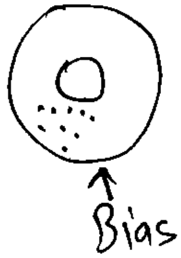
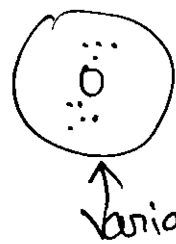
BIAS-VARIANCE TRADEOFF (supervised Learning)





Error = Bias^2 + Variance + Irreducible Error

Increasing Bias will decrease variance
Increasing variance will decrease bias.

Bias : Assumptions about the form of target function

Variance : Prediction error if different training set is used

Parametric algorithms have high bias & low variance
E.g. Linear Regression, Logistic regression, LDA

Non-parametric algorithms have low bias & high variance
E.g. Decision trees, KNN, SVM

Examples: If value of K is high in KNN → increases bias
Depth of Decision tree inc → variance increased.

Bias → under-fitting on training data (under-catching signal)

✓ Ways to address it:

1) Train longer
2) Use more ~~complexity~~ complex model
3) Add features
4) Decrease regularization

Variance → over-fitting in ~~test~~ validation/dev set (over-catching signal i.e. noise too) memorizing training data and not well "generalizing" on unseen/test data

✓ Ways to address it

1) Add more data
2) Decrease #features
3) Increase regularization.
4) Use simpler model (less complex)
5) Early-stopping

Example: Linear Regression model may ~~under~~ under-fit the training data but can generalize well in some situations (can perform well in ~~test~~ validation or test set)

Polynomial Regression model may fit the training data well but can fail to generalize in valid/test data.

<u>Questions</u>: What is bias?
What is variance?
What is bias-var tradeoff?
What is underfitting? ways to address it?
What is overfitting? ways to address it

HANDLING IMBALANCED DATASETS

1) Extract test set before applying any techniques to handle imbalance (test set can be imbalanced as data coming for scoring may be imbalanced)

(Remember: It is not "necessary" to balance classes e.g. using tree based approaches)

2) Get the evaluation metrics right (If data is imbalanced)

✓
- Not accuracy ✗
- Kappa : class accuracy normalized by imbalance ~~of~~ classes
- Precision / Recall (of both classes individually)
- AUCPR (if +ve class is more important)
- AUCROC (if both classes are important but can be optimistic for +ve minority class)

3) Fixing in Data

✓ (Making classes balanced)

- Oversampling of Minority Class
  - ✓ Random oversampling (duplicating samples)
  - ✓ SMOTE (may add noise) (Synthetic Minority Over-sampling technique)
  - ADASYN (may add noise)
- Undersampling of Majority Class
  - ✓ Random Undersampling
  - ✓ Edited Nearest Neighbor (cleans, not Undersamples)
  - Tomek links (cleans, not undersamples)

If data is fixed for balance → regular classification metrics

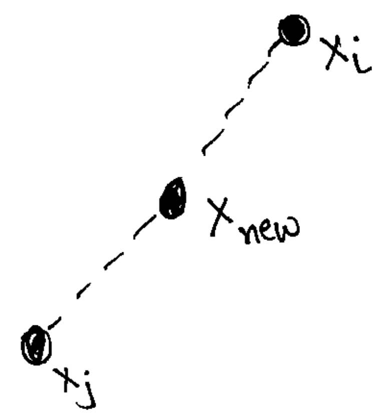
4) Letting ML Algo take care of class imbalance ✓

- One Class SVM (trains on only minority class, maj. class can be used for testing/evaluation)
- Cost Sensitive Algos: (penalizes minority class mistakes heavily)
  - custom obj/loss fn (example dependent cost)
  - specify class wt. (class-dependent cost) (assigning diff. wts to samples from diff. classes)
- Tree-based Algos: Tree based Algos generally fare well on imbalanced datasets as if/else tree addresses both classes (better than parameterized algos)

OVERSAMPLING TECHNIQUES

1) **SMOTE**: (Synthetic Minority Over-Sampling Technique)

→ For a given obs x_i, a new (synthetic) observ. is generated by interpolating between one of K-nearest neighbors of x_i, e.g. x_j



$$x_{new} = x_i + \lambda(x_j - x_i)$$
where $\lambda = [0,1]$

→ To select x_i, three options:

[random] 1) regular : randomly select x_i

[KNN] 2) borderline : Separate all x_i into 3 classes using K nearest neighbor

a) noise : all nearest-neighbors are of diff. class than x_i

b) in danger: atleast half of nearest neighbors are of same class as x_i

c) safe : all nearest neighbors are of same class as x_i

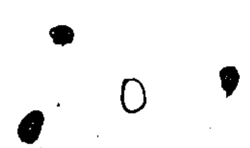
[SVM] 3) svm : Uses svm to identify samples close to decision boundary and selects x_i from these points

→ **Disadv**: Can generate noisy samples by interpolating new pts b/w marginal outliers and inliers.

Sol^n: After oversampling, use under-sampling techniques such as Edited Nearest Neighbor or Tomek links to clean up

2) **ADASYN**: (Adaptive Synthetic Sampling)

→ For a given x_i, the number of new samples generated $\propto$ number of nearby samples which do not belong to the same class as x_i



$\Rightarrow$ #new samples $\propto 3$

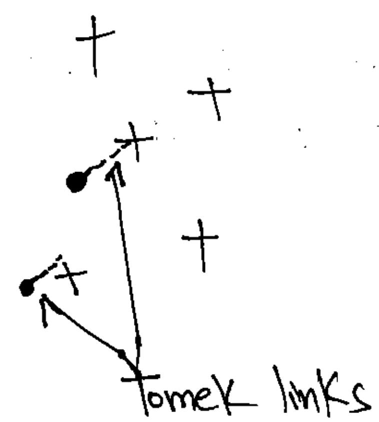
→ **Disadv**: Focusses on generating new samples which are outliers

Sol^n: After oversampling use under sampling tech such as Edited Nearest Neighbor or Tomek links to clean up

UNDERSAMPLING TECHNIQUES

1) TOMEK LINKS:

- A tomek link exists b/w 2 observations if they are of different classes and nearest neighbor of each other



- For undersampling, we remove any obs. from majority class for which tomek links are identified

- It doesn't create balance between classes, it simply "cleans" the dataset by removing noisy obs. of majority class, which may result in easier classification problem

2) EDITED NEAREST NEIGHBORS:

- For the majority class samples, K-nearest neighbors are computed and if the samples do not agree "enough" to their K-nearest neighbors, they are removed

- This simply "cleans" the noisy samples from majority class and does not necessarily "balance" the classes

Question: Ways to handle imbalanced dataset?

SPARSE DATA / FEATURES

SPARSE Vs. MISSING:

✓ — Sparse: many of the values are zero and you "know" they are zero

✓ — Missing: value is "unknown" → may be zero or non-zero

HOW TO HANDLE SPARSE DATA:

- Features with high sparsity % → remove
- Encode sparse data with some const value (can be zero) so that the ML algo will pick signals corresponding to this const. value, if any w.r.t. labels

✓ — Convert to lower & dense dimensions e.g. PCA

2. ✓ — Embeddings

3. ✓ — Use Algos that can handle sparse data e.g. Xgboost, FM/DeepFM, Wide & Deep Model

1. ✓ — Discretization (for continuous) / Feature-hashing (for categorical)

- Feature-crossing (for categorical)

ENCODING NUMERICAL FEATURES

- Normalization
- Standarization
- Bucketing / Discretization

ENCODING CATEGORICAL FEATURES

- one-hot (if #categories is less)
- Embeddings
- Feature-hashing. (multiple categories hashed to a single bucket)

Question: Ways to encode?

1. Sparse data
2. Categorical data
3. Numerical data

MISSING DATA

- Some Algorithms handle missing data (robust to missing data) e.g. tree based methods (Decision tree)

- Decide b/w Row-wise imputation or Column wise.

Column Wise

Discard { 1) Below threshold % data are missing - Discard

Impute {
2) Mean, median, mode imputation
3) Nearest neighbor fit (resource intensive)
4) Predictive modeling - ~~cannot do w/ time series data~~
5) Use a global constant
Impute & add a column to signify missing values
6) Interpolation using adjacent values (time-series data)
7) In presence of collinearity, correlated features that have high % of missing values can be deleted entirely

Row Wise

Discard { 1. Discard if below threshold %.

Impute { 2. Impute based on other predictors

- Practically, impute by any method then use Cross-validation to check if evaluation metric is not giving desired results.

- It is important to note that as soon as we impute → introducing bias

Question: How to handle missing data?

THEORY BEHIND MISSING VALUES

1) Missing completely at Random (MCAR):
The propensity of a data point to be missing is completely random i.e. it doesn't depend on any values in the dataset, missing or observed.

→ Little' test to find out MCAR

2) Missing At Random (MAR):
The propensity for a data point to be missing is not related to missing data per se, but to other observed data in the dataset.
E.g. in a dataset for a survey of wt, women are less likely to fill the values of wt → MAR (depends on other field 'sex')

→ Create a dummy var and plot other predictors against this dummy var → if relationship exists = MAR

3) Missing Not At Random (MNAR):
The propensity of a data point to be missing depends on the value of missing data itself.
E.g. in a survey of depressed individuals, the column containing depression level value missing value in → bcz of depression level itself
i.e. people who have high depression levels are more likely to leave it blank

→ No way to detect it

Bias decreases →	Mean	Regression	EM (like MLE)	MI (Multiple Imputation) – Most preferred ⇓ Python R packages
MCAR	✗	✓	✓	✓
MAR	✗	✓	✓	✓
MNAR				→ cannot detect

GENERAL STRATEGY: 0: Study relationship of missing values to other predictors & create a predictive model using those predictors

OR

1. Impute a really high or low value to the missing entry

2. Create a new column, put 1 if the original column had ~~some~~ no value else put 0

GENERATIVE MODEL : → mean imputation is seen (or median if some outliers in available data)

✓

Reason: Generative models estimate parameters from the distribution of data, imputing mean does not change the existing dist. of data

DISCRIMINATORY MODEL : → 1) Impute with a global constant value (high or low)

✓

2) Create a new derived column put 1 if original column is empty

Reason: Adding a discriminatory feature so that the model will learn the discriminatory decision boundary

OUTLIERS

When not to worry about Outliers?
- Tree based Model
- Non-parametric test

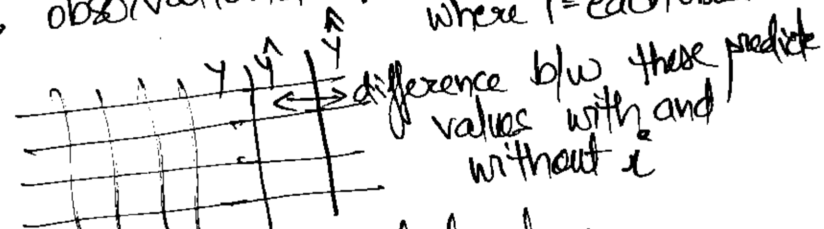
When to worry about outliers?
- Especially regression

DETECTING OUTLIERS:

1) Univariate approach (Column wise): $IQR = Q_3 - Q_1$
   1.5 times IQR below Q_1
   1.5 times IQR above Q_3

2) Multivariate (Row-wise): Compute the distance of observations from euclidean origin & plot

3) Multivariate (Row-wise) especially useful in regression: Cook's distance: Difference in predicted ~~& actual~~ values for each observations with & without i, where i = each observ.



Generally, Mean difference in predicted values for observation i > 4 ≡ outlier

Outliers package in R

HANDLING OUTLIERS:

1. 3' ✓ Use a robust error metric : Instead of MSE, use MAE
2. Unless sure about that the outlier is measurement/data-entry error, do not drop it.
3. 1' ✓ Cap the outlier
4. 2' ✓ Transforming data - log or sqrt
5. If outliers form a major chunk in data - ✓ it may point of ~~two~~ multiple underlying phenomenon
   - Model them separately
6. Compare results with and without outliers & decide from there
7. Use a robust Algo for outliers e.g. RANSAC (Random Sample Consensus)
   - It takes a subset of data that are inliers
   - Fits the model
   - Test all other data points against the fitted model and add those points that fall within user given tolerance e.g. residual threshold
   - Iterates until a user given iterations are reached or the tolerance is exceeded.

Question: Ways to handle outliers?

MULTI-COLLINEARITY

WHEN NOT TO CARE ABOUT MULTI-COLLINEARITY?

i) If its a prediction problem (and not interpretation)

ii) Model is non-linear (eg. tree based models or NN)

⇓

By defn collinear means "linear dependence" b/w predictors

WHAT IS MULTI-COLLINEARITY?

- Multi-collinearity occurs when predictors are co-related with other predictors

DIAGNOSIS / DETECTION OF MULTI-COLLINEARITY

1. Variance Inflation factor (VIF >5 or >10)

$$VIF = \frac{1}{1-R^2}$$

- VIF is defined for individual predictor variable
Conceptually, one uses the predictor as the dependent var in a regression model on all other predictors and calculates $1-R^2$ from the regression as the "usable" fraction of that predictor in the full regression model.

- If $1-R^2$ i.e. usable fraction is low e.g 0.1 then $\frac{1}{1-R^2}$ will become big

2. Standard error of coeff is large

- VIF is not natively defined for categorical data but after transforming categorical data to numerical representation e.g. one hot encoding, VIF can be applied
  - One-hot encoding introduces multi-collinearity between diff levels of the categorical data e.g. if column = sex/Gender then its value is either M or F

Gender	Gender_M	Gender_F
M	1	0
F	0	1
M	1	0
F	0	1

One-hot encoding

These 2 columns are -ve co-related since presence of one makes another absent

WAYS TO HANDLE MULTI-COLLINEARITY

1. Correlation Matrix : i) Remove one of the highly co-related var (between 2 var)
   ii) Cluster based on co-relation (coeff) (e.g. dist b/w Var A & var B = co-relation (coeff)) and take 1 from every cluster
2. L1 : will select one b/w 2 co-related var
3. PCA : will combine all predictors as orthogonal components

NOTE: Testing collinearity b/w categorical var : Chi-square test
Testing collinearity b/w continuous & categorical var : t-test (if category = 2), ANOVA (if category > 2)

<u>FEATURE SELECTION</u> WAYS

1. FILTER: Features are selected based on statistical tests e.g. information gain, correlation coeff, mutual information etc.

2. L1 Regularization

3. WRAPPER : Features are selected based on their performance in a model. E.g. feature importance

HYPERPARAMETER TUNING METHODS

1. RANDOM SEARCH:
   - We manually specify a set of possible values for each hyperparam and num_of_iterations
   - Random search "randomly" selects one value from each of the possible values of each hyperparam and builds a model and calculates score (offline metric)

     This is done num_of_iterations times and the combination of hyperparams which yields the best score is selected
   - Not all combinations of hyperparam values are searched, limited to num_of_iterations

2. GRID SEARCH
   - We manually specify a set of possible values for each hyperparam
   - Grid search "exhaustively" selects all the combination of hyperparam values, builds model, calculates score and picks the combination of hyperparam values that yields the best score
   - All combinations of hyperparams are searched exhaustively i.e. it is time-consuming and becomes impractical for large number of hyperparams

3. BAYESIAN ('Hyperopt' package uses Bayesian approach)

- Issue with Random & Grid Search:
  1. Each time we try diff hyperparams, we have train the model, make predictions on validation and then calc. metric i.e. evaluating the objective fn to find the score is extremely expensive
  2. Grid & Random Search are completely "uninformed" by past evaluations

- Sol^n in Bayesian:
  1. Uses a "surrogate" model that maps hyperparam values to score i.e. P(score | hyperparams)
     The "surrogate" model is much easier to optimize ~~xxxx~~
  2. Keeps track of past evaluation results to "inform" the next set of hyperparams to try (explore-exploit)

- Steps:
  1 → Build a "surrogate" prob. model of the true objective fn
  2 → Find hyperparams that perform best on this "surrogate" model
  3 → Apply these hyperparams to the true obj. fn
  4 → Update the surrogate model based on results from step #3
  5 → Repeat steps #2-#4 until max iteration is reached

- Bayesian approach can find better hyperparams in less time bcz they take into account past evaluations

NORMS

Norm of a Vector:

- measures the size/length of vector
- measures the distance from origin to the point x
- maps vectors to non-neg. values (scalar) (x being the vector)
- Mathematically: L^p norm

$$\|x\|_p = \left(\sum_i |x_i|^p\right)^{1/p} \quad \text{for } p \in \mathbb{R},\ p \geq 1$$

NORM:
- A norm is a function from vector space to non-neg. real numbers that behave like the distance from the origin
- A norm is any function f that satisfies:

1) $f(x) = 0 \Rightarrow x = 0$ (definiteness)
2) $f(x+y) \leq f(x) + f(y)$ (triangle ineq.)
3) $\forall \alpha \in \mathbb{R}$, $f(\alpha x) = |\alpha| f(x)$ (Absolute homogenity)

   (α ← scalar)

4) $f(x) \geq 0$ for all x (Non-negativity)

L1 norm: – Let $x = [1, 2, 3]$

$$\|x\|_1 = 1 + 2 + 3 = 6$$

- It is the Manhattan dist. from origin to point identified by x
- L1 norm is commonly used in machine learning when the difference between zero and non-zero elements is very imp. Every time an element of x moves away from zero by ϵ, L1 norm increases by ϵ

L2 Norm :- Let $x = [1, 2, 3]$

$$\|x\|_2 = \sqrt{1^2+2^2+3^2}$$
$$= \sqrt{14}$$

- Also called Euclidean norm
- It is euclidean distance from origin to point identified by x

- L2 norm is not preferred in settings where it is important to discriminate b/w elements that are exactly zero and elements that are small but non-zero. reason → L2 norm increases very slowly near origin

e.g. in above if $x = [0.9, 2, 3]$

$$\|x\|_2 = \sqrt{2^2+3^2+(0.9)^2}$$
$$= \sqrt{13+(0.9)^2}$$

↑ very small diff to $\sqrt{14}$

For L1 : $[1,2,3] \rightarrow 6$ ↕ bigger diff
$[0.9,2,3] \Rightarrow 5.9$

- The squared L2 norm is more convenient to work with mathematically and computationally than L2 norm itself. E.g. the derivative of squared L2 norm w.r.t. each element of x each depend only on the corresponding element of x while all of the derivates of L2 norm depend on entire vector
- Squared L2 norm can be calc. with vector operation $x^T x$

L0 Norm:

- Count of non-zero elements e.g $[0,3,0] \to 1$
- Not technically a norm since scaling the vector by α does not change the number of non zero entries

MAX NORM: (L^∞ norm)

- max abs. magnitude of element in vector e.g. $[1,9,3] \to 9$

FROBENIUS NORM

- Used in context of matrix, not vector
- $\|A\|_F = \sqrt{\sum_{i,j} A_{ij}^2}$
- Analogous to L2 norm but for matrix
- Intuitively, if the matrix is rolled into vector (1d) then euclidean norm of this vector = Frobenius Norm

REGULARIZATION USING L1/L2

Lasso Ridge

→ To solve over-fitting

→ Types:
- → Lasso (L1)
- → Ridge (L2)
- → Elastic.Net (linear combination of L1 & L2)

→ Penalizes complexity of model by adding a penalty term

→ The penalty term in a way acts as a "bias"
(For given error, increasing bias reduces variance which in turn reduces overfitting)

→ Works with any parametric algo (e.g. neuralnet)

L1 Regularization:

$$Loss_{L_1} = \underbrace{\Sigma (y_i - \hat{y}_i)^2}_{SSE} + \underbrace{\lambda \Sigma |\beta_i|}_{\lambda \text{ times sum of abs. value of coeff}}$$

will depend on β_i

λ ← hyperparam

→ L1 regularization shrinks some parameters to zero: feature selection

→ Increasing λ will cut features (coeff.) one by one until no variable remains

L2 Regularization:

$$\text{Loss}_{L2} = \underbrace{\sum (y_i - \hat{y}_i)^2}_{\text{SSE}} + \lambda \underbrace{\sum \beta_i^2}$$

$\hat{y}_i$ → will depend on β_i

λ times sum of square of coeff.

λ → hyperparam

→ L2 regularization will force the parameters (coeff.) to be relatively small / shrink (by increasing λ)

→ Practically performs better at prediction than L1

→ L2 is preferred over L1 when
  i) prediction performance is concerned and
  ii) Features have multi-collinearity
  When two predictors are highly co-related L1 simply picks one of them leaving other. However in L2, it keeps both of them and jointly shrinks the corresponding coeff

ELASTIC NET:

$$\text{Loss}_{\text{elastic net}} = \sum (y_i - \hat{y}_i)^2 + \alpha \underbrace{\text{Ridge Penalty}}_{\lambda |\beta_i|^2} + (1-\alpha) \underbrace{\text{lasso Penalty}}_{\lambda |\beta_i|}$$

SELECTING λ & α: Cross-Validation

WHY L1 ENCOURAGES ZERO COEFF. BUT L2 SHRINKS COEFF

Multiple ways of explain:

1) Using loss function optimization
2) Using contour plot

i) USING LOSS FUNCTION OPTIMIZATION:

L2:

Consider a model with single coeff β, then

$$L_2 = (y - x\beta)^2 + \lambda\beta^2$$
$$= y^2 - 2xy\beta + x^2\beta^2 + \lambda\beta^2$$

To minimize this eq., take derivative w.r.t. β and equate to zero (to get coeff's optimal value)

$$\frac{\partial L_2}{\partial \beta} = 0$$

$$\Rightarrow \frac{\partial (y^2 - 2xy\beta + x^2\beta^2 + \lambda\beta^2)}{\partial \beta} = 0$$

$$\Rightarrow 0 + (-2xy) + 2x^2\beta + 2\lambda\beta = 0$$

$$\Rightarrow -2xy + 2x^2\beta + 2\lambda\beta = 0$$

$$\Rightarrow \beta(x^2 + \lambda) = xy$$

$$\Rightarrow \beta = \frac{xy}{x^2 + \lambda}$$

To make $\beta = 0$, $\lambda \to \infty$ (since it is addition). And $\therefore$ β will be as low as possible but will not become 0

L1:

$$L_1 = (y - x\beta)^2 + \lambda|\beta|$$
$$= y^2 - 2xy\beta + x^2\beta^2 + \lambda|\beta|$$

To minimize, take derivative w.r.t. β and equate to 0
(to get optimal value of β)

For demonstration purpose, let $\beta > 0$

$$\Rightarrow \frac{\partial L_1}{\partial \beta} = 0$$

$$\Rightarrow -2xy + 2x^2\beta + \lambda = 0$$

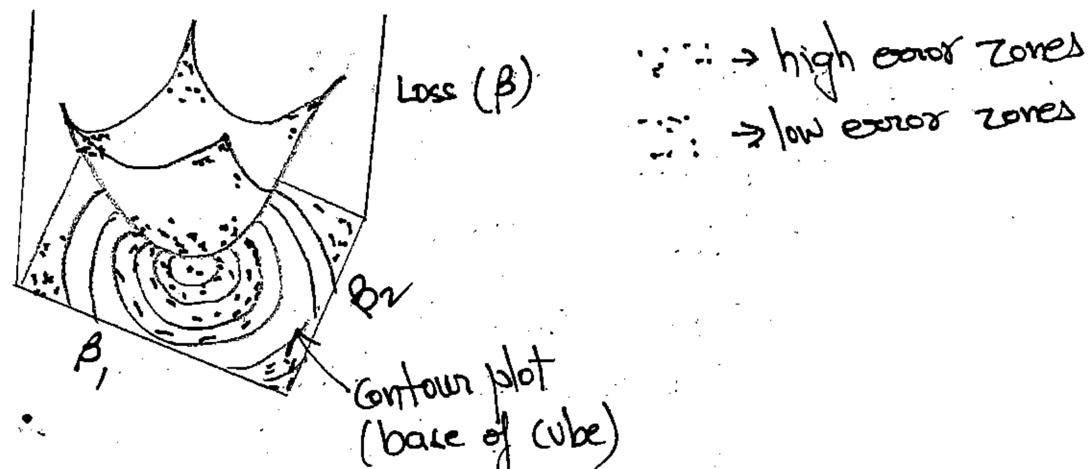
$$\Rightarrow 2x^2\beta = 2xy - \lambda$$

$$\Rightarrow \beta = \frac{2xy - \lambda}{2x^2}$$

To make $\beta = 0$, anytime $\lambda = 2xy$, the condition will be satisfied
& this can happen a lot

ii) USING CONTOUR PLOT:

Gradient Descent represented as Contour plot



L2 ~~Reg~~ Regularization Term: $\lambda \beta^2$

- For 2 variables/features, this will translate to $\beta_1^2 + \beta_2^2$
- For illustration/simplicity, let's take $\lambda = 1$

$\therefore$ Reg. term for L2 = $\beta_1^2 + \beta_2^2$ (if $\lambda = 1$)

L1 Regularization Term: $\lambda |\beta|$

- For 2 var/features, this will translate to $\lambda(|\beta_1| + |\beta_2|)$
- For simplicity/illustration, let's take $\lambda = 1$

$\therefore$ Reg. term for L1 = $|\beta_1| + |\beta_2|$ (if $\lambda = 1$)

- Error/Loss = bias^2 + variance + irreducible error

To reduce variance / over-fitting for a given/constant error
$\Rightarrow$ bias needs to be increased

In fact, that's what the regularization terms are doing
i.e. increasing bias

$\therefore$ Regularization Term = Bias term

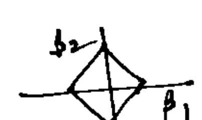
- Regularization parameter λ:
    - Variance depends on the wts (β's)
    - Bias depends on the wts (β's) as well
    - Now since we are trying to reduce variance by increasing bias (both of which depend on β's), we have to add an additional parameter that can regulate the size of bias term.

    - This regularization param is a hyper param else gradient descent will set it to 0 and travel to the global minimum.
      Hence, control of λ cannot be given to gradient descent and needs to be kept out

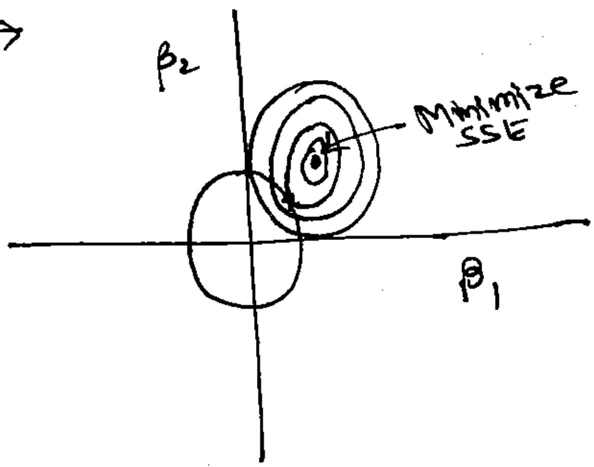
- If bias is constant/same (i.e. comb. of β_1 & β_2 that generate the same bias)

L2: $\beta_1^2 + \beta_2^2$ = constant $\rightarrow$ Circle of radius constant

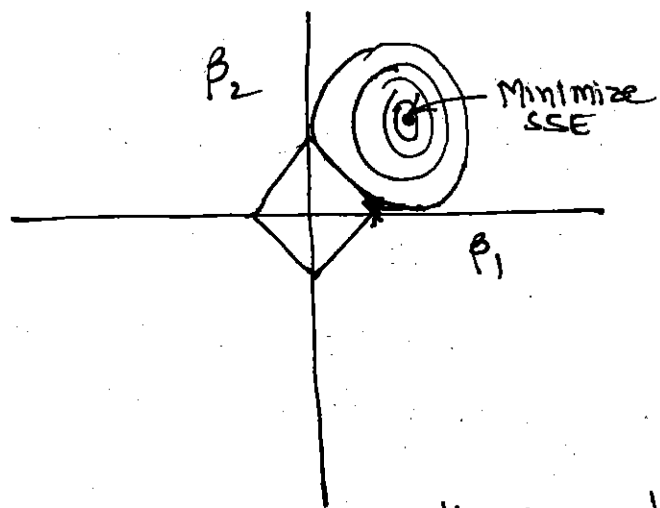
L1: $|\beta_1| + |\beta_2|$ = constant $\rightarrow$ Diamond



→



L2 Norm with GD Contour



L1 Norm with GD contour

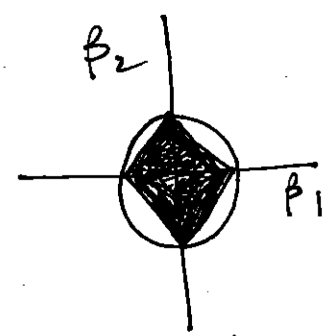
2 Forces at play here:

i) Bias term pulling β_1 & β_2 to lie on circle (L2) or diamond (L1)

ii) GD trying to travel to global minimum of cost fn (without bias term) indicated by green dot

Both the forces pull and finally settle near the point of intersection indicated by 'black cross' (x)

X → For the same amount of bias term generated, the area occupied by L1 norm is smaller and closer to axes compared to L2 norm.
This is what causes the point of intersection b/w L1 norm & GD contour to converge / intersect on the axes leading to feature selection



L1 & L2 Norm

DIFFERENCE BETWEEN L1 & L2 REGULARIZATION:

- L1 reg. penalizes the sum of abs. values of wts
  whereas
  L2 reg. penalizes the sum of squares of wts

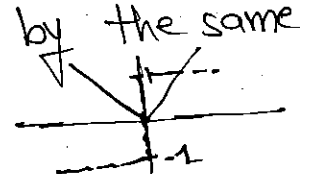
- L1 reg. solution is sparse
  whereas
  L2 reg. solution is not sparse

- L1 reg. performs feature selection
  whereas
  L2 reg doesn't perform feature selection (shrinks wts. close to zero, not zero)

- L1 reg. is robust to outliers
  whereas
  L2 is not robust to outliers

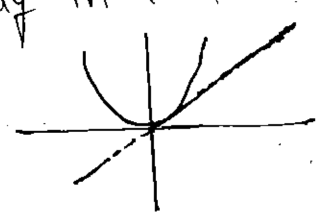
WHY L1 make wts. sparse but L2 does not?

- Gradient of L1 is -1 or 1 (except where the wt. is zero) so the penalty moves the value closer to zero by the same increment, regardless of wt's current value



  whereas

- Gradient of L2 is a linear fn that decreases to zero as the wt. approaches zero, resulting in a penalty that diminishes as the wt. value becomes smaller



Which is better L1 or L2?

- L1 is more robust : L1 takes abs. value of wts so the cost of outliers only increases linearly
  whereas L2 takes square of the wts. so cost of outliers increases exponentially

- L2 is computationally less expensive : L2 takes square of wts, has closed sol^n and can be solved in terms of matrix math
  L1 takes abs. value of wts, has no closed form sol (non-differentiable) and cannot be solved in terms of matrix math

LIKELIHOOD & PROBABILITY:

PROBABILITY:

$$\text{Reality Model Population} \xrightarrow{\text{Probability}} \text{Data sample}$$

e.g. What is the chance of observing particular data or sample given a specific model or population (whose parameters we know)

Knowing parameters → Prediction of outcome

Prob. (data | distribution)

LIKELIHOOD:

$$\text{Reality Model population} \xleftarrow{\text{Likelihood}} \text{Data sample}$$

e.g. Given observed data, what is the chance that a given reality or model is true?

If you observe x, what is the best most likely distribution possible with its parameters?

Observation of data → Estimation of parameters

L (distribution | data)

$$L(\theta|x_i) = \prod_{i=1}^{n} P(x_i|\theta)$$

⇒ The likelihood of true parameters being a certain value given data

= Prob. of observing the data given some true parameter values.

⇓

probability of observing $x_1, x_2, \ldots x_n$ given θ

$P(x_1, x_2, x_3 \ldots x_n|\theta)$ = If samples are independent $P(A \text{ and } B) = P(A) \cdot P(B)$

$$= P(x_1|\theta) \cdot P(x_2|\theta) \cdot \ldots$$

$$= \prod_{i=1}^{n} P(x_i|\theta)$$

MLE (MAXIMUM LIKELIHOOD ESTIMATION)

- Maximizing the likelihood fn i.e. Max. $L(\theta|x_i) = \prod f(x_i|\theta$
- Find the parameter values that make the observed data most likely.
- E.g. if we assume the pop. is normal then given data what is the best estimate of μ and σ^2 (parameters of normal distribution)
- To make a new prediction, we simply evaluate pdf using the best parameters found
- The pop. can be binomial, gaussian, exponential etc.
- MLE does not tell "how" to find the optimal value of θ
  → it just tells how one value of θ is "more likely" than other
  - optimization algo like GD tell "how" to find this optimal value

EXPECTATION MAXIMIZATION:

- EM is a technique to compute MLE
- EM is MLE with hidden states
- EM can work with any distribution

MLE defines the objective function whereas EM solves it in iterative way.

GAUSSIAN MIXTURE MODELS:

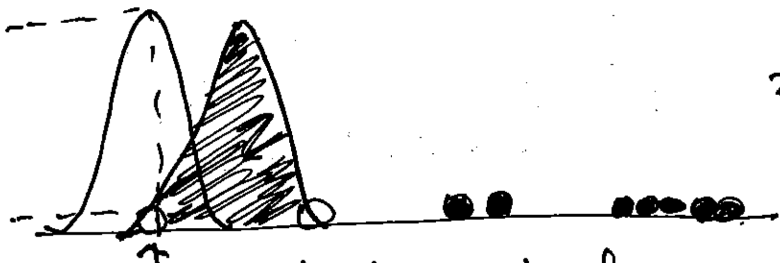
- Hint for using Mixture model: data looks multi-modal i.e. there is more than one peak in the dist. of data

- probabilistic model for representing normally distributed subpopulations
- Mixture models in general don't require knowing which subpopulation a data point belongs to allowing the model to learn the subpopulations automatically
- Unsupervised

EM & GMM

EM is used to estimate the parameters of GMM

1) Start with arbitrary distributions parameters (μ, δ^2) for a given no. of distributions



2) Compute the likelihood that each distribution produced that data point → Expectation step

This point has higher likelihood of belonging to white distribution (its pdf)

Likelihood of white = pdf on white gaussian distribution

Likelihood of black = pdf on black gaussian distribution

Total likelihood = Likelihood of white + Likelihood of black

black wt = likelihood of black / Total likelihood

white wt = likelihood of white / Total likelihood

3) Now take all data points having black portion & compute new μ & δ^2 → Maximization step

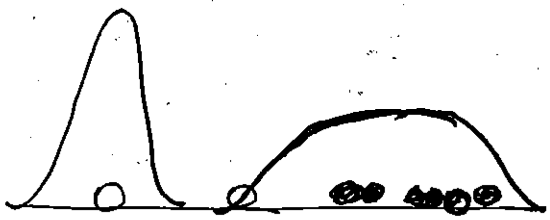
$$\mu = \frac{\Sigma(wt \times \text{data point})}{\Sigma wt}$$

$$\delta^2 = \frac{\Sigma(wt \times (\text{data pt} - \mu))}{\Sigma wt}$$

Similarly do it for white data points

4) The distribution will shift

5) Now remove/forget what portion of which data point belonged to which distribution i.e. start fresh with these updated distributions & unassigned data points



6) Again calculate likelihood of each data point with respect to updated distributions
i.e. Repeat steps 2 & 3 until a threshold of pdf or likelihood is reached

RELATION TO K-MEANS:

K-Means performs "hard" assignment of data points to clusters, in which each data pt is associated uniquely with one cluster
EM algo makes a "soft" assignment of data pts to all clusters probabilistically

---

BAYESIAN or MAXIMUM A POSTERIORI (MAP):

- We have some knowledge of data (prior)
- There is no point estimate which best explains the data instead there are multiple values of a parameter e.g. μ & σ^2
- This gives us multiple models with the same prior
- To predict a new example/data point, we have to compute "weighted sum" of these predictions

MLE & MAP

→ Both MLE & MAP are used to estimate some variable in the setting of probability distributions.

→ The likelihood fn, in case of MLE

$$\theta_{MLE} = \underset{\theta}{\operatorname{argmax}} \prod_i P(x_i|\theta)$$

θ → parameter we want to infer

x_i → data

Taking ~~log~~, as each $P(x_i|\theta)$ is a no. between 0 and 1 and multiplying them together when no. of x_i's are large (approach to infinity)

→ will result in underflow

$$\theta_{MLE} = \underset{\theta}{\operatorname{argmax}} \log \prod_i P(x_i|\theta)$$

$$= \underset{\theta}{\operatorname{argmax}} \sum \log P(x_i|\theta) \quad (1)$$

$\prod$ → product changed to Σ → summation bcz of log

$\log xy = \log x + \log y$

→ The ~~Posterior~~ fn, in case of MAP

Using Bayes Rule:

$$\underbrace{P(\theta|x)}_{\text{Posterior}} = \frac{P(X|\theta)\,P(\theta)}{P(X)}$$

ignoring the normalization constant in denominator

Maximizing

$$\propto \underbrace{P(X|\theta)}_{\text{likelihood}}\,\underbrace{P(\theta)}_{\text{Prior}}$$

$\therefore \theta_{MAP} = \underset{\theta}{\operatorname{argmax}} \prod_i P(x_i|\theta)\, P(\theta)$

Taking log,

$= \underset{\theta}{\operatorname{argmax}} \log \prod P(x_i|\theta)\, P(\theta)$

$= \underset{\theta}{\operatorname{argmax}} \sum \log P(x_i|\theta)\, P(\theta)$ (2)

Comparing (1) and (2), the only difference is inclusion of prior $P(\theta)$ in MAP.

$\Rightarrow$ This means that the likelihood is now weighted with some wt coming from prior

$\therefore$ MLE is a special case of MAP

$\rightarrow$ If $P(\theta)$ = constant or uniform [but not some distribution fn like gaussian where depending on the region of distribution, probability is high or low i.e. never always the same] Then we can ignore the constant term

$\theta_{MLE} = \theta_{MAP}$

$\rightarrow$ MAP equivalent to Ridge regression when i) weight prior $P(\theta)$ Normal ii) errors belong to Normal dist.

NOTE: It is important to remember that MLE or MAP assumes samples/observations are independent then only

$P(x_1, x_2, x_3 \ldots x_n|\theta) = P(x_1|\theta) \cdot P(x_2|\theta) \ldots P(x_n|\theta) = \prod_i P(x_i|\theta)$

NOTE: If you have some idea about the parameter that you are estimating $\rightarrow$ use MAP with the prior where prior = idea that you have abt the parameter.

NOTE: Maximizing log likelihood = Minimizing deviance = -2 ~~log~~-likelihood fn = Minimizing squared error (if the errors belong to normal dist)

→ Many of the penalized (regularization) max. likelihood techniques.

||

MAP with certain parameter priors

i) Quadratic wt. decay (shrinkage, L2) → Gaussian prior

ii) Absolute wt. decay (lasso, L1) → Laplace prior

GENERALIZED LINEAR MODELS

- extension of linear regression
- linear regression assumes the error terms/residuals ~~outcome var. Y,~~ are normally distributed

  vs

  Generalized linear models assume error terms/residuals ~~the outcome var. Y,~~ are not normally distributed ~~is generated from a particular dist. in exponential family (e.g. normal, poisson, binomial etc.)~~

- GLM generalizes linear regression by allowing the linear model to be related to the response var. via a "link function"

- "Linear" bcz : predictors affect the dist. of outcome only thru linear comb. of $x_i^T\beta$

- **Link fn :** (usually non-linear) Link fn transforms this linear comb. of predictors into outcome's space e.g. logistic fn

- Example: Logistic Regression, Linear Regression

- Linear Regression may be viewed as a special case of generalized linear model with identity link ~~and responses normally distributed~~ and error terms are normally distributed

GENERATIVE Vs DISCRIMINATIVE MODELS

GENERATIVE	DISCRIMINATIVE
→ learns the joint probability distribution $p(x,y)$	learns the conditional probability distribution $p(y\|x)$

Example: (1,0) (1,0) (2,0) (2,1)

$p(x,y)$:

	$y=0$	$y=1$
$x=1$	$2/4$	0
$x=2$	$1/4$	$1/4$

$$= \frac{\text{No. of times } (x,y) \text{ appears}}{\text{Total no. of } (x,y) \text{ pairs}}$$

$p(y|x)$:

	$y=0$	$y=1$
$x=1$	$2/2$	$0/2$
$x=2$	$1/2$	$1/2$

$$= \frac{\text{No. of times } (x,y) \text{ appears}}{\text{Total no. of } (x) = \text{given value}}$$

→ Models how the data was generated in order to categorize a signal

Does not care how the data was generated, tries to find the "discrimination" between classes.

→ E.g. Naive Bayes
GMM

E.g. Linear Regression
Logistic Regression
Decision Trees
SVM
Neural Networks

We are only intrested in $p(y|x)$

NOTE: $p(x,y)$ can be transformed into $p(y|x)$ using Bayes conditional probability rule:

$$p(x,y) = p(y|x) \cdot p(x)$$

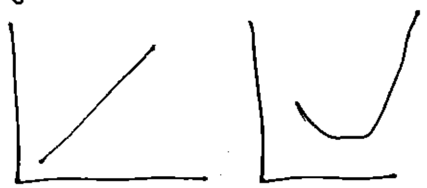
but $p(x)$ calculation can be an extra step and irrelevant for the task in hand so directly calculate $p(y|x)$

LINEAR Vs NON-LINEAR MODELS

Linear:

→ Linear in "parameters" (not predictors)

→ E.g. $y = \beta_0 + \beta_1 x_1 + \beta_2 x_1 x_2$

→ 

→ Decision boundary is linear in case of classification

→ E.g. Linear Regression
Logistic Regression

Non-linear:

everything else

E.g. $y = e^{ax}$

Tree based Models
Neural networks (bcz of non linear activation fns)

PARAMETRIC Vs NON-PARAMETRIC MODELS

Parametric:

→ No. of parameters are finite

→ E.g. Linear Regression
Logistic Regression
Linear SVM

→ "There is a distribution that the data follows"

Non-parametric:

No. of parameters are infinite
i.e. complexity of model grows with training data

E.g. KNN, Decision Trees, RBF Kernel SVM, Neural networks

→ "Distribution-free model"

CLASSIFICATION METRICS

		Actual 1	Actual 0
Predicted	1	TP	FP
	0	FN	TN

1) Accuracy : $\frac{(TP+TN)}{TP+FP+TN+FN}$

2) Precision : $\frac{TP}{TP+FP}$: If your classifier predicts 1, how likely it is to be true

3) Recall/Sensitivity/TPR : $\frac{TP}{TP+FN}$: Out of all the +ve classes, how much did your classifier catch

4) F1 score : $\frac{2}{\frac{1}{P}+\frac{1}{R}} = \frac{2PR}{P+R}$

Why harmonic mean, not simple average?

1) If $P = 1$ & $R \approx 0$

100	0
10,000	Some value

Avg = 0.5
which is not good, as there are lots of FN

2) Since numerator is same for both precision and recall, it makes sense to take avg. of denominator

3) Harmonic mean encourages similar values for precision and recall ie. more the precision and recall deviate from each other, the worse the harmonic mean

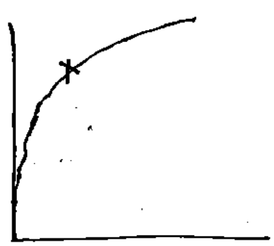
5) Specificity : $\frac{TN}{TN+FP}$

6) FPR : 1 - specificity = $\frac{FP}{TN+FP}$

7) TPR : $\frac{TP}{P} = \frac{TP}{TP+FN}$

ROC vs PR

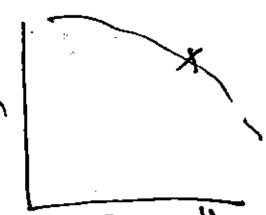
ROC Curve



TPR

FPR

PR Curve



Precision

Recall

Point that maximizes area under ROC curve → top left
[tpr − (1−fpr) ≈ 0]

Point that maximizes area under PR curve → top right
[precision − recall ≈ 0]

Both are derived from confusion matrix by varying the thresholds, so every point on ROC curve has corresponding point on PR curve

→ Recall = TPR
so the diff is FPR & Precision

PR curve must be preferred over ROC curve when:

1) only concerned with +ve class (no TN in PR curve!!)

→ ROC curve is immune to imbalanced data (doesn't change)

2) Class imbalance : If +ve class is less in number and -ve class dominates in dataset

→ FPR is negative-class only metric (all -ve instances)
i.e. change in FP ∝ change in FP+TN

$FPR = \frac{FP}{FP+TN}$ → large change in FP, FPR will change only slightly

→ Thus, if underlying data dist. changes, ROC doesn't change in imbalanced datasets

but Precision : $\frac{TP}{TP+FP}$ → will change dramatically

ROC Curve

TPR & FPR are defined at every threshold

ROC Curve is ~~plotted~~ plotted by calculating TPR & FPR at every threshold

Interpretation (AUC): probability that a random +ve is assigned a higher score than a random -ve

AUCROC is used to compare classifiers independent of threshold

Classifier with no discrimination ability b/w +ve & -ve class will have ROC Curve = diagonal (even in imbalanced datasets)

PR Curve

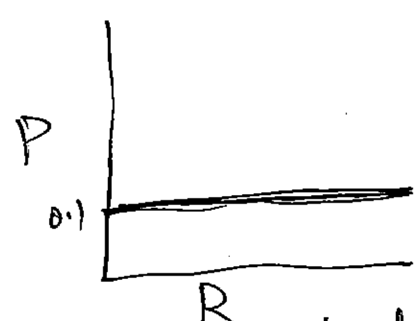
P & R are defined at every threshold

PR curve is plotted by calculating P & R at every threshold

Interpretation (AUC):
Average precision, where the avg. is taken across all thresholds

AUCPR is used to compare classifiers independent of threshold

Classifier with no discrimination ability b/w +ve & -ve class will have PR curve:
= horizontal line at ht. proportional to +ve cases in dataset



(e.g. 10% +ve cases, 90% -ve)

(For balanced datasets = 0.5)

UNDERSTANDING THE DIFFERENCE B/W ROC & PR Curves

ROC Curve uses : TP, TN, FP & FN i.e. all elements of confusion matrix. ⇒ ROC curve focusses on both classes

PR Curve uses : TP, ~~FN~~ & FP i.e. one element FN is conveniently left out off confusion matrix ~~&~~ -ve class is in abundance for imbalanced datasets ⇒ PR curve focusses on "minority" class

Even if distribution of +ve class & -ve class changes (i.e. proportion of +ve to -ve instances), ROC AUC does not change as it employs all elements of confusion matrix and use TP rate & FP rate which is strict columnar ratio in confusion matrix so do not depend on class distributions.

However, when class distribution changes, PR AUC changes as it does not take into account **FN**

It is possible that in imbalanced datasets, ROC AUC is high but PR AUC is low since PR AUC only focusses on "minority" class.

Note that: PR curve & ROC curve have one axis in common i.e. TPR = Recall

RANKING METRICS

1. PRECISION @ K $= \dfrac{\text{Number of recommended item @K that are relevant}}{\text{Number of recommended items @K}}$

E.g.

Rank=1 2 3 4 5

— Relevant
— Not relevant

$P@1 = \frac{1}{1}$.

$P@2 = \frac{1}{2}$

$P@3 = 1/3$

$P@4 = 2/4$

$P@5 = 3/5$

- Disadv : Fails to take into account the relative ordering of relevant docs in top K (order unaware metric)
  e.g. in above example, first 3 items could be relevant and last 2 could be not relevant but still P@K would be 3/5

- Range [0,1]
- Mostly used for binary relevance. Can be adapted to graded relevance (i.e. non-binary relevance, e.g. relevance score) by thresholding and converting to binary relevance

2: RECALL@K = $\dfrac{\text{No. of recommended item@K that are relevant}}{\text{Total no. of relevant items (in the recommended list since we are mostly interested in recommending top-K items)}}$

✓ Rank = 1 2 3 4 5

E.g.

- relevant
- not relevant

Recall@1 = $\frac{1}{3}$

Recall@2 = $1/3$

Recall@3 = $2/3$

Recall@4 = $3/3$

Recall@5 = $3/3$

- Disadv: 1) Fails to take into account relative ordering of relevant docs in top K (order unaware metric)
  ✓ E.g. in above example, if first 2 docs were relevant and next 1 doc was not relevant then also Recall@3 would be 2/3

  2) Another disadv: By increasing K to N i.e. K = No. of recommended items(N) we can return a perfect score everytime

  ✓3) Smaller K value makes it harder to score well with Recall@K metric

- Range [0,1]

- Mostly used for binary relevance. Can be adapted to graded relevance by thresholding and converting to binary relevance.

3. AVERAGE PRECISION :

- Metric that tells how much of the relevant docs are concentrated in the highest ranked predictions ✓

- $$\frac{\sum_{K=1}^{n} P@K \times relevance(K)}{\text{No. of relevant docs}}$$ where $relevance(K) = \begin{cases} 0 \\ 1 \end{cases}$ depending on whether doc at rank K is relevant or not ✓

- This metric is able to give more wts. to errors that happen high up in the recommended list. Conversely, it gives less wt to errors that happen deeper in the recommended list ✓

- One of the methods to calculate Area Under P-R Curve for one class [obj. detection use case] (other method is using trapezoidal rule)

- Defined for 1 user (1 list)

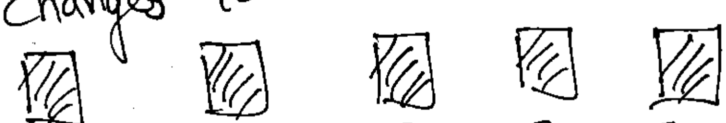
E.g.

- relevant
- not relevant

User1 / list1:

$\frac{1}{1}\times1 \quad \frac{1}{2}\times0 \quad \frac{2}{3}\times1 \quad \frac{3}{4}\times1 \quad \frac{3}{5}\times0 \quad = \frac{\frac{1}{1}+\frac{2}{3}+\frac{3}{4}}{3} = 0.8$

Now, if it changes to



$\frac{1}{1}\times1 \quad \frac{2}{2}\times1 \quad \frac{2}{3}\times0 \quad \frac{3}{4}\times1 \quad \frac{3}{5}\times0 \quad \frac{\frac{1}{1}+\frac{2}{2}+\frac{3}{4}}{3} = \frac{2.75}{3} = 0.916$ (increased)

- Range: [0, 1]
- Can be adapted to graded relevance by thresholding

4. MAP (Mean Avg. Precision)

- Avg. precision over a set of users/lists

- $$\frac{\sum_{u=1}^{U} \text{Avg. Precision } (u)}{\text{Number of Users / lists}}$$ where $U = \#$ of users

- relevant
- not relevant

User 1: $\frac{1}{1} \times 1$ $\frac{1}{2} \times 0$ $\frac{2}{3} \times 1$ $\frac{3}{4} \times 1$ $\frac{3}{5} \times 0$ $= \dfrac{\frac{1}{1} + \frac{2}{3} + \frac{3}{4}}{3} = 0.8$

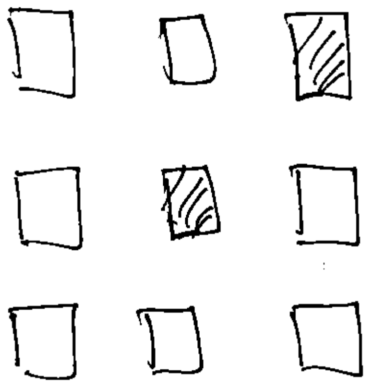
User 2: $\frac{0}{1} \times 0$ $\frac{0}{2} \times 0$ $\frac{0}{3} \times 0$ $\frac{1}{4} \times 1$ $\frac{2}{5} \times 1$ $\dfrac{\frac{1}{4} + \frac{2}{5}}{2} = 0.325$

$$MAP = \frac{0.8 + 0.325}{2} = 0.56$$

- Defined for a group of users/lists
- Mean of Area Under PR curve for all classes of each obj. [obj. detection use case]
- Con: 1) Since all classes are not of interest (e.g. class 0) or equal (in imbalanced dataset), it may not give complete/accurate picture)
  2) Primarily, for binary relevance
- Range [0, 1]
- Can be adapted to graded relevance by thresholding

5. RECIPROCAL RANK : Inverse rank where the first relevant doc is found

E.g.



RR = 1/3

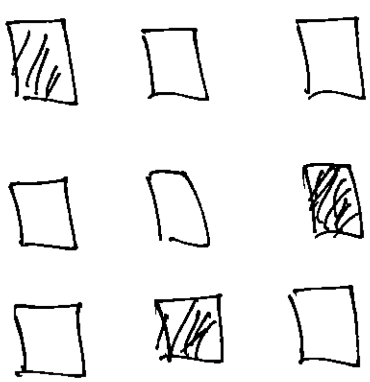
RR = 1/2

RR = 0

- Con: Only cares about the first relevant doc's rank
- [0,1] Range
- Can be adapted to graded relevance by thresholding

6. MEAN RECIPROCAL RANK

- Avg. of RR for multiple users/lists



RR = 1

RR = 1/3

RR = 1/2

$$MRR = \frac{1 + 1/3 + 1/2}{3} = 0.61$$

- Con : Only cares about first relevant doc's rank
- [0,1] Range
- Can be adapted to graded relevance by thresholding

※ All the previous 6 metrics are usually used for binary relevance

7. NORMALIZED DISCOUNTED CUMULATIVE GAIN

- Used when relevancy is not binary, instead it is a real number (can also be used for binary relevance)

- Gives more importance to correctly predicted ranks at top and "discounts" mistakes as you down the ranks

- Compared to MAP, NDCG further tunes the recommended lists evaluation. Since relevance is a real number, instead of binary, it is able to use the fact that some documents are "more" relevant than others.

$$CG_p = \sum_{i=1}^{p} rel_i$$

where p = # of elements in the recommended list

rel_i = graded relevance of result at position i

$$DCG_p = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)}$$

DCG at position p

(not normalized hence difficult to compare)
e.g. longer list will have higher DCG compared to shorter list

or

$$\sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

(Used in industry)

$\log_2(i+1)$ = logarithmic reduction factor to penalize proportionally to the position of result

$$IDCG_p = \sum_{i=1}^{|REL_p|} \frac{rel_i}{\log_2(i+1)}$$

ideal DCG at position p

or

$$\sum_{i=1}^{|REL_p|} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

$|REL_p|$ = list of relevant docs sorted by relevance ~~upto~~ position p (High rel to low rel.)

- IDCG is "ideal DCG" so the formula should remain consistent

→ $NDCG_p = \dfrac{DCG_p}{IDCG_p}$

- Per User/list metric, we need to avg it out for all users in test set

- Range $[0,1]$

E.g.

Predicted Rank =	1	2	3	4	5	6
True Rating	$\boxed{3}$	$\boxed{2}$	$\boxed{3}$	$\boxed{0}$	$\boxed{1}$	$\boxed{2}$

Predicted Ranking with True Ratings

$CG_6 = 3 + 2 + 3 + 0 + 1 + 2 = 11$

DCG_6 (using $\dfrac{rel_i}{\log_2(i+1)}$) $= \dfrac{3}{\log_2(1+1)} + \dfrac{2}{\log_2(2+1)} + \dfrac{3}{\log_2(3+1)} + 0 + \dfrac{1}{\log_2(5+1)} + \dfrac{2}{\log_2(6+1)}$

$= 6.8$

To get IDCG, sort the list according to true rating

Rank =	1	2	3	4	5	6
	$\boxed{3}$	$\boxed{3}$	$\boxed{2}$	$\boxed{2}$	$\boxed{1}$	$\boxed{0}$

$IDCG_6$ (using $\dfrac{rel_i}{\log_2(i+1)}$) $\dfrac{3}{\log_2(1+1)} + \dfrac{3}{\log_2(2+1)} + \dfrac{2}{\log_2(3+1)} + \dfrac{2}{\log_2(4+1)} + \dfrac{1}{\log_2(5+1)} + 0$

$= 7.141$

$NDCG_6 = \dfrac{DCG_6}{IDCG_6} = \dfrac{6.8}{7.141} = 0.961$

CONS OF NDCG: (occurs when unequal size of returned list)

1) Does not penalize for <u>missing docs</u> in the recommended list

Query 1 returns : 1, 1, 1

Query 2 returns : 1, 1, 1, 1, 1

$NDCG_3(\text{query } 1) = NDCG_5(\text{query } 2)$

To fix this: i) enforce fixed size of result set
ii) Use minimum scores for missing docs

Query 1 : 1, 1, 1, 0, 0

Query 2 : 1, 1, 1, 1, 1

$NDCG_5(\text{query } 1) < NDCG_5(\text{query } 2)$

2) Does not penalize <u>bad docs</u> in the recommended list

Query 1 returns : 1, 1, 1

Query 2 returns : 1, 1, 1, 0

$NDCG_3(\text{query } 1) = NDCG_4(\text{query } 2)$

NLP METRICS

- should be defined based on task (task-specific)

1. BLEU (Bilingual Evaluation Understudy)
   - n-gram overlap b/w O/P sentence & Reference sentence
   - n-gram can be unigram, bigram, trigram, ...
   - precision kind of metric, range [0,1]
   - CLIP $\frac{\text{No. of words matched}}{\text{No. of words in O/P}}$ (from O/P to ref)

   ↓

   Clips no. of words matched by no. of times the word appeared in reference

Example

	Ref	O/P
1. हम स्कूल जा रहे	I am going to school	school school

If no CLIP in definition $= \frac{2}{2} = 1 \rightarrow$ misleading

for unigram matches: If CLIP, BLEU-1 $= \frac{1}{2} = 0.5$

	Ref	O/P
2. हम स्कूल जा रहे	I am going to school	I school here

for unigram matches: BLEU-1 $= \frac{2}{3}$ → penalizes extra word in O/P

- Issues: 1) missing correct words from O/P is not penalized
  2) Semantics ✗
  3) order of words is not taken into account for unigram matches

     soln: n-gram

- Usually BLEU score = Geometric mean of all 4 n-gram precisions i.e. unigram, bigram, trigram & quadgram

$$= \sqrt[4]{P_1 P_2 P_3 P_4}$$

2. METEOR: Overcomes drawbacks of BLEU, which are
1) Does not take recall into account
2) Allows only exact n-gram matching

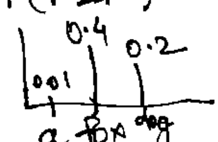
METEOR does synonym/stemmed match + computes recall as well
$\xrightarrow{\text{finally}}$ computes F-score

3. ROUGE
- Recall related metric [0,1]
- $\text{ROUGE-1 recall} = \dfrac{\text{No. of word matches}}{\text{No. of words in reference}}$
- $\text{ROUGE1 precision} = \dfrac{\text{No. of word matches}}{\text{No. of words in O/P}}$

In practice, ROUGE1 is the F1 score calculated on top of ROUGE-1 precision & recall

- Similarly for bigram, trigram : ROUGE-2, ROUGE-3
- ROUGE-L : - Least Common subsequence matching
  - no exact n-gram matching
  - same order but not necessarily continuous

4. PERPLEXITY : - Used in lang. generation tasks
- How confused/uncertain the model is in generating next token/text
- Lower is better, range : [0,1]
- $e^{\text{cross-entropy}}$
- Let's say vocab has 6 words and prob. distribution of next word is available given a word e.g. $P(W_2|a)$



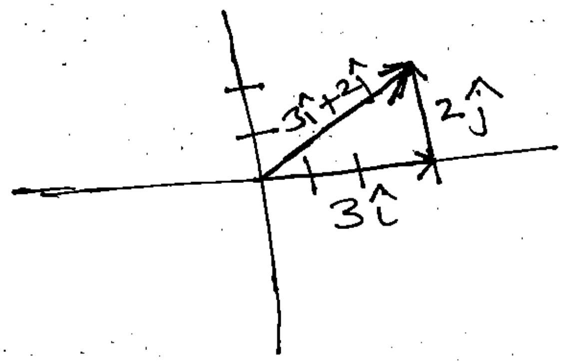
$\therefore$ Prob of sentence "a red fox"
$= P(a) * P(red|a) * P(fox|a\ red)$

Finally, normalize it by length of sentence/text

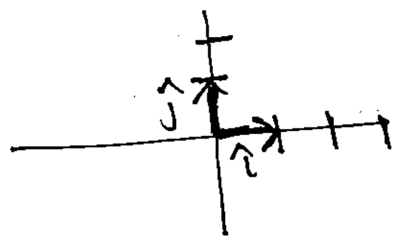
5. Cosine similarity b/w O/P & reference can be used as a metric as well

VECTORS:

$$\begin{bmatrix} 3 \\ 2 \end{bmatrix} = 3\hat{i} + 2\hat{j}$$



BASIS VECTORS: Vectors that define a co-ordinate system
$\hat{i}$ & $\hat{j}$ are the basis vectors of xy co-ord. system



technical defn: → Set of linearly independent vectors that span the full space

Hence whenever we specify a vector e.g $\begin{bmatrix} 3 \\ 2 \end{bmatrix}$, implicitly there is a notion of basis vectors and the vector we specified is a "scaled" version of basis vectors

$$\begin{bmatrix} 3 \\ 2 \end{bmatrix} = 3 \text{ times } \underbrace{\hat{i}}_{\substack{\text{basis vector} \\ \text{in x-dir}}} + 2 \text{ times } \underbrace{\hat{j}}_{\substack{\text{basis vector} \\ \text{in y-dir}}}$$

LINEAR COMBINATION OF VECTORS:

Let $\vec{v}$ & $\vec{w}$ be two vectors then their linear combination is defined as

$a\vec{v} + b\vec{w}$ (note: scalar multiplication + vector addition)

a, b ← scalar

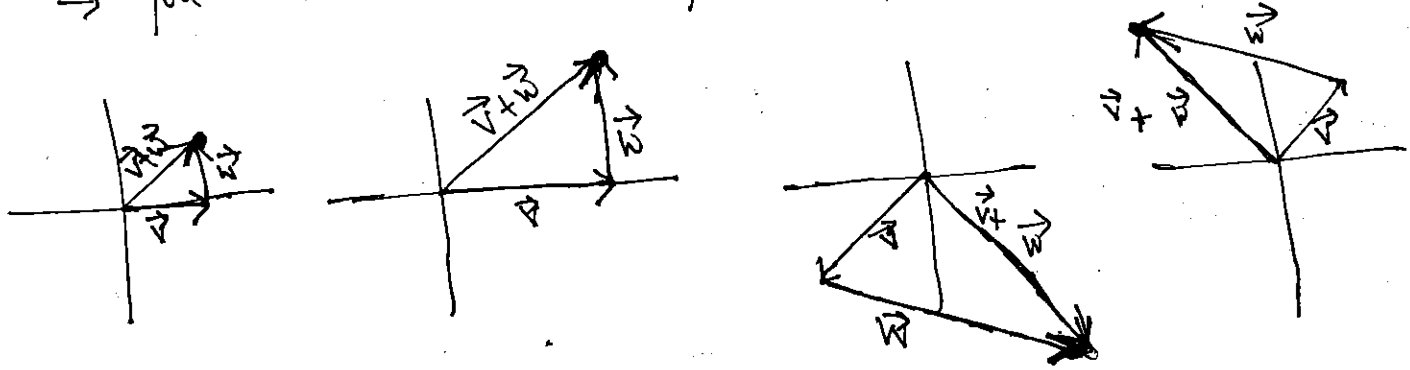
If a & b can take any values (real numbers) 3 possibilities

i) $a = 0$, $b = 0$

$\Rightarrow a\vec{v} + b\vec{w} =$ origin

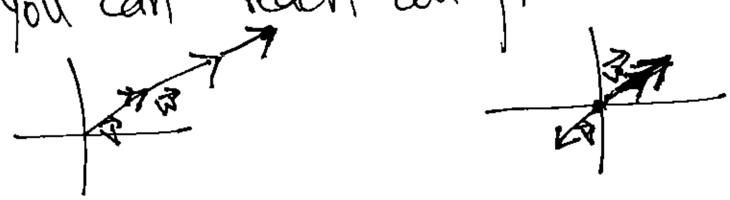
ii) $a \neq 0$, $b \neq 0$ and $\vec{v}$ & $\vec{w}$ do not line up

$\Rightarrow$ you can reach all pts on a plane



(iii) $a \neq 0$, $b \neq 0$ and $\vec{v}$ & $\vec{w}$ line up

$\Rightarrow$ you can reach all pts on the same line as $\vec{v}$ & $\vec{w}$



SPAN OF VECTORS:

The span of $\vec{v}$ & $\vec{w}$ is the set of all pts reachable via their linear combinations

$$a\vec{v} + b\vec{w}$$

scalars that can be any real number

If $\vec{v}$ & $\vec{w}$ do not line up (point in diff. directions)

$\Rightarrow$ span $\rightarrow$ 2d plane

Similarly,
For 3 vectors $\vec{v}, \vec{w}$ & $\vec{u}$, if they do not line up

$\Rightarrow$ span $\rightarrow$ 3d space

If two of these vectors line up

$\Rightarrow$ span $\rightarrow$ 2d plane

If all three of these vectors line up

$\Rightarrow$ span $\rightarrow$ line

LINEAR DEPENDENCE: In a grp of vectors, if a vector does not add anything to the span or is redundant (since it is a scaled version of any other vector or linear comb. of other vectors) $\rightarrow$ Linearly dependent vector

$$\underbrace{\vec{u}}_{\text{linearly dependent}} = \underbrace{a\vec{v} + b\vec{w}}_{\rightarrow \text{Linear comb.}}$$

LINEARLY INDEPENDENT: If a vector adds another dimension to span in a grp of vectors or cannot be expressed as a linear comb. of other vectors in the grp

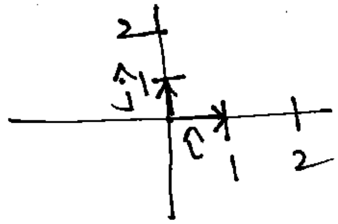
→ Linearly Independent vector

$$\vec{u} \neq a\vec{v} + b\vec{w}$$

For "all" values of a & b

MATRICES AS LINEAR TRANSFORMATION OF SPACE:

Recall that in a ~~xy~~ co-ord. system, $\hat{i}$ & $\hat{j}$ are the basis vectors



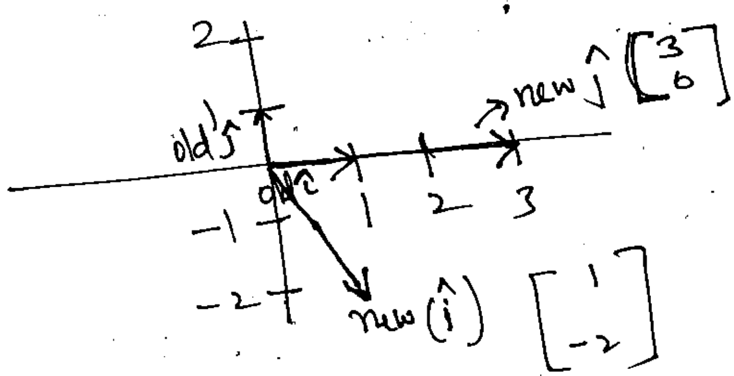
in matrix format

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$\hat{i}$ co-ord. $\hat{j}$ co-ord

Now a matrix $\begin{bmatrix} 1 & 3 \\ -2 & 0 \end{bmatrix}$ represents a linear transformation of $\hat{i}$ & $\hat{j}$ where new $\hat{i} = 1(\text{old } \hat{i}) - 2(\text{old } \hat{j})$

new $\hat{j} = 3(\text{old } \hat{i}) + 0(\text{old } \hat{j})$



Note: "Linear" in linear transformation: 1) origin is fixed before & after transformation of space

2) new $\hat{i}$ & new $\hat{j}$ is still a line and not curved

3) grid lines are parallel & evenly spaced even in the transformed space

MULTIPLICATION OF VECTOR & MATRIX

(in the original space)

– Can be thought of as where the vector lands after the transformation of space represented by matrix

Let $v = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$ & $M = \begin{bmatrix} 1 & 3 \\ -2 & 0 \end{bmatrix}$

new $\hat{i}$ lands new $\hat{j}$ lands

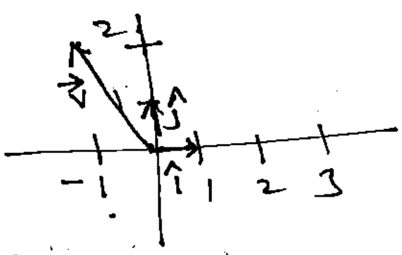
$\Downarrow$ means

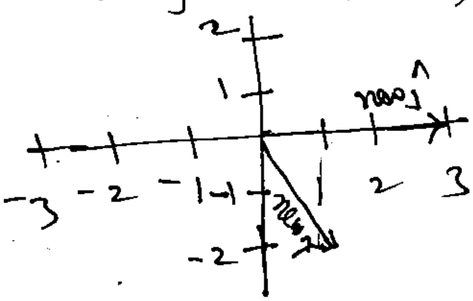
$\vec{v} = -1\hat{i} + 2\hat{j}$

$\Downarrow$ means

new $\hat{i}$ = $1(\text{old } \hat{i}) + -2(\text{old } \hat{j})$

new $\hat{j}$ = $3(\text{old } \hat{i}) + 0(\text{old } \hat{j})$



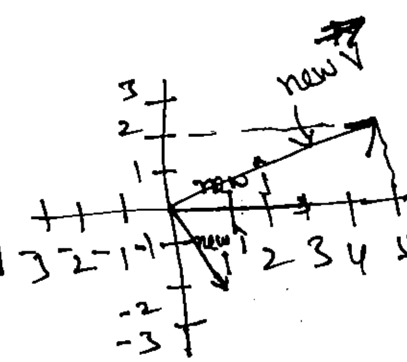


→ Transformed $\vec{v}$ = -1 (transformed $\hat{i}$) + 2 (transformed $\hat{j}$)

$\Downarrow$

multiplication of matrix by a vector $= -1 \begin{bmatrix} 1 \\ -2 \end{bmatrix} + 2 \begin{bmatrix} 3 \\ 0 \end{bmatrix}$

$= \begin{bmatrix} -1 \\ 2 \end{bmatrix} + \begin{bmatrix} 6 \\ 0 \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \end{bmatrix}$



MULTIPLICATION OF MATRIX BY MATRIX:

Since matrix represents a transformation of space matrix multiplication can be thought of as consecutive transformation of space i.e. one transformation followed by another

Condition: $m \times \underline{n} \quad \underline{n} \times p \Rightarrow m \times p$

# of cols = # of rows

Dot product

Eg. $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ & \end{bmatrix}$

$2\times3 \quad 3\times2 \quad 2\times2$

FORMAL DEF^n of LINEARITY (Linear Transformation)

Let L be the transformation:

I) Additivity: $L(\vec{v} + \vec{w}) = L(\vec{v}) + L(\vec{w})$

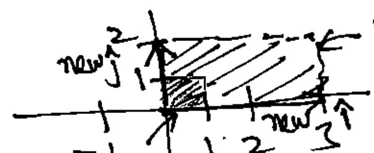
II) scaling: $L(c\vec{v}) = cL(\vec{v})$

DETERMINANT OF A MATRIX

We know matrix represents a transformation of space
Now this transformation can increase the space or decrease the space (w.r.t. original space)

The factor by which the original space increases or decreases $\Rightarrow$ determinant of matrix

$$\begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \rightarrow$$



In 2d terms, space = area
original area = 1
new area $= 3 \times 2 = 6$

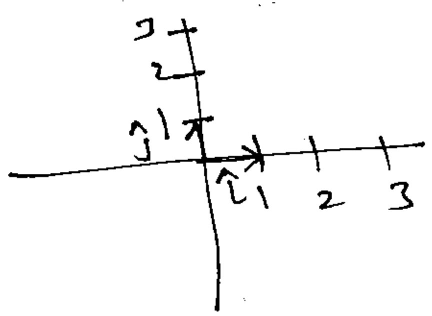
$$\begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \rightarrow 3 \times 2 - 0 \times 0 = 6$$

So in 2d, |determinant of a matrix| = factor by which unit area in original space changes

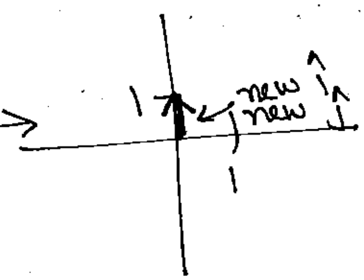
Now determinant can be -ve : if the new transformed space is obtained by flipping original space. Eg. $\begin{bmatrix} 3 & 0 \\ 0 & -2 \end{bmatrix}$

ZERO DETERMINANT :

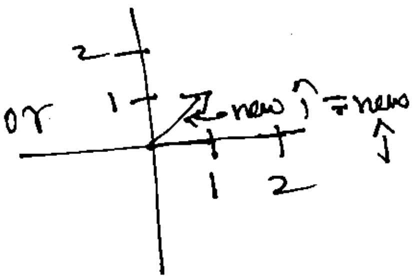
When will the area in the transformed space $= 0$



$\rightarrow$

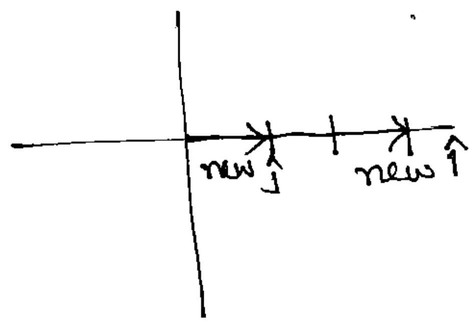


or



$$\begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} \text{ or } \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

or



$$\begin{bmatrix} 3 & 1 \\ 0 & 0 \end{bmatrix}$$

$\Rightarrow$ If the columns in the matrix are linearly dependent : determinant becomes zero

$\Rightarrow$ "Singular Matrix": When the determinant is zero (i.e. inverse does not exist)

IDENTITY MATRIX :

Recall, matrix is a transformation of space

Identity matrix is a special type of transformation that does nothing.

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \begin{aligned} \text{new } \hat{i} &= 1(\text{old } \hat{i}) + 0(\text{old } \hat{j}) = \text{old } \hat{i} \\ \text{new } \hat{j} &= 0(\text{old } \hat{i}) + 1(\text{old } \hat{j}) = \text{old } \hat{j} \end{aligned}$$

a) Identity Matrix is a square matrix (# of rows = # of columns)

b) Identity Matrix has 1s on the main diagonal & 0s everywhere else

Multiplying a vector by identity matrix = leaves the vector unchanged

Multiplying a matrix by inverse of the matrix = identity matrix

$\parallel$

equivalent to doing no transf. at all

$$A^{-1} A = I$$

or $\quad A A^{-1} = I$

$(A \times I = A$ or $I \times A = A)$

INVERSE OF A MATRIX:

✓ Inverse of a matrix exists only when

1) matrix is a square matrix

2) determinant of the matrix $\neq 0$

To understand why $\det(\text{matrix}) \neq 0$:

$\det(\text{matrix}) = 0$ when transformation of space represented by matrix squishes the original space

$A \rightarrow$ represents a transformation of space

$A^{-1} \rightarrow$ represents a reverse transformation to A

So essentially $A^{-1} \cdot A \Rightarrow$ does not change the space

However when $\det(A) = 0$, the transformation squishes the space e.g. 2d space becomes 1d (line) and you cannot recover 2d space back from line.

<u>2x2 Matrix</u>: $\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad-bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$

1) swap the positions of a & d 2) put -ve in front of b & c
3) Divide everything by ad-bc (determinant)

<u>Why do we need Inverse?</u> Because with matrices, we don't divide! (no concept of dividing by a matrix)
But we can multiply by an inverse, which achieves the same thing

$$XA = B$$

To find matrix X: $XAA^{-1} = BA^{-1}$

$$X = BA^{-1}$$

SYSTEM OF EQUATIONS

$$\left.\begin{aligned} 2x + 3y + 9z &= 10 \\ 10x + 11y + 12z &= 20 \\ 39x + 14y + 27z &= 62 \end{aligned}\right\}$$

This can be thought of as:
$x, y, z \rightarrow$ wts
$2, 3, 9$, etc. $\rightarrow$ feature values

In matrix form:

$$\underbrace{\begin{bmatrix} 2 & 3 & 9 \\ 10 & 11 & 12 \\ 39 & 14 & 27 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x \\ y \\ z \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 10 \\ 20 \\ 62 \end{bmatrix}}_{b}$$

To have a "unique" sol^n
For a system of m linear eq^n with n unknowns:

i) $m = n$

ii) inverse of matrix A should exist

$$Ax = b$$
$$A^{-1}Ax = A^{-1}b$$
$$Ix = A^{-1}b$$
$$x = A^{-1}b$$

This is called "analytically" solving system of linear eq^n

Eg. A group took a trip on a bus at \$3 per child and \$3.20 per adult for a total of \$118.40

They took the train back at \$3.50 per child and \$3.60 per adult for a total of \$135.20

How many children & how many adults?

Let x_1 = # of children x_2 = # of adults

$$3x_1 + 3.20\,x_2 = 118.40$$

$$3.5x_1 + 3.60\,x_2 = 135.20$$

$$\begin{bmatrix} 3 & 3.20 \\ 3.5 & 3.60 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 118.4 \\ 135.2 \end{bmatrix}$$

$$A\,X = B$$

$$A^{-1}AX = A^{-1}B$$

$$IX = A^{-1}B$$

$$X = A^{-1}B$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 & 3.20 \\ 3.5 & 3.60 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 118.4 \\ 135.2 \end{bmatrix}$$

$$= \begin{bmatrix} 16 \\ 22 \end{bmatrix}$$

Solving $x_1 = 16$

$x_2 = 22$

ELEMENT-WISE OPERATION / HADAMARD PRODUCT

1. Operands (vector/matrix) should be of same dimension
= operator applied to corresponding elements of each operand

2. Exception : M = Matrix V = Vector
$\Rightarrow$ rows should be of same dimension

$$M = \underset{\underline{2\times 3}}{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}} \quad V = \underset{\underline{2}\times 1}{\begin{bmatrix} 1 \\ 2 \end{bmatrix}} = \begin{bmatrix} 1 & 2 & 3 \\ 8 & 10 & 12 \end{bmatrix}$$

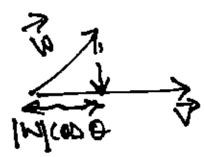
$\downarrow$ broadcast 2×3

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{bmatrix}$$

Now perform element-wise prod

3. * or np.multiply

DOT-PRODUCT :



$\vec{v}\cdot\vec{w} = |v||w|\cos\theta$

1. no. of columns of first operand = no. of rows of second operand

$$n\times m \quad m\times r \quad \Rightarrow \quad n\times r$$

2. If $\vec{a} = a_x + b_y + c_z$ & $\vec{b} = b_x + b_y + b_z$

$$\text{np.dot}(a,b) = a_x b_x + a_y b_y + c_y c_z$$

3. If both are matrix $\Rightarrow$ matrix multiplication

4. One V & One M

$$\underset{3\times 3}{\begin{bmatrix} 3 & 0 & 2 \\ 2 & 0 & -2 \\ 0 & 1 & 1 \end{bmatrix}} \underset{3\times 1}{\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}} = \underset{3\times 1}{\begin{bmatrix} 9 \\ -4 \\ 5 \end{bmatrix}} \begin{matrix} (3\times 1 + 0\times 2 + 2\times 3) \\ (2\times 1 + 0\times 2 + -2\times 3) \\ (0\times 1 + 1\times 2 + 1\times 3) \end{matrix}$$

DERIVATIVE OF SIGMOID

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

$$\frac{d}{dx}(\sigma(x)) = \frac{d}{dx}\left(1+e^{-x}\right)^{-1}$$

$$= -\left(1+e^{-x}\right)^{-2}\left(-e^{-x}\right)$$

$$= \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= \frac{e^{-x}}{1+e^{-x}} \cdot \frac{1}{1+e^{-x}}$$

$$= \frac{1+e^{-x}-1}{1+e^{-x}} \cdot \frac{1}{1+e^{-x}}$$

$$= 1 - \frac{1}{1+e^{-x}} \cdot \frac{1}{1+e^{-x}}$$

$$= (1-\sigma(x))(\sigma(x))$$

---

tanh is rescaled sigmoid fn

$$\tanh(x) = 2\sigma(2x) - 1$$

$$\frac{d}{dx}(x^n) = nx^{n-1}$$

$$\frac{d}{dx}(e^x) = e^x$$

$$\frac{d}{dx}(e^{-x}) = -e^{-x}$$

$$\frac{d}{dx}(ax) = a$$

$$\frac{d}{dx}(e^{4x}) = e^{4x} \cdot \frac{d}{dx}(4x)$$

$$= e^{4x} \cdot 4$$

$$\frac{d}{dx}(c+x) = 1$$

CLOSED FORM SOLN : (ANALYTICAL SOLN)

Closed form solⁿ of equation → finite number of elementary operations $(+, -, *, /, \sqrt{\ }, e^x)$ to get solⁿ

Eg. $ax^2 + bx + c = 0$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Closed form solⁿ are not preferred due to very expensive operations. Hence gradient descent methods are used

e.g. for linear reg. closed form solⁿ

$$B = \underbrace{(X^T \cdot X)}_{\text{expensive}}{}^{-1} \cdot X^T \cdot y$$

(inverse) expensive

bcz

$X =$ K columns (K predictors)
N rows (no. of observations)

can be very big matrix itself

KOLMOGOROV COMPLEXITY:

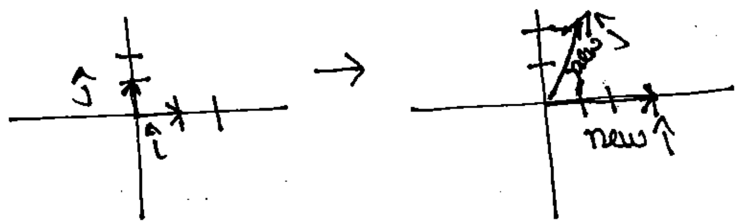
- length of optimal specification of an algo or object (string)
- 11111111111 → Less Kolmogorov complexity as there is pattern
  0A59c3deZa → High Kolmogorov complexity as there is no obvious pattern

- Kol. complexity of x = shortest program outputting x
  If you fix a lang, e.g. python & write program to output above two strings then since first string has an obvious pattern, it will require shorter program to o/p it

∴ It describes how "compressible" a string is

EIGEN VALUES & EIGEN VECTORS

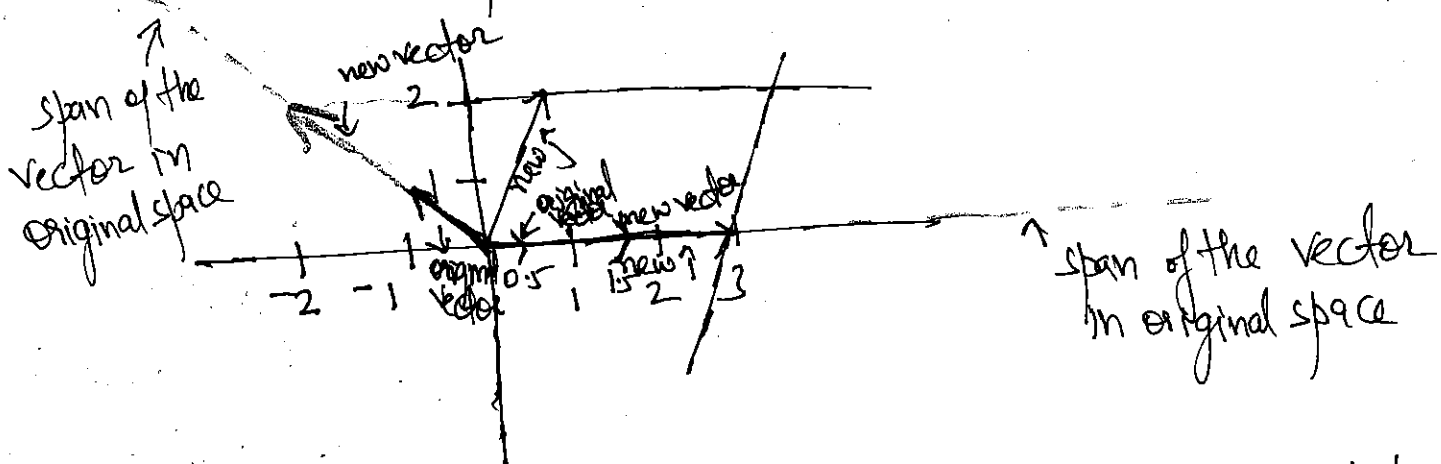
Recall matrix is a transformation of space



$$= \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix}$$

new $\hat{i}$ new $\hat{j}$

Now think what happens to a vector when multiplied by this matrix meaning where will this vector land in the transformed space (w.r.t. original space)



If the vector in original space was on $\hat{i}$, it will still remain on $\hat{i}$-axis as $\hat{i}$ is now $3\hat{i}$ (transformation) but will be stretched by a factor of 2.

Similarly the vector $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$ when multiplied by $\begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix}$

$$\Rightarrow \begin{bmatrix} -1 \\ 1 \end{bmatrix} \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} = -1 \begin{bmatrix} 3 \\ 0 \end{bmatrix} + 1 \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} -3 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \end{bmatrix} = 2 \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

will remain on its "span" but will be stretched by a factor of 2

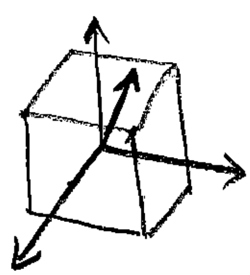
Eigen Vectors: The vectors which remain on their original span even after the matrix transformation are called eigen vectors

Note: eigen vectors are generally normalized to unit length

Eigen Values: The factor by which eigen vectors are stretched or contracted in the new transformed space are called eigen values.

Eigen Vectors = Axis of Rotation

If a vector stays in the same orientation (span) when the space is transformed, it is in effect the axis of rotation



Now if this cube rotates but the vector still points in the same dir → axis of rotation (or opp dir but on the same span)

FINDING EIGEN VECTOR & EIGEN VALUE OF A MATRIX

Where $\vec{v}$ = eigen vector
λ = eigen value

$$\underbrace{A\vec{v}}_{\text{Matrix-Vector multiplication}} \xrightarrow{\text{is same as}} \underbrace{\lambda\vec{v}}_{\text{scaling the vector}}$$

Matrix-Vector multiplication is same as scaling the vector

LHS = Matrix · vector
RHS = Scalar · vector

writing scalar in matrix format = diagonal matrix with diagonal element $= \lambda$

$$\begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} = \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$A\vec{v} = \lambda I \vec{v}$$

$$(A - \lambda I)\vec{v} = 0$$

Now we don't want $\vec{v} = 0$ since we want non-zero eigen vector

$$\therefore \underbrace{(A - \lambda I)}_{\text{Matrix}}\underbrace{\vec{v}}_{\text{vector}} = 0$$

Now a non-zero vector when transformed via a matrix is zero only when the matrix transformation "squishes" the vector into lower dimension

$$\Rightarrow \det(A - \lambda I) = 0$$

Note: det. of matrix = 0 when the space transformation done by matrix squished into lower dimension get the value of λ by solving $\det(A-\lambda I)=0$ and substitute in $(A-\lambda I)\vec{v}=0$ to get value of $\vec{v}$

E.g. $A = \begin{bmatrix} 3 & 1 & 4 \\ 1 & 5 & 9 \\ 2 & 6 & 5 \end{bmatrix}$

$$A-\lambda I = \begin{bmatrix} 3-\lambda & 1 & 4 \\ 1 & 5-\lambda & 9 \\ 2 & 6 & 5-\lambda \end{bmatrix}$$

$$\det(A-\lambda I) = \det\left(\begin{bmatrix} 3-\lambda & 1 & 4 \\ 1 & 5-\lambda & 9 \\ 2 & 6 & 5-\lambda \end{bmatrix}\right) = 0$$

Simpler example: $A = \begin{bmatrix} 2 & 2 \\ 1 & 3 \end{bmatrix}$

$$A-\lambda I = \begin{bmatrix} 2-\lambda & 2 \\ 1 & 3-\lambda \end{bmatrix}$$

$$\det(A-\lambda I)=0 \Rightarrow (2-\lambda)(3-\lambda)-(2)(1)=0$$

$$\therefore \lambda = 1$$

$$(A-\lambda I)\vec{v}=0$$

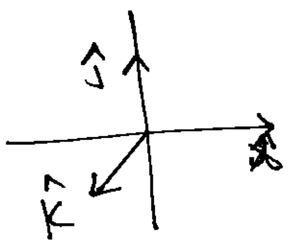
$$\Rightarrow \begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow x\begin{bmatrix} 1 \\ 1 \end{bmatrix} + y\begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

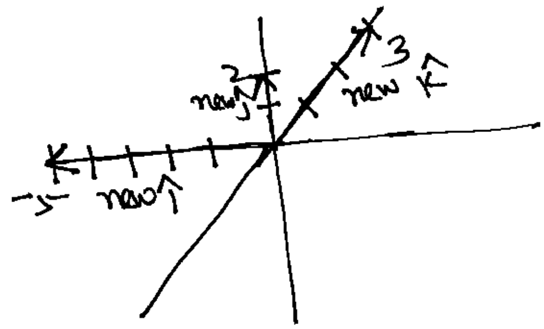
$$= \begin{bmatrix} x \\ x \end{bmatrix} + \begin{bmatrix} 2y \\ 2y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

here x & $y = 0$ but this is just an example

DIAGONAL MATRIX INTERPRETATION IN TERMS OF EIGEN VECTORS & EIGEN VALUES

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} -5 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$





Original space . . . Transformation

$$\begin{bmatrix} -5 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

Eigen vectors as they stay on their span, may be expand or contract

Eigen values - factor by which eigen vectors expand or contract in the transformed space

EIGEN DECOMPOSITION (MATRIX FACTORIZATION TECHNIQUE)

Eigendecomposition of a matrix is a type of decomposition that involves decomposing a ~~squared~~ symmetric matrix into a set of eigenvectors and eigen values

If the eigenvectors of a matrix A are $v_1, v_2, \dots v_n$ and $v_1, v_2, \dots v_n$ are stacked column-wise to create a matrix ~~vector~~ V, ~~and~~ $\lambda_1, \lambda_2, \dots \lambda_n$ are eigen values, stacked diagonally ~~to~~ create matrix λ; then

$$\boxed{A = V \, \mathrm{diag}(\lambda) \, V^{-1}}$$

eigen decomposition of matrix A
(A needs to be square & symmetric)

Eigendecomposition is used to simplify calc. of other complex matrix operations. Also used in PCA.

The decomposition can be derived from fundamental property of eigen vectors: $A\vec{v} = \lambda\vec{v}$

↓ if you pack all v vectors into a matrix V column-wise

$$\Rightarrow AV = \lambda V$$
$$\Rightarrow AV = V\lambda$$
$$\Rightarrow A = V\lambda V^{-1}$$

UNITARY MATRIX: square matrix where conjugate transpose = Inverse
For real matrix, conjugate transpose = transpose

$$U^{\dagger} = U^T = U^{-1} \quad , \text{if } U \text{ is real matrix}$$

unitary matrix symbol

Since V in eigendecomposition is unitary, eigen decomposition can also be written as:

$$\boxed{A = V \lambda V^T}$$

VECTOR SPACE: A space where the rules of vector algebra apply.

LINEAR REGRESSION

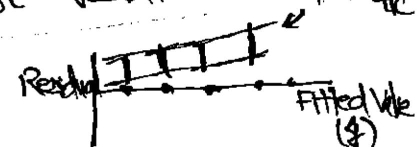
(No assumption abt normal dist. of Independent or dependent var.)

ASSUMPTIONS:

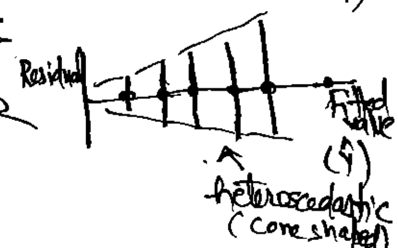
1. Linear Relationship b/w X & Y
   - Use Scatter plot Matrix to check
   - If not linear relationship, transform variables
     e.g. right skew - log transform
     left skew - reciprocal then log transform

2. No multi-collinearity (if interpretation is the objective else not needed) between predictors
   - Use correlation matrix to check
   - If VIF > 10, handle multi-collinearity
     1) identify var which are co-related & remove one of them
     2) highest VIF var → remove
     (+ other methods provided in multicollinearity notes)

3. No-Auto correlation b/w residuals
   - Use Residual plot to check (residuals should be i.i.d random var.) → no pattern or trend
   - Auto-correlation: residuals should not be dependent on ~~past~~ another residuals



4. Homoscedasticity :- Equal var of residuals
   - Use Residual plot to check



5. Normality Assumption of Residuals:
   - ~~Use~~ Residual plot to check
   - Residuals' histogram → Normal

5a) Regression is sensitive to outliers: Check it from residual plot

Linear Regression Eq^n :

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n \quad (1)$$

or

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

or

$$\hat{y} = \theta^T \cdot X$$

→ contains θ_0, bias term

Loss Fn: $SSE = \Sigma(y_i - \hat{y}_i)^2$

or more generally $MSE = \frac{1}{m}\Sigma(y_i - \hat{y})^2$

(1) Solving for $\beta_0, \beta_1 \dots \beta_n$ using Var - Cov. method:
(OLS or Line of best fit method)

$$\beta = \frac{Cov(X,Y)}{Var(X)}$$

where $Cov(X,Y) = \frac{\Sigma(x_i - \bar{x})(y_i - \bar{y})}{n-1}$ where N = # of data values/examples

$$var(x) = \frac{\Sigma(x_i - \bar{x})^2}{n-1}$$

After calculating β's, substitute in eq (1) to get β_0

→ Cov. is a measure of how much two variables change together linearly

→ If there is no linear relationship between two variables, their covariance will be equal to zero; the variables are linearly uncorrelated but not necessarily independent.

In vector form: Normal Equation

$$\hat{\theta} = (X^T \cdot X)^{-1} \cdot X^T \cdot y$$

(2) Gradient Descent Way of solving

$$SSE = \frac{1}{2} \Sigma (y-\hat{y})^2$$

$\frac{1}{2}$ ⇒ for mathematical convenience as it helps in calculating gradient easily

Update rules:

$$\theta^{(\text{next step})} = \theta^{(\text{prev.})} - \text{learning rate} \times \frac{\partial SSE}{\partial \theta}$$

e.g. $\hat{y} = a + bx$

$$\text{New } a = a - \text{learning rate} \times \frac{\partial SSE}{\partial a}$$

$$\text{New } b = b - \text{learning rate} \times \frac{\partial SSE}{\partial b}$$

POLYNOMIAL REGRESSION: – Linear Model (We assume the form of function as linear in coeff.)

$$\rightarrow \hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 \underbrace{x_1 x_2}_{\text{interaction terms}} + \dots$$

→ For given predictors e.g. x_1 & x_2 how to generate polynomial features

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree = 2)
X_poly = poly.fit_transform(X)
```

→ for x_1, x_2 it will generate features all comb. of x_1 & x_2 = $x_1, x_2, x_1x_2, x_1^2, x_2^2$ = all comb including original x_1, x_2

→ Now fit linear Regression Model

ASSESSING REGRESSION MODELS:

1) R^2 : $\dfrac{\text{Variance Explained}}{\text{Total variance}}$ → Interpretation

→ Always between 0 & 1

$$R^2 = 1 - \frac{SS_{res}}{SS_{Total}}$$ where $SS_{res} = \Sigma\,(y_i - \hat{y}_i)^2$

$SS_{total} = \Sigma\,(y_i - \bar{y})^2$ → Mean of observed true values of response variable (not predicted)

2) RMSE or MSE → Prediction

$$RMSE = \sqrt{\frac{\Sigma\,(y_i - \hat{y}_i)^2}{n}}$$

$$RMSE = \sqrt{MSE}$$

→ easier to calc. gradients (compared to MAE)

3) MAE : Mean Absolute Error $= \Sigma |y_i - \hat{y}_i|$

(In case of outliers)

↓

reduces the effect of outliers

FLOWCHART FOR REGRESSION

1) Start with Linear Regression:
   1) Linear relationship b/w X & y → scatterplot + Correlation matrix
   2) No multi-collinearity → Residual plot

2) Once we get the Residual plot, look for two things:
   1) Pattern – There shouldn't be pattern, should be random
   2) Outliers – Few if more → 1) RANSAC or tree based model
      2) Evaluation metric
         – MAE (reduces the effect of outliers)

3) If there are patterns in residual plot, it means we are unable to capture some explanatory information

⇓

Polynomial Regression : to account for interaction terms

or

Non-Linear Model (e.g. RF regressor) : to capture non-linear relationship b/w predictors

LOGISTIC REGRESSION

Assumptions:

1) Binary outcome
2) linear relationship between independent variables and log odds of outcome
3) Observations need to be independent
4) No or little multi collinearity between independent variables
5) Large sample size (to obtain reliable estimates)
6) No influential outliers that excessively influence the estimation of model parameters

Why not use linear regression for predicting probabilities

1. Response / target is binary, not continuous in LR
2. Error terms are not normally distributed in LR
   (if error terms normally distributed MLE = LS)
   MLE: max. likelihood estimation; LS: least squares

ODDS, LOG ODDS, LOGIT FN & LOGISTIC (SIGMOID) FN:

ODDS = ratio of something happening to something not happening

odds = $\frac{p}{1-p}$ where p = prob. of something happening

Probability is between 0 & 1 but when transforming prob. to odds, it removes the upper bound (upper bound = $+\infty$) but lower bound is still 0.

Taking log makes the lower bound $-\infty$ $\left(\begin{array}{l}\log 1 = 0 \\ \log 0 = -\infty\end{array}\right)$

So log (odds) range $[-\infty, +\infty] \Rightarrow$ range of values linear combination of independent variables will output

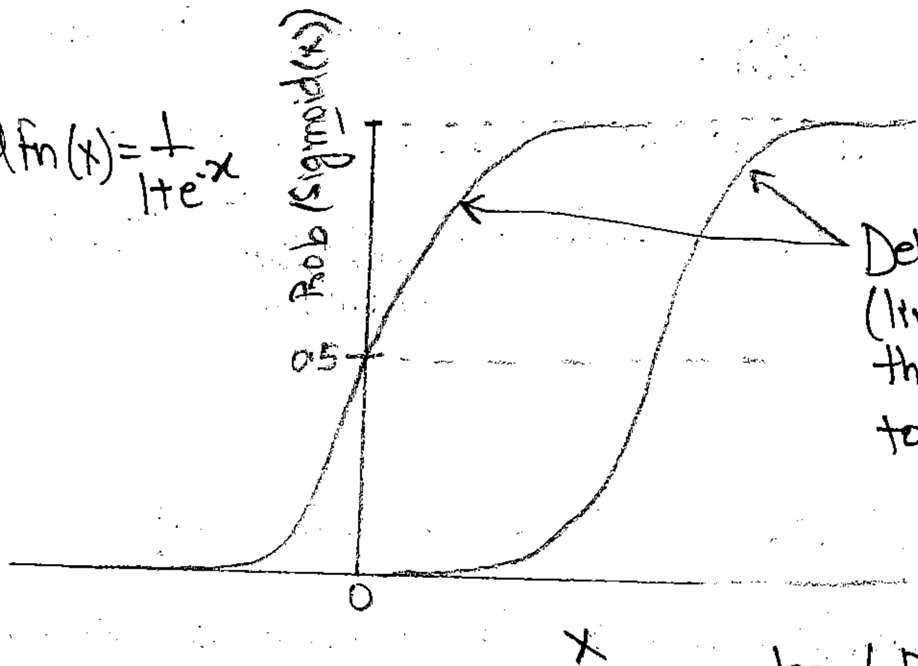
Also $\log(\text{odds}) = \log\left(\frac{p}{1-p}\right) = \text{logit}(p)$

i.e. logit fn converts prob. to log of odds

Logistic/Sigmoid fn is the inverse of logit fn and converts log (odds) into probability
i.e. squashes values from $[-\infty, +\infty]$ to $[0, 1]$

Logistic/Sigmoid ~~(log(odds))~~ $= \dfrac{1}{1+e^{-\log(\text{odds})}} = \dfrac{1}{1+e^{-\log\left(\frac{p}{1-p}\right)}}$

Sigmoid fn $(x) = \dfrac{1}{1+e^{-x}}$



Depending on range of x^* (linear comb. of independent var) the shape can be shifted towards left or right

Derivation of Above Formula:

$$\log\left(\frac{p}{1-p}\right) = \log(\text{odds})$$

Exponentiate both sides

$$\frac{p}{1-p} = e^{\log(\text{odds})}$$

$$p = (1-p)\, e^{\log(\text{odds})}$$

$$p = e^{\log(\text{odds})} - p\, e^{\log(\text{odds})}$$

$$p + p\, e^{\log(\text{odds})} = e^{\log(\text{odds})}$$

$$p = \frac{e^{\log(\text{odds})}}{1+e^{\log(\text{odds})}}$$

∴ Logistic Regression eq^n

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right) = w_1x_1 + w_2x_2 + \dots + w_0$$

or

$$\beta_1 x_1 + \beta_2 x_2 + \dots \beta_0$$

Exponentiating

$$\frac{p}{1-p} = e^{w_0x_0 + w_1x_1 + \dots}$$

$$p = \frac{e^{W^T \cdot x}}{1+e^{W^T \cdot x}}$$

$$p = \frac{1}{1+e^{-(W^T \cdot x)}}$$

↑ prob. of outcome 1

INTERPRETATION : $\log \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots$

$$\frac{p}{1-p} = e^{\beta_0 + \beta_1 x_1 + \beta_2 x_2}$$

↓ odds

Quantitative Var: For every unit change in predictor, the odds of event (with outcome = 1) change by a factor of $e^{\langle \text{coeff. value} \rangle}$

If $e^{\text{coeff value}} > 1$, change → increase

$e^{\text{coeff value}} < 1$, change → dec.

e.g. Let's assume only one prediction β_1

$= e^{\beta_1(x_1+1)} / e^{\beta_1 x_1}$ ⇒ odds of event=1 change for 1 unit change in predictor (in terms of factor)

$= e^{\beta_1 x_1} \cdot e^{\beta_1} / e^{\beta_1 x_1}$

$= e^{\beta_1}$

<u>Qualitative Var:</u> E.g. $x_2 = \begin{cases} 1 & \text{for gold} \\ 0 & \text{for silver} \end{cases}$ } for one-hot coding

If x_2 = gold when compared to silver ($x_2 = 0$),
i.e. $x_2 = 1$

the odds of event (=1) change by the factor of $e^{\text{coeff value}}$ ∴

when $e^{\text{coeff. value}} > 1$, change → increase
$e^{\text{coeff val}} < 1$, change → decrease

<u>COST FUNCTION</u>: (LogLoss function or negative log likelihood)

We know that $L(\underset{\text{parameter}}{\theta} | \overset{\text{outcome of } i^{th} \text{ instance}}{y_i}) = P(y_i | \theta)$

i.e. The likelihood of true parameters being a certain value given data (i.e. outc
= Prob. of observing data (i.e. outcome) given some true parameter values

MLE: maximizes LHS, but since LHS = RHS above, we maximize $P(y_i | \theta)$

Prob. of i^{th} sample being 1 is given by logistic fn (with some params)

$$P(x_i) = \frac{1}{1 + e^{-\theta x_i}}$$

Mathematically, for samples labeled as '1', we try to estimate θ such that prod. of probability $p(x)$ is as close to 1 and for samples labeled as '0', we try to estimate θ such that product of all prob. is as close to 0. i.e. $1-p(x)$ should be close to 1

(product of prob is taken since all samples are independent i.e. $P(A \text{ and } B) = P(A) \cdot P(B)$)

$\therefore$ For samples labelled 1 : $\prod_{s \in y_i = 1} p(x_i)$

For samples labelled 0 : $\prod_{s \in y_i = 0} (1 - p(x_i))$

$$L(\theta) = \prod_{s \in y_i = 1} p(x_i) \prod_{s \in y_i = 0} (1 - p(x_i))$$

→ likelihood fn that needs to be maximized

$$L(\theta) = p(x_i)^{y_i} \left(1 - p(x_i)\right)^{1-y_i}$$

If 1 sample prob $= p(x_i)$
2 sample prob $= (p(x_i))^2$
$\vdots$
y_i sample prob $= (p(y_i))^{y_i}$

Taking log

$$\log L(\theta) = y_i \log p(x_i) + (1-y_i) \log (1 - p(x_i)) \quad (i)$$

Maximize above (i) or minimize the -ve of above eq (i)

$$= -\left[y_i \log p(x_i) + (1-y_i) \log (1 - p(x_i)) \right]$$

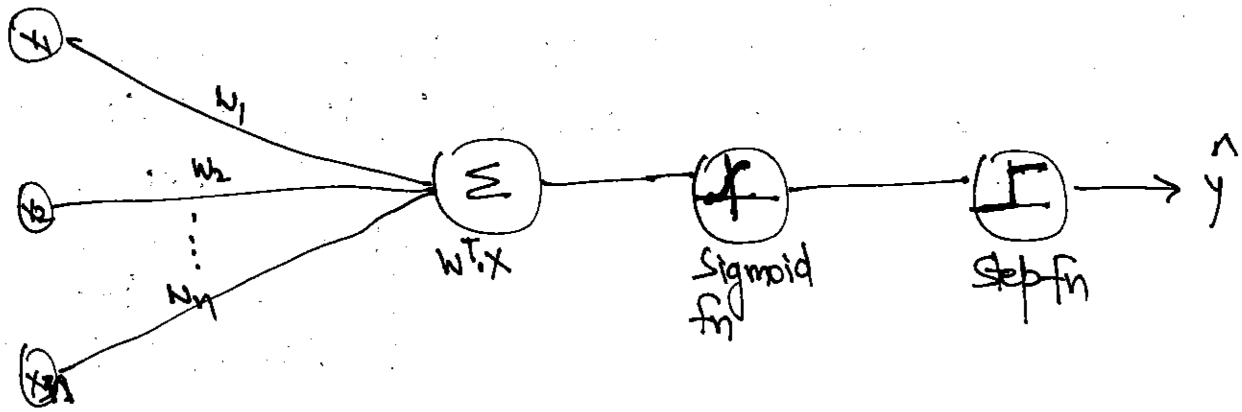
y_i = true label (either 0 or 1)

Average ~~over~~ over n samples

Log loss fn $= -\frac{1}{n} \left[y_i \log p(x_i) + (1-y_i) \log (1 - p(x_i)) \right]$

Logloss heavily penalizes classifiers that are confident about incorrect predictions

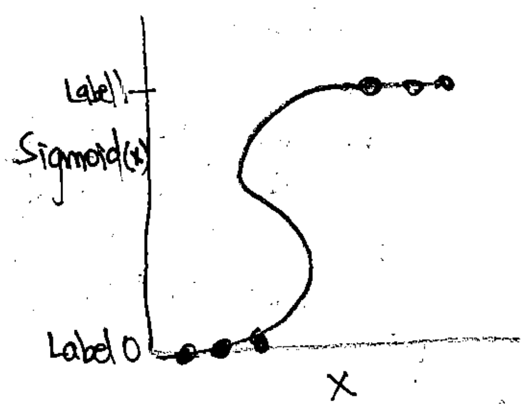
In Neural N/W Terms:

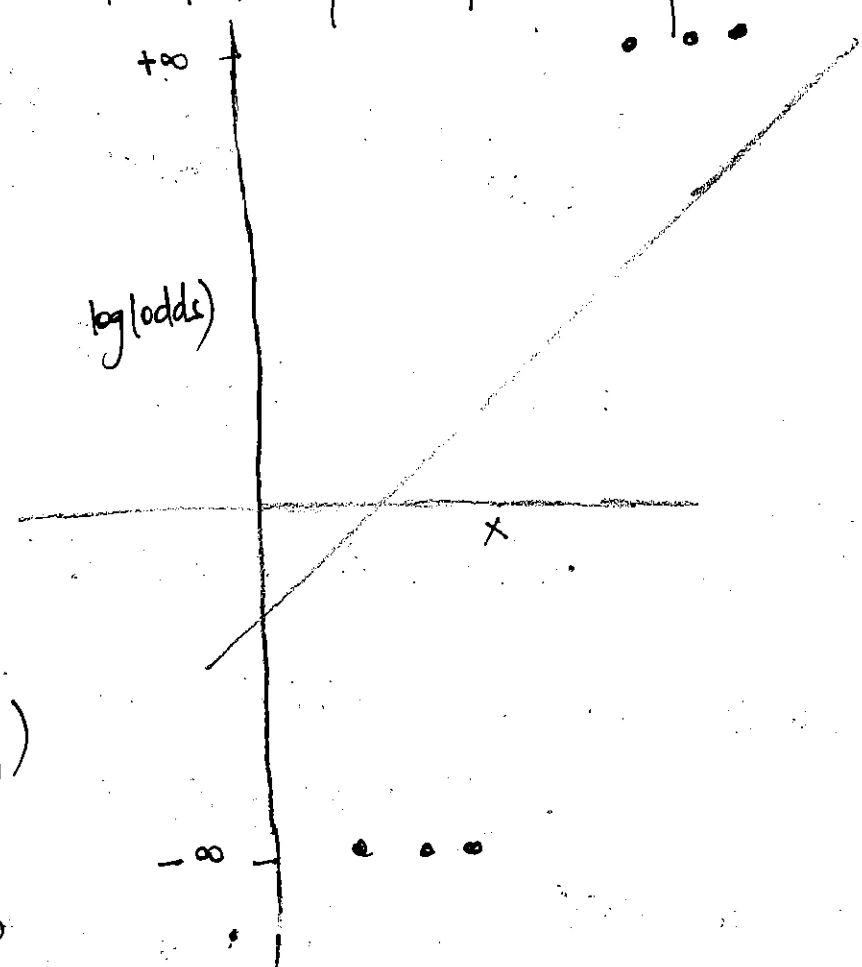


UNDERSTANDING MAXIMUM LIKELIHOOD ESTIMATION

Goal in logistic regression is to find the best fitting S curve for given data points

& In logistic regression, we transform the y-axis from the probabilities to log(odds)





If $p = 1$ (true label)

$$\log\left(\frac{p}{1-p}\right) = \log\left(\frac{1}{1-1}\right) = \log\left(\frac{1}{0}\right)$$

$$= \log 1 - \log 0$$

$$= 0 - -\infty = +\infty$$

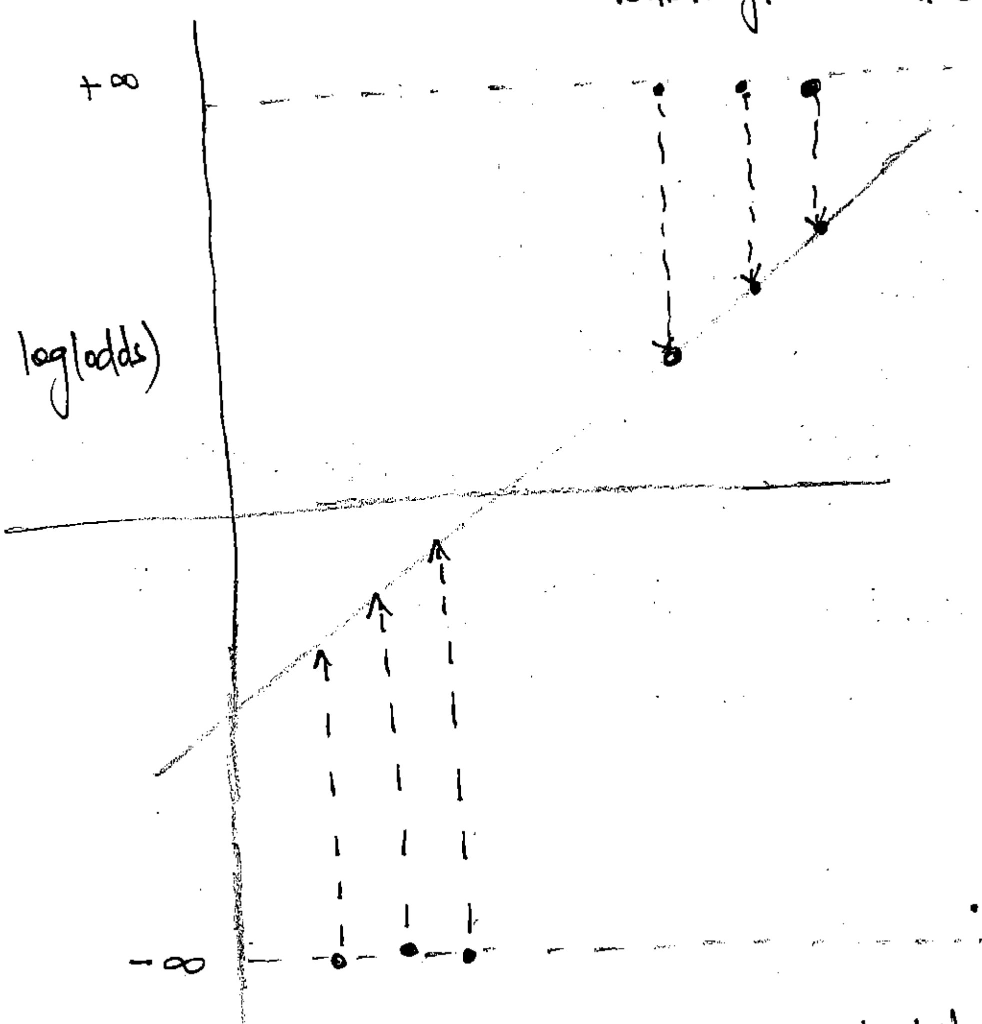
Similarly if $p=0$, $\log \frac{0}{1-0} = \log 0 = -\infty$

The log(odds) transformation pushes the data points to +ve & -ve infinity

<u>Side Note</u>: That's why we can't use least squares to find best fitting line as the residuals are also equal to +ve and -ve infinity

MLE → A likelihood fn is defined that calculates the probability of observing the outcome given input data & model (maximized)

Log(odds) of data pts is taken, now project these data pts onto the log(odds) line which gives each data pt, a log(odds) value



Then transform this log(odds) value to probabilities using

$$p = \frac{e^{\log(odds)}}{1 + e^{\log(odds)}}$$

After calculating prob, plot them on S-curve

Now keep rotating the log(odds) line
projecting data pts onto it
Transforming it to probabilities
Calculating the log likelihood (until maximized) (e.g. using logit fn)

Note: The algorithm that find the line with max. likelihood does so in a way that it increases log likelihood each time it rotates the line. So in few rotations, we get optimal fit

Methods that maximize log likelihood:

- Newton Raphson method
- Fixed point iteration
- Bisection method
- Muller's method
- Gradient Descent (minimizes negative log likelihood)

In general, to maximize log likelihood, take derivative of likelihood fn & equate it to zero

MLE does not tell "how" to find the optimal value of θ, it just tells how one value of "θ" is "more likely" than others (optimization algos like GD tell "how" to find this optimal value)

DECISION TREES

- AIM: The aim of decision trees → reduce impurity i.e. splitting the nodes using features which lead to maximum improvement in impurity i.e. moving towards "pure" nodes
(performing a split when building a decision tree ⇒ dividing up the feature space)

- IMPURITY MEASURES: (SPLITTING CRITERIA)

1) GINI SCORE: or GINI INDEX

prob. of incorrectly classifying an element (node value from start of tree), if randomly chosen

$$G_i = 1 - \sum_{K=1}^{n} p_{i,K}^2$$

G_i → Gini score at node i

$p_{i,K}$ = ratio of class K instances to total instances at i^{th} node
& n = total no. of classes.

- (does binary splits only)
- Higher Gini = ~~higher~~ lower homogeneity

e.g. $1 - \left(\frac{0}{54}\right)^2 - \left(\frac{49}{54}\right)^2 - \left(\frac{5}{54}\right)^2 \approx 0.168$

2) Entropy: $H_i = - \sum_{K=1}^{n} p_{i,K} \log(p_{i,K})$

measure of disorder; range [0, 1] (min, max)

Higher entropy = Lower homogeneity

H_i → Entropy at node i

$p_{i,K}$ = ratio of class K instances to total instances at i^{th} node

e.g. $-\frac{49}{54} \log \frac{49}{54} - \frac{5}{54} \log \frac{5}{54}$

Information gain = reduction in impurity = Entropy$_{parent}$ − Entropy$_{children}$ (Max)
(wt. avg of all subnodes that a parent node split into)
i.e. $\frac{n_{child1}}{n_{parent}} \cdot Entropy_{child1} + \ldots$
max gain amongst all features

∴ AIM of DT: impurity → 0

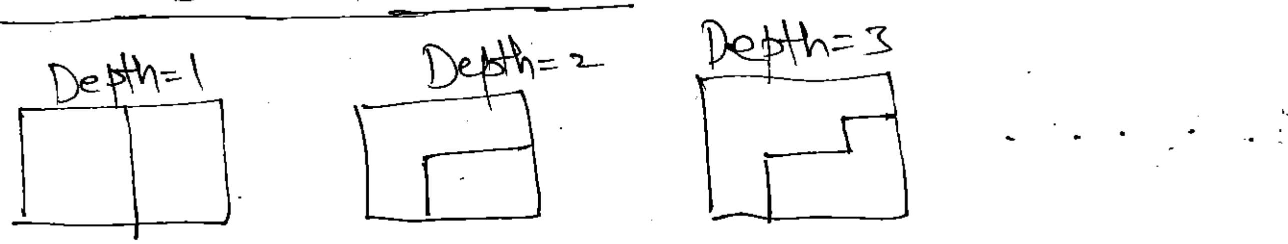
NOTE: Start with the node (feature) having most ~~least~~ purity (e.g. least gini score) & then select nodes which reduces impurity maximally

Stopping Cond.
1) max depth is reached
2) impurity does not decrease
3) Min num. of samples required to split an internal node is reached
4) Min. num of samples required to be at leaf node is reached

DT Algos:

1. ID3 – can produce decision trees with nodes that have more than 2 children
2. C4.5 } → use entropy / inf. gain
3. CART (Classification & Regression Tree)
   - greedy algorithm: searches for optimum split at each level without checking whether the split will lead to lowest possible impurity several levels down
   - produces only binary trees
   - uses gini

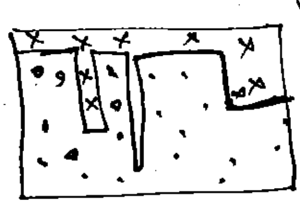
DECISION BOUNDARY OF DT:



- Decision boundary is parallel to some axes ~~or~~ perpendicular to other axes

REGULARIZATION:

- If depth of decision tree is increased, DT is prone to overfitting



- To regularize decrease max_* params or increase min_* params

PRUNING:

- Pruning is done to avoid overfitting
  If not pruned, the tree will come very large and will overfit the data
- Pruning deletes unnecessary nodes
  Remove the nodes whose child nodes provide no purity improvement i.e. if purity improvement is not statistically significant
  χ^2 test: Null hypothesis: Improvement is due to chance
  Hence if p-value > 0.05, prune the node

ESTIMATING CLASS PROBABILITIES: (INFERENCE | PREDICTION)

- Traverse the tree to find leaf node for the instance
- ratio of instances of class K to total instance at this node

ESTIMATING TARGET CLASS: Majority vote at leaf node
(PREDICTION)

NON-PARAMETRIC:

- DT are non-parametric: as they do not make an assumption on the distribution of data
  $\Rightarrow$ generally do not require centering or scaling

NON-LINEAR:

- DT are non-linear: non-linear relationship b/w dependent & independent variables

REGRESSION TREES: (outcome is continuous)
not feature

- Minimize MSE, instead of impurity
- Prediction is simply the average target value of instance associated with the leaf node

ADVANTAGES

1. Easy to understand & interpret
2. Standardizing the data is not required
3. "Feature Selection" inbuilt as less informative features are not part of decision tree
4. Can handle missing values

DISADVANTAGES:

1. Prone to over-fitting
2. Sensitive to small changes in data : a slight change in data can result in a very different tree

DIFFERENCE B/w STOPPING COND. & PRUNING:

Stopping cond. = criteria for stopping the growth of tree as soon as cond. is met

Pruning = look few steps ahead & decide whether we want to stop

1) make DT to a large depth
2) start at the bottom & start removing leaves which are giving -ve returns compared to the top
3) Suppose a split is giving -10 gain (i.e. loss) and next split gives gain of 20. A simple DT will stop at -10 but in pruning we will see overall gain of +10 and keep both leaves

Note: sklearn DT does not support pruning but xgboost supports it.

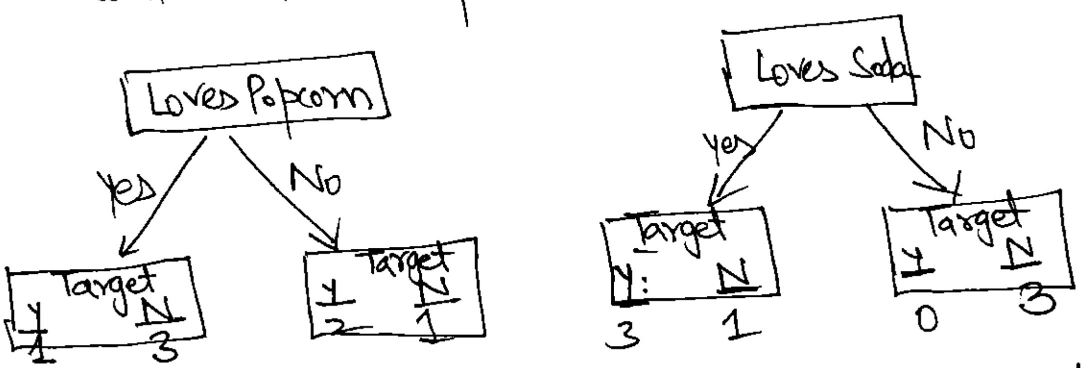
NOTE: In tree-based models, the nodes at the top of tree provide max gain → this fact can be utilized to get feature importance.

UNDERSTANDING DECISION TREES

Consider the dataset

F1	F2	F3	Target
Loves Popcorn	Loves Soda	Age	Loves Cool As Ice
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

1. Find the feature (along with its split) with lowest impurity (Gini or Entropy)
   Note that when we are looking at a feature, we have to take into account its all possible values



Looking at the diagram, we can see "Loves Soda" ("No" branch) results in pure node whereas none of the branches of "Loves Popcorn" results in pure node. Hence "Loves Soda" does a better job in predicting the target ("Loves Cool As Ice")

Quantifying it via Gini impurity:

Gini Impurity for "Yes" branch of "Loves Popcorn"

$$= 1 - (\text{prob. of "Y" in target var})^2 - (\text{prob. of "N" in target var})^2$$

$$= 1 - \left(\frac{1}{1+3}\right)^2 - \left(\frac{3}{1+3}\right)^2$$

$$= 0.375$$

Gini Impurity for "No" branch of "Loves Popcorn"

$= 1 - (\text{prob of "Yes" in target var})^2 - (\text{prob. of "No" in target var})^2$

$= 1 - \left(\frac{2}{2+1}\right)^2 - \left(\frac{1}{2+1}\right)^2$

$= 0.444$

$\therefore$ Total Gini Impurity for "Loves Popcorn" (includes both branches)

= Weighted avg. of both branches

$= \left(\frac{1+3}{1+3+2+1}\right) \times 0.375 + \left(\frac{2+1}{1+3+2+1}\right) \times 0.444$

$= 0.405$

Similarly, Total Gini Impurity for "Loves Soda" (both branches)

$= 0.214$

Hence, we choose "Loves Soda" as the root node

The Final tree looks like this: (upto this pt.)

"Yes" branch will now only see the dataset where "Loves Soda" value is "Yes".

Similarly "No" branch will now only see the controlled dataset where "Loves Soda" value is "No".

Loves Popcorn	Loves Soda	Age	Target
Yes	Yes	7	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes

2. Now on this controlled dataset, use other features (e.g. Age or Loves Popcorn) to find next best feature for splitting.

NOTE: We excluded "Age" in our discussion for simplicity. In reality, we would compute impurity for "Age" as well. Since it is a numerical feature, the splits creation is different.

Creating splits for "Age" feature

i) Sort the "Age" feature

Age	Target Loves Cool As Ice
7	No
12	No
18	Yes
35	Yes
38	Yes
50	No
83	No

→ 9.5
→ ~~17~~ 15
→ 26.5

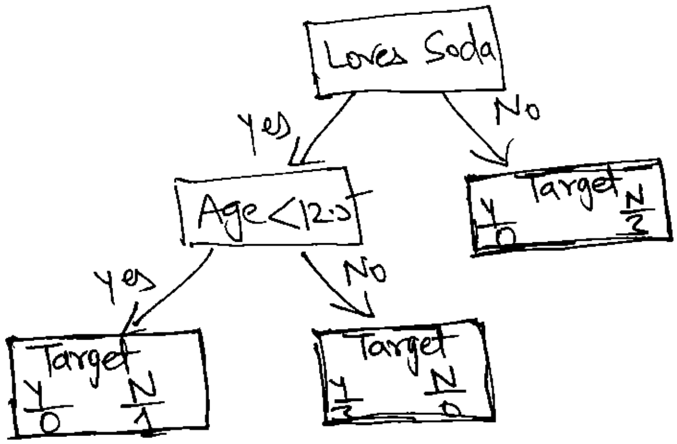
ii) Take avg of consecutive "age" feature value as split points

e.g.

iii) Calc. gini impurity of all split points & wt. avg. for "Age" feature

---

Let's say the next best feature to split is "Age" and we end up with a tree like below:



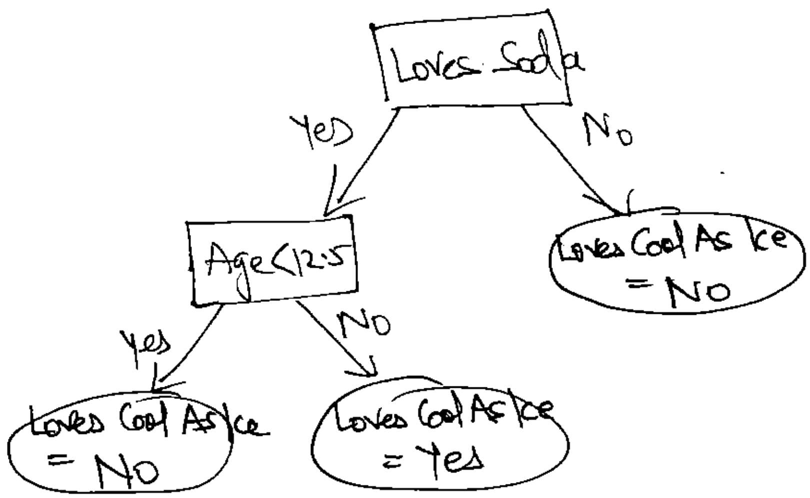
3. Since all nodes are pure, there is no need to continue building tree (in practice, stopping condition is used to end building tree)

4. Now, we need to assign output values to each leaf:

Classification : Majority vote

Regression : Mean of values

So, the final tree is:

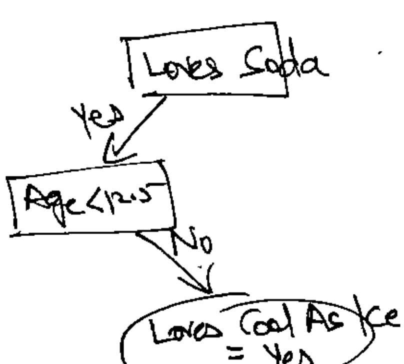


INFERENCE / PREDICTION:

If a new data point comes along with the following values:

F1 Loves Popcorn	F2 Loves Soda	F3 Age	Target Loves Cool As Ice
Yes	Yes	15	??

Traversing the tree:



BAGGING & BOOSTING

Bagging: sampling with replacement i.e. duplicate records/examples in diff. samples
(bootstrap aggregation)

Pasting: sampling without replacement i.e. no duplicate records/examples in different samples

Boosting:
- combining several weak learners into strong learner
- Train models sequentially, each trying to correct the mistakes of its predecessor

Note: row sampling & column sampling facility is provided by most implementation (before each boosting iteration)

BAGGING Vs BOOSTING: DIFFERENCE

BAGGING (Not RF since RF = bagging + random subset of features for each tree)	BOOSTING (All kinds of boosting but nothing specific to one boosting algo e.g. Catboost, GBM, Xgboost)
1. Parallel ensemble: Each model is built independently on diff. subset of data	1. sequential ensemble: next model to be trained corrects previous model's errors
2. AIM: To decrease variance, not bias	2. AIM: Primarily to decrease bias. However, variance is also reduced due to combining multiple trees
3. Trees are built independently in a parallel fashion hence easily parallelizable/distributed	3. Although trees are built sequentially, construction of diff branches or split finding procedure can be parallelized/distributed (available in Xgboost but not in standard GBM implementation)

WHY BAGGING MODELS like RF reduce variance & not bias whereas BOOSTING MODELS like GBDT/Xgboost reduce primarily bias and also variance?

RF

RF uses "fully grown" DTs (low bias, high variance)
The trees are made uncorrelated to maximize decrease in variance but the algo cannot reduce bias : hence need for large, unpruned trees so that bias is initially as low as possible (No pruning happens in RF)

GBDT/Xgboost

Boosting is based on "weak learners" i.e. "shallow trees (high bias, low variance), sometimes even as small as decision stumps (trees with 2 leaves). By sequentially training on residuals i.e. fixing mistakes, bias is reduced.

However, combining multiple trees reduces variance too.

RANDOM FOREST

RF = Bagging + "Feature selection" (random subset of features for each tree)

- Build n parallel & independent trees by:

  for each tree:
  1. selecting bootstrap sample (samples with replacement)
  2. selecting "random" subset of features

  random in Random Forest

  1 & 2 help in reducing over-fitting

- Combine predictions using
  - → majority voting : classification
  - → Avg. : regression

- SPLITTING CRITERIA: Same as decision trees (e.g. Gini, Entropy → class. or MSE → Regression) (based on target var not predictor)

- WHY SUBSET OF FEATURES & NOT SIMPLE BAGGING

  1) If subset of features not different for each tree
     - → learners (individual trees) will be highly correlated
     - → "Diversity" in ensemble decreases
     - → All learners will point in the same dir i.e. not a great way to combine

  2) Using subset of features helps in ↓ (reducing) over-fitting

- WHY NO PRUNING:
  - RF is robust to noise from each individual tree

- HYPERPARAMS
    1) Num. of trees (usually sufficient to tone)
    2) Size of bootstrap sample
    3) Num. of features for each tree

- EASILY PARALLELIZABLE:
    - Since trees are bilt independently in parallel

GBDT / GBM (GRADIENT BOOSTED DECISION TREES / GRADIENT BOOSTING MACHINES)

- Sequentially adds model/tree to an ensemble, each one correcting the errors made by the predecessor model
- In particular, the new model is fit to the "residual" error i.e. difference between true label and predictions from prev. model
- This "residual" is called "pseudo-residual" since it is the same as taking $\underbrace{\text{-ve gradient}}_{\text{"GRADIENT" in GBDT/GBM}}$ of loss fn (e.g. MSE) w.r.t. predictions

$$L_{MSE}(y, \hat{y}) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Now, let's take partial derivative of loss fn with respect to specific $\hat{y}_j$

$$\frac{\partial L_{MSE}(y, \hat{y})}{\partial \hat{y}_j} = \frac{\partial}{\partial \hat{y}_j} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Note: We can now remove summation since partial derivative of Loss fn for $i \neq j$ is 0

$$= \frac{\partial}{\partial \hat{y}_j} (y_j - \hat{y}_j)^2$$

$$= 2(y_j - \hat{y}_j) \frac{\partial}{\partial \hat{y}_j} (y_j - \hat{y}_j)$$

$$= 2(y_j - \hat{y}_j)(-1)$$

$$= -2(y_j - \hat{y}_j)$$

Dropping constant -2

$$= (y_j - \hat{y}_j) \rightarrow \text{"residual"}$$

i.e. Chasing Residual vector in GBM (magnitude + dir.) = Chasing (-ve) gradient of loss fn via gradient descent

[-ve sign is imp. since we want to move in opp dir of gradient for minimization in gradient descent]

- Each new tree corresponds to another step of gradient descent
- GBM/GBDT uses Loss Fn and gradient of loss fn (i.e. pseudo-residual) to Chase the "residual" error
- GBM/GBDT have both split criteria (how to build one tree) and loss fn (how to build next tree)
- GBM/GBDT does not use Gini or Entropy as splitting criteria instead it usually is based on loss fn e.g. MSE.
  (reason: trees in gbm predict gradient of loss (even for classification) which is a numerical value hence MSE is used)
- Loss fn like mse, mae (regression) or like fns logloss (classification) is used to calculate the residual on which the next tree is fit
- <u>How GBM is trained:</u>

1) Usually the first tree takes avg. of true label as the initial prediction for all examples/instances (P_0)
2) Next, residual or (true label − initial prediction) is calculated (R_0) (by taking gradient of loss fn)
3) Next, 2^{nd} tree is trained to predict the residuals calc. (i.e. residuals become the target) in step 2)
   (Splitting criteria : usually based on loss fn)
4) To make new prediction: (P_1)

$$P_1 = \underbrace{P_0}_{\text{(initial model pred e.g. avg.)}} + \text{Learning-Rate} \times (\text{Predictions from } 2^{nd} \text{ tree})$$

5) Again calc. residuals and repeat from step #2

X_1	X_2	X_3	Label/True Value (Y)	P_0	R_0	$P_1 = P_0 + LR \times$ (pred. from next tree)	R_1	...

- **PREDICTION USING GBM:**

Prediction = <u>sum of predictions of all models weighted by learning rate</u>

i.e. 1^{st} Model Pred + LR $\times$ 2^{nd} Model Pred.
+ LR $\times$ 3^{rd} Model Pred

⇒ "sum" since models are trained on residuals

⇒ Learning Rate: How quickly the error is corrected from one tree to next.

$0 < LR < 1$

If LR is low: more trees are needed to fit the training set
but
usually results in better generalization

- Over-fitting problem is usually seen in GBMs

- Boosting can be considered as "<u>Gradient Descent" in Prediction space</u>
since we are optimizing the combined model predictions and not model parameters (as in neural networks)

- HYPERPARAMS:
  - Num. of trees
  - Max depth of tree
  - Learning Rate
  - Row sampling & Column sampling facility provided by most implementations before each boosting round

XGBOOST

LOSS FUNCTION:

Xgboost, rather than explicitly fitting the pseudo-residuals (as in GBM), aims to minimize the following objective at each iteration

$$L^{(t)} = \sum_{i=1}^{n} l\Big(\underbrace{y_i}_{\text{label for instance } i},\ \underbrace{\hat{y}_i^{(t-1)}}_{\text{prediction from ensemble of trees until prev time step } (t-1) \text{ for instance } i} + \underbrace{f_t(x_i)}_{\text{prediction from the new tree at time } t \text{ for instance } i}\Big) + \underbrace{\Omega(f_t)}_{\text{regularization term for the new tree}}$$

The above eq. is a fn of fns i.e. includes fn as a parameter to another fn and cannot be optimized using traditional optimization methods in euclidean space [as mentioned in xgboost paper]

Hence, 2nd order approximation is used to optimize the above loss fn utilizing Taylor Series Expansion

Note: Taylor Series Expansion is used to calc. the value of entire fn at every point, if the value of the fn and all of its derivatives are known at a single point

Since y_i & $\hat{y}_i^{(t-1)}$ are constants in above eq.

$$L^{(t)} = \sum_{i=1}^{n} l\big(f_t(x_i)\big) + \Omega(f_t)$$

Now applying Taylor Series Expansion

$$= \sum_{i=1}^{n} \Big[\underbrace{g_i}_{\text{gradient of loss fn w.r.t prediction for instance } i} f_t(x_i) + \frac{1}{2} \underbrace{h_i}_{\text{hessian of loss fn w.r.t. prediction for instance } i} f_t^2(x_i)\Big] + \Omega(f_t)$$

GRADIENT & HESSIAN CALCULATION:

To minimize the objective fn, XGBoost uses gradient descent. In each iteration, the first and second order derivatives (gradient and Hessian) of the loss fn are calculated w.r.t. predicted output of each instance in the dataset, giving vectors 'g' and 'h' with gradient & hessian values for each instance

Note: Hessian provides a more precise estimate of the dir. of highest decrease of loss fn which allows the model to converge faster.

CONSTRUCTION OF TREES:

In the t^{th} iteration, the best tree f_t that minimizes the objective fn is added using the calculated gradients and Hessians in a greedy fashion (or approx. greedy for large datasets)

- Start with single ~~node~~ (containing residuals of all instances) (obtained from default initial pred.)
- Iterate over i) all features and values per feature for greedy or ii) cand. split pts. obtained from approx. greedy / weighted quantile sketch and evaluate possible split loss reduction [For larger datasets, Xgboost partitions features into quantiles instead of scanning all feature values]

$$\text{GAIN} = \text{loss}_{\text{parent}} - (\text{loss}_{\text{left branch/child}} + \text{loss}_{\text{right branch/child}})$$

[Gain must be > min_split_gain hyperparam (Gamma) else stop growing the branch]

Gain is fn of gradient & hessian of left and right branches + λ (lambda) (L2 regularization term) ~~(and γ (gamma) to determine min. loss reduction required for a split)~~

Gain eq^n combines both loss reduction & regularization term helping prevent over-fitting & making optimal trade-off b/w complexity & predictive power

ILLUSTRATION

Feature X	Target Y
10	100
15	90
20	110
25	75

Step 1: Let's say initial Prediction = avg. of target values
(same as GBM)

$$= \frac{100+90+110+75}{4} = \frac{375}{4} = 93.75$$

Step 2: Calc. residuals for each instance
(same as GBM)

Feature X	Target Y	Residual
10	100	6.25
15	90	-3.75
20	110	16.25
25	75	-18.75

Step 3: All the residuals go to the root node
(Diff. than GBM)
from now on

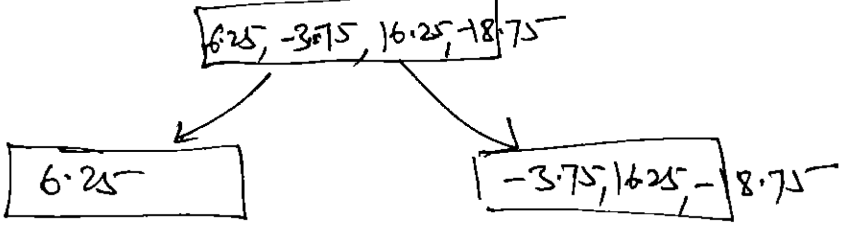
$\boxed{6.25, -3.75, 16.25, -18.75}$ → Root Node

Step 4: Now the first cand. split pt is taken (either from greedy or approx. greedy / weighted quantile sketch method)
Let's say it is 12.5
then all residuals whose ~~insta~~ feature value is < 12.5 go in left subtree
& remaining i.e. ≥ 12.5 go in right subtree

$\boxed{X < 12.5}$

left subtree → 6.25   -3.75, 16.25, -18.75 ← right subtree

Step 5: Now loss is calc. for parent (with all residuals) and left & right subtree.
i.e.

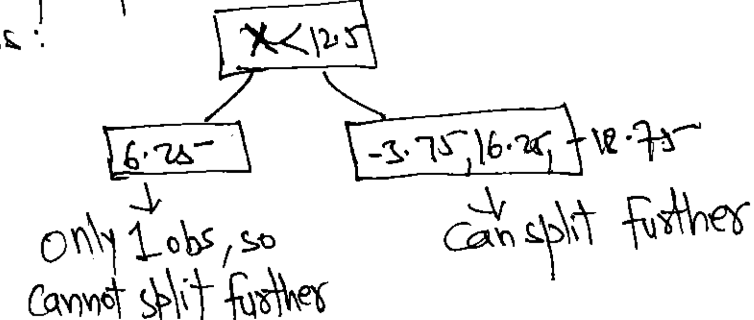


loss is a fn of gradient & hessian of instances in respective nodes + reg. param (λ)

& finally, gain is calculated

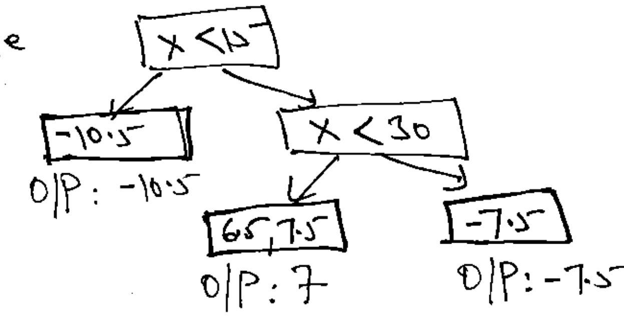
$$\text{GAIN} = \text{loss}_{\text{parent}} - \left(\text{loss}_{\text{left subtree}} + \text{loss}_{\text{right subtree}}\right)$$

Step 6: Gain is calc. for all candidate split pts and the cand split pt. with max gain is chosen as the actual/true split pt.
Let's say optimum split pt is $X < 12.5$ then the tree looks like this:



HOW TO PREDICT / OBTAIN OUTPUT VALUE FROM LEAF NODES:

Let's say, this is the final tree



(Leaf nodes are residuals)

$$\text{O/P value from leaf node} = \frac{\text{Sum of values (i.e. residuals)}}{\text{No. of values (residuals)} + \lambda \text{ (reg. pa}}$$

($\lambda = 0$ in calculation of O/P values for this tree)

HOW IS HESSIAN CALCULATED, SINCE IT IS COMPUTATIONALLY EXPENSIVE TO CALCULATE 2nd Order Derivatives

Ans: Xgboost uses a diagonal approximation to the Hessian.
A diagonal 'n×n' matrix has atmost 'n' non zero elements.
The diagonal approximation scales nicely bcz it grows linearly in 'n', as opposed to dense Hessian which grows quadratically.

HANDLING OF MISSING VALUES IN XGBOOST → SPARSITY-AWARE SPLIT FINDING ALGO

At the time of training, the instances with missing values of a feature are placed on both left and right branch of split and Gain is calculated. The side where gain is maximized, becomes the default dir. whenever this feature value is missing (e.g. during prediction if the example has this feature value missing)

Illustration:

Feature → X	Target → Y	Residual
10	-7	-6
20	8	-8
25	10	+12
30	12	+19
?	18	-10
?	10	+8

split into → 2 tables (w/o missing values & only with missing values)

X	Y	Residual
10	-7	-6
20	8	-8
25	10	12
30	12	19

At each split point for X e.g 15, 22.5, 27.5 all the missing value instances are first placed on left then right and "default" placed wherever gain is maximized

X	Y	Residual
?	18	-10
?	10	8

Note: Residuals are used in finding best split pts.

<u>SPLIT FINDING</u> < Exact Greedy
Approx. Greedy (→ Weighted Quantile → Sparsity Aware Split findi
↓ makes Xgboost parallelizable/distributed

Xgboost provides a hyperparam to choose either i) exact greedy
ii) Approx. greedy

EXACT GREEDY → Enumerates 'all' possible splits on 'all' features
→ In order to do so for a feature, all values of a feature must be stored and sorted → becomes an issue when dealing with continuous features in large datasets

→ Most single machine boosting implementations e.g. GBM in sklearn, R' gbm or single machine version of Xgboost support exact greedy

→ Though exact greedy can be parallelized to some extent (one feature on each node, if node can accomodate all values of one feature, usually not possible in large datasets), it is computationally expensive and parallelization gains are much lower than approximate greedy

(PARALLEL LEARNING)
APPROX GREEDY: & (WEIGHTED QUANTILE SKETCH)
→ Enumerating 'all' possible splits on 'all' features is impossible to efficiently compute when the data <u>does not fit in memory or split needs to be found in distributed setting</u> ⟹ Approx. Greedy

→ Cand. split points are proposed based on the ~~percentiles~~ quantiles of feature distribution. E.g. if a continuous feature has values sorted from 1...100, then the cand. split points are 10, 20, ... 90.

→ If the dataset is huge, sorting and finding quantile cannot be done on a single machine. So the dataset is chopped and put on diff. machines. (distributed)
↓
<u>Quantile Sketch Algorithm</u> combines the quantiles value from each machine to make an approx. histogram, which in turn is used to calc. approx. quantiles of the full dataset

→ When every instance in the dataset has equal wt → Quantile sketch
When instances in the dataset have unequal wt → <u>Weighted Quantile Sketch</u>

E.g. In regression, all instances have equal weight
however, in classification, instances could have wts in imbalanced datasets
(e.g. scale_pos_weight hyperparam)

→ The 'weight' in Weighted Quantile sketch is a fn of hessian for the instance

DIFFERENCE BETWEEN XGBOOST VS GBM

Unique Features of Xgboost (not found in GBM) [Also Advantages]

- Regularization term in Loss fn + Gamma : min loss needed for split
- Use of Hessian → xgboost converges faster
- Approximate Greedy Algo → for larger datasets (+ exact greedy for smaller datasets) (GBM uses only exact greedy algo)
- Parallel Learning (Xgboost parallelizes the construction of 'one' tree by building several nodes at a given depth simultaneously; standard GBM does not have this parallelizing implementation)
- Weighted Quantile Sketch
- Sparsity Aware Split Finding
- Cache-aware access (Xgboost caches gradient & hessian in CPU cache for quick access)
- Blocks for out-of-core computation (Xgboost - If dataset is too large to fit in memory, it must be stored on hard drive but reading & writing to hard drive is super slow, so Xgboost ~~compresses~~ compresses the data while storing on hard drive. decompressing the data while reading is faster than reading the uncompressed data)

- Pruning difference: Xgboost splits upto max-depth and then start 'pruning' the tree backwards & remove split beyond which there is no 'net' gain
  GBM uses greedy approach and stops as soon as '-ve' gain is seen at a split

CONS OF XGBOOST:

- Doesn't support categorical features natively i.e. categorical features need to be encoded before feeding to xgboost algo
- Somewhat sensitive to outliers as new trees are fit to fix errors made by prev. trees (though regularization helps)

'EXTREME' in Xgboost: Refers to the engineering goal to push the limit of computation resources for boosted tree algos

POPULAR HYPERPARAMS TO TUNE:

1. Scale_pos_weight: Useful for imbalanced datasets should be set to $\frac{\#\text{majority class instances}}{\#\text{minority class instances}}$
2. Eta (shrinkage or Learning Rate)
3. L1 (alpha), L2 (lambda)
4. Gamma: Min. loss reduction to create a split
5. Subsample: Fraction of data to train on
6. Colsample: Fraction of features to train on

<u>K-MEANS</u> <u>CLUSTERING</u>

Step1 Pick centroids

Step2 Calc. dist of every pt to ~~each~~ centroid and assign pts to nearest centroid (cluster)

Step3 Now for every cluster, calculate the mean of all data points in that cluster → New centroid

---

Repeat step2 Calc. dist of every pt (irrespective of current cluster) to the new centroid and re-assign pts to nearest centroid (cluster)

Repeat step3 Now for every cluster, calc. the mean of all data points in that cluster → New centroid

- Continue until:
  1) Threshold for the difference b/w cost fn value b/w iterations is reached
  2) Threshold for the difference b/w ~~position~~ of centroid b/w subsequent iterations is reached.

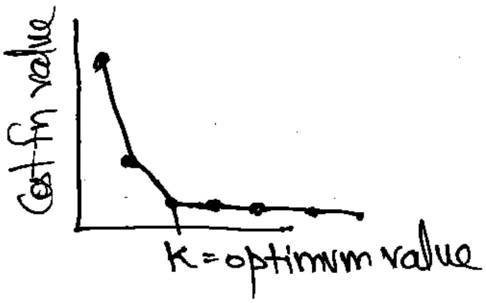
<u>COST FN OF K-MEANS:</u> $\sum^{K} | x_i - \mu_k |^2$ → within cluster SSE

μ_k ← centroid of the cluster where pt i belongs

<u>HOW TO FIND K of K-Means:</u> ELBOW METHOD:

- The elbow method plots value of cost fn produced by different K.

- Intuitively : if K increases, distortion or within cluster SSE will decrease.



HYPERPARAMETER TO TUNE: K

Advantages

+ Scalable

$O(nKi)$ — n: no. of data pts; K: no. of clusters; i: no. of iterations

DisAdv

- K beforehand
- Non deterministic, depending on initial choice of centroids
- Centroids can be non-existent sample

AFFINITY PROPAGATION

EXEMPLARS: Data points representative of clusters

SIMILARITY FN: – Euclidean Distance (in general, can be other fn too)

Pairwise similarities between data points are taken as input and clusters are found by maximizing the similarity between data points and Exemplars.

IMPORTANT HYPERPARAMETERS:

Preference: Threshold Similarity value b/w Exemplars and other data points in the cluster

∴ Low preference = cluster size will be large but fewer clusters

High preference = cluster size will be smaller (compact) but large no. of clusters

Advantages:

+ No K beforehand
+ Deterministic
+ Exemplars are real data points

DisAdvantages:

- Not scalable
  $O(n^2 i)$
  n = no. of data points
  i = iteration

DBSCAN

Each data point:

1) Core point: if Min. no. of neighbouring points (MinPts) fall within a specific radius ε

2) Border point: Falls within radius ε of a core point but has fewer neighbours than Minpts

3) Noise point: Except core & border point

Notion of Density: no. of points within radius ε

- Form a separate cluster for each core point or connected grp of core points
- Assign border point to the cluster of its corresponding core point

HYPERPARAMETERS TO TUNE

1) MinPts 2) ε - radius

Adv:

+ Arbitrary shaped clusters
+ No K beforehand
+ Notion of Noise - assigned -1 - useful in anomaly detection

DisAdv:

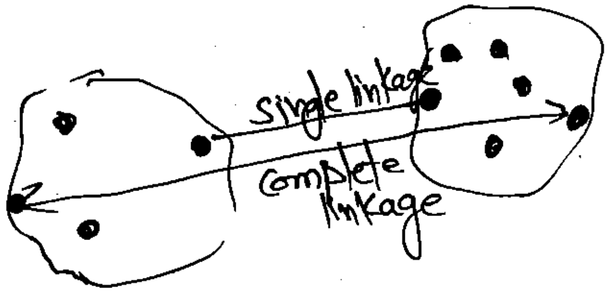
- Hyperparameter ε & MinPts to tune
- Not entirely deterministic, assignment of border pts can be to either neighbouring cluster depending on order of data being processed
- Not scalable

HIERARCHICAL CLUSTERING

Agglomerative
Every pt is a cluster & then merge

Divisive
All pts belong to one cluster and then break

CRITERIA FOR MERGING:



1) Single Linkage – minimum
2) Complete Linkage – maximum
3) Average Linkage – $\frac{min + max}{2}$
4) Ward's Linkage: ~~Those~~ clusters are merged which leads to minimum increase of total within-cluster SSE

1) Compute dist matrix of all data pts
2) Merge according to criteria (min, max, avg, ward)
3) Update dist. matrix
4) Repeat 2 & 3 until one cluster remains.

HYPERPARAMETER: LINKAGE

– In sklearn implementation of Agglomerative clustering, K can be pre-specified

Adv

+ No K beforehand
+ Deterministic
+ Dendograms to understand structure in data

DisAdv

- Members/Data pts when clustered, do not unmerge
- Not scalable

Time complexity of Agglomerative with single linkage:

Worst case: $O(n^3)$ Best case: $O(n^2 \log n)$

i) dist. from every pt. to every other pt needs to be calc. (n^2)

ii) search thru this distance matrix to find closest pt (n or $\log n$)

linear search binary search

GMM:

- Probabilistic Clustering
  - ~~Deterministic~~
- K - No. of Gaussian Mixture (Gaussian Distr.) needs to be pre-specified.

✓ ASSESSING THE QUALITY OF CLUSTERING:

1) Visualization, if possible (if no. of dim > 3, difficult)
2) Silhouette plots & scores
3) If labels present ⇒ all classification metrics

SILHOUETTE PLOTS:

Silhouette coeff is calculated for every data point

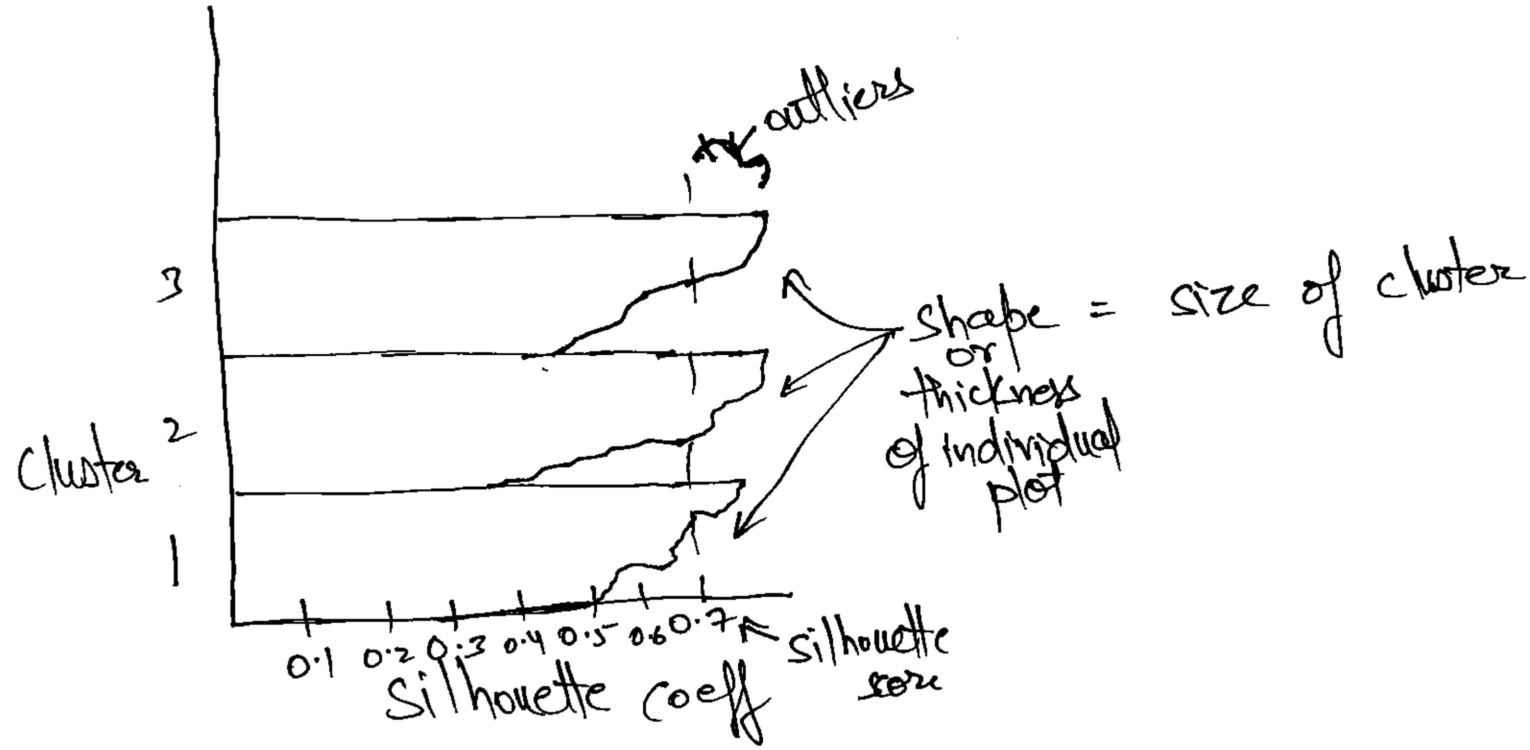
$$s = \frac{b-a}{\max\{b,a\}} = [-1, 1]$$

a = cluster cohesion, avg dist b/w a data point and all other data points in same cluster

b = cluster separation, avg dist b/w a data point and all other data points in nearest cluster

If $s = 0$ if cluster separation and cohesion are equal

Silhouette score (for a cluster) : avg. silhouette coeff of all data point in a cluster



outliers
3
Cluster 2
1
Shape or thickness of individual plot = size of cluster
0.1 0.2 0.3 0.4 0.5 0.6 0.7
Silhouette coeff
silhouette score

KNN

1. Choose the number K and a <u>distance metric</u>
2. Find K nearest neighbors of the sample that we want to classify.
3. Assign the class label by majority vote. (avg in case of regression)

In case of a tie (if K = even) the answer is implementation specific. In sklearn, the algorithm will prefer the neighbours with closer distance to the sample.

CHOICE OF K:

1) <u>Odd</u> (so that majority can be calculated) in case of 2 class problem
2) $K = \sqrt{n}$ where n = no. of samples
3) If K is very high → under-fitting
   K is small → over-fitting
   Perform <u>cross-validation</u>

WEIGHTED KNN

wts could be assigned to K nearest neighbors to make the final prediction.

COMPUTATIONAL COMPLEXITY OF KNN (test/prediction time)

If d = no. of dimensions

$O(dn)$

However, efficient data structures like KD-tree can reduce time complexity at the cost of increased training time.

LAZY LEARNER & INSTANCE BASED LEARNING

Models based on instance based learning are characterized by memorizing the training dataset and lazy learning is a special case of instance based learning that is associated with no(zero) cost during the learning process.

NON-PARAMETRIC:

- Does not make explicit assumption about the form of target fn.
- It cannot be characterized by a fixed set of params and the number of params grow with training data

CURSE OF DIMENSIONALITY:

- Prone to over-fitting

  This is because when no. of features increase compared to size of training set, even closest neighbors are too far away in a high-dimensional space to give a good estimate

<u>PROS:</u>

1. Immediately adapts to the new training data
2. Somewhat insensitive to outliers
3. No assumption about data

<u>CONS:</u>

1. Computational complexity is high for classifying new samples.
2. Storage req are high as all data is needed at prediction time.
3. Sensitive to scale in data i.e. standarization must be done before applying it.

DISTANCE / SIMILARITY METRIC

1. Euclidean (real-valued)

L2 norm : $d(x,y) = \sqrt{\Sigma (x_k - y_k)^2}$

2. Manhattan (real valued)

L1 norm : $d(x,y) = \Sigma |x_k - y_k|$

3. Cosine Similarity (real valued) $\cos\theta = \frac{\vec{a} \cdot \vec{b}}{|a|\,|b|}$ $[-1,1]$

judgement of orientation rather than mag.

$= 1$ similar

$= 0$ dissimilar

$= -1$ in opp. dir

$[1,2]$ $[100,200]$ → cosine similarity high but euclidean distance → far off points

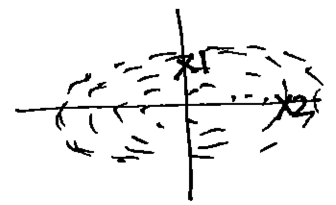
4. Pearson correlation coeff (real valued) (trend)

$$r = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\Sigma (x_i - \bar{x})^2} \sqrt{\Sigma (y_i - \bar{y})^2}}$$

$[-1,1]$

5. Mahalanobis distance (real-valued)

- Useful if features/attributes are correlated
- Accounts for the fact that variances in each dir is different



In euclidean space X1 is near to origin

· but variance in X1 dir is less to origin

∴ X2 is near if mahalanobis distance is considered since normalized by variance in that direction

6. Jaccard Similarity (categorical)

$$\frac{A \cap B}{A \cup B}$$

NOTE: If observation contains both real & cat. values

real → dist. metric for real valued

cat. values → dist. metric for categorical

∴ compare observations by real & cat. dist. metric separately.

NAIVE BAYES:

$P(A \text{ and } B) = P(A) * P(B)$ if events A & B are independent

Conditional probability:

$P(A \text{ and } B) = P(A|B) * P(B)$ if events A & B are not independent

Bayes Theorem:

$$\underbrace{P(O|E)}_{\text{Posterior prob.}} = \frac{\overbrace{P(E|O)}^{\text{Likelihood}}\ \overbrace{P(O)}^{\text{Class prior probability}}}{\underbrace{P(E)}_{\text{Predictor prior probability}}}$$

It is easier to calculate conditional probability in one direction; often we know freq of some evidence, given known outcome, we use it to calculate the reverse

E.g. Prob. of Disease D given Test-positive

$$= \frac{\text{Prob (Test is +ve | Disease)} * \text{Prob(Disease)}}{\text{(scaled by) Prob (Testing +ve, with or without disease)}}$$

NAIVE BAYES: In reality, there are multiple evidence and in naive bayes we calculate class probabilities of each outcome & select the outcome with higher prob.

$$P(C|X) = \frac{P(X|C)\ P(C)}{P(X)}$$

$P(X)$ → need not calc. since common to all classes / outcomes

If all predictors / attributes / features are assumed to be independent, → Naive of "Naive Bayes".

$$P(C|X_i) = P(X_1|C)\ P(X_2|C) \ldots P(X_n|C) \times P(C)$$

(Multi-variate setting) If 0 then everything = 0 (zero freq problem)

Intuition behind multiplying by prior is to given higher prob to more common outcome → base rate

Ex1: Swine Flu is suspected to affect 1 in 10,000 people. Test is 99% accurate with FP = 1%, FN = 0. You test +ve. What is the probab. that you have swine flu?

$P(D)$ = prob. of swine flu $\quad P(T)$ = prob. of +ve test

$$\therefore P(D|T) = \frac{P(T|D)P(D)}{P(T)}$$

which can be rewritten as:

$$P(D|T) = \frac{P(T|D)\,P(D)}{P(T|D)P(D) + P(T|ND)\,P(ND)}$$

where $P(ND)$ = prob. of not having swine flu

$P(D) = 1/10000 = .0001 \rightarrow$ prior

$P(T|D) = 1$

$P(T|ND) = .01$ (1% chance of false +ve)

$P(ND) = 1 - P(D) = .9999$

		Actual T	Actual F
Pred	T		1
	F	0	

$$P(D|T) = \frac{1 \times .0001}{1 \times 0.0001 + 0.01 \times 0.9999} \approx 0.01$$

$\therefore$ even though your test is +ve, your chance of having swine flu is 1% (compared relative to population)

Ex2:

Weather	Play
Sunny	No
Overcast	No
Rainy	No
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No

Prob. of play=Yes if weather is sunny

$$P(\text{Yes}|\text{Sunny}) = \frac{P(\text{Sunny}|\text{Yes}) \cdot P(\text{Yes})}{P(\text{Sunny})}$$

$$= \frac{(2/3) \times 3/7}{3/7} = \frac{2}{3} = 0.667$$

$$= 66.67\%$$

Adv. of NB: → Fast

Disadv of NB → a) Indepence of features may not be true

b) Zero problem:

$$P(C|X_i) = \underset{0}{P(X_1|C)} \times P(X_2|C) \ldots$$

then overall $= 0$

→ MAP is the foundation for Naive Bayes

→ In case of two classes, we are assuming our data is drawn from two distributions.

i.e. we have a bunch of data where we know the class, and want to be able to predict $P(\text{class} \mid \text{data point})$

$$P(\text{spam} \mid w) \propto \prod_i p(w_i \mid \text{spam})\, p(\text{spam})$$

$$P(-\text{spam} \mid w) \propto \prod_i p(w_i \mid -\text{spam})\, p(-\text{spam})$$

Whichever is bigger of two, wins!
(Maximizing posterior)

→ Since $P(C \mid x_i) = \underbrace{P(x_1 \mid C_1) \times P(x_2 \mid C_2) \times P(x_3 \mid C_3) \times \cdots \times P(C)}$

Can lead to numerical underflow as $[0,1] * [0,1] * \cdots =$ very small value

Take log:

$$= \log P(C) + \sum \log P(x_i \mid C)$$

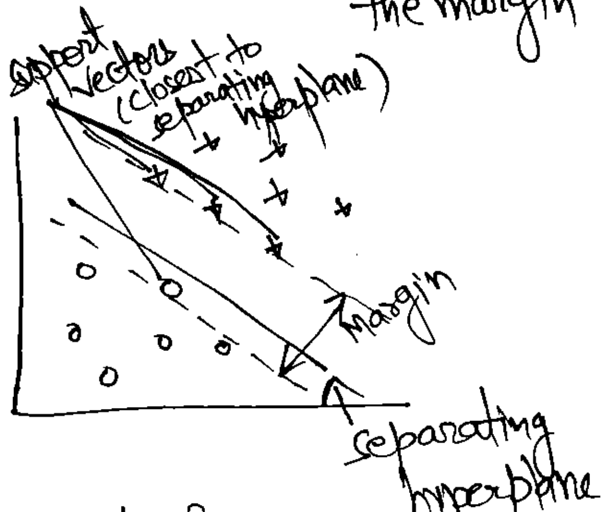
→ If ind. variables are continuous = Use Gaussian NB which calculates prob. using a continuous function

<u>SVM</u>

Objective Fn: $\underbrace{\frac{1}{2}\|w\|^2}_{\text{Maximize the margin}} + \underbrace{C\sum\xi}_{\text{penalty on misclassification}}$

→ also called as ~~[illegible]~~ margin constant

⇒ Large C (hard margin) → higher penalty on misclassification ∴ the left & right margins shrink
= low bias, high variance



Small C (soft margin) → lower penalty on misclassification, ∴ the left & right margins expand
= High bias, low variance

Hard Margin classifier: no points are allowed to cross the margin

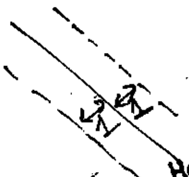
Soft Margin classifier: some points are allowed to cross the margin

A Linear classifier is of the form

$$f(x) = w^T x + b$$

(essentially an eq. of a line)

↑ dot product of two vectors: wt vector and training vector



Hard Margin: $\begin{cases} w^T \cdot x + b \geq 1 \rightarrow \text{one class} \\ w^T \cdot x + b \leq -1 \rightarrow \text{other class} \end{cases}$
only if data is linearly separable without misclassification

Soft margin: $w^T \cdot x + b > 0 \rightarrow class_1$, $w^T \cdot x + b < 0 \rightarrow class_2$

The decision boundary defined by a hyperplane is said to be linear because it is linear in input space.

Linear SVM works well if decision boundary is linear in input space

What if decision boundary is non-linear in input space?
— Kernel trick

Transform the training data into higher dimensional space via a mapping function ϕ. and train a 'linear' SVM model to classify data in this new feature space. Then new unseen data can be transformed using the same mapping function ϕ to classify using linear SVM. ⇓ computationally very expensive
soln → use Kernel trick

Note: The hyperplane always remains linear in higher dimensional space. However, in the input feature space, this hyperplane takes the form of a curve.

Kernel trick: A method to map features to new feature space, computing dot product (essentially producing a scalar) without being computationally expensive ⇒ achieve this without explicitly doing it thru a diff. series of operations

A Kernel is a function that returns the same value as the dot product of its corresponding mapped feature vectors given original feature vectors.
In the new feature space, the classifier becomes $f(x) = w^T \phi(x) + b$

E.g. polynomial Kernel, RBF (Gaussian) Kernel

Kernel can be interpreted as → similarity fn b/w a pair of samples
very similar ≈ 1, 0 is very dissimilar

Learning the coeff. of hyperplane: such that margins are maximized & misclassification is low

⇓

Quadratic Programming Problem
solved using
SMO (sequential Minimal Optimization)
sub gradient descent

Adv of SVM:

1. Can classify data points when decision boundary is non-linear
2. Less overfitting as obj fn has regularization parameter C

Disadv. of SVM:

1) Can be computationally very expensive – both memory wise & training time wise
2) Choosing the right kernel & tuning hyperparameters C, gamma can be tricky
3) Not very robust to outliers
4) Fundamentally a binary classifier, in multiclass setting – one vs Rest paradigm.

PRACTICAL CONSIDERATIONS WHEN APPLYING SVM:

1. Choosing C.
2. Choosing Kernel

Linear Polynomial Gaussian

A smaller gamma in higher dimensions = pointed bump & vice-versa for larger gamma
↑ low bias & high variance & vice-versa for larger gamma

3. Choosing Gamma (in case of Gaussian Kernel)

gamma = 0.1 gamma = 1 gamma = 10

Note: If multiple of the above has to be chosen: Gridsearch

Kernel Trick: Kernel methods represent the data only thru a set of "pairwise similarity" comparisons between the original data observations X (with the original co-ordinates in the lower dimensional space), instead of explicitly applying the transformation $\phi(x)$ and representing the data by these transformed co-ordinates in the higher dimensional feature space.

In Kernel methods, the dataset X is represented by $n \times n$ Kernel matrix of pairwise similarity comparisons where the entries (i,j) are defined by kernel function $K[x_i, x_j]$

Two observations in the dataset

The Kernel function acts as modified dot product.

The ultimate benefit of the kernel trick is that the obj. function we are optimizing to fit the higher dimensional decision boundary only includes the dot product of transformed feature vectors. ∴ we just substitute these dot product terms with the kernel fn and we don't even use $\phi(x)$

→ Let w be the minimizer of the SVM problem for some dataset with m examples $\{x_i, y_i\}$

Then for $i = 1, \ldots m$ there exists $\alpha_i \geq 0$ such that optimum w can be written as $w = \sum_{i=1}^{m} \alpha_i x_i y_i$

Prediction: $\text{sign}(w^T x)$

&

Wt. matrix: $w = \sum_{i=1}^{m} \alpha_i y_i x_i$

$\therefore \quad w^T x = \sum_{i=1}^{m} \alpha_i y_i \boxed{x_i^T x} \Rightarrow$ we only need to compute dot products b/w training examples and new example

This is true even if we map examples to high dimensional space

$$w^T \phi(x) = \sum_i \alpha_i y_i \underbrace{\phi(x_i)^T \phi(x)}_{\substack{\text{This is provided by} \\ \text{Kernel trick}}}$$

HINGE LOSS (incorporates dist. from classification boundary into cost calc)
(soft margin SVM) accounts for classification errors

$$H(y) = \max(0, 1 - t\,y)$$

where t = true label (1 or −1)
y = pred. prob (output of classifier)

Ex1: outcome = 1, pred = 0.5

$$H(y) = \max(0, 1 - 1 \times 0.5) = 0.5$$

Ex2 outcome = −1, pred = 0.5

$$H(y) = \max(0, 1 - (-1) \times 0.5) = 1.5$$ (Higher compared to above)

COST FUNCTION OF SVMS:

Maximize margin
or
Minimize w (inv. prop. to margin) which is a vector orthogonal to the data-separating hyperplane onto which we project our data pts
+
Primal form of Hard Margin SVM

For soft margin SVM, we combine this minimization obj with a loss fn such as hinge loss

DIMENSIONALITY REDUCTION

→ Dimensions in ML = Features

Reducing dimension = reducing feature space

→ Why dimensionality reduction?

- If the feature space is very large, the distance b/w data pts increases hence models may not generalize well

- Computation time for algos increase in large feature space hence to reduce the <u>computation time</u>

→ Main Approaches for dimensionality Reduction

- Projection (projecting onto lower d-dimensions)
- Manifold Learning (Learning the lower dimension manifold on which training instances lie e.g. a 2D manifold is a 2D shape that can be twisted & bent in higher dimensional space)

→ All dimensionality redn techniques suffer from "<u>reconstruction error</u>"

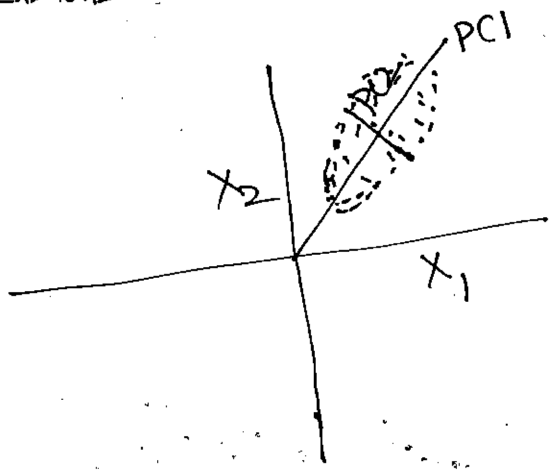
NOTE: HYPERPLANE (in context of classification)

1) n-1 dimencions
2) Flat
3) Divides the space (n dimension feature space) into two

PRINCIPAL COMPONENT ANALYSIS (PCA) (Linear, Unsupervised)

IDEA: The basic idea behind PCA is to rotate the axis of dataset towards directions that maximize the variance along the new axis

Ex: in 2 dimensions:
Scatter plot



PC1 is the axis = max. variance

PC2 is the axis = second max variance

Note: 1) All these axes are orthogonal (linearly independent)

2) PC_i = linear combination of features

e.g. $PC_i = ax_1 + bx_2$

real numbers

You take top m PCs where m < no. of features and do a dot product with Original dataset to get the projection of data on these m axes

STEPS TO CALCULATE PCA

1) Standardize the data
   - meaning subtract the mean of a dimension/feature from each of the value in that dimension/feature and divide the result with std. dev of the dimension/feature

   i.e. $\frac{x_i - \bar{x}}{\sigma_x}$ for each feature value x_i

   where $\bar{x}$ = mean of feature

   σ_x = std. dev of feature

2) Calculate Covariance Matrix of all features

$$\mathrm{Cov}(\underbrace{X_1, X_2}_{\text{features}}) = \frac{\sum_{i=1}^{n} (X_{1i} - \bar{X}_1) \cdot (X_{2i} - \bar{X}_2)}{n-1}$$

where $\bar{X}_1$ = mean value of feature X_1

$\bar{X}_2$ = mean value of feature X_2

X_{1i} = i^{th} feature value of feature X_1

X_{2i} = i^{th} feature value of feature X_2

n = no. of data points/examples

Sign of covariance = +ve : both features ↑ together

-ve : one feature ↑ another feature ↓

0 : no relation (independent features)

$$\underset{\substack{\uparrow \\ \text{Covariance matrix}}}{C} = \begin{bmatrix} Cov(X_1, X_1) & Cov(X_1, X_2) & \dots & Cov(X_1, X_n) \\ \vdots & & & \\ Cov(X_n, X_1) & \dots & \dots & Cov(X_n, X_n) \end{bmatrix}$$

IMP: For a matrix X, co-var. matrix in code can be computed as

```
C = np.dot(X.T, X)/n-1
```

Square matrix & symmetric along diagonal

IMP: The reason we create Covariance matrix is bcz eigen decomposition requires square & symmetric matrix & co-var. matrix has this property

Note $Cov(X_1, X_1)$ = variance of X_1

3) Calculate Eigen vectors and Eigen Values

Recall Eigen vectors of a matrix are the vectors that stay on their span and eigen values are the factors by which eigen vectors stretch or squish after the matrix transformation. Also eigenvectors = axis of rotation

Finding Eigen vectors of covariance matrix amounts to finding axes that remain unchanged during the matrix transformation of space according to co-variance of features. Now eigen values "stretch or squish" the eigen vectors during this transformation based on co-variance of features, eigen values = amt. of variance corresponding to axes

Selecting eigen vectors based on their eigen values
$\equiv$ finding axes based on amt. of variance

Recall eigenvectors and eigen values are found using eq^n

$$C\vec{v} = \lambda \vec{v}$$

or in eigen decomposition form:

$$C = V \operatorname{diag}(\lambda) V^{-1}$$

We take top-d eigen vectors based on eigen values and stack them column wise to build eigen vector matrix

4) Project I/P data on the eigen vector matrix

Input $\cdot$ top-d eigen vector matrix = projection of input on the hyperplane defined by d-axis

dot product

<u>Note</u>: eigen vector of covariance matrix = principal component
(usually both are unit size)

eigen vector with highest eigen value = 1^{st} PC
eigen vector with 2^{nd} highest eigen value = 2^{nd} PC

USING SVD TO CALCULATE EIGEN VECTORS

- Recall eigen decomposition of a matrix is only possible when the matrix is square & symmetric

$$A = X \lambda X^T$$

needs to be symmetric (& square)

eigen vectors stacked column-wise

diagonal matrix where diagonal elements = eigen values

- However, ~~where~~ observations/examples for a ML task may not be square & symmetric

$$\underset{\text{I/P}}{\begin{bmatrix} 2 & 1 & 4 \\ 3 & 2 & 9 \\ 4 & 11 & 12 \\ 14 & 17 & 19 \end{bmatrix}} \underset{\text{wts}}{\begin{bmatrix} x \\ y \\ z \end{bmatrix}} = \underset{\text{labels}}{\begin{bmatrix} 2 \\ 5 \\ 13 \end{bmatrix}}$$

- SVD is a way to factorize *any* matrix (even if not symmetric + square)

$$\underset{m\times n}{A} = \underset{m\times r}{U}\ \underset{r\times r}{\Sigma}\ \underset{r\times n}{V^T}$$

PARALLEL BETWEEN EIGEN DECOMPOSITION & SVD DECOMPOSITION

SVD decomposition $\quad A = U \Sigma V^T \quad (1)$

Now since eigen decomposition is only defined for symmetric, square matrices

let's convert A into symmetric square matrix so that it can be compared to eigen decomposition

$A^T \cdot A$ yields symmetric, square matrix for any matrix A

So,

$$A^T \cdot A = (U \Sigma V^T)^T \cdot U \Sigma V^T$$

Covariance matrix of A

$$= (V^T)^T \Sigma^T U^T \cdot U \Sigma V^T$$

$$= V \Sigma^T U^T \cdot U \Sigma V^T$$

$$= V \Sigma^T \Sigma V^T$$

$$= \underset{X}{\underset{\sim}{V}}\ \underset{\lambda}{\underset{\sim}{\Sigma^2}}\ \underset{X^T}{\underline{V^T}}$$ → This resembles eigen decomposition

$(AB)^T = B^T \cdot A^T$

$(A^T)^T = A$

$U^T \cdot U = I$ since U is orthonormal (columns & rows of U are orthogonal unit vectors)

$\Sigma^T \cdot \Sigma = \Sigma^2$ since Σ is diagonal matrix

Hence, V = Matrix of Eigen vectors as columns

Σ = Diag. Matrix of $\sqrt{\text{eigen values}}$ (or singular values)

$\therefore$ from SVD decomposition (1), using V & Σ we can compute eigen vectors and eigen values

WHY SVD METHOD IS USED TO CALCULATE EIGENVECTORS & EIGEN VALUES IN PCA?

1. Calc covariance matrix is computationally expensive and can result in rounding errors. SVD way is numerically stable. Another way to say it is - singular values are more stable than eigen values
2. SVD can work on sparse matrices

SVD (SINGULAR VALUE DECOMPOSITION)

→ Matrix Factorization technique

SINGULAR VECTORS & SINGULAR VALUES :

For a given matrix A :

$A^T \cdot A$ and $A \cdot A^T$ results in Symmetric, square matrix and both matrices have same positive (≥ 0) eigenvalues

Also $\underbrace{A^T \cdot A = A \cdot A^T}_{\text{These are in fact, covariance matrix of } A}$ (dot product is commutative i.e. $a \cdot b = b \cdot a$)

$\Rightarrow |a||b| \cos\theta = |b||a| \cos\theta$

Fundamental PROPERTY of Symmetric Matrices :

✓ For symmetric matrices, we can choose its eigenvectors to be orthonormal

Two vectors are said to be orthonormal, if

i) they are of unit length

ii) perpendicular to each other (orthogonal)

Let's name eigenvectors of $AA^T = u_i$

eigen vectors of $A^TA = v_i$

u_i, v_i → singular vectors of A

Both matrices AA^T & A^TA have same +ve (or zero) eigenvalues

✓ $\sqrt{\text{eigen values}}$ = singular values

SVD DECOMPOSITION:

Any matrix A can be decomposed as

$$A = U \Sigma V^T$$

where U = orthonormal eigen vectors of AA^T

V = orthonormal eigen vectors of A^TA

Σ = diagonal matrix with $\sqrt{\text{eigen values}}$ in decreasing order

Dimension-wise:

$$A_{m \times n} = U_{m \times m} \Sigma_{m \times n} V^T_{n \times n}$$

$$= \begin{bmatrix} \uparrow & & \uparrow \\ u_1 & \cdots & u_m \\ \downarrow & & \downarrow \end{bmatrix} \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ 0 & & & \sigma_n \end{bmatrix} \begin{bmatrix} \uparrow & & \uparrow \\ v_1 & \cdots & v_n \\ \downarrow & & \downarrow \end{bmatrix}$$

eigen vectors of AA^T — $\sigma_1 \geq \sigma_2 \geq \cdots \sigma_n$ — eigen vectors of A^TA

Since $\sigma_1 \geq \sigma_2 \cdots \geq \sigma_n$, if we want to ignore some very low $\sqrt{\text{eigen values}}$, we can take first r values. In that case

$$A_{m \times n} = U_{m \times r} \Sigma_{r \times r} V^T_{r \times n} \Rightarrow \text{dimensionality reduction}$$

Note: Orthogonal matrices are useful as they inverse = transpose which means calc. their inverse is computationally cheap & stable

GEOMETRICAL INTERPRETATION OF SVD

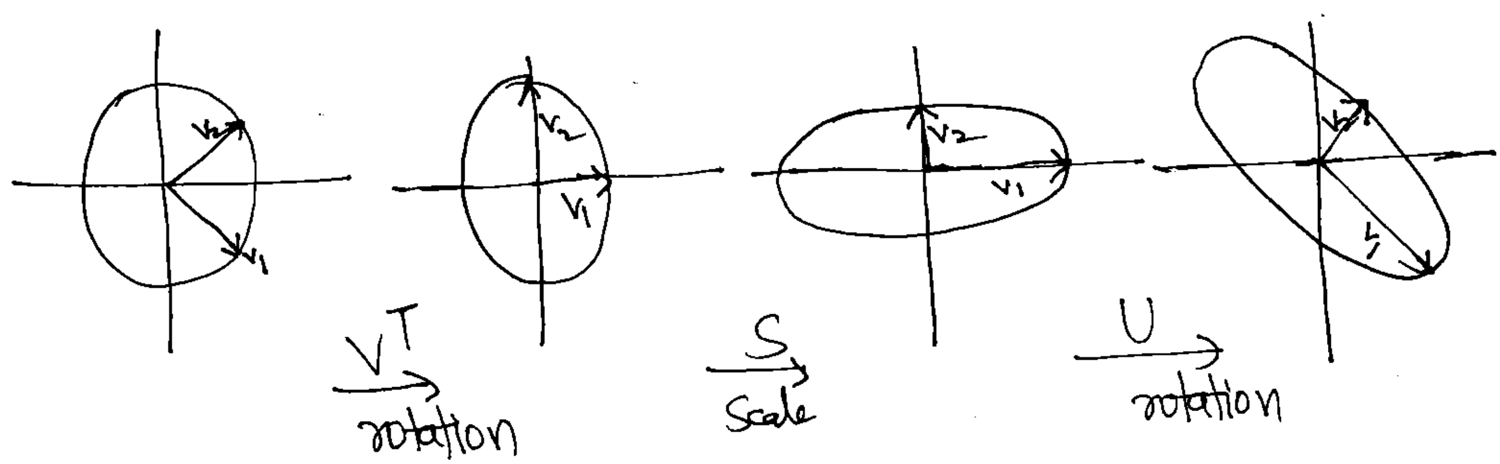
$$A = U \Sigma V^T$$

Since A is a matrix = Transformation of space

This transformation can be expressed as three transformations in sequence: rotation (V^T), scaling (Σ) and again rotation (U)

Since U & V are orthonormal matrix and orthonormal matrix just change the coordinate axes via some rotation but no scaling

Σ is a diagonal matrix and hence just scales the dimensions



SVD in RECOMMENDATION SYSTEMS

$$A_{m \times n} = U_{m \times r} \, \Sigma_{r \times r} \, (V_{n \times r})^T$$

- $A_{m \times n}$ → user-item rating matrix
- $U_{m \times r}$ → columns are unit vectors & orthogonal
- $\Sigma_{r \times r}$ → diagonal matrix, diagonal entries = singular values
- $V_{n \times r}$ → columns are unit vectors & orthogonal

then

$U \to$ user to concept

$\Sigma \to$ strength of concept

$V \to$ item to concept

Ques: Given a data pt, how can we find which concept it relates to?

$$\underset{1 \times n}{d = [0 \ 4 \ 5 \ 0 \ 0]} \cdot \underbrace{V_{n \times r}}_{\text{projecting into concept space}} = [\underset{\text{Higher affinity to this concept}}{5.2} \quad 0.4]$$

Note: SVD is defined for sparse matrices but not for missing values