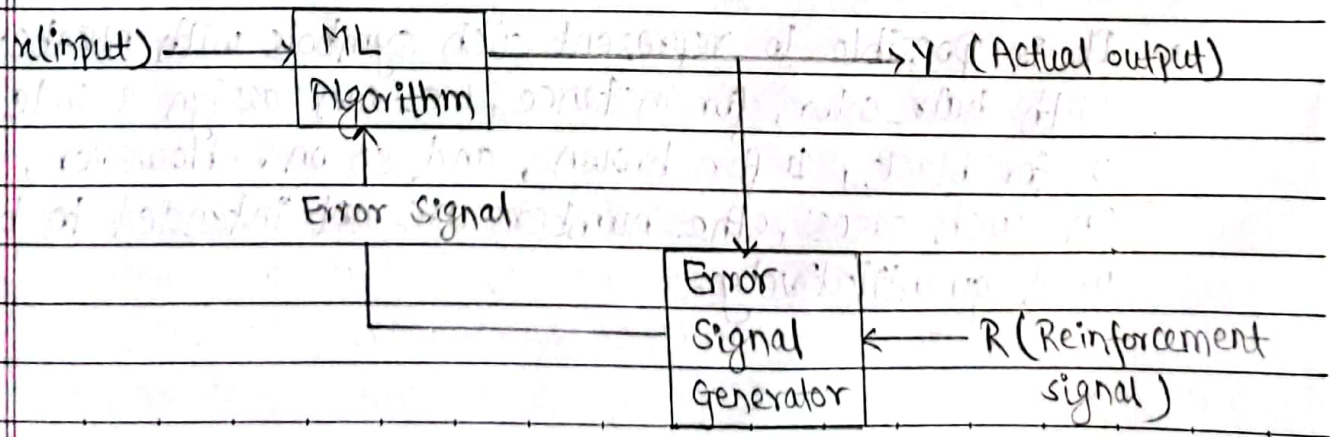
- **Supervised Learning:**
  - In this learning paradigm, we present examples of correct input-output pairs to the ML algorithms during the training phase.
  - This training set of examples is equivalent to the teacher for the ML algorithms.
  - During the training of ML algorithm under supervised learning, then it takes input vector and computes output vector.
  - An error signal is generated, if there is a difference between the computed output and the desired output vector.
  - On the basis of this error signal, the model parameters are adjusted until the actual output is matched with the desired output.
  - Supervised machine learning is used for performing tasks like: Regression and Classification.



- **Unsupervised Learning:**
  - In unsupervised learning, ML algorithm is provided with dataset without desired output.
  - The ML algorithm then attempts to find structure in the data by extracting useful features and analyzing its structure.
  - Unsupervised learning algorithms are widely used for tasks like: clustering, dimensionality reduction, association mining, etc.



- **Reinforcement Learning:**
  - In reinforcement learning, we do not provide the machine with examples of correct input-output pairs, but we do provide a method for the machine to quantify its performance in the form of a reward signal.
  - Reinforcement learning methods resemble how humans and animals learn: the machine tries a bunch of different things and is rewarded with performance signal.
  - Reinforcement learning algorithms are widely used for training agents interacting with its environment.



Data Objects and Attribute Types

Types of Attributes

On the basis of set of possible values, attributes can be divided into following types :

- Nominal Attributes
- Ordinal Attributes
- Interval-scaled Attributes
- Ratio-scaled Attributes

Nominal Attributes

- The values of a nominal attribute are symbols or name of things.
- Each value represents some kind of category, code, or state, and so nominal attributes are also referred to as categorical. The values do not have any meaningful order.
- Examples of nominal attributes :
  - → Hair color :- possible values are : {black, brown, red, grey, white}
  - → Marital status :- possible values are : {Married, Single, Divorced, Widowed}
  - → Customer ID :- possible values are : Combination of numbers
- It is possible to represent such symbols with numbers. With hair_color, for instance, we can assign a code of 0 for black, 1 for brown, and so on. However, in such cases, the numbers are not intended to be used quantitatively.

Ordinal Attributes

- An ordinal attribute is an attribute with possible values that have a meaningful order or ranking among them, but the magnitude between successive values is not known.
- Examples of ordinal attributes:
  - → Grades :- possible values are : {A+, A, A-, B+, B, B- and so on}
  - → Height :- possible values are : {Tall, Medium, Short}
- The values have a meaningful sequence (which corresponds to increasing height); however, we cannot tell from the values how much bigger, say, a medium is than a short.
- Note that, nominal and ordinal attributes are qualitative. That is, they describe a feature of an object without giving an actual size or quantity.
- We can compute median and mode of ordinal attributes. However, we cannot compute mean.
- But we can only compute mode of nominal attributes.

Interval-Scaled Attributes

- Interval-scaled attributes are numeric attributes. A numeric attribute is quantitative; that is, it is a measurable quantity, represented in integer or real values.
- The values of interval-scaled attributes have order and can be positive, 0, or negative. Thus, in addition to providing a ranking of values, such attributes allows us to compare and quantify the difference between values.
- Because interval-scaled attributes are numeric, we can compute their mean value, in addition to the median and

mode measures of central tendency.

- A temperature attribute is interval-scaled. Suppose that we have the outdoor temperature value for a number of different days, where each day is an object. By ordering the values, we obtain a ranking of the objects with respect to temperature.
- In addition, we can quantify the difference between values. For example, a temperature of 20°C is five degrees higher than a temperature of 15°C.
- Calendar dates are another example. For instance, the years 2002 and 2010 are eight years apart.
- Temperatures in Celsius and Fahrenheit do not have a true zero-point that is, neither 0°C nor 0°F indicates "no temperature".
- Although we can compute the difference between temperature values, we cannot talk of one temperature value as being a multiple of another.
- Without a true zero, we cannot say, for instance, that 10°C is twice as warm as 5°C. That is, we cannot speak of the values in terms of ratios. Similarly, there is no true zero-point for calendar dates.

Ratio-Scaled Attributes

- A ratio-scaled attribute is a numeric attribute with an inherent zero-point.
- That is, if a measurement is ratio-scaled, we can speak of a value as being a multiple (or ratio) of another value.
- In addition, the values are ordered, and we can also compute the difference between values, as well as the

mean, median, and mode.

- Temperature in kelvin, length, counts, elapsed time, etc. are examples of ratio scaled attributes.

Differences between 3 machine learning algorithms:

Criteria	Supervised ML	Unsupervised ML	Reinforcement ML
Definition	• Learn by using labelled data	• Trained using unlabelled data without any guidance	• Works on interacting with the environment
Types of data	• Labelled data	• Unlabelled data	• No predefined data
Types of Problems	• Regression and Classification	• Association and Clustering, dimensionality reductio	• Exploitation or Exploration
Supervision	• Extra Supervision	• No supervision	• No supervision
Algorithms	• Linear regression, Logistic regression, SVM, KNN, etc.	• K-mean C-mean Apriori	• Q-Learning SARSA
Aim	• Risk Evaluation, Forecast sales	• Recommendation system, Anomaly Detection	• Self Driving Cars, Gaming, Healthcare

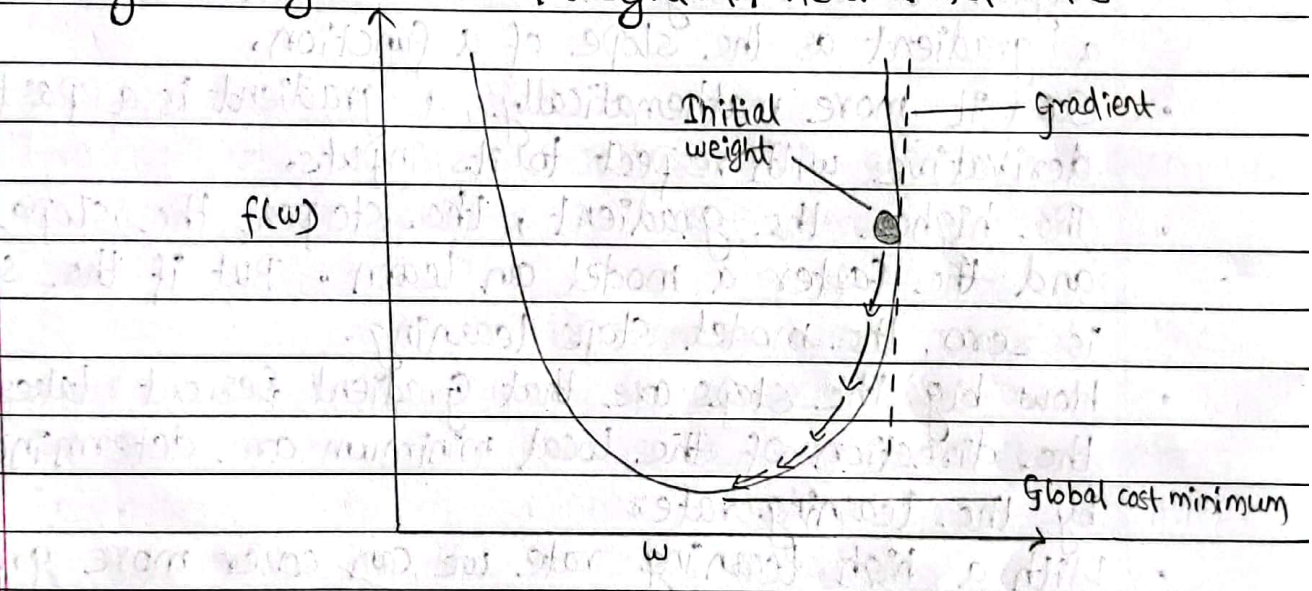
Classmate
Date 02/24
Page

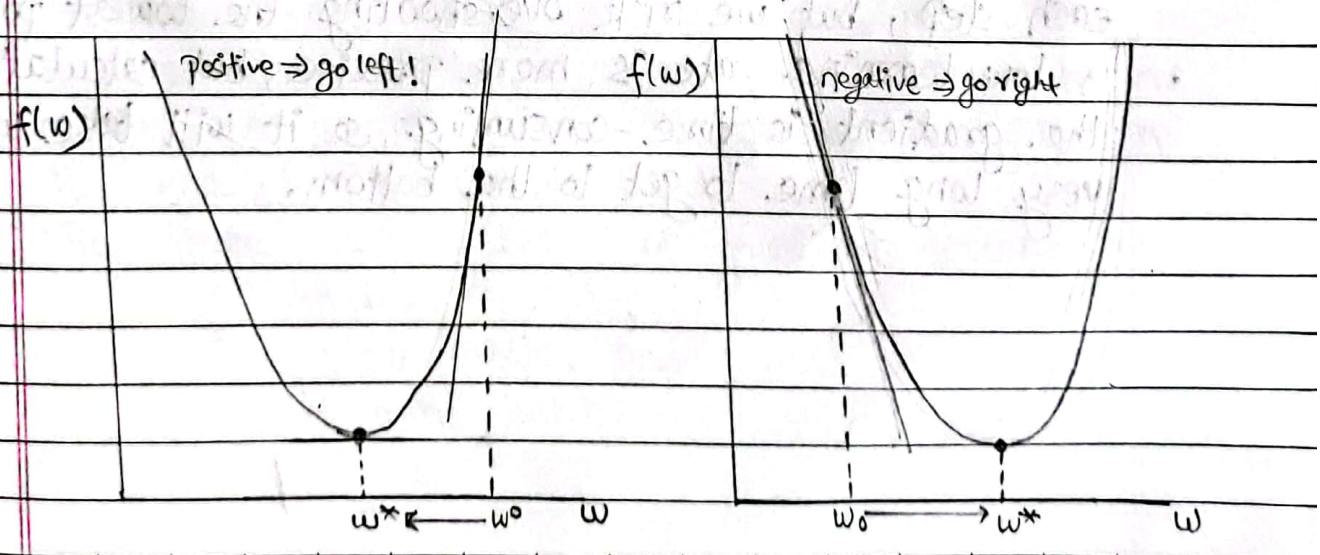
Unit 2: Supervised Learning

Gradient Descent

- Gradient Descent is an optimization algorithm used to minimize some convex function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient.

- In machine learning, we use gradient descent to update the parameters of our model. Parameters refer to coefficients in Logistic Regression and weights in neural networks.



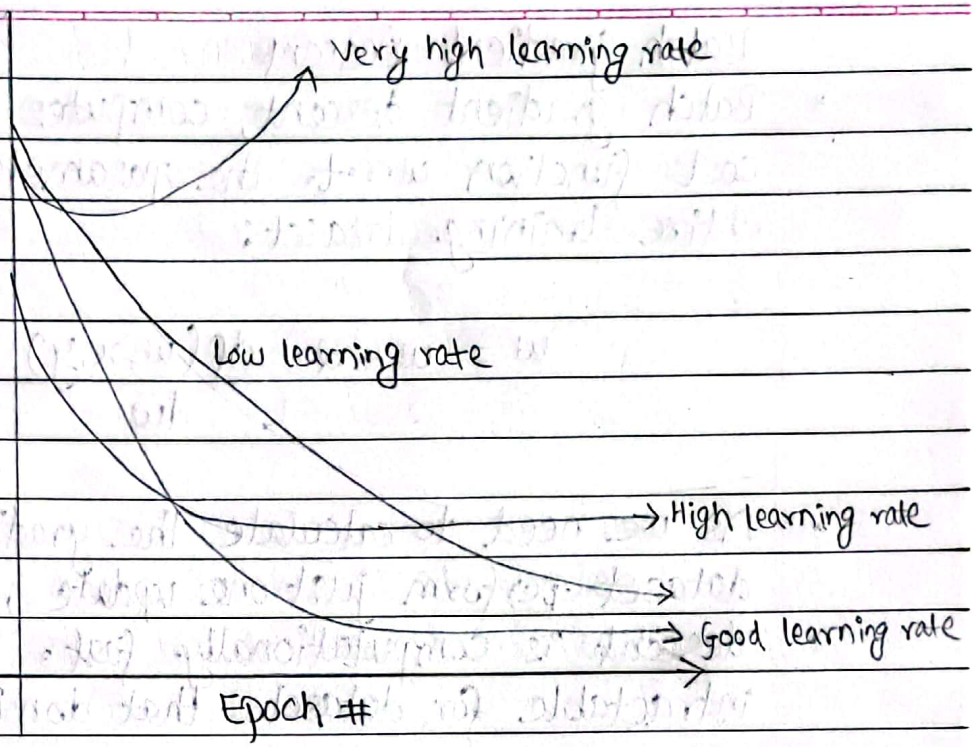


- If f is function to be minimized (cost function), Gradient descent changes the parameters of learning model iteratively as below:

$$w = w - \alpha \frac{df}{dw}$$

where,
α is learning rate and
w is parameter to be optimized

- GD simply measures the changes in all weights with regard to the change in error. We can also think of a gradient as the slope of a function.
- Said it more mathematically, a gradient is a partial derivative with respect to its inputs.
- The higher the gradient, the steeper the slope and the faster a model can learn. But if the slope is zero, the model stops learning.
- How big the steps are that Gradient Descent takes into the direction of the local minimum are determined by the learning rate.
- With a high learning rate we can cover more ground each step, but we risk overshooting the lowest point.
- A low learning rate is more precise, but calculating the gradient is time-consuming, so it will take us a very long time to get to the bottom.



Gradient Descent Variations

- Gradient descent is an optimization algorithm often used for finding the weights or coefficients of machine learning algorithms.
- It uses the error on the predictions to update the model in such a way as to reduce the error.
- The goal of the algorithm is to find model parameters (e.g. coefficients or weights) that minimize the error of the model on the training dataset.
- It does this by making changes to the model that move it along a gradient or slope of errors down toward a minimum error value. This gives the algorithm its name of "gradient descent".
- The three main flavors of gradient descent are
  - → batch,
  - → stochastic, and
  - → mini-batch

Batch Gradient Descent

- Batch gradient descent, computes the gradient of the cost function w.r.t. the parameters w for the entire training dataset:

$$w = w - \alpha \frac{df(w, x, y)}{dw}$$

- As we need to calculate the gradients for the whole dataset perform just one update, batch gradient descent is computationally fast. However, it is intractable for datasets that don't fit in memory.

- Batch gradient descent also doesn't allow us to update our model online, i.e. with new examples on-the-fly.

- Pseudocode of batch gradient descent looks like below:

```
for i in range (#epochs):
    grad = evaluategradient(data, para)
    para = para - learning_rate * grad
```

- Batch gradient descent is guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces.

Stochastic Gradient Descent (SGD)

- Stochastic gradient descent (SGD) in contrast performs a parameter update for each training example $x^{(i)}$ and label $y^{(i)}$. Therefore, learning happens on every example.

$$w = w - \alpha \frac{df(w, x^{(i)}, y^{(i)})}{dw}$$

- The term stochastic indicates that the one example comprising each batch is chosen at random.
- Stochastic gradient descent allow us to update our model online.
- Pseudocode for Stochastic Gradient Descent looks like below:

```
for i in range(#epochs):
    np.random.shuffle(data)
    for d in data:
        grad = compute_gradient(d, params)
        params = params - learning_rate * grad
```

- SGD updates the model much frequently, which is more computationally expensive than other configurations of gradient descent.
- Thus, it takes significantly longer to train models on large datasets. At the same time we lose speedup due to vectorization.
- These frequent updates can result in a noisy gradient signal, which may cause the model parameters jump around (have a higher variance over training epochs).
- At the same time this behavior helps to jump to another min
- This ultimately complicates convergence to the exact minimum, as SGD will keep overshooting.

Mini-Batch Gradient Descent

- Mini-batch gradient descent is a variation of the gradient descent algorithm that splits the training dataset into small batches that are used to calculate model error and update model coefficients.

$$w = w - \alpha \frac{df(w, x^{(i:i+n)}, y^{(i:i+n)})}{dw}$$

- Implementations may choose to sum the gradient over the mini-batch or take the average of the gradient which further reduces the variance of the gradient.

- Pseudocode of Mini-batch gradient descent looks like below:

```
for i in range (#epochs):
    np.random.shuffle(data)
    for batch in data:
        grad = compute_gradient(batch, params)
        params = params - learning_rate * grad
```

- Mini-batch gradient descent takes the best of both Batch Gradient and SGD.
  - Reduces the variance of the parameter updates which can lead to more stable convergence
  - Performs less frequent parameter updates and hence is not much time consuming.

- Mini-batch gradient descent is the recommended variant of gradient descent for most applications.

Comments on Batch-Size

- Mini-batch requires the configuration of an additional batch size Hyperparameter for the learning algorithm.
- Small values results in faster learning process at the cost of noise in the training process.
- Large values results in slow learning process with accurate estimates of the error gradient.

02/29

Linear Regression

- Regression analysis is the process of curve fitting in which the relationship between the independent variables and dependent variables are modeled in the m^{th} degree polynomial.
- Polynomial Regression models are usually fit with the method of least mean square (LMS) rule or Widrow-Hoff rule.
- If we assume that the relationship is a linear one and only one variable, then we can use linear equation given as:

$$y = f(x) = w_0 + w_1 x$$

- Let us suppose that training set contains n data points. Error function or cost function for the n data points is given by:

$$E = \frac{1}{2n} \sum_{i=1}^{n} e_i^2$$

$$E = \frac{1}{2n} \sum_{i=1}^{n} (y^i - w_0 - w_1 x^i)^2$$

Predicted value $= w_0 + w_1 x^i$, actual (given value) $= y^i$

$\therefore$ Actual - predicted i.e. $(y^i - w_0 - w_1 x^i)$

- Now, coefficients can be determined or updated using gradient decent method as below.

$$w_0 = w_0 - \alpha \frac{\partial E}{\partial w_0} = w_0 + \alpha \frac{1}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i)$$

$$w_1 = w_1 - \alpha \frac{\partial E}{\partial w_1} = w_1 + \alpha \frac{1}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i) x_i$$

Example:

Fit a straight line through the following data using SGD. Show one epoch of training.

x	1	2	3	4
f(x)	3	5	7	9

soln:

General form of linear regression equation is :

$$y = w_0 + w_1 x$$

Let us assume that initial values of parameters are:

$$w_0 = w_1 = 0$$

Iteration 1: $x = 1$, $y = f(x) = 3$, $\alpha = 0.01$

$w_0 = w_0 + \alpha (y - w_0 - w_1 x) = 0 + 0.01 \times 3 = 0.03$

$w_1 = w_1 + \alpha (y - w_0 - w_1 x) \cdot x = 0 + 0.01 \times 3 = 0.03$

Iteration 2: $x = 2$, $y = f(x) = 5$, $\alpha = 0.01$

$w_0 = w_0 + \alpha (y - w_0 - w_1 x)$

$= 0.03 + 0.01 (5 - 0.03 - 0.03 \times 2)$

$= 0.03 + 0.01 \times 4.91$

$= 0.0791$

$w_1 = 0.03 + 0.01 \times 4.91 \times 2$ $\{ \because w_1 + \alpha (y - w_0 - w_1 x) \cdot x \}$

$= 0.1282$

Iteration 3: $x=3$, $y=f(x)=7$, $\alpha=0.01$

$$w_0 = 0.0791 + 0.01(7 - 0.0791 - 0.1282 \times 3)$$
$$= 0.0791 + 0.01 \times 6.5363$$
$$= 0.1445$$

$$w_1 = w_1 + \alpha (y - w_0 - w_1 x) \cdot x$$
$$= 0.1282 + 0.01 \times 6.5363 \times 3$$
$$= 0.3243$$

Iteration 4: $x=4$, $y=f(x)=9$, $\alpha=0.01$

$$w_0 = w_0 + \alpha (y - w_0 - w_1 x) \cdot x$$
$$= 0.1445 + 0.01 (9 - 0.1445 - 0.3243 \times 4)$$
$$= 0.2200$$

$$w_1 = w_1 + \alpha (y - w_0 - w_1 x) \cdot x$$
$$= 0.3243 + 0.01(9 - 0.1445 - 0.3243 \times 4) \times 4$$
$$= 0.6266$$

Derive Weight Update rule for Linear Regression for t
variables.

Locally Weighted Regression

- Linear regression is a supervised learning algorithm used for computing linear relationships between input (x) and output (y).
- This algorithm cannot be used for making predictions when there exists a non-linear relationship between x and y. In such cases, locally weighted linear regression is used.
- Locally weighted linear regression is a non-parametric algorithm, that is, the model does not learn a fixed set of parameters as is done in ordinary linear regression.
- Rather the parameters are computed individually for each query point x. While computing parameters a higher preference is given to the points in the training set lying in the vicinity of x than the points lying far away from x.
- This results in the model fitting a straight line only to the data which is near or close to the query point.
- The cost function for the locally weighted linear regression algorithm is given as below:

$$E = \frac{1}{2n} \sum_{i=1}^{n} w_i (y_i - a_0 - a_1 x_i)^2$$

- Here, w_i is a non-negative weight associated with training point x^i.
- For x^i lying closer to the query point the value of w_i is large, while for x^i lying away to the query point value of w_i is small. A typical choice w_i of is:

$$w_i = \exp\left(\frac{-(x^i - x)^2}{2\tau^2}\right)$$

- where, τ called the bandwidth parameter and controls the rate at which w_i falls with distance from x.

- Clearly, if $|x^i - x|$ is small, w_i is close to 1 and if $|x^i - x|$ is large, w_i is close to 0. Thus, the training-set-points lying closer to the query point x contribute more to the cost function than the points lying far away from x.

- Once we have cost function, we can use gradient descent algorithm to train the LWR algorithm.

Example:
Consider a query point $x = 6$ and let $x^1 = 5$, $x^2 = 4$, and $x^3 = 3$ are three points in the training set. Find cost function for the Locally weighted linear regression.

Sol^n:

Given,

$x = 6,\ x^1 = 5,\ x^2 = 4,\ x^3 = 3$

Use $\tau = 1$.

We know,

$$w_i = \exp\left(\frac{-(x^i - x)^2}{2\tau^2}\right)$$

So,

$$w_1 = e^{\left(\frac{-(x^1 - x)^2}{2\tau^2}\right)} = e^{\left(\frac{-(5-6)^2}{2\times 1^2}\right)} = e^{\left(\frac{-1}{2}\right)} = 0.60$$

$$w_2 = e^{\left(\frac{-(x^2 - x)^2}{2\tau^2}\right)} = e^{\left(\frac{-(4-6)^2}{2\times 1^2}\right)} = e^{\left(\frac{-4}{2}\right)} = 0.135$$

$$w_3 = e^{\left(\frac{-(x^3 - x)^2}{2\tau^2}\right)} = e^{\left(\frac{-(3-6)^2}{2\times 1^2}\right)} = e^{\left(\frac{-9}{2}\right)} = 0.011$$

$$E = \frac{1}{2n} \sum_{i=1}^{n} w_i (y_i - a_0 - a_1 x_i)^2$$

Thus, cost function for LWR is:

$$E = \frac{1}{2\times 3} \left\{ \begin{array}{l} 0.606 (y^1 - a_0 - a_1 x^1)^2 + \\ 0.135 (y^2 - a_0 - a_1 x^2)^2 + \\ 0.011 (y^3 - a_0 - a_1 x^3)^2 \end{array} \right\}$$

#

- Coefficient update rule for the locally weighted linear regression is given below.

$$a_0 = a_0 - \alpha \frac{\partial E}{\partial a_0} = a_0 + \alpha \frac{1}{n} \sum_{i=1}^{n} w_i (y_i - a_0 - a_1 x_i)$$

$$a_1 = a_1 - \alpha \frac{\partial E}{\partial a_1} = a_1 + \alpha \frac{1}{n} \sum_{i=1}^{n} w_i (y_i - a_0 - a_1 x_i) x_i$$

Example:
Consider following dataset and show one epoch of training using LWR.

x	3	4	5	6
y	10	15	24	?

Soln:

Assume $a_0 = a_1 = 0$

Use $\tau = 1$, $\alpha = 0.01$

Here,

$x = 6$

$w_1 =$

$w_2 =$

$w_3 =$

SGD

Iteration 1:

$a_0 = a_0 + \alpha (y^1 - a_0 - a_1 x^1) w_1$

=

=

$a_1 = a_1 + \alpha (y^1 - a_0 - a_1 x^1) w_1 x^1$

=

=

Iteration 2:

$a_0 = a_0 + \alpha (y^2 - a_0 - a_1 x^2) w_2$

=

=

$a_1 = a_1 + \alpha (y^2 - a_0 - a_1 x^2) w_2 x^2$

=

=

Iteration 3:

$a_0 = a_0 + \alpha (y^3 - a_0 - a_1 x^3) w_3$

classmate
Date 03/01
Page

Logistic Regression

- Logistic regression is one of the most popular machine learning algorithms for binary classification. This is because it is a simple algorithm that performs very well on a wide range of problems.

- We want to predict a variable $\hat{y} \in \{0, 1\}$, where 0 is called negative class, while 1 is called positive class. Such task is known as binary classification.

- The heart of the logistic regression technique is logistic function and is defined as:

$$f(x) = \frac{1}{1+e^{-x}}$$

- Logistic function transforms the input into the range [0,1]. Smallest negative number results in values close to zero and the larger positive numbers results in values close to on

- If there are two input variable, logistic regression has two coefficients just like linear regression.

$$y = w_0 + w_1 x_1 + w_2 x_2$$

- Unlike linear regression, the output is transformed into a probability using the logistic function:

$$\hat{y} = \sigma(y) = \frac{1}{1+e^{-y}}$$

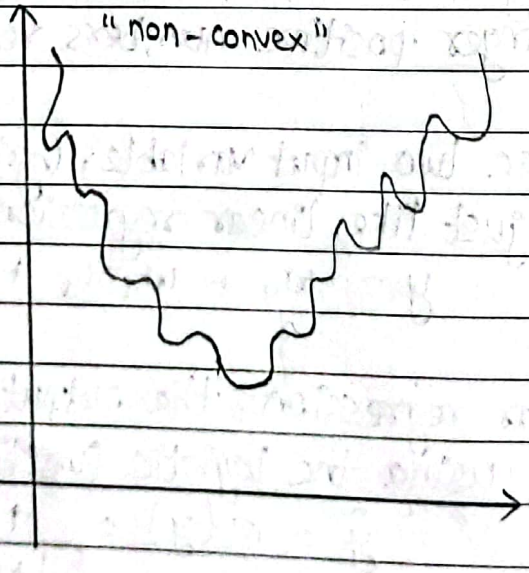
- If the probability is > 0.5 we can take the output as a prediction for the class 1, otherwise the prediction is for the class 0.

- The job of the learning algorithm will be to discover the best values for the coefficients (w_0, w_1, and w_2) based on the training data.

Logistic Regression Cost Function

- Unfortunately, we can't use the cost function MSE (Mean Squarred Error) in logistic regression. It is because our prediction function is non-linear (due to sigmoid transform).

- Squarring this prediction as we do in MSE results in a non-convex function with many local minimums. If our cost function has many local minimums, gradient descent may not find the optimal global minimum.



- Given the training set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots\ldots, (x^{(m)}, y^{(m)})\}$

  we want $\hat{y}^{(i)} \approx y^{(i)}$

- For logistic regression, we use following loss function or error function

$$L(\hat{y}, y) = \begin{cases} -\log(\hat{y}) & \text{if } y = 1 \\ -\log(1-\hat{y}) & \text{if } y = 0 \end{cases}$$

- In case $y = 1$, the output (the cost to pay) approaches to 0 as $\hat{y}$ approaches to 1. Conversely, the cost to pay grows to infinity as $\hat{y}$ approaches to 0.

- This is a desirable property: we want a bigger penalty as the algorithm predicts something far away from the actual value. The same intuition applies when $y = 0$.

- Thus, cost function for Logistic regression is given as:

$$L(y, \hat{y}) = -y \log \hat{y} - (1-y) \log(1-\hat{y})$$

Logistic Regression GD / derivatives

$$\frac{\partial L(y,\hat{y})}{\partial \hat{y}} = \frac{-y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$

$$\frac{\partial L(y,\hat{y})}{\partial y} = \frac{\partial L}{d\hat{y}} \frac{\partial \hat{y}}{\partial y} = \left\{ \frac{-y}{\hat{y}} + \frac{1-y}{1-\hat{y}} \right\} \{\hat{y}(1-\hat{y})\} = \hat{y} - y$$

∵ if $g(x)$ is logistic function $g'(x) = g(x)(1-g(x))$

$$\frac{\partial L}{\partial w_0} = \frac{\partial L}{d\hat{y}} \frac{\partial \hat{y}}{\partial y} \frac{\partial y}{\partial w_0} = (\hat{y} - y)$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{d\hat{y}} \frac{\partial \hat{y}}{\partial y} \frac{\partial y}{\partial w_1} = x_1(\hat{y} - y)$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{d\hat{y}} \frac{\partial \hat{y}}{\partial y} \frac{\partial y}{\partial w_2} = x_2(\hat{y} - y)$$

Thus, parameters of Logistic Regression are updated as below:

$$w_0 = w_0 - \alpha \frac{\partial L}{\partial w_0} = w_0 - \alpha(\hat{y} - y)$$

$$w_1 = w_1 - \alpha \frac{\partial L}{\partial w_1} = w_1 - \alpha(\hat{y} - y)\, x_1$$

$$w_2 = w_2 - \alpha \frac{\partial L}{\partial w_2} = w_2 - \alpha(\hat{y} - y)\, x_2$$

Example:

Fit the logistic regression model through the following data. Show one epoch of training.

x_1	x_2	Class(y)
0.78	0.69	1
0.67	1.00	1
0.00	0.00	0
0.22	0.14	0

Soln:

General form of logistic regression equation is:

$$(z)\quad y = w_0 + w_1 x_1 + w_2 x_2$$

Let us assume that initial values of parameters are:

$$w_0 = w_1 = w_2 = 0$$

Iteration 1:

$$x_1 = 0.78, \quad x_2 = 0.69, \quad y = 1, \quad \alpha = 0.1$$

$$y = w_0 + w_1 x_1 + w_2 x_2 = 0$$

$$\hat{y} = \frac{1}{1+e^{-y(z)}} = \frac{1}{1+e^{-0}} = \frac{1}{1+1} = \frac{1}{2} = 0.5$$

$$w_0 = w_0 - \alpha(\hat{y}-y) = 0 - 0.1 \times (0.5-1) = 0.05$$

$$w_1 = w_1 - \alpha(\hat{y}-y)x_1 = 0 - 0.1(0.5-1) \times 0.78 = 0.039$$

$$w_2 = w_2 - \alpha(\hat{y}-y)x_2 = 0 - 0.1(0.5-1) \times 0.69 = 0.0345$$

Iteration 2:

$x_1 = 0.67, \quad x_2 = 1.00, \quad y = 1, \quad \alpha = 0.1$

$$y = w_0 + w_1 x_1 + w_2 x_2$$
$$= 0.05 + 0.039 \times 0.67 + 0.0345 \times 1$$
$$= 0.05 + 0.026 + 0.0345$$
$$= 0.1105$$

$$\hat{y} = \frac{1}{1+e^{-y}} = \frac{1}{1+e^{-0.1105}} = 0.5276$$

$$w_0 = w_0 - \alpha(\hat{y} - y) = 0.05 - 0.1(0.5276 - 1) = 0.097$$
$$w_1 = w_1 - \alpha(\hat{y} - y)x_1 = 0.039 - 0.1(0.5276 - 1)0.67 = 0.0706$$
$$w_2 = w_2 - \alpha(\hat{y} - y)x_2 = 0.0345 - 0.1(0.5276 - 1)1 = 0.0817$$

Iteration 3:

$x_1 = 0.00, \quad x_2 = 0.00, \quad y = 0, \quad \alpha = 0.1$

$$y = w_0 + w_1 x_1 + w_2 x_2$$
$$= 0.097 + 0.0706 \times 0 + 0.0817 \times 0$$
$$= 0.097$$

$$\hat{y} = \frac{1}{1+e^{-y}} = \frac{1}{1+e^{-0.097}} = 0.524$$

$$w_0 = w_0 - \alpha(\hat{y} - y) = 0.097 - 0.1(0.524 - 0) = 0.0446$$
$$w_1 = w_1 - \alpha(\hat{y} - y)x_1 = 0.0706 - 0.1(0.524 - 0)0 = 0.0706$$
$$w_2 = w_2 - \alpha(\hat{y} - y)x_2 = 0.0817 - 0.1(0.524 - 0)0 = 0.0817$$

Iteration 4:

$x_1 = 0.22$, $x_2 = 0.14$, $y = 0$, $\alpha = 0.1$

$$y = w_0 + w_1 x_1 + w_2 x_2$$
$$= 0.0446 + 0.0706 \times 0.22 + 0.0817 \times 0.14$$
$$= 0.0446 + 0.0155 + 0.0114$$
$$= 0.0715$$

$$\hat{y} = \frac{1}{1+e^{-y}} = \frac{1}{1+e^{-0.0715}} = 0.5178$$

$w_0 = w_0 - \alpha(\hat{y} - y) = 0.0446 - 0.1(0.5178 - 0) = -0.0072$

$w_1 = w_1 - \alpha(\hat{y} - y)x_1 = 0.0706 - 0.1(0.5178 - 0) \times 0.22 = 0.0592$

$w_2 = w_2 - \alpha(\hat{y} - y)x_2 = 0.0817 - 0.1(0.5178 - 0) \times 0.14 = 0.0744$

Performance Evaluation of Classification

Before selecting a classification algorithm to solve a problem, we need to evaluate its performance. Confusion matrix is widely used for computing performance measures of classification algorithm.

- Confusion Matrix

- A confusion matrix is an N×N matrix used for evaluating the performance of a classification model, where N is the number of target classes.

- The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making.

- For a binary classification problem, we would have a 2×2 matrix as shown below with 4 values:

		Actual Classes	
		Positive	Negative
Predicted Classes	Positive	TP	FP
	Negative	FN	TN

- True Positive (TP)
  - It represents correctly classified positive classes.
  - Both predicted and actual class are positive here.

False Positive (FP)
- It represents incorrectly classified positive classes.
- Predicted class is positive but actual class is negative.
- Type I error.

False Negative (FN)

- It represents incorrectly classified negative classes.
- Predicted class is negative but actual class is positive.
- Type II error.

True Negative (TN)

- It represents correctly classified negative classes.
- Both predicted and actual class are negative.

Performance Measures of Classification

Four widely used performance measures used for evaluating classification models are: Accuracy, Recall, Precision, F1-score.

- **Accuracy:**
  - It is the percentage of correct predictions made by the model and is given as below:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$= \frac{TP + TN}{\text{Total number of instances}}$$

- **Precision:**
  - It is the percentage of predicted positives that are actually positive and is given as below:

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{TP}{\text{Total Number of Predicted Po}}$$

70% precision means :- Out of 100 predicted positives, only 7 are correct, remaining 30 are not.

10%. recall = out of 100 positive, only 10 are positive

Recall (Sensitivity):

- It is the percentage of actual positives that are correctly classified by the model and is given as below:

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{TP}{\text{Total number of actual positives}}$$

F1-Score:

- It is the harmonic mean of recall and precision. It becomes high only when both precision and recall are high. This score is given by:

$$F1 = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}}$$

specificity

- Rate of true negative.

$$\text{Specificity} = \frac{TN}{TN + FP}$$

Example:

page-50
program

Overfitting and Underfitting

Overfitting

- Overfitting is the result of using an excessively complicated model.
- It happens when the model learns the details and noise of the training data to the extent that it negatively impacts the performance of the model on new data.
- This means that the model learns noise or random fluctuations in the training data as concepts.
- This negatively impacts the model's ability to generalize because these concepts do not apply to new data.

- complex model
- Too much training
- Learns noise and random fluctuations in data as pattern
- Not able to generalize in new data.
  - Training accuracy - high
  - Test accuracy - low

Underfitting

- Underfitting is the result of using an excessively simple model or using very few training samples.
- In such situations, a machine learning algorithm can not capture the underlying trend of the data.
- Thus, it refers to a model that can neither model the training data nor generalize to new data.
- Obviously an underfitted machine learning model is not a suitable model for making predictions because it has poor performance on the training data too.

- simple model, - less training data
- Not able to capture pattern in the training data.
  - Training accuracy - low
  - Test accuracy - low

Matrix Derivatives

- Let A be a $m \times n$ matrix and $f: R^{m \times n} \to R$ be a mapping function from m-by-n matrices to a real number.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

- Now, gradient of f with respect to A is also a $m \times n$ matrix and is given as below.

$$\nabla_A f_A = \begin{bmatrix} \frac{\partial f}{\partial a_{11}} & \frac{\partial f}{\partial a_{12}} & \cdots & \frac{\partial f}{\partial a_{1n}} \\ \frac{\partial f}{\partial a_{21}} & \frac{\partial f}{\partial a_{22}} & \cdots & \frac{\partial f}{\partial a_{2n}} \\ \vdots & & & \vdots \\ \frac{\partial f}{\partial a_{m1}} & \frac{\partial f}{\partial a_{m2}} & \cdots & \frac{\partial f}{\partial a_{mn}} \end{bmatrix}$$

gradient of A w.r.t. f

Example:

Let $A = \begin{bmatrix} w & x \\ y & z \end{bmatrix}$ and $f(w, x, y, z) = 2w + 3x + yz$.

Find $\nabla_A f_A$.

Sol:

$$\begin{bmatrix} \frac{\partial f}{\partial w} & \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} & \frac{\partial f}{\partial z} \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ z & y \end{bmatrix}$$

#

Trace of Matrix

- For a $n \times n$ matrix A_1, trace of A is defined as sum of diagonal elements.

$$\text{i.e. } \text{tr } A = \sum_{i=1}^{n} a_{ii}$$

- For a real number a $\text{tr}.a = a$.
- If multiplication of A and B is square matrix then

$$\text{tr } AB = \text{tr } BA$$

Verify that $\text{tr}.AB = \text{tr}.BA$

Let A and B are matrices of order 2×3 and 3×2 respectively.

i.e.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}$$

Now,

$$AB = \begin{bmatrix} a_{11}\cdot b_{11} + a_{12}\cdot b_{21} + a_{13}\cdot b_{31} & a_{11}\cdot b_{12} + a_{12}\cdot b_{22} + a_{13}\cdot b_{32} \\ a_{21}\cdot b_{11} + a_{22}\cdot b_{21} + a_{23}\cdot b_{31} & a_{21}\cdot b_{12} + a_{22}\cdot b_{22} + a_{23}\cdot b_{32} \end{bmatrix}$$

$$\text{tr}.AB = a_{11}\cdot b_{11} + a_{12}\cdot b_{21} + a_{13}\cdot b_{31} + a_{21}\cdot b_{12} + a_{22}\cdot b_{22} + a_{23}\cdot b_{32}$$

Similarly

$$BA = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

$$= \begin{bmatrix} b_{11}\cdot a_{11} + b_{12}\cdot a_{21} & b_{11}\cdot a_{12} + b_{12}\cdot a_{22} & b_{11}\cdot a_{13} + b_{12}\cdot a_{23} \\ b_{21}\cdot a_{11} + b_{22}\cdot a_{21} & b_{21}\cdot a_{12} + b_{22}\cdot a_{22} & b_{21}\cdot a_{13} + b_{22}\cdot a_{23} \\ b_{31}\cdot a_{11} + b_{32}\cdot a_{21} & b_{31}\cdot a_{12} + b_{32}\cdot a_{22} & b_{31}\cdot a_{13} + b_{32}\cdot a_{23} \end{bmatrix}$$

$$\text{tr}.BA = \begin{bmatrix} b_{11}\cdot a_{11} + b_{12}\cdot a_{21} + b_{21}\cdot a_{12} + b_{22}\cdot a_{22} + b_{31}\cdot a_{13} + b_{32}\cdot a_{23} \end{bmatrix}$$

$$= [a_{11}b_{11} + a_{12}\cdot b_{21} + a_{13}\cdot b_{31} + a_{21}\cdot b_{12} + a_{22}\cdot b_{22} + a_{23}\cdot b_{32}]$$

Thus, $\text{tr}.AB = \text{tr}.BA$

Corollories of tr. AB = tr. BA

tr. ABC = tr. CAB = tr. BCA

tr. ABCD = tr. DABC = tr. BCDA

tr. ABC =

Prove that : tr. CAB = tr. BCA

tr. ABC = tr. (AB)C

= tr. CAB

Similarly,

tr. ABC = tr. A(BC)

= tr BCA

Thus,

tr. ABC = tr. CAB = tr. BCA

proved :.

Prove that : tr. ABCD = tr. DABC = tr. BCDA

⇒

If A and B are square matrices and a is a real number then trace operator exhibits following properties:

tr. A = tr. A^T

tr. ABCD = tr. (ABC) D

= tr. DABC

Similarly,

tr. ABCD = tr. A (BCD)

= tr. BCDA

Thus,

tr. ABCD = tr. DABC = tr. BCDA

proved ..

If A and B are square matrices and a is a real number then trace operator exhibits following properties:

- $tr.A = tr.A^T$
- $tr.(A+B) = tr.A + tr.B$
- $tr.\ aA = a\ tr.A$

- $tr.A = tr.A^T$

Proof:

Let $A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$

$$tr.A = \sum_{i=1}^{n} a_{ii}$$

Again,

$$A^T = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{n1} \\ a_{12} & a_{22} & \cdots & a_{n2} \\ \vdots & & & \\ a_{1n} & a_{2n} & \cdots & a_{nn} \end{bmatrix}$$

$$tr.A^T = \sum_{i=1}^{n} a_{ii}$$

Thus, $tr.A = tr.A^T$ $\left(\text{i.e. } \sum_{i=1}^{n} a_{ii}\right)$

• $tr(A+B) = tr.A + tr.B$.

Let $A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$, $B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & & & \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$

Now,

$$A+B = \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \cdots & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{22} & \cdots & a_{2n}+b_{2n} \\ \vdots & & & \\ a_{n1}+b_{n1} & a_{n2}+b_{n2} & \cdots & a_{nn}+b_{nn} \end{bmatrix}$$

$$tr.(A+B) = a_{11}+b_{11}+a_{22}+b_{22}+\cdots+a_{nn}+b_{nn}$$

$$tr.(A+B) = \sum_{i=1}^{n} a_{ii}+b_{ii}$$

Then,

$$tr.A = a_{11}+a_{22}+\cdots+a_{nn} \quad \left(\sum_{i=1}^{n} a_{ii}\right)$$

$$tr.B = b_{11}+b_{22}+\cdots+b_{nn} \quad \left(\sum_{i=1}^{n} b_{ii}\right)$$

$$tr.A + tr.B = (a_{11}+a_{22}+\cdots+a_{nn}) + (b_{11}+b_{22}+\cdots+b_{nn})$$
$$= a_{11}+b_{11}+a_{22}+b_{22}+\cdots+a_{nn}+b_{nn}$$

$$tr.A + tr.B = \sum_{i=1}^{n} a_{ii} + \sum_{i=1}^{n} b_{ii}$$

$$= \sum_{i=1}^{n} a_{ii}+b_{ii}$$

Thus,

$$tr.(A+B) = tr.A + tr.B = \left(\text{i.e. } \sum_{i=1}^{n} a_{ii}+b_{ii}\right)$$

- $tr.\ aA = a\ tr.\ A$

- Let,

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

$$aA = \begin{bmatrix} a.a_{11} & a.a_{12} \\ a.a_{21} & a.a_{22} \end{bmatrix}$$

$$\boxed{tr.\ aA = a.a_{11} + a.a_{22}}$$

Now,

$$tr.\ A = a_{11} + a_{22}$$

$$\boxed{a.\ tr.A = a.a_{11} + a.a_{22}}$$

Thus,

$$\boxed{tr.\ aA = a\ tr.A}\ \#$$ [i.e. $a.a_{11} + a.a_{22}$]

Some facts about matrix derives

- $\nabla_A\ tr.AB = B^T$
- $\nabla_{A^T}\ f(A) = (\nabla_A \cdot f(A))^T$
- $\nabla_A\ tr.ABA^TC = CAB + C^TAB^T$
- $\nabla_A\ |A| = |A|(A^{-1})^T$

Prove that $\nabla_A \text{tr.}\, AB = B^T$

Sol:

Let order of A is 3×2 and order of B is 2×3.

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}, \quad B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix}$$

$$AB = \begin{bmatrix} a_{11}\cdot b_{11} + a_{12}\cdot b_{21} & a_{11}\cdot b_{12} + a_{12}\cdot b_{22} & a_{11}\cdot b_{13} + a_{12}\cdot b_{23} \\ a_{21}\cdot b_{11} + a_{22}\cdot b_{21} & a_{21}\cdot b_{12} + a_{22}\cdot b_{22} & a_{21}\cdot b_{13} + a_{22}\cdot b_{23} \\ a_{31}\cdot b_{11} + a_{32}\cdot b_{21} & a_{31}\cdot b_{12} + a_{32}\cdot b_{22} & a_{31}\cdot b_{13} + a_{32}\cdot b_{23} \end{bmatrix}$$

$$\text{tr.}\, AB = a_{11}\cdot b_{11} + a_{12}\cdot b_{21} + a_{21}\cdot b_{12} + a_{22}\cdot b_{22} + a_{31}\cdot b_{13} + a_{32}\cdot b_{23}$$

And,

$$\nabla_A \text{tr.}\, AB = \begin{bmatrix} b_{11} & b_{21} \\ b_{12} & b_{22} \\ b_{13} & b_{23} \end{bmatrix}$$

Now,

$$B^T = \begin{bmatrix} b_{11} & b_{21} \\ b_{12} & b_{22} \\ b_{13} & b_{23} \end{bmatrix}$$

Thus,

$$\nabla_A \text{tr.}\, AB = B^T$$

i.e. $\begin{bmatrix} b_{11} & b_{21} \\ b_{12} & b_{22} \\ b_{13} & b_{23} \end{bmatrix}$

#

Date 03/08
Page

Deriving Normal Equations for Linear Regression

Given a training set, Let X be a $m \times n$ matrix that contains input values of training examples in its rows and let $\vec{y}$ be a m dimensional vector containing target values.

$$\begin{bmatrix} — (x^1)^T — \\ — (x^2)^T — \\ \vdots \\ — (x^m)^T — \end{bmatrix} \qquad \vec{y} = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^m \end{bmatrix}$$

We know that,

$$y = f(x) = (x^i)^T w;$$

where, w is coefficient vector.

Now,

$$\text{error} = f(x) - \vec{y}$$
$$= xw - \vec{y}$$

$$= \begin{bmatrix} (x^1)^T w \\ (x^2)^T w \\ \vdots \\ (x^m)^T w \end{bmatrix} - \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^m \end{bmatrix}$$

Now,

Loss function can be written as:

$$L = \frac{1}{2} \sum_{i=1}^{m} \left(f(x^i) - y^i \right)^2$$

$$= \frac{1}{2} (xw - \vec{y})^T (xw - \vec{y})$$

To minimize loss, we need to calculate derivative / gradient of L w.r.t w.

$$\nabla_w L = \frac{1}{2} \nabla_w (Xw - \vec{y})^T (Xw - \vec{y})$$

$$= \frac{1}{2} \nabla_w (w^T X^T - \vec{y}^T)(Xw - \vec{y})$$

$$= \frac{1}{2} \nabla_w (w^T X^T Xw - w^T X^T \vec{y} - \vec{y}^T Xw + \vec{y}^T \vec{y})$$

$$= \frac{1}{2} \nabla_w \, tr\,(w^T X^T Xw - w^T X^T \vec{y} - \vec{y}^T Xw + \vec{y}^T \vec{y})$$

$\nabla_w \vec{y}^T \vec{y} = 0$

$tr\, A = tr\, A^T$

$\nabla_A tr\, AB = B^T$

$\nabla_A tr\, ABA^T C = B^T A^T C^T + BA^T C$

$tr(w^T X^T \vec{y}) = tr(w^T X^T \vec{y})^T$

$= tr\, \vec{y}^T Xw$

$$= \frac{1}{2} \nabla_w (tr\, w^T X^T Xw - 2\, tr\, \vec{y}^T Xw)$$

$$= \frac{1}{2} (X^T Xw + X^T Xw - 2X^T \vec{y})$$

$$= X^T Xw - X^T \vec{y}$$

At minima, gradient is always zero.

$\therefore X^T Xw - X^T \vec{y} = 0$

$\Rightarrow w = \frac{X^T \vec{y}}{X^T X} = (X^T X)^{-1} X^T \vec{y}$

This is called normal equation for linear regression.

Example:

Consider the following training data.

x_1	1	2	1	3
x_2	1	1	2	3
y	3	5	2	5

Use normal equation to find coefficient of linear regression line fitted through above data.

Sol:

$$X = \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 1 & 2 \\ 3 & 3 \end{bmatrix}$$

Augment X for w_0.

$$X = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \\ 1 & 3 & 3 \end{bmatrix}$$

$X = 4 \times 3$

$X^T = 3 \times 4$

$(X^T X)^{-1} = 3 \times 3$

$\underbrace{(X^T X)^{-1}}_{3\times3} \underbrace{X^T}_{3\times4} = 3 \times 4$

$\underbrace{(X^T X)^{-1}}_{3\times4} \underbrace{X^T \vec{y}}_{4\times1} = 3 \times 1$

$$\vec{y} = \begin{bmatrix} 3 \\ 5 \\ 2 \\ 5 \end{bmatrix}$$

$$W = (X^T X)^{-1} X^T \vec{y}$$

$$= \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 3 \\ 1 & 1 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \\ 1 & 3 & 3 \end{bmatrix} \right)^{-1} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 3 \\ 1 & 1 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 5 \\ 2 \\ 5 \end{bmatrix}$$

$$= \begin{bmatrix} 1+1+1+1 & 1+2+1+3 & 1+1+2+3 \\ 1+2+1+3 & 1+4+1+9 & 1+2+2+9 \\ 1+1+2+3 & 1+2+2+9 & 1+1+4+9 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 3 \\ 1 & 1 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 5 \\ 2 \\ 5 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 7 & 7 \\ 7 & 15 & 14 \\ 7 & 14 & 15 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 3 \\ 1 & 1 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 5 \\ 2 \\ 5 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 7 & 7 \\ 7 & 15 & 14 \\ 7 & 14 & 15 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 3+5+2+5 \\ 3+10+2+15 \\ 3+5+4+15 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 7 & 7 \\ 7 & 15 & 14 \\ 7 & 14 & 15 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 15 \\ 30 \\ 27 \end{bmatrix}$$

Parametric Vs. Non-parametric Algorithms

Parametric Algorithms | Models

- General form of learning model is assumed.
- Fixed number of parameters, we just need to adjust values of parameters.
- Fast compared to non-parametric.
- can show better performance with few training data.
- Example: linear regression, locally weighted logistic regression.

General form of linear regression is:

$$y = w_0 + w_1 x^{(1)} + w_2 x^{(2)}$$

Non-parametric

- General form of mapping function is not assumed. Mapping function can take any form.
- Size of parameters may not be fixed.
- Slow compared to parametric algorithms.
- Need large volume of training data to show better performance.

Example: K-nearest neighbor (KNN) algorithm

Probabilistic Implementation of Least square Regression

In case of least square regression, input and output variables are related via following equation.

$$y = w^T x^{(i)} + \epsilon^{(i)}$$

where,

$\epsilon^{(i)}$ is error term that represents unmodeled effect and random noise.

Assume that $\epsilon^{(i)}$ follows IID (Independently and Identically Distributed). Thus, for zero mean σ^2 standard deviation, probability density function for gaussian distribution can be written as below:

$$P(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(\frac{-(\epsilon^{(i)})^2}{2\sigma^2}\right)$$

$$\Rightarrow P(y^i / x^i, w) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(\frac{-(\epsilon^{(i)})^2}{2\sigma^2}\right)$$

If we view above function as function of w, it is called likelihood function and is given as below:

$$L(w) = P(\vec{y} / x, w)$$

$$= \prod_{i=1}^{m} P(y^i | x^i, w)$$

$$= \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(\frac{-(\epsilon^{(i)})^2}{2\sigma^2}\right)$$

$$= \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(\frac{-(y^{(i)} - w^T \cdot x^{(i)})^2}{2\sigma^2}\right)$$

Taking log on both sides;

$$\log(L(w)) = \log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - w^T \cdot x^{(i)})^2}{2\sigma^2}\right)$$

$$l(w) = \sum_{i=1}^{m} \log\left\{\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - w^T \cdot x^{(i)})^2}{2\sigma^2}\right)\right\}$$

[where, $l(w) = \log(L(w))$]

$$= \sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi}\sigma} + \sum_{i=1}^{m} \log\left(\exp\left(\frac{-(y^{(i)} - w^T \cdot x^{(i)})^2}{2\sigma^2}\right)\right)$$

$$= m \cdot \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{i=1}^{m} (y^{(i)} - w^T \cdot x^{(i)})^2$$

Thus,

$$l(w) = m \cdot \log \frac{1}{\sqrt{2\pi}\cdot\sigma} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - w^T \cdot x^{(i)})^2$$

From eqⁿ ①, we can say that likelihood is maximized when the term $\frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - w^T \cdot x^{(i)})^2$ is minimize

Hence, we can say that maximizing likelihood is equivalen to minimizing squarred sum of error or maximizing likelihood is equivalent to least square regression.

Perceptron Learning Rule

- Perceptron is single neuron used for binary classification.
- It commonly classify linearly separable patterns.
- In case of logistic regression, output is predicted as below:

$$\hat{y} = g(y) = \frac{1}{1+e^{-y}} \quad \text{(logistic activation function)}$$

where,

$$y = w_0 + w_1 x^{(1)} + w_2 x^{(2)} + \dots$$

- If we use following function 'g', above equation works for perceptron.

$$g(y) = \begin{cases} 1 & \text{if } y \geq 0 \\ 0 & y < 0 \end{cases}$$

Thus, weight update rule for logistic regression is same as weight update rule for perceptron, which is given as below:

$$w_0 = w_0 + \alpha \frac{\partial L}{\partial w_0}$$

$$w_1 = w_1 + \alpha \frac{\partial L}{\partial w_1} x^{(1)}$$

$$w_2 = w_2 + \alpha \frac{\partial L}{\partial w_2} x^{(2)}$$

where,

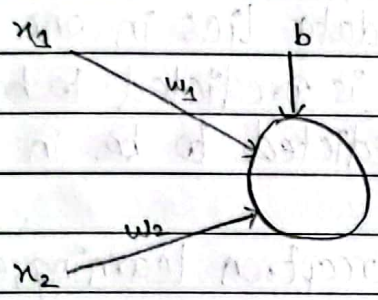
$$\frac{\partial L}{\partial w_0} = \hat{y} - y$$

$$\frac{\partial L}{\partial w_1} = (\hat{y} - y) x^{(1)}$$

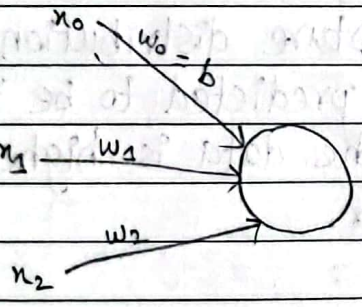
$$\frac{\partial L}{\partial w_2} = (\hat{y} - y) x^{(2)}$$

Now,
Generalized perceptron learning rule can be written as below:

$$w_i = w_i + \alpha(\hat{y} - y)\, x^{(i)}$$



$$v = w_1 x_1 + w_2 x_2 + b$$
$$y = f(v)$$



$$v = \underbrace{w_0 x_0}_{b} + w_1 x_1 + w_2 x_2$$

Date 03/15

Gaussian Discriminant Analysis (GDA)

There are two broad categories of supervised classification algorithms:

- Discriminative Learning Algorithms
- Generative Learning Algorithms

⇒ Discriminative learning algorithms finds decision between classes. For a new data, if the data lies in one side of the decision boundary then it is predicted to be in one class otherwise it is predicted to be in another class.

Example: Logistic regression, perceptron learning, etc.

⇒ Generative learning algorithms capture distribution of each class. For a new data, it is predicted to be in the class for which resemblance of the data is higher.

Example: Naive Baye's Classifier.

Naive Baye's Classifier

- also called Bayesian classifier
- It is based on Baye's theorem, which is given as below:

$$P(H|x) = \frac{P(x|H) \cdot P(H)}{P(x)}$$

likelihood

posterior probability

prior probabilities

- Let D be the dataset and $C_1, C_2, \ldots, C_m$ are m classes. For this, Baye's rule can be written as below:

$$P(C_i|X) = \frac{P(X|C_i) \cdot P(C_i)}{P(X)}$$

where,

X is a tuple containing n attributes

i.e. $X = \{x_1, x_2, \ldots, x_n\}$

- Let us assume the input attributes are independent of each other. Now, the probability $P(X|C_i)$ can be written below:

$$P(X|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \ldots \times P(x_n|C_i)$$

$$= \prod_{k=1}^{n} P(x_k|C_i)$$

$$\Rightarrow \boxed{P(C_i|x) = \frac{P(C_i) \times \prod_{k=1}^{n} P(x_k|C_i)}{P(X)}}$$

In the above equation, denominator is same for all classes. Therefore, it can be ignored.

Finally, while making prediction, X is predicted to be in class C_i for which $P(C_i|X)$ is maximum.

MA = Middle Aged

Example:

Age	Income	Student	Credit Rating	Buys Computer
Youth	High	No	Fair	No
Youth	High	No	Excellent	No
MA	High	No	Fair	Yes
Senior	Medium	No	Fair	Yes
Senior	Low	Yes	Fair	Yes
Senior	Low	Yes	Excellent	No
MA	Low	Yes	Excellent	Yes
Youth	Medium	No	Fair	No
Youth	Low	Yes	Fair	Yes
Senior	Medium	Yes	Fair	Yes
Youth	Medium	Yes	Excellent	Yes
MA	Medium	No	Excellent	Yes
MA	High	Yes	Fair	Yes
Senior	Medium	No	Excellent	No

Predict class label for X = {Age = Youth, Income = medium, Student = yes, Credit Rating = Fair}.

Sol?

$P(Buys = Yes) = 9/14$

$P(Buys = No) = 5/14$

For Age:

$P(Age = Youth \mid Buys = Yes) = 2/9$

$P(Age = Youth \mid Buys = No) = 3/5$

For Income:

$P(Income = Medium \mid Buys = Yes) = 4/9$

$P(Income = Medium \mid Buys = No) = 2/5$

For Student:

$P(\text{Student} = \text{Yes} \mid \text{Buys} = \text{Yes}) = 6/9$

$P(\text{Student} = \text{Yes} \mid \text{Buys} = \text{No}) = 1/5$

For Credit Rating:

$P(\text{Credit Rating} = \text{Fair} \mid \text{Buys} = \text{Yes}) = 6/9$

$P(\text{Credit Rating} = \text{Fair} \mid \text{Buys} = \text{No}) = 2/5$

Now,

$$P(\text{Buys} = \text{Yes} \mid X) = P(\text{Buys} = \text{Yes}) \times \prod_{k=1}^{n} P(x_k \mid \text{Buys} = \text{Yes})$$

$= P(\text{Buys} = \text{Yes}) \times P(\text{Age} = \text{Youth} \mid \text{Buys} = \text{Yes}) \times$
$P(\text{Income} = \text{Medium} \mid \text{Buys} = \text{Yes}) \times$
$P(\text{Student} = \text{Yes} \mid \text{Buys} = \text{Yes}) \times$
$P(\text{Credit Rating} = \text{Fair} \mid \text{Buys} = \text{Yes})$

$$= \frac{9}{14} \times \frac{2}{9} \times \frac{4}{9} \times \frac{6}{9} \times \frac{6}{9}$$

$$= 0.028$$

Similarly,

$$P(\text{Buys} = \text{No} \mid X) = P(\text{Buys} = \text{No}) \times \prod_{k=1}^{n} P(x_k \mid \text{Buys} = \text{No})$$

$= P(\text{Buys} = \text{No}) \times P(\text{Age} = \text{Youth} \mid \text{Buys} = \text{No}) \times$
$P(\text{Income} = \text{Medium} \mid \text{Buys} = \text{No}) \times$
$P(\text{Student} = \text{Yes} \mid \text{Buys} = \text{No}) \times$
$P(\text{Credit Rating} = \text{Fair} \mid \text{Buys} = \text{No})$

$$= \frac{5}{14} \times \frac{3}{5} \times \frac{2}{5} \times \frac{1}{5} \times \frac{2}{5}$$

$$= 0.007$$

Since, $P(\text{Buys} = \text{Yes} \mid X) > P(\text{Buys} = \text{No} \mid X)$;
Predicted class label for the given data is Buys Computer = Yes.

Variants of Naive Baye's Classifier

1. Gaussian Naive Bayes

- Gaussian Naive Bayes is used when features of data set have gaussian distribution.
- If features of dataset have continuous values, they follow gaussian distribution.
- In this case, probability of feature x_i belonging to class C is estimated using gaussian probability distribution function as given below:

$$P(x_i|C) = \frac{1}{\sqrt{2\pi}\,\sigma_c} \exp\left(-\frac{(x_i-\mu_c)^2}{2\sigma_c^2}\right)$$

where,

μ_c and σ_c are mean and standard deviation of the feature belonging to class C.

2. Multinomial Naive Bayes

- It is used when input features follows multinomial distribution. It is primarily used in text classifications.
- It calculates probability of feature x_i belonging to class C as below:

$$P(x_i|C) = \frac{N_{ic} + \alpha}{N_c + \alpha \times n}$$

where,

N_{ic} is no. of occurrences of feature x_i in class C.
N_c is the no. of samples belonging to class C.
α is smoothing factor (Normally $\alpha = 1$ is used).
n is no. of dimensions.

3. Bernoulli Naive Bayes

- It is used when input features follows multivariate Bernoulli distribution. When input features of dataset are binary-value, they follow Bernoulli distribution.

- It calculates probability of feature x_i belonging to class C as below:

$$P(x_i|C) = P(x_i|C)x_i + (1 - P(x_i|C))(1 - x_i)$$

Laplace Smoothing

- It is the technique used to handle zero probability in Naive Bayes classifier.
- It estimates probability of feature x_i belonging to class C as below: (Case of multinomial Naive Bayes)

$$P(x_i|C) = \frac{N_{ic} + \alpha}{N_c + \alpha \times n}$$

where,

N_{ic} is no. of occurrences of feature x_i in class C.
N_c is the no. of samples belonging to class C.
α is smoothing factor (Normally $\alpha = 1$ is used).
n is no. of dimensions.

Example:

Consider the following dataset.

Email	Class
Send us your password	Spam
Send your account	Spam
Review your account	Ham
Review your password	Ham
Review our profile	Spam
Send us money	Spam
Review your profile	?

Sol:

Assume $\alpha = 1$.

Vocabulary = {send, us, your, password, account, review, our, profile, money}

$P(\text{Spam}) = 4/6$

$P(\text{Ham}) = 2/6$

Send: $P(\text{send} \mid \text{Spam}) = \frac{3+1}{4+1\times 9} = \frac{4}{13}$

$P(\text{send} \mid \text{Ham}) = \frac{0+1}{2+1\times 9} = \frac{1}{11}$

us: $P(\text{us} \mid \text{Spam}) = \frac{2+1}{4+1\times 9} = \frac{3}{13}$

$P(\text{us} \mid \text{Ham}) = \frac{0+1}{2+1\times 9} = \frac{1}{11}$

your: $P(\text{your} \mid \text{Spam}) = \frac{2+1}{4+1\times 9} = \frac{3}{13}$

$P(\text{your} \mid \text{Ham}) = \frac{2+1}{2+1\times 9} = \frac{3}{11}$

Password: $P(\text{Password} \mid \text{spam}) = \frac{1+1}{4+1\times9} = \frac{2}{13}$

$P(\text{Password} \mid \text{Ham}) = \frac{1+1}{2+1\times9} = \frac{2}{11}$

account: $P(\text{account} \mid \text{spam}) = \frac{1+1}{4+1\times9} = \frac{2}{13}$

$P(\text{account} \mid \text{Ham}) = \frac{1+1}{2+1\times9} = \frac{2}{11}$

review: $P(\text{review} \mid \text{spam}) = \frac{1+1}{4+1\times9} = \frac{2}{13}$

$P(\text{review} \mid \text{Ham}) = \frac{2+1}{2+1\times9} = \frac{3}{11}$

our: $P(\text{our} \mid \text{spam}) = \frac{1+1}{4+1\times9} = \frac{2}{13}$

$P(\text{our} \mid \text{Ham}) = \frac{0+1}{2+1\times9} = \frac{1}{11}$

Profile: $P(\text{profile} \mid \text{spam}) = \frac{1+1}{4+1\times9} = \frac{2}{13}$

$P(\text{profile} \mid \text{Ham}) = \frac{0+1}{2+1\times9} = \frac{1}{11}$

money: $P(\text{money} \mid \text{spam}) = \frac{1+1}{4+1\times9} = \frac{2}{13}$

$P(\text{money} \mid \text{Ham}) = \frac{0+1}{2+1\times9} = \frac{1}{11}$

Now,

$P(\text{Spam} \mid \text{"Review your profile"}) = P(\text{Spam} \mid [0,0,1,0,0,1,0,1,0])$

$= P([0,0,1,0,0,1,0,1,0] \mid \text{Spam}) \times P(\text{Spam})$

$= \left(\frac{1-4}{13}\right) \times \left(\frac{1-3}{13}\right) \times \frac{3}{13} \times \left(\frac{1-2}{13}\right) \times \left(\frac{1-2}{13}\right) \times \frac{2}{13} \times \left(\frac{1-2}{13}\right) \times \frac{2}{13} \times \left(\frac{1-2}{13}\right) \times \frac{4}{6}$

$= 0.000994.$

$P(\text{Ham} \mid \text{"Review your profile"}) = P(\text{Ham} \mid [0,0,1,0,0,1,0,1,0])$

$= P([0,0,1,0,0,1,0,1,0] \mid \text{Ham}) \times P(\text{Ham})$

$= \left(\frac{1-1}{11}\right) \times \left(\frac{1-1}{11}\right) \times \frac{3}{11} \times \left(\frac{1-2}{11}\right) \times \left(\frac{1-2}{11}\right) \times \frac{3}{11} \times \left(\frac{1-1}{11}\right) \times \frac{1}{11} \times \left(\frac{1-1}{11}\right) \times \frac{2}{6}$

$= 0.00103$

Since, $P(\text{Ham} \mid \text{"Review your profile"}) > P(\text{Spam} \mid \text{"Review your profile"})$.

Predicted class label for the given email message is "Ham".

classmate
Date 03/22
Page

Decision Tree Classifier

- Decision tree is an algorithm that constructs model in the form of decision tree from training data.
- Decision tree is the tree structured model where internal nodes represents test on attribute, edges represents test outcomes, and leaf nodes represent class label.
- At the time of prediction, we just need to trace path from root node to leaf node on the basis of attribute values given in the tuple. Class label of the leaf node will be predicted class label for the given tuple.

Outline of Decision tree Algorithm

1. Initially, all training tuples are considered in root node.
2. Partition the tuples recursively on the basis of selected attribute.
3. If all samples of the given node belongs to same class,
   - Label the class.
4. Else if there are no remaining attributes for further partitioning
   - Use majority voting to label the class
5. Else
   - Go to step 2.

- There are many variations of decision tree. Some of them are : ID3, C 4.5, CART, etc.

- There are different attribute selection measure used by various decision tree algorithm. Some of the attribute selection measures are : Information Gain, Gain Ratio, Gini Index, etc.

ID3 Algorithm

- ID3 stands for iterative dichotomiser-3. It uses information gain as attribute selection measure. Information gain is calculated from entropy.

- Entropy is the measure of homogeniety of data sample and is calculated as below:

$$E(\mathcal{D}) = \sum_{i=1}^{m} -P_i \log_2 P_i$$ (before partitioning)

where,

m is no. of classes

P_i is probability of tuple in $\mathcal{D}$ belonging to class C_i.

- P_i can be calculated as below:

$$\frac{|C_{i,\mathcal{D}}|}{|\mathcal{D}|}$$

where,

$|C_i, \mathcal{D}|$ is no. of tuple belonging to class C_i and $|\mathcal{D}|$ is total no. of tuples in data set.

- Suppose dataset $\mathcal{D}$ is partitioned on the basis of attribute A having v distinct values. Thus, V partition are created $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_v\}$. Now total entropy of the dataset after partitioning on A can be calculated as below:

$$E_A(\mathcal{D}) = \sum_{i=1}^{v} \frac{|\mathcal{D}_i|}{|\mathcal{D}|} E(\mathcal{D}_i)$$

- Information gain is reduction in entropy after partitioning. Thus, information after partitioning on A can be calculated as below.

$$\boxed{\text{Gain}(A) = E(\theta) - E_A(\theta)}$$

- ID3 selects attribute with highest information for partitioning.

Example: (eg. of Naive Baye's)

soln:

Entropy of the dataset before partitioning is given as below:

$$E(\theta) = \frac{-9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14}$$

$$= 0.94$$

Entropy of the dataset after partitioning on "Age".

$$E_{Age}(\theta) = \frac{5}{14}\left\{\frac{-2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right\} +$$

$$\frac{4}{14}\left\{\frac{-4}{4}\log_2\frac{4}{4} - \frac{0}{4}\log_2\frac{0}{4}\right\} +$$

$$\frac{5}{14}\left\{\frac{-3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}\right\}$$

$$= 0.694$$

Thus, Information Gain after partitioning on Age is

$$\text{Gain}(Age) = E(\theta) - E_{Age}(\theta)$$

$$= 0.246$$

Similarly,

Gain (Income) = 0.029

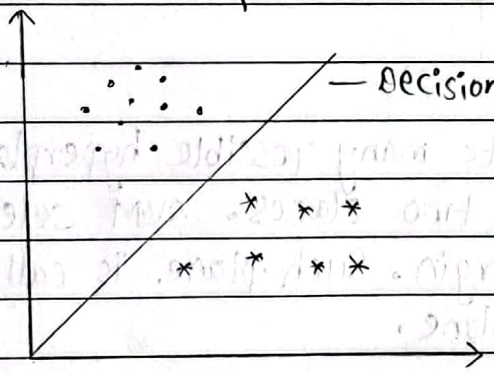
Gain (Student) = 0.151

Gain (Credit Rating) = 0.048

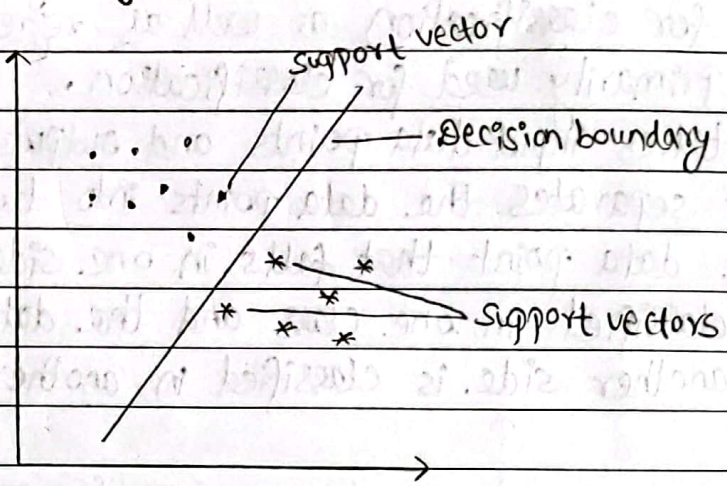
Since, information gain of age is highest, it is selected as partitioning attribute and the decision tree looks like below.

Support Vector Machine (SVM)

- SVM is the supervised learning algorithm that can be use for classification as well as regression. However, it is primarily used for classification.
- It takes input data points and outputs a hyperplane that best separates the data points into two classes.
- Any data point that falls in one side of the hyperplane is classified in one class and the data points that falls in another side is classified in another class.



- Support vectors are the data points that are closest to the decision boundary.



- There may exists many possible hyperplane that divides data points into two classes. SVM selects the plane with largest margin. Such plane is called maximal marginal hyperplane.

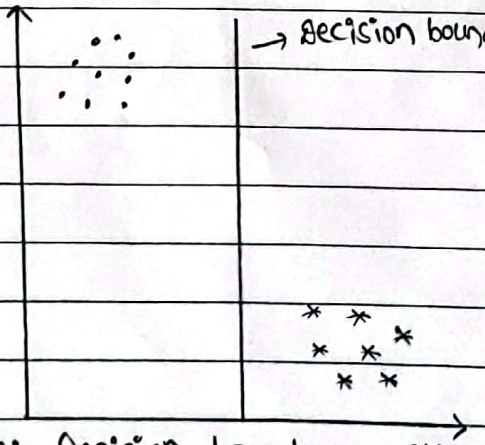


Fig: Decision boundary with small margin

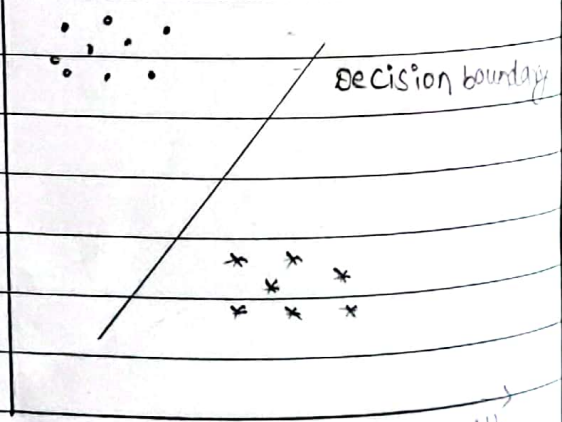
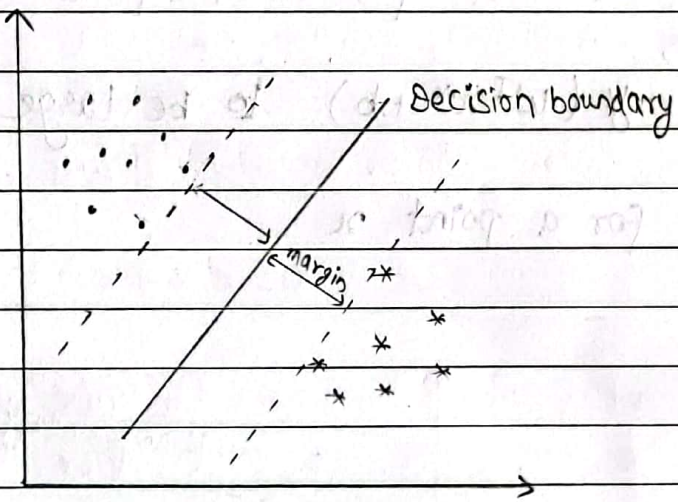


Fig: Decision boundary with large margin

- Let H_1 and H_2 are two hyperplanes that passes through support vectors and parallel to the hyperplane of decision boundary. Margin is the distance H_1 and H_2 such that distance between H_1 and decision boundary must be equal to the distance between H_2 and the decision boundary.



Mathematical Formulation of SVM :

Let equation of decision is

$$w_1x_1 + w_2x_2 + b = 0$$

$$\Rightarrow w^Tx + b = 0$$

For a new point x^*

if $w^Tx^* + b = 0$ x^* is on the decision boundary

if $w^Tx^* + b > 0$ x^* is above the decision boundary.

if $w^Tx^* + b < 0$ x^* is below the decision boundary

For classification,

If $w^Tx^* + b > 0$ x^* belongs to +ve class (i.e. $y = 1$)

If $w^Tx^* + b < 0$ x^* belongs to -ve class (i.e. $y = -1$)

$\Rightarrow y(w^Tx^* + b) > 0$

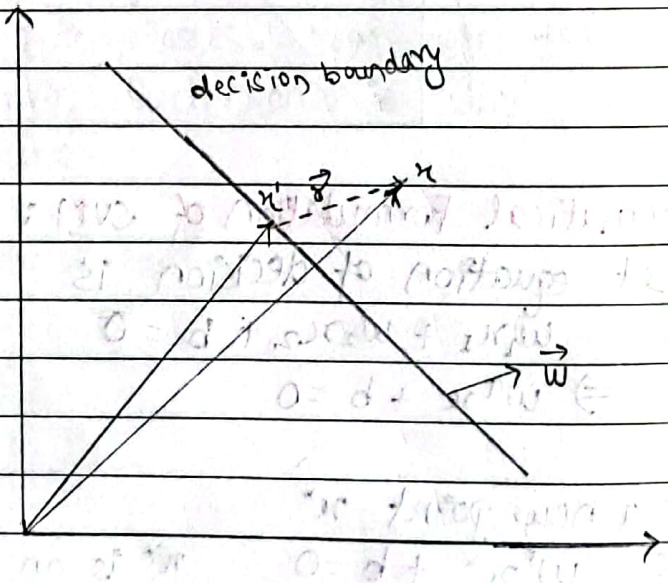
We want to find the decision boundary such that all data points belonging to both classes are far from the boundary so that class label of every data point can be predicted with higher confidence.

i.e. We want $|w^T x^* + b|$ to be large

or,

$y(w^T x^* + b)$ to be large.

Now, for a point x



$$\vec{r} = \vec{x} - \vec{x'} \quad \text{——— (i)}$$

For x', $w^T x' + b = 0$ ——— (ii)

Let $|\vec{r}| = r$

Since $\vec{r}$ is parallel to $\vec{w}$

$$\Rightarrow \vec{r} = r \frac{\vec{w}}{||\vec{w}||}$$

From eqn (i)

$$\vec{x'} = \vec{x} - \vec{r} = \vec{x} - r \frac{\vec{w}}{||\vec{w}||}$$

Now, eq^n (11) can be written as below:

$$w^T\left(\vec{x} - \gamma\frac{\vec{w}}{\|\vec{w}\|}\right) + b = 0$$

$$\Rightarrow w^T\vec{x} - \frac{\gamma\,\vec{w}^T\vec{w}}{\|\vec{w}\|} + b = 0$$

$$\Rightarrow w^T\vec{x} - \frac{\gamma\,\|w\|^2}{\|\vec{w}\|} + b = 0$$

$$\Rightarrow w^T\vec{x} - \gamma\|w\| + b = 0$$

$$\Rightarrow \gamma = \frac{w^T\vec{x} + b}{\|w\|}$$

Similarly, for a -ve point

$$\gamma = -\frac{(w^T\vec{x} + b)}{\|w\|}$$

$\Rightarrow$ In general,

$$\gamma = y\left(\frac{w^T\vec{x} + b}{\|w\|}\right)$$

For the point on H_1.

$$\gamma = \frac{w^T\vec{x} + b}{\|w\|} = \frac{1}{\|w\|}$$

and for the point on H_2

$$\gamma = \frac{w^T\vec{x} + b}{\|w\|} = \frac{1}{\|w\|}$$

$\Rightarrow$ Total margin:

$$= \frac{1}{\|w\|} + \frac{1}{\|w\|} = \frac{2}{\|w\|}$$

Our goal is to maximize the margin.

Maximizing $\frac{2}{\|w\|}$ is equivalent to minimizing $\|w\|$.

Thus, we can formulate following optimization problem.

$$\underset{w,b}{\text{minimize}} \quad \frac{\|w\|^2}{2}$$

$$\text{subject to } y(w^T x_i + b) \geq 1 \quad ; i = 1, 2, \ldots n.$$

Above optimization problem can be solved using quadratic programming.

Functional and Geometric Margins of SVM

Functional: We can write SVM classifier as:

$$h_{w,b}(x) = g(\underbrace{w^T x + b}_{(z)})$$

where,

$$g(z) = 1, \text{ if } z \geq 0 \text{ and } g(z) = -1 \text{ otherwise.}$$

Given a training example (x^i, y^i), functional margin of the decision boundary parameterized by (w, b) is defined as below:

$$\gamma^i = y^i (w^T x^i + b) \quad \text{---} \quad (1)$$

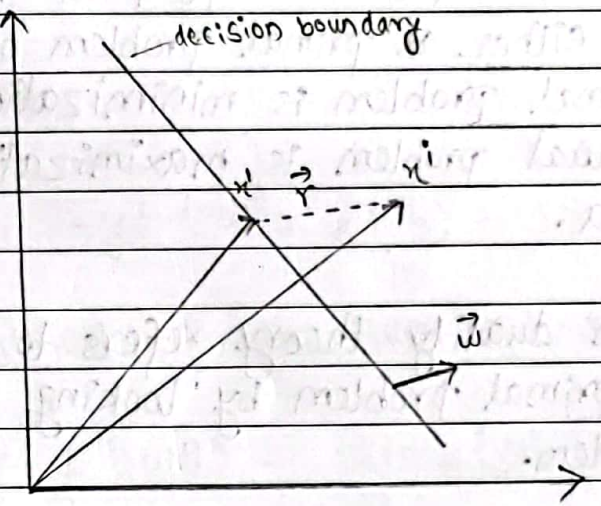
If we replace w by $2w$ and b by $2b$ then

$$g(w^T x + b) = g(2w^T x + 2b)$$

$\Rightarrow$ We can make functional margin arbitrarily large without changing meaning. This is drawback of functional margin.

Geometric Margin

For a point x belonging to +ve class.



$$w^T\left(\vec{x}i - \frac{y^i \vec{w}}{\|w\|}\right) + b = 0$$

$$\Rightarrow y^i = \frac{w^T x^i + b}{\|w\|} = \left(\frac{w}{\|w\|}\right)^T x^i + \frac{b}{\|w\|}$$

Thus, geometric margin of the decision boundary parametarized by (w,b) is given as below:

$$\gamma^i = y^i\left(\frac{w^T x^i}{\|w\|} + \frac{b}{\|w\|}\right)$$

Lagrange Duality

- In mathematical optimization theory, duality is the principle that states that optimization problems can be viewed either as primal problem or dual problem. If the primal problem is minimization problem then the dual problem is maximization problem and vice-versa.

- Lagrangian duality theory refers to the way to solve a primal problem by looking at the corresponding dual problem.

Karush - Kuhn - Tucker (KKT) conditions

- KKT conditions are tests on first derivatives and other regularity conditions that needs to be satisfied for solving a non-linear programming problem.

- KKT conditions for solving SVM optimization problem are given below:

$\frac{\partial L}{\partial w} = 0$	$\frac{\partial L}{\partial b} = 0$
$\alpha_i g_i(w) = 0$ $g_i(w) \leq 0$ $\alpha_i \geq 0$	

Solving SVM optimization Problem

SVM optimization problem is

$$\min_{w,b} \frac{1}{2}\|w\|^2$$

subject to condition

s.t.c $y^i(w^T x^i + b) \geq 1$ for $i = 1, 2, \ldots, n$

$\Rightarrow g_i(w) = -y^i(w^T x^i + b) + 1 \leq 0$

Lagrangian function for the given optimization problem is:

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i (w^T x^i + b) - 1\right) \quad \text{---(1)}$$

where, α is lagrange multiplier

Thus, the lagrangian dual problem is

$$\theta_D(\alpha) = \min_{w,b} L(w, b, \alpha)$$

Now,

$$\frac{\partial L}{\partial w} = 0 \Rightarrow w - \sum_{i=1}^{n} \alpha_i y^i x^i = 0$$

$$\Rightarrow w = \sum_{i=1}^{n} \alpha_i y^i x^i \quad \text{---(2)}$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{i=1}^{n} \alpha_i y^i = 0 \quad \text{---(3)}$$

Substituting the value of w in eqn (1), we get,

$$L(w, b, \alpha) = \frac{1}{2}\left(\sum_{i=1}^{n} \alpha_i y^i x^i \cdot \sum_{j=1}^{n} \alpha_j y^j x^j\right) - \sum_{i=1}^{n} \alpha_i \left(y^i \left(\sum_{j=1}^{n} \alpha_j y^j x^j\right) x^i + b\right) - 1$$

$$\Rightarrow L(w,b,\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y^i y^j \alpha_i \alpha_j (x^i x^j)$$

Thus, the task is to solve the following optimization problem.

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y^i y^j \alpha_i \alpha_j (x^i x^j)$$

s.t.c

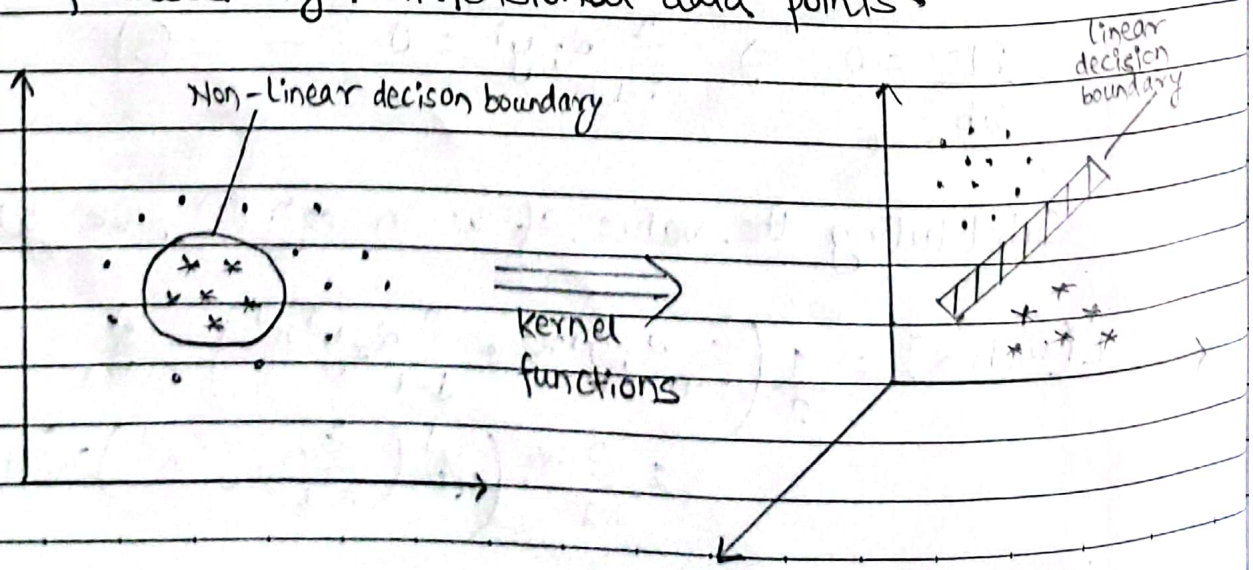
$$\alpha_i > 0$$

$$\sum_{i=1}^{n} \alpha_i y^i = 0$$

Once we get value of α, we substitute the value in eq^n (2) to get w.

SVM Kernels

- SVM uses a set of mathematical functions that are called kernels functions or simply kernels.
- These functions are used to transform non-linearly separable low dimensional data points to linearly separable high dimensional data points.



Definition of Kernel functions

Kernel function is a mathematical function that takes vector in original space as input and returns dot product of vectors in transformed space.

i.e. $K(a,b) = (a \cdot b)^2 = \phi(a) \cdot \phi(b)$

Consider the following kernel function

$$K(a,b) = (a \cdot b)^2 = \phi(a) \cdot \phi(b)$$

where,

$$\phi\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1^2 \\ \sqrt{2}\, x_1 x_2 \\ x_2^2 \end{pmatrix}$$

Now,

$$\phi(a) = \phi\begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} a_1^2 \\ \sqrt{2}\, a_1 a_2 \\ a_2^2 \end{pmatrix}$$

$$\phi(b) = \phi\begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} b_1^2 \\ \sqrt{2}\, b_1 b_2 \\ b_2^2 \end{pmatrix}$$

Computing dot product of $\phi(a)$ and $\phi(b)$.

$$\phi(a) \cdot \phi(b) = \begin{pmatrix} a_1^2 \\ \sqrt{2}\, a_1 a_2 \\ a_2^2 \end{pmatrix} \cdot \begin{pmatrix} b_1^2 \\ \sqrt{2}\, b_1 b_2 \\ b_2^2 \end{pmatrix}$$

$$= a_1^2 b_1^2 + 2 a_1 a_2 b_1 b_2 + a_2^2 b_2^2$$

$$= (a_1 b_1 + a_2 b_2)^2$$

$$= \left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}\right)^2$$

$$= (a \cdot b)^2$$

Example:

Consider the kernel function $k(a, b) = (a \cdot b)^2$ and mapping function $\phi(x) = \begin{pmatrix} x_1^2 \\ \sqrt{2}\, x_1 x_2 \\ x_2^2 \end{pmatrix}$

Show that kernel function gives dot product of data points in transformed space. Assume $a = (1,2)$ and $b = (2,3)$

Types of Kernels

1. Linear kernel

$$K(a,b) = (a \cdot b),$$

where, a and b are data points in original space.

2. Polynomial kernel

$$K(a,b) = (a \cdot b + c)^d,$$

where, d denotes degree of polynomial

3. RBF kernel

RBF - Radial Basis Function

$$K(a,b) = \exp\left(-\gamma \|a-b\|^2\right)$$

where, γ is the parameter whose value ranges between 0-1.

4. Gaussian kernel

$\|a-b\|^2$ → Euclidean distance

$$K(a,b) = \exp\left(-\frac{\|a-b\|^2}{2\sigma^2}\right)$$

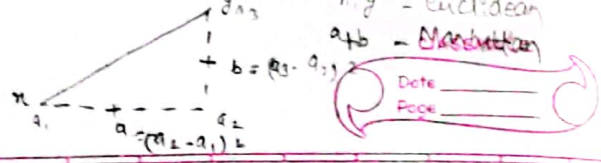
where, σ is bandwidth

5. Sigmoid kernel

$$K(a,b) = \tanh\left(\alpha a^T b + c\right)$$

where, α is the parameter whose value ranges between 0-1.

- It is equivalent to two layer perceptron model of neural network.



6. Exponential kernel

Manhattan distance

$$k(a,b) = \exp\left(-\frac{\|a-b\|}{2\sigma^2}\right)$$

Kernel Mercer Theorem

- Kernel matrix is a symmetric matrix that is obtained by applying kernel function k in every pair of points in the dataset.

i.e. $K_{i,j} = K(x_i, x_j)$, $i,j = 1,2,3,\ldots n$.

or,

$$K_{i,j} = \begin{bmatrix} K(x_1,x_1) & K(x_1,x_2) & \cdots & K(x_1,x_n) \\ K(x_2,x_1) & K(x_2,x_2) & \cdots & K(x_2,x_n) \\ \vdots & & & \\ K(x_n,x_1) & K(x_n,x_2) & \cdots & K(x_n,x_n) \end{bmatrix}$$

- Kernel Mercer theorem states that necessary and sufficient condition for a kernel K to be valid kernel is that it's kernel matrix must be symmetric positive semi-defnit.

SMO Algorithm

- SMO stands for sequential minimal optimization.
- SMO algorithm can be used to solve SVM dual problem.

Coordinate Ascent Algorithm

- This algorithm is used to solve unconstrained optimization problem of the form.

$$\max_{\alpha} \; w(\alpha_1, \alpha_2, \ldots, \alpha_n)$$

Algorithm:

Loop until convergence

{

for $i = 1$ to n

{

$$\alpha_i = \arg\max_{\hat{\alpha}_i} W(\alpha_1, \alpha_2, \ldots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \ldots, \alpha_n)$$

}

}

SMO Algorithm:

SVM dual problem that we have to solve is

$$W(\alpha) = \max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y^i y^j \alpha_i \alpha_j (x^i . x^j) \quad \text{---} (1)$$

s.t.c $\alpha_i \geq 0$ — (2)

$$\boxed{\sum_{i=1}^{n} \alpha_i y^i = 0} \quad \text{---} (3)$$

Coordinate ascent algorithm discussed above can not be used to solve SVM dual problem because if we optimize α_1 by keeping $\alpha_2 - \alpha_n$ fixed, it leads to

$$\alpha_1 y^1 + \sum_{i=2}^{n} \alpha_i y^i = 0$$

$$\Rightarrow \alpha_1 = -\frac{1}{y^1} \sum_{i=1}^{n} \alpha_i y^i$$

This violates constraint specified in eq^n (2).

Thus, we must update at least two α_i simultaneously to keep constraint satisfying.

⇒ SM algorithm for solving SVM dual problem is given as below:

Algorithm

Repeat until convergence

{

- select pair of α_i and α_j
- optimize $w(\alpha)$ w.r.t. α_i and α_j by keeping other α's fixed.

}