Lecture 13

`After MidSem`

Name: Harsh Sanjay Roniyar

Roll Number: 22B3942

MidSem Solution Discussion

Now, continuing with classifiers -

SVM for distribution-free learning

Empirical risk: risk of misclassifying training data == structural risk

Minimizing empirical risk implies maximizing margin, i.e. having support vectors close to the separating hyperplanes.

$$\max_{w,b}[\min_i||\mathbf{w}^T\mathbf{x}_i + b||]$$

is the constrained optimization problem subject to the constraints : $||\mathbf{w}|| = 1$ and
$\forall i, t_i(w^T x_i + b) \geq 0$.
$t_i \in \{-1, 1\}$

The above method was one type of formulation for this, another formulation can be:

$$\min_{v,c}||\mathbf{v}||^2$$

such that $\forall i, t_i(\mathbf{v}^T\mathbf{x}_i + c) \geq 1$ and $\mathbf{v}/c = \mathbf{w}/b$.

Lecture 14

Name: Harsh Sanjay Roniyar

Roll Number: 22B3942

`Assignment Questions Overview`

SVMs

Continuing from the previous lecture,
We saw multiple (convex) optimization objective functions in Lecture 13

For mathematical convenience,

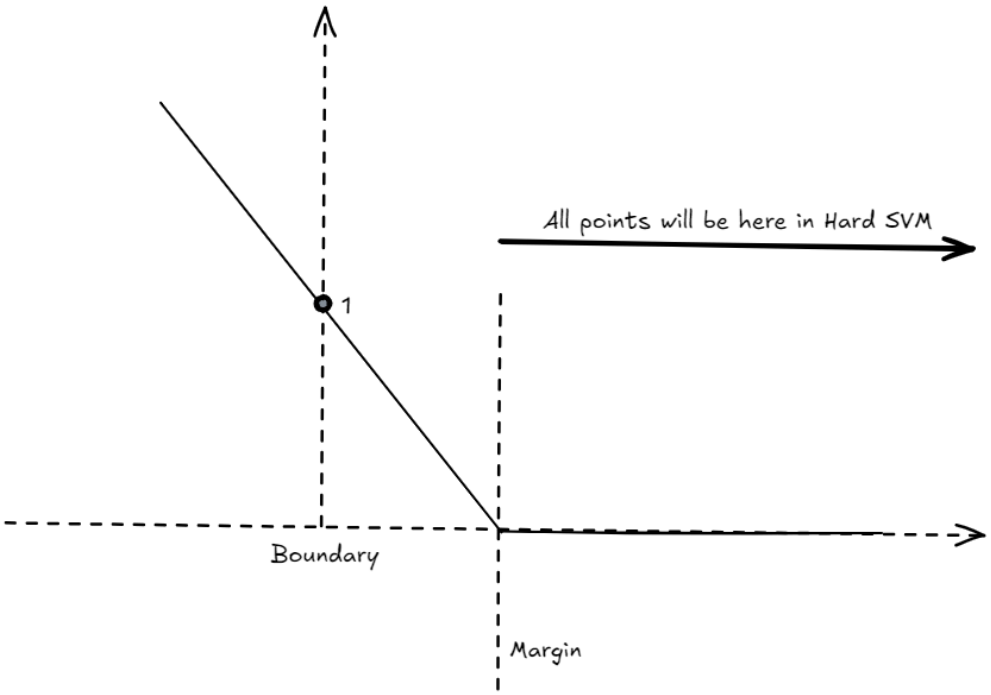
$$\max[0, y] \equiv [y]_+$$

and

$$max[0, 1 - y] \equiv l_{hinge}[y]$$

Up until now, we have discussed `Hard SVM` where, we do not allow points between the margins/support vectors (and the loss between the margins increases linearly up to the boundary, where the loss would already be 1, beyond boundary would be > 1). If the classification point lies beyond the margins, then the loss is definitely zero.

But, if we add newer points which lies between the current support vectors, then the new points closest to the boundary would end up becoming the new margins (`support vector`).



y-axis in this plot is the l_{hinge}. And from this plot, it can be seen that the loss function is convex, using `Jensen's Inequality`

Soft-Margin SVM

Formulation:

$$\min \left(\|w\|_2^2 + C \sum_i \zeta_i \right)$$

where, ζ_i are the slack variables subject to -

- $\forall i \; t_i[w^T x_i + b] \geq 1 - \zeta_i$
- $\forall i \; \zeta_i \geq 0$

Equivalently,

$$\min_{w,b} \sum_i l_{hinge}(t_i[w^T x_i + b]) + Regularization$$

where we can choose the regularization of our choice, $L1$ or $L2$.
`As can be noted again, this optimization is also a convex problem`

For Logistic Regression: $\min \text{CE} + \frac{\lambda}{2}\|w\|_2^2$

Values of ζ_i:

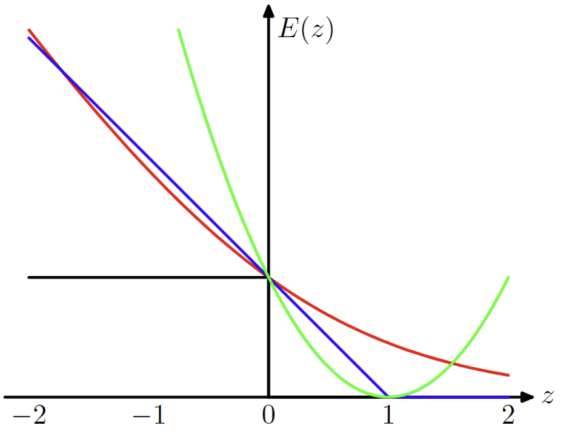
- Correct side of margin = 0
- On the margin = 0
- Inside the margin > 0

- On the boundary = 1
- Beyond the boundary > 1

Comparison with Hard SVM:

In Hard SVMs, support vectors define the hyperplane and are closest to the hyperplane, which is usually a sparse set of points. The remaining points do not matter since they cannot affect the values of w and b.

In Soft SVMs, all the misclassified points are support vectors too, and consequently all points with $\zeta_i > 0$ or just at the cusp (read `margin`).



Plot of the 'hinge' error function used in support vector machines, shown in blue, along with the error function for logistic regression, rescaled by a factor of $1/\ln(2)$ so that it passes through the point $(0, 1)$, shown in red. Also shown are the misclassification error in black and the squared error in green.

The red curve is the cross entropy over sigmoid of $w^T x_i + b$. CE(Sigmoid())

`A small caveat to keep in mind is that we considered targets as {-1, +1} but for cross entropy over sigmoid would have to relabel to {0, +1} for usage`

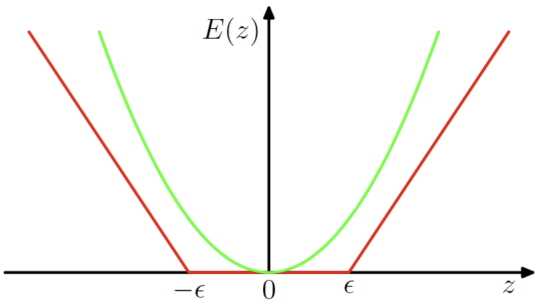
SV Regression

We change the mean squared error function to a ϵ-insensitive (fault-tolerant) error function.

$$C\sum_{n=1}^{N} E_\epsilon(y(x_n) - t_n) + \frac{1}{2}\|w\|^2$$

where,

$$E_\epsilon(y(x) - t) = \begin{cases} 0 & \text{if } |y(x) - t| < \epsilon \\ |y(x) - t| - \epsilon & \text{otherwise} \end{cases}$$



Plot of an ϵ-insensitive error function (in red) in which the error increases linearly with distance beyond the insensitive region. Also shown for comparison is the quadratic error function (in green).

Now, defining the slack variables (for fixed ϵ) -
$t_i \leq y_i + \epsilon + \xi_i$ for $\xi \geq 0$
$t_i \geq y_i - \epsilon - \hat{\xi}_i$ for $\hat{\xi}_i \geq 0$

Then, the cost function can be written as -

$$\min_{w,b} \mathrm{C} \sum_i \left(\xi_i + \hat{\xi}_i\right) + \frac{1}{2}\|w\|_2^2$$

Lecture 15

Name: Harsh Sanjay Roniyar

Roll Number: 22B3942

`Assignment Discussion/Doubts`

Now beginning with a shorter topic

Feature Extraction and Selection

Increasing or decreasing the number of features and how they affect our classification for the data

Variable Transformation

Say, we have a heavy tailed distribution (i.e. not Gaussian, usually for income distribution), then we can create a new feature, $\phi(x) = \log(x + \epsilon)$.

Some other transformations are:

- Power: $\phi(x) = x^p$
- Exponential: $\phi(x) = e^{ax+b}$

These transformations that help in approximate transformation of some other distribution into a Gaussian Distribution, since all our techniques were based on Gaussian.

So, how would we do an **exact** transformation:
$x \sim p_x(x)$
Then, find a transformation $\hat{x} = f(x)$ such that $q_x(\hat x) is a desired (and well-behaved) distribution.

Goal: Find the transformation $f(x)$ -

$$\int_{x_1}^{x_2} p(x)dx = \int_{f(x_1)}^{f(x_2)} q(\hat{x})d\hat{x}$$

This reduces to,

$$q(\hat{x}) = p(f^{-1}(\hat{x})) \left| \frac{df^{-1}(\hat{x})}{d\hat{x}} \right|$$

which is equivalent to,

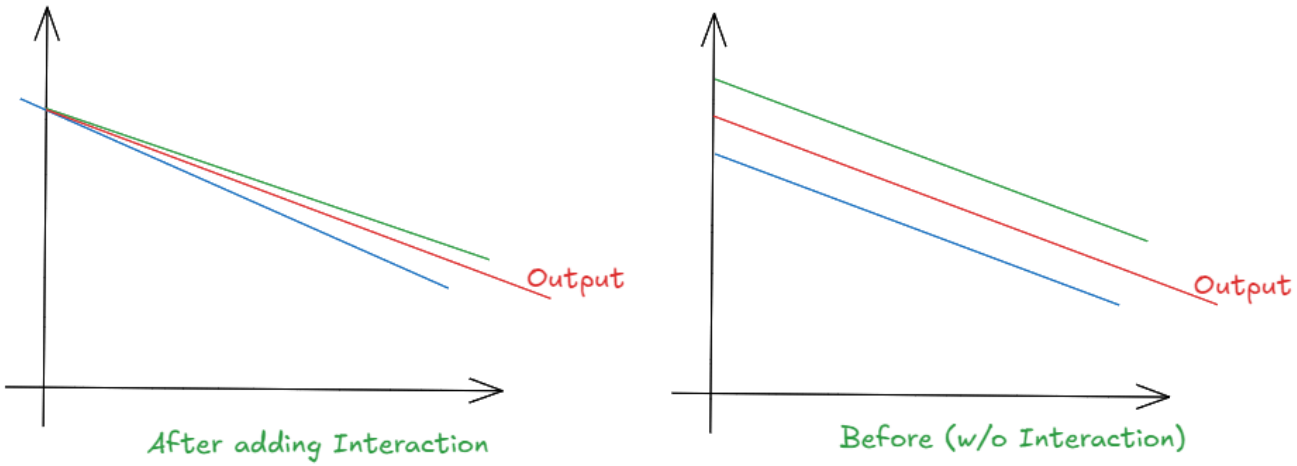
$$q(\hat{x}) = p(x) \left| \frac{dx}{d\hat{x}} \right|$$

Interaction Variables

x_1	x_2	$x_3 = x_1 x_2$
Row 1, Col 1	Row 1, Col 2	Row 1, Col 3
Row 2, Col 1	Row 2, Col 2	Row 2, Col 3
Row 3, Col 1	Row 3, Col 2	Row 3, Col 3

We can model either as $w_1 x_1 + w_2 x_2 + b$, or as $w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + b$.

In the second case, this will allow us to model x_1 and x_2 separately and their combination is now part of a new feature, since from the initial model we can see that the output would be something midway between both x_1 and x_2. So adding x_3 allows the two features to interact in some way, and we would be able to represent the relation between them `via` x_3.



Modelling discrete variables

Assume that the discrete variable has C unique values.
So, there are two scenarios -

1. The variable is an input to a model: Then we will have to convert the variable to binary form for our model ($\because$ we would have C binary variables)
2. The variable is an output of a model $\implies$ **One-Hot Encoding**

Cross Entropy Loss $= -\sum_{i=1}^{N} \sum_{j=1}^{C} t_{ij} \log p_{ij}$,
where t_{ij} is the j^{th} one-hot encoding for the i^{th} sample

But, this one-hot encoding would not work for the input since that would just add redundancy, leading to instability in the data, because, for example $w_1 x_1 + w_2 x_2 + w_3 x_3 + b$,

after one hot encoding would be $w_1x_1 + w_2x_2 + w_3(1 - x_1 - x_2) + b$, where the third term is redundant.

Therefore for input, we use **C-1** binary columns $\implies$ Dummy Encoding

`Note: Do not use integers to encode discrete variables`

An example illustrating the difference between the encoding depending on whether it is an input to the model or output of the model.

Pet Type	x_1, x_2 (Input)	x_1, x_2, x_3 (Output)
Dog	1 0	1 0 0
Dog	1 0	1 0 0
Cat	0 1	0 1 0
Parrot	0 0	0 0 1

Image Features

- Pixel-level
  - Gray-scale histograms
  - Color histograms
    But these representations won't retain shape of the images
- Shape-based
  - Hue invariant moments
- Texture-based
  - Fourier descriptors

Audio Features

- MFCC
  - Windowing
  - DFT: Power Spectral Density
  - Filter Bank
  - DCT: Discrete Cosine Transform ($\sim$ to Fourier Transform without complex components)

Text Features

TF-IDF and then using SVM was the classical way of processing text earlier.

TF-IDF

TF-IDF stands for *Term Frequency - Inverse Document Frequency*, used to evaluate how important a word is to a document in a corpus.

Term Frequency (TF)

The term frequency (tf) of a term t in a document d is calculated as:

$$\mathrm{tf}(t, d) = \frac{f(t, d)}{\sum_z f(z, d)}$$

where:

- $f(t, d)$ is the frequency of term t in document d.
- $\sum_z f(z, d)$ is the sum of frequencies of all terms in the document.

Inverse Document Frequency (IDF)

The inverse document frequency (idf) of a term t in a set of documents D (corpus) is calculated as:

$$\mathrm{idf}(t, D) = \log\left(\frac{|D|}{1 + |\{d \in D : t \in d\}|}\right)$$

where:

- $|D|$ is the total number of documents.
- $|\{d \in D : t \in d\}|$ is the number of documents containing the term t.

TF-IDF

The final TF-IDF value is the product of TF and IDF:

$$\text{TF-IDF} = \mathrm{tf}(t, d) \times \mathrm{idf}(t, D)$$

Lecture 16

Name: Harsh Sanjay Roniyar

Roll Number: 22B3942

Continuing from last lecture, where we wanted to find a transformation function so that it converts to an easier version.

> which is equivalent to,
>
> $$q(\hat{x}) = p(x) \left| \frac{dx}{d\hat{x}} \right|$$

Now, moving on

Feature Reduction:

- **Filter**: For each feature decide keep v/s discard. Then train your model with the kept features $\implies$ Best Subset Selection over multiple subsets
- **Wrapper**: For some n subsets, Forward Selection and Backward Elimination

Forward Selection:
start with an empty set S, set of remaining features R include all x_i's (i goes from 1 to D) at this point.

loop over j in D: `Note: alternative stopping criteria are allowed`
loop over elements x_i of R:
evaluate utility of including x_i in S
pick the best feature, include in S, remove from R
you get the subset features S and R

Backward Elimination:
S is the whole set and R is empty

loop over j:
loop over elements x_k of S:
evaluate utility of removing x_j from S
remove the worst feature from S, add to R
you get the subset S and R

- **Embedded**:
  Examples: LASSO (L1), Elastic Net (L1+L2)
  Learning and Subset Selection are integrated.

L2 SVM:

$$\frac{\lambda}{2}\|w\|_2^2 + \sum_i [1 - t_i y_i] + \text{ Hinge Loss}$$

L1 SVM:

$$\lambda \|w\|_1 + \sum_i [1 - t_i y_i] + \text{ Hinge Loss}$$

$$= \lambda \sum_j |w_j| + \sum_i [1 - t_i (w^T x_i + b)] + \text{ Hinge Loss}$$

where the first term represents sparsity in j and the other term represents sparsity in i

Principal Component Analysis

`Feature Reduction Technique`
$x_3 = ax_1 + bx_2$

Now, starting with Kernelized SVMs

Kernelized Support Vector Machines

`Detour:`
$y = f(w_1 x_1 + w_2 x_2 + \ldots + w_D x_D + w_0)$
If Gaussian distribution is assumed, then perform standard normalization (mean = 0, variance = 1).
If you assume uniform, perform min-max normalization.

Objectives:

- dual form of SVM
- replacing raw data with kernels
- derive support vector regression objective

Kernel - way to compare two samples in similarity.
$k(x, \hat{x}) \rightarrow$ high

$k(x, \tilde{x}) \rightarrow$ low

Example:

1. $k(x, \hat{x}) = exp(-a\|x - \hat{x}\|_2^2)$: Gaussian Kernel (Radial Basis Function (RBF))
2. $\frac{|x.\hat{x}|}{\|x\|\|\hat{x}\|}$ Cosine Kernel

Lecture 17

Name: Harsh Sanjay Roniyar

Roll Number: 22B3942

Kernels

$K(x_i, x_j) \rightarrow$ Similarity between x_i and x_j

$k(x_i, x_j) = \phi(x_i)^T \phi(x_j) \leftrightarrow$ Ensures that Gram Matrix is P.S.D. (positive semi-definite)

k is a simple computable expression, e.g. $e^{-k\|x_i - x_j\|_2^2}$ and the output is scalar,
$k(x_i, x_j) = k(x_j, x_i)$ and
$\phi(x_i)$ is the theoretical vector feature computed using x_i, and can have infinite dimension.

This above kernel example is known as `Mercer Kernel`.
i.e.

1. Implements an inner product in Hilbert space
2. Has a positive semi-definite Gram matrix.

These kernels are generally simple to compute although the vector features might not be easily computable

Kernel Trick

1. Take an objective function (such as a loss function) of x and try to manipulate it, so that it becomes a function of $x_i^T x_j$
2. Replace $x_i^T x_j$ with $\phi(x_i)^T \phi(x_j)$, which is basically $k(x_i, x_j)$

`Primal Form` - Objective Function of x
`Dual Form` - Function of $x_i^T x_j$

This trick is used to make linear problems non-linear in x-space, but will be easier to solve in the kernel space.

Primal Form of SVM

$$L(w, b) = \frac{1}{2}\|w\|_2^2 - \sum_i a_i[t_i(w^T x_i + b) - 1]$$

s.t. $\forall i\ a_i \geq 0$

Optimization Background:

$$L(x, \lambda) = f(x) + \lambda g(x), \lambda \geq 0$$

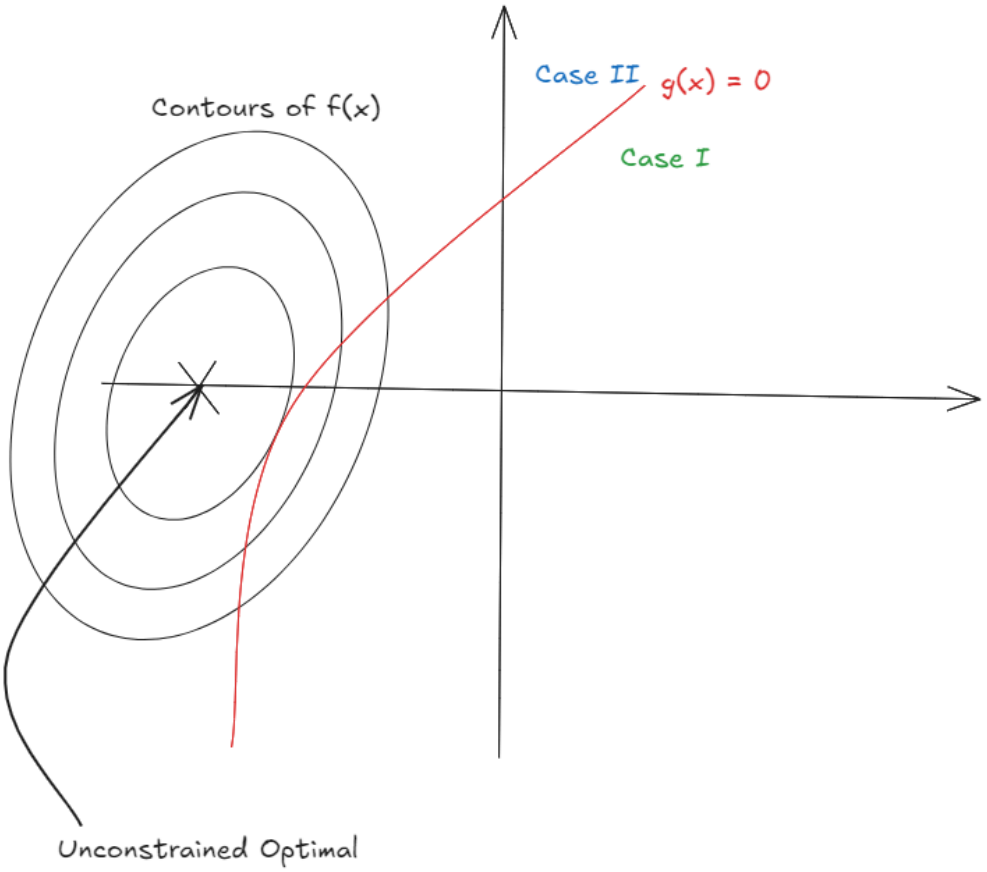
which is equivalent to

$$\max f(x), \text{s.t. } g(x) \geq 0$$

This is a single-constraint case.

There are two cases possible here:

1. $g(x) > 0$ is above the $g(x) = 0$ curve
2. $g(x) > 0$ is below the $g(x) = 0$ curve



For Case I, $\nabla f(x)$ is in opposite direction to $\nabla g(x) \implies \lambda > 0$
For Case II, $\nabla g(x)$ does not matter $\implies \lambda = 0$, since now we can directly find the unconstrained optimal value as we are guaranteed to lie on the safe side for $\lambda = 0$

The final optimization would look like

$$\min L(w, b, a) = \frac{1}{2}\|w\|_2^2 - \sum_i a_i[t_i(w^T x_i + b) - 1], a_i \geq 0$$

$$\frac{\partial L}{\partial w} = 0 \implies w = \sum_i a_i t_i x_i$$

$$\frac{\partial L}{\partial b} = 0 \implies 0 = \sum_i a_i t_i$$

Substituting the values, from above in the loss function $\rightarrow$
Therefore, the `Dual Formulation` in a will be -

$$L(a) = \sum_i x_i - \frac{1}{2} \sum_i \sum_j a_i a_j t_i t_j x_i^T x_j$$

second term derived from

$$\left[\sum_i a_i t_i x_i \right]^T \left[\sum_j a_j t_j x_j \right]$$

and we will replace $x_i^T x_j$ with our kernel function $k(x_i, x_j)$ which would apply $\phi(x_i)^T \phi(x_j)$.

Now, we will also need to compute the class, hence

$$y = \left(\sum_i a_i t_i x_i^T x_i \right) + b$$

Now, for eliminating b,

$$b = \frac{1}{N_S} \sum_{i \in S} \left[t_i - \sum_{j \in S} a_j t_j x_j^T x_j \right]$$

where, S is the set of support vectors ($a > 0$)

Case 1	Case 2
$g(x) = 0$ $\lambda > 0$	$g(x) > 0$ $\lambda = 0$
$\lambda g(x) = 0$	$\lambda g(x) = 0$

1. $a_i \geq 0, \forall i$
2. $t_i y(x_i) - 1 \geq 0$
3. $a_i \left[t_i y(x_i) - 1 \right] = 0$ `where the first term` (a_i) `is zero for non support vector and` $(t_i y(x_i) - 1 = 0)$ `for support vectors`.

These three conditions are known as the `KKT Conditions` (Karush-Kuhn-Tucker)

For Soft SVM

Primal Form:

$$L(w, b, a, \mu) = \frac{1}{2}\|w\|_2^2 - \sum_i a_i \left[t_i y(x_i) - 1 + \xi_i\right] + C\sum_i \xi_i - \sum_i \mu_i \xi_i, \text{ for } a_i \geq 0, \mu_i \geq 0 \forall i$$

where the last term is the Lagrangian, and now we will have `six` KKT conditions, since now we will also have a condition from

$$\frac{\partial L}{\partial \xi_i} = 0$$

The conditions are -

1. $a_i \geq 0$
2. $t_i y(x_i) - 1 + \xi_i \geq 0$
3. $a_i \left[t_i y(x_i) - 1 + \xi_i\right] = 0$
4. $\mu_i \geq 0$
5. $\xi_i \geq 0$
6. $\mu_i \xi_i = 0$

After following a similar process as before for the Hard SVM, we get the following `Dual Formulation` -

$$\tilde{L}(a) = \sum_i a_i - \frac{1}{2}\sum_i \sum_j a_i a_j t_i t_j x_i^T x_j$$

where, $0 \leq a_i \leq c$ and $\sum_i a_i t_i = 0$

`If` $a_i \neq 0$ `then implies not a support vector` $\implies$ away from margin
`Else If` $0 < a_i < c \implies$ support vector but on the margin
`Else` $a_i = c \implies$ inside the margin but support vector depending on value of ξ_i.

Lecture 18

Name: Harsh Sanjay Roniyar

Roll Number: 22B3942

Continuing from last lecture, these are the steps that we follow -

1. Change objective to a Lagrangian
2. Equate derivatives wrt w, b to 0
3. Substitute
4. Manipulate the loss equation
5. Rewrite the constraints as KKT Conditions

We performed the above steps in Lecture 17.

For Soft SVM, we got box constraints for the result. Classification with respect to that is as follows -

Box constraints: Four types of points

- **Inside the margin (and correctly classified)**
  - $a_n = 0$
- **On the margin (and correctly classified)**
  - $0 < a_n < C$, $\xi_n = 0$
- **Across the margin and correctly classified**
  - $a_n = C$, $\xi_n \leq 1$
- **Across the margin and misclassified**
  - $a_n = C$, $\xi_n > 1$

For Hard SVM, we will have $C = \infty$. (`More regularization means smaller C`)

The hyperparameters in this setting are -

1. . C
2. Kernel hyperparameters (K and d)

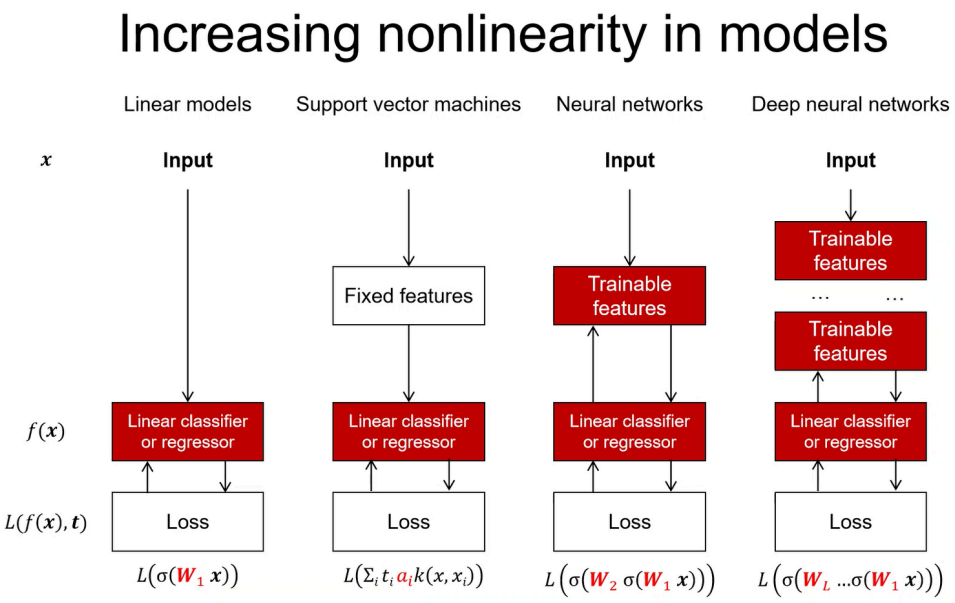
Thus, we conclude the Kernel Methods, Kernelized SVMs.

We now move to begin Neural Networks

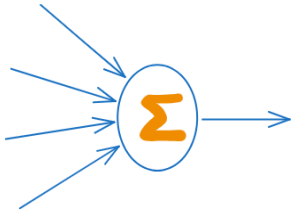
Introduction to Neural Networks

key terms : `hidden layers`, `mathematical modelling`

A very informative representation of all models in order of increasing non-linearity.



Tries to mimic the behavior of a biological neuron.



Layered structure of neuron -

$$y = w_3^{Q\times P}\sigma\left(w_2^{P\times M}\sigma\left(w_1^{M\times N}\mathbf{w}^{N\times 1}\right)\right)$$

where w_1, w_2, w_3 represent the weights for the layers 1, 2 and 3 respectively and σ is a non-linear function known as the activation function (since we want to perform non-linear modelling).

Application of chain rule for gradient update/descent is known as back-propagation.

Importance of hidden layers

The hidden layers iteratively extract the features from the previous stage of features until the desired output stage is reached.

$$f(x) = \sum_{i\in S} w_i k(x, x_i)$$

Universal Approximation Theorem

The Universal Approximation Theorem states that a neural network with at least one hidden layer of a sufficient number of neurons, and a non-linear activation function can approximate any (`Lipschitz`) continuous function to an arbitrary level of accuracy, with the approximation error controlled by the Lipschitz constant of the function being approximated.

Types of activation functions (σ)

- **Step**
- **Sigmoid**
- **ReLU** (Rectified Linear Unit)
- **Softmax**
- **tanh**
- **Linear**

> Formally, the standard (unit) softmax function $\sigma : \mathbb{R}^K \to (0,1)^K,\ \text{where } K \geq 1$, takes a vector $\mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$ and computes each component of vector $\sigma(\mathbf{z}) \in (0,1)^K$ with
>
> $$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Softmax Function - Wikipedia.

`PS: Refer to this` Neural Networks - Harsh S Roniyar `amazing link for more such content`

Lecture 19

Name: Harsh Sanjay Roniyar

Roll Number: 22B3942

`Assignment 3 has been released`

Recap:

- Hidden Layers
- Universal Approximation (Cybenko's) Theorem
- Activation Functions
  - Softmax is generalization of sigmoid for multi-class classification
  - e.g. $\text{Softmax}\left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix}$, where $0 \leq p_j \leq 1$ and $\sum_{j=1}^{c} p_j = 1$ with

$$p_j = \frac{e^{x_j}}{\sum_{i=j}^{c} e^{x_j}}$$

  - The CE loss is $\text{CE} = -\sum_{j=1}^{c} t_j \log p_j$, where t_j is the target class. This is an extension of the binary cross entropy loss discussed earlier (when $c = 2$).

Multi-Class SVM

Choice 1:

Class i vs Class j SVMs $\implies \binom{c}{2}$ SVMs

Choice 2:

Class i vs Rest SVMs $\implies c$ SVMs

$\arg\max_j \left(w_j^T x + b_j\right)$ would give us the decided class and j is the class.

Basic Structure of a Neural Network

- Output Layer
- Hidden Layer(s)

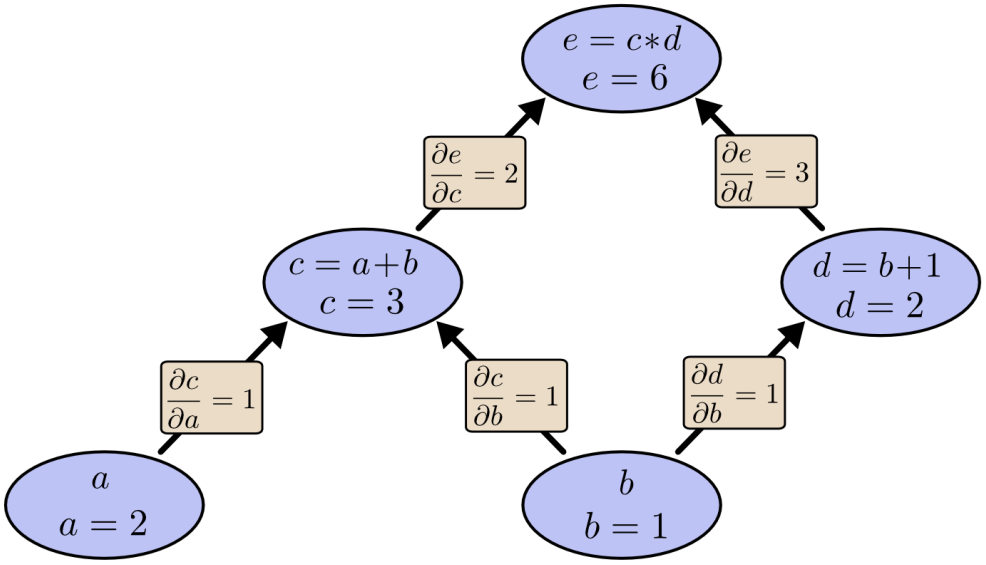
- Input Layer

Backpropagation

as a method to update the gradient weights

Chain Rule of Differentiation

Computation Graph



Source: Neural Networks - Harsh S Roniyar

Vector Valued Functions

$$f(x) = \begin{bmatrix} f_1(\bar{x}) \\ f_2(\bar{x}) \end{bmatrix} = \begin{bmatrix} f_1(x_1, x_2, x_3) \\ f_2(x_1, x_2, x_3) \end{bmatrix}$$

Jacobians

The Jacobian for a function $f(x)$ as defined above would be -

$$J(f) = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \frac{\partial f}{\partial x_3} \end{bmatrix}$$

Issues with Gradient Descent

- Need to find good step-size
- Lots of computation before each update
- Can get stuck in local minima

Summary of GD

Loop until stopping criterion

$$\text{loss } l \leftarrow 0,\ \nabla l \leftarrow 0$$

Loop over training samples:

$$l_i \leftarrow \text{loss}(\theta, x_i, t_i)$$

$$\nabla l_i \leftarrow \text{grad}(\theta, x_i, t_i)$$

$$\nabla l \mathrel{+}= \nabla l_i \ ; l \mathrel{+} = l_i$$

$$\theta \leftarrow \theta - \eta \nabla l$$

Lecture 20

Name: Harsh Sanjay Roniyar

Roll Number: 22B3942

Discussing some issues with Gradient descent, as understood in Lecture 19.

We use **Stochastic Gradient Descent** (SGD) to escape local minima and other critical points.

`SGD:`

```
loop{
        i <- random sample [N]     --- sgd

        l_i <- l(t_i, x_i, theta)  --- scalar
        grad_theta(l_i)            --- vector

        theta_new <- theta_old - eta * grad_theta(l_i)
}
until{
        stopping criterion
}
```

Batch Gradient Descent

`BGD:`

```
batch formation{
        Randomly permute training samples
        Divide training data into batches of size N/B samples
}

epoch loop{
        for each batch b in [B] {
                L_batch = sum(l_i) for i in batch
                grad_theta(L_batch) = (sum(grad_theta(l_i)) for i in batch)
/ (N/B)
                theta_new = theta_old - eta * grad_theta(L_batch)
        }
}
until{
        stopping criterion
}
```

Now, lets see how we can combine models

Combining Models

`ensemble` models

$\rightarrow$ for classification

- model averaging:
  - assumption 1: probability that any model is correct if p > 0.5
  - assumption 2: h_1, h_2, h_3 are independent of each other

nearest neighbor
splits into voronoi regions
label is the same as the data point

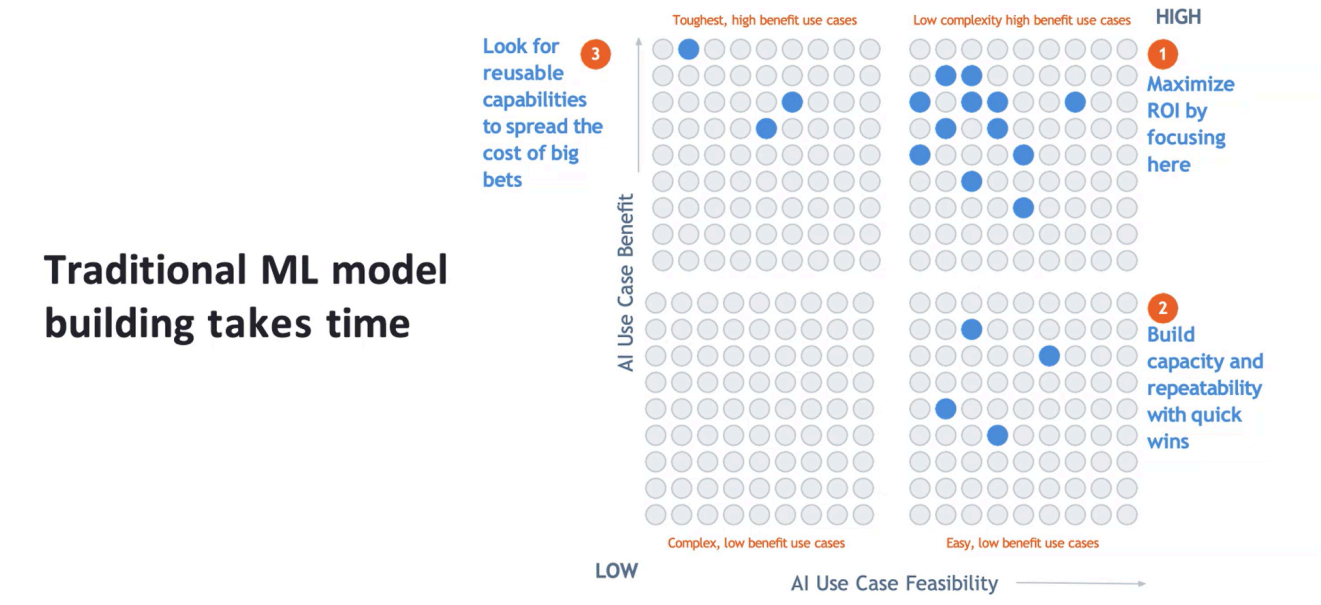
Lecture 21

Name: Harsh Sanjay Roniyar

Roll Number: 22B3942

Guest Lecture by Dr. Shreyas Chavan

AutoML (Automated Machine Learning)



What is AutoML?

> AutoML — short for "automated machine learning" — is a technology invented by DataRobot to automate many of the tasks needed to develop artificial intelligence (AI) and machine learning applications. AutoML helps accelerate the value delivery of AI.

Ref. Slides by the presenter

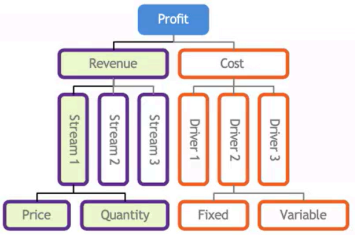
Value Estimation of a project

Exercise: Value estimation

Assess the value potential of the AI solution: Methodology

Step 1 — **Estimate the addressable base for each use case**

The addressable baseline is usually a specific driver of cost or revenue. **Make and state assumptions**, these can be refined later.



Step 2 — **Evaluate the opportunity for improvement through AI**

Estimate the potential for improvement on the baseline by:

Qualitatively assessing current practices

- Consistency of approach
- Analytical rigor
- Level of oversight

-OR- Benchmarking against best-ever performance

Sophistication of current practices	Typical improvement from AI[1]
Closely managed, well defined standard work, sophisticated analytics	0-5%
Simple business rules in place, limited management oversight	5-10%
Processes not formalized, minimal oversight	10-20%+

1 Rules of thumb based on past DataRobot experience - opportunity will vary by business domain and use case

Sizing examples

Churn

Step 1 — **Estimate the addressable baseline**

TOP-DOWN

- **$1B** — Annual revenue
- **5%** — Annual customer churn
- **$50M** — Addressable revenue base

Step 2 — **Evaluate the opportunity for improvement with AI**

QUALITATIVE

- Current churn reduction approach is manual, heuristic, without formal process
- Churn is measured, without accountability
- ⬆ **$5-10M** — Opportunity is ***high***: 10-20% of $50M base

Fraud

BOTTOMS-UP

- **$100** — Average cost of fraud transaction
- **1 out of 2** — Customers defrauded Annually
- **25M** — Customers

$1.25B Annual fraud cost base

BENCHMARKING

- **$95** — Best lowest cost of fraud achieved with early detection
- ⬇ **$63M** — Saved from lower annual fraud cost with AI

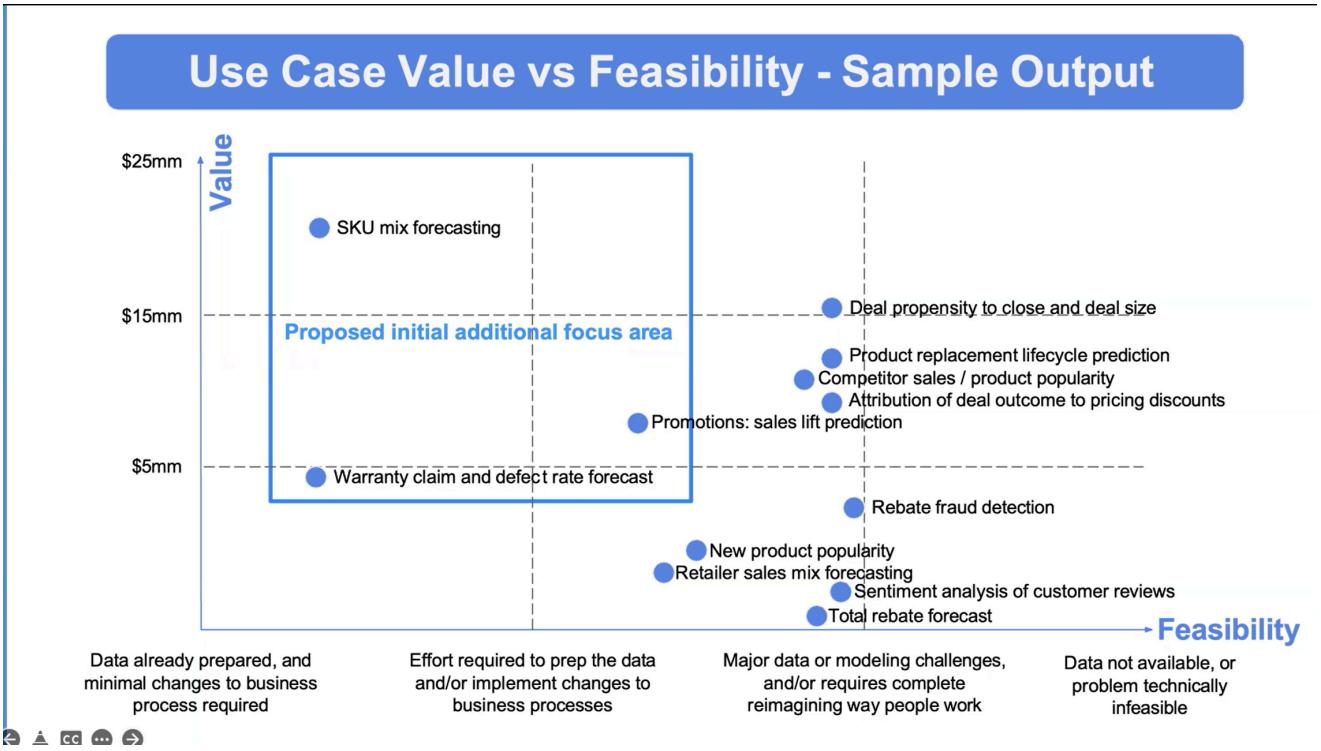
Using AI to consistently achieve best performance on all fraud transactions

Feasibility Estimation

Feasibility has multiple dimensions - many aren't "technical"

	Less feasible		More feasible
	No project champion has yet been identified	**Sponsorship** Is there an executive sponsor for this project?	There is a natural executive sponsor already identified who can champion this project and help overcome blockers
	Obtaining access to data will involve a potentially time-consuming request and/or retrieval process	**Data** Is the needed data easily accessible?	Data is readily available and primary clean; access could be granted and data shared in short order
	Alignment is still required to secure resources to execute the project	**Resourcing** Are the resources available?	The analysts, data scientists, IT, and subject matter experts needed to deliver the project could readily dedicate time as needed
	Additional steps required (e.g. optimization or business rules) to make predictions actionable	**Operationalization** What is required to make predictions actionable or insightful?	Predictions require only minimal (e.g. sorting) post-processing to be consumed
	Mindsets, behaviors, and processes will need to change for value to be achieved	**Adoption** Will people's workflows or behaviors need to change?	Predictions can be directly incorporated into existing processes with minimal changes
	Alignment is required to identify resources that will support the production model	**Monitoring** Will it be effortful to support the model once in production?	Resources have already been identified and secured for long term support of the model or model does not require continuous monitoring

Comparison



Lecture 22

Name: Harsh Sanjay Roniyar

Roll Number: 22B3942

`Assignment Deadline extended to 3rd Nov`

`Assignment Doubts Discussed`

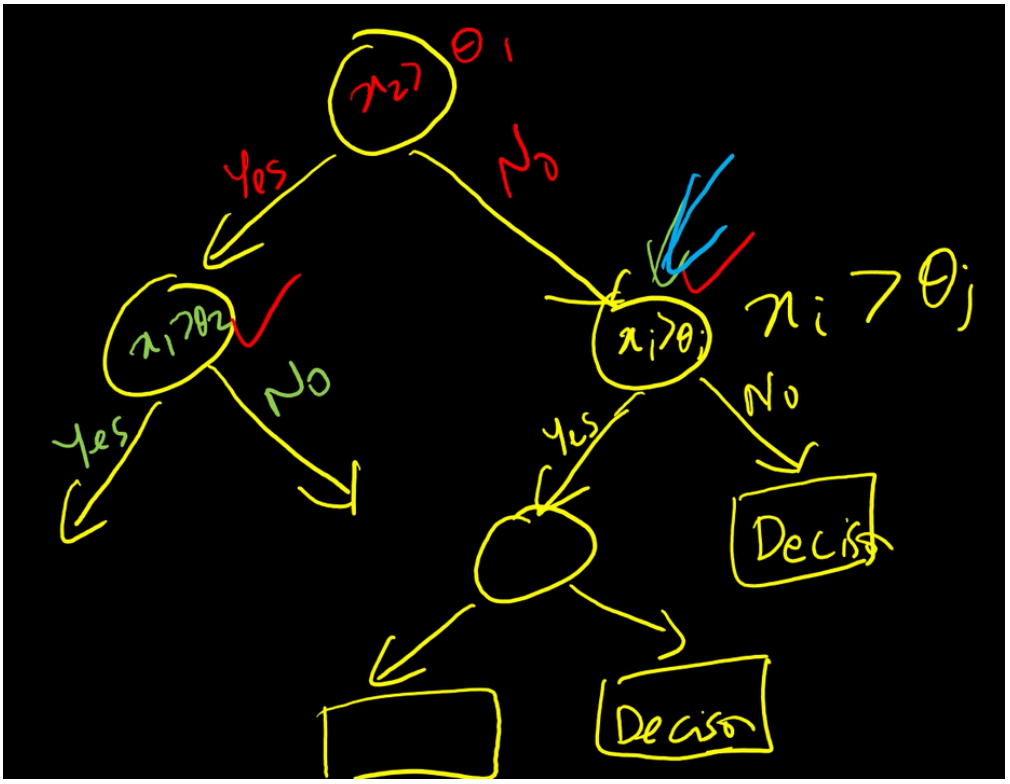
Cascade of Models (Decision Trees)

Binary Tree

Root
Internal nodes
Leaf nodes

Parent Node
Child Node

Threshold-based Decision Tree



Q1. How to choose a variable for a decision node?
Q2. How to select a threshold?

Q3. When to insert leaf nodes?

O(N) thresholds for each variables and hence O(ND) total number of thresholds, where N is the number of training points and D is the dimension.

Criteria to compare threshold classifiers

Maximal reduction in uncertainty

Greedy Tree Learning is the approach used.

- Among all thresholds evaluated, pick the one that maximally reduces the uncertainty in the data going to each of its children
- Splitting of subsets

Example: Weighted average entropy of D_1 and D_2 is minimized

Classification: $D \equiv D_0$ having label 0 or 1 with counts n_{00} and n_{01} respectively.
$|D_0| = n_0 = n_{00} + n_{01}$

The entropy is thus

$$\text{Entropy}[D_0] = -\sum_{c \in \{0,1\}} p_c \log(p_c) = -\left[\frac{n_{00}}{n_0}\log\left(\frac{n_{00}}{n_0}\right) + \frac{n_{01}}{n_0}\log\left(\frac{n_{01}}{n_0}\right)\right]$$

Criteria
Weighted average Entropy:

$$\frac{|D_1|\text{Ent}[D_1] + |D_2|\text{Ent}[D_2]}{|D_1| + |D_2|}$$

```
NodeSplit[D_i]:

If all D_i have same label then return None
Else
        For all dims n
        {
                For all pts in D_i
                {
                        Split D_i according to threshold into D_j and D_k
                        Compute wt. avg. entropy of the split D's
                }
        }
        Best combination <- arg min_{n,m} (x_n, theta_nm)
Return Best combo, corresponding D_j and D_k
```

```
TreeLearn[D_i]:
        Recursively split D_i
```

```
        Until [None] is returned
        Make LeafNode
```

Maximum number of leaf nodes = N
Maximum depth of tree = N

Balanced - Unbalanced Binary Trees

Deep trees can overfit
Regularization:

1. Stop splitting below a certain entropy
2. Prune trees beyond a certain depth

Purity/Impurity Criteria

For Classification:

1. Entropy: $-\sum_c p_c \log p_c$
2. Gini Index: $1 - \sum_c p_c^2$
3. $1 - \max p_c$
4. $p_1(1 - p_1)$

For Regression:

1. Variance of labels (target values)

Random Forest

It is an ensemble of decision trees

Randomize:

1. **Bagging**: Training on random subsets of samples. Out-of-bag samples (OOB) are not used
2. At each node consider only a random subset of features/dimensions/variables

Our hyperparameters are:

- number of trees
- number of features to consider at each node
- max depth
- percent of samples in a bag

Properties of random forests

1. **Can get good idea of generalization**
    - For each tree, identify OOB (out of bag) samples and their predicted label
    - Average OOB labels across trees
    - Converges to theoretical generalization error due to law of large numbers
2. **Guarantee against overfitting**
    - Since each sample is OOB for some trees, OOB generalization estimate gives a good idea of test accuracy
3. **Give feature importance**
    - Randomly permute a feature across samples and measure drop in accuracy
4. **Can handle both discrete and continuous variable**

`Random Forest - Brieman`

`Now, while I was planning on making text boxes in my notes like the one below, I stumbled upon this gem`

> **On the importance of sentence length**
>
> This sentence has five words. Here are five more words. Five-word sentences are fine. But several together bocome monotonous. Listen to what is happening. The writing is getting boring. The sound of it drones. It's like a stuck record. The ear demands some variety.
>
> Now listen. I vary the sentence length, and I create music. Music. The writing sings. It has a pleasent rhythm, a lilt, a harmony. I use short sentences. And I use sentences of medium length. And sometimes when I am certain the reader is rested, I will engage him with a sentence of considerable length, a sentence that burns with energy and builds with all the impetus of a crescendo, the roll of the drums, the crash of the cymbals -- sounds that say listen to this, it is important.
>
> **- Gary Provost** *(100 Ways to Improve Your Writing, 1985)*

Lecture 23

Name: Harsh Sanjay Roniyar

Roll Number: 22B3942

- Ensemble
- Bagging
- Cascade (Tree)
- Boosting

Assumption: Weak Learners

Adaboost

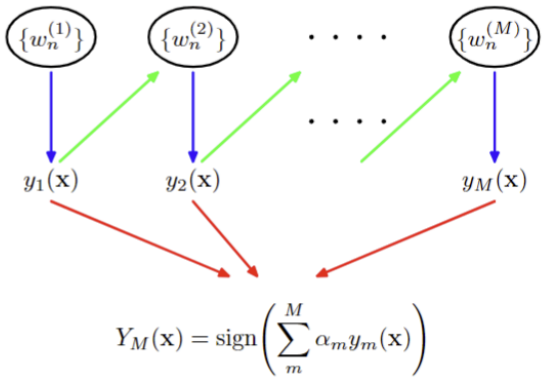


Figure 14.1 Schematic illustration of the boosting framework. Each base classifier $y_m(\mathbf{x})$ is trained on a weighted form of the training set (blue arrows) in which the weights $w_n^{(m)}$ depend on the performance of the previous base classifier $y_{m-1}(\mathbf{x})$ (green arrows). Once all base classifiers have been trained, they are combined to give the final classifier $Y_M(\mathbf{x})$ (red arrows).

> Christopher M. Bishop - Pattern Recognition and Machine Learning-Springer (2011), page 627

**The Algorithm:

1. Initialize the data weighting coefficients $\{w_n\}$ by setting $w_n^{(1)} = \frac{1}{N}$ for $n = 1, \ldots, N$.
2. For $m = 1, \ldots, M$:
   a. Fit a classifier $y_m(\mathbf{x})$ to the training data by minimizing the weighted error function

$$J_m = \sum_{n=1}^{N} w_n^{(m)} \mathbb{1}(y_m(\mathbf{x}_n) \neq t_n)$$

   where $I(y_m(\mathbf{x}_n) \neq t_n)$ is the indicator function and equals 1 when $y_m(\mathbf{x}_n) \neq t_n$ and 0 otherwise.

   2. Evaluate the quantities

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} \mathbb{1}(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}} = \frac{J_m}{\sum_{n=1}^{N} w_n^{(m)}}$$

and then use these to evaluate the model weights (`assumption:` $\epsilon < \frac{1}{2}$ `i.e. a weak learner`)

$$\alpha_m = \ln\left(\frac{1-\epsilon_m}{\epsilon_m}\right).$$

3. Update the data weighting coefficients

$$w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m \mathbb{1}(y_m(\mathbf{x}_n) \neq t_n)\}.$$

3. Make predictions using the final model, which is given by

$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(\mathbf{x})\right).$$

Adaboost is minimizing exponential error.

Overall Loss:

$$E = \sum_{n=1}^{N} \exp\{-t_n f_m(\mathbf{x_n})\}$$

where, $f_m(\mathbf{x})$ is the weighted average given by -

$$f_m(\mathbb{x}) = \frac{1}{2}\sum_{l=1}^{m} \alpha_l y_l(\mathbf{x})$$

Also,

$$w_n^{(m)} = \exp\{-t_n f_{m-1}(\mathbf{x_n})\}$$

and thus, E reduces to

$$E = \sum_{n=1}^{N} w_n^{(m)} \exp\left\{-\frac{1}{2} t_n \alpha_m y_m(\mathbf{x_n})\right\}$$

T_m: correctly classified
M_m: misclassified subset

$$\boxed{w_n^{(m+1)} = w_n^{(m)} \exp(-\alpha_m/2) \exp\{\alpha_m \mathbb{1}(y_m(\mathbf{x_n}) \neq t_n)\}}$$

Unsupervised Learning

Input: D-dim continuous

1. Clustering (Output: Discrete Brackets)
2. Dimension Reduction (Output: Continuous Numbers)

Principle:

1. Reduce data precision required in the output
2. Still be able to reconstruct the input from output.

Error Function: Reduce the reconstruction loss of the L2 Norm

Clustering

- Identify if there are any naturally visible groups, sometimes it might not be obvious and hence need mathematical ways.
- Clustering can also help in reducing data, by using only a cluster descriptor parameter for the data-points belonging to the cluster.

K-means clustering

Divides the region into K-hard partitions.

Input: $X \equiv \{x_1, x_2, \ldots, x_N\}$, K

Initialize: $c_1, c_2, \ldots, c_K$

Loop
{
$y_n \leftarrow \arg\min_k d(x_n, c_k)$
$c_k \leftarrow \frac{\sum_{n=1}^{N} \mathbb{1}\{y_n=k\} x_n}{\sum_{n=1}^{N} \mathbb{1}\{y_n=k\}}$ for all n, k
}
Until no change in cluster assignment

Loss: $\sum_n \sum_k \mathbb{1}\{y_n = k\} \|x_n - c_k\|_2^2$

Lecture 24

Name: Harsh Sanjay Roniyar

Roll Number: 22B3942

`A4 Released - Due after EndSem`

---

k-means initialization

```
Initialize c_1 to c_k randomly
Loop
{
        y_i <- arg min_k ||x_i - c_k||**2, for all i (Cluster Assignments)
        c_k <- (sum_i 1{y_i = k}*x_i)/(sum_i 1{y_i = k}), for all k
(Calculate Centroids)
}
```

`1{}` is the indicator function.

Initialization of cluster centers

1. Pick c_1 randomly from X itself
2. Pick c_2 from X itself that is furthest from c_1
3. Pick c_3 that is furthest from c_1 and c_2.
4. . . .

Fuzzy c-means

In k-means, X_i and c_j's are mapped by hard partitioning, that is x_i can belong to only one class. i.e. $w_{ij} \in \{0, 1\}$

$$w_{ij} = \mathbb{1}\{d(x_i, c_j)\} = \min_k [d(x_i, c_j)]$$

In fuzzy c-means, x_i's can belong to every class with some probability, hence soft partitioning, i.e. $w_{ij} \in [0, 1]$

$$w_{ij} = \frac{1}{\sum_{k=1}^{c} \left[\frac{\|x_i - c_j\|}{\|x_i - c_k\|} \right]^{\frac{2}{m-1}}}$$

$\lim m \to 1$ will make the weights approach k-means.
Higher the hyperparameter value of m, higher the fuzziness.

In fuzzy c-means, the cluster centers will be a weighted mean.

$$c_j = \frac{\sum_i w_{ij} x_i}{\sum_i w_{ij}}$$

Problem

With both k-means and fuzzy c-means, both depend on Euclidean distance from cluster centers. That means, both prefer hyper-spherical clusters (isotropic) of equal radii

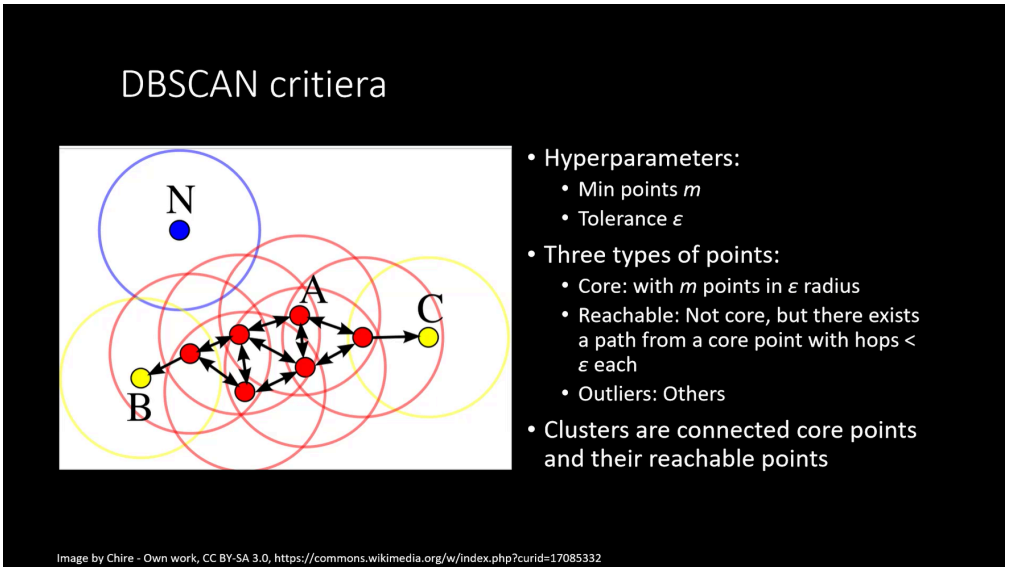
Density-based clustering

An example of which is `DB-SCAN`.

DB-SCAN

3 types of data points, clustered into

- High-density -> cores of arbitrary shapes
- Medium-density -> peripheries
- Low-density -> outlier regions



ϵ radius (`tolerance`) and m minimum points

Steps:

1. Form a graph with ϵ neighbor hard (`tolerance region`)
2. Identify core points (number of neighbors $\geq$ m)
3. Identify periphery/reachable points (non-core points, but at least one core-point neighbor, i.e. within ϵ reach)
4. Other points become the outliers

So, now we have two hyper-parameters in this problem

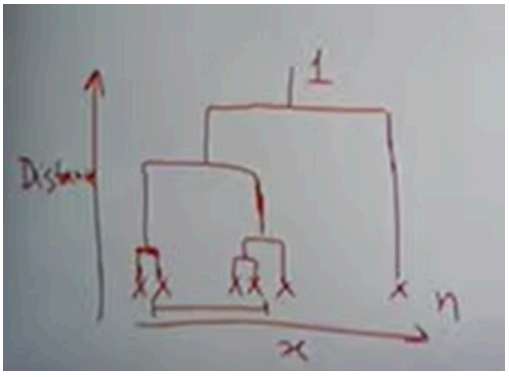
DB-SCAN Algorithm

- For each sample x_i
  - For each other sample x_j
    - Mark j as neighbor of i, if $d_{ij} < \epsilon$
    - Increment number of neighbors n_i of x_i
- For each sample x_i
  - If neighbors $\geq m$ then mark as **core**
  - If neighbors > 0 then mark as **reachable**
  - If neighbors $= 0$, then mark as **outlier**
- For each sample x_i
  - If core and un-clustered, then mark all connected samples with this cluster

Hierarchical Clustering

- Start with each sample as its singleton cluster
- For each merge iteration
  - For each cluster
    - For each other cluster
      - Compute inter-cluster distance
  - Merge two closest clusters

We get a resulting dendo-gram similar to the following



Now, what is inter-cluster distance?
Choices:

1. Distance between nearest points from the clusters: `Single-Linkage`
2. Distance between furthest: `Complete-Linkage`
3. Distance between centroids: `Average-Linkage`

Clustering Metrics

Variation explained
Low intra-cluster variation
High inter-cluster variation

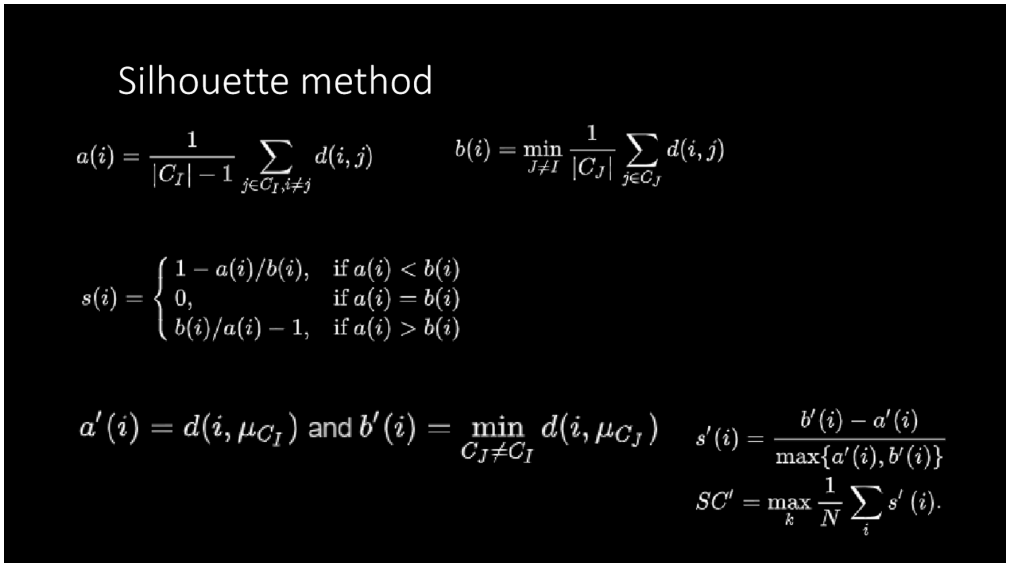
There are multiple methods for such analyses, e.g.

- Elbow method (Variation explained vs No. of clusters) - We observe an elbow at 10% from 100% variation explained, and pertaining to that we get a good number of clusters from the method.
- Silhouette method

How would we measure the variation:

- Avg. distance from centroid
- Avg. distance from all other points
- Fit an isotropic gaussian
  These methods won't work for DB-SCAN

- **Silhouette** method:



  $a(i)$ is the intra-cluster parameter
  $b(i)$ is the inter-cluster parameter
  $S = \underset{i}{\text{mean}}\ s(i)$

- **Davies-Bouldin** index:
  minimize ratio of intra-cluster versus inter-cluster variation

$$\underset{i}{\text{mean}}\ \underset{j}{\max} \frac{S_i + S_j}{M_{ij}}$$

  S_i is variation in cluster i and M_{ij} is the variation in cluster i and j combined.