I. Univariate, Bivariate and Multivariate Analysis

Univariate Analysis: We just pick up one feature and try to see / classify those points w.r.t output. In the y-axis, we would have nothing as there is just one feature and the points would appear in a line parallel to the x-axis.
For example: we choose weight feature - then we get the clusters in line as slim, fit and obese just with the help of weight.
But, these easily classifiable points will not always be there; there would be many overlaps.
To tackle that, we would go with bivariate and multivariate analysis.

Bivariate Analysis: For example, we take height and weight features. Based on these points are classified. We see certain clusters with some overlaps. This overlap will also help us understand which machine learning algorithm we can use to classify. For example if there is lesser overlap, we can use logistic regression because in it we are using sigmoid function and we would get higher errors. So, in case of higher overlaps, we would choose non-linear algorithms such as decision tree, KNN, random forest etc.

Multivariate analysis: What if we have multiple features. So, in order to analyze such data, we use multivariate analysis. Seaborn pairplot helps us to visualize such data. For example, features: age, height, weight. This pairplot can lead to correlation. Whenever we see age and height, can we find out if age is increasing, if height is increasing. This would show positive correlation. If we are not able to find positive or negative correlation, it might have zero correlation. We can use Pearson correlation (value ranges from -1 to +1). Should I use some non-linear classifying plane/line to classify? - all these questions would be answered by a pairplot.

II. Histograms

Histograms help us to visualize the number of points within a particular category in univariate analysis as those points are not quite evident in a univariate analysis. Default bin count is 10. Y-axis shows the count of the number of values in a particular range. We can use the Matplotlib hist() function or seaborn histogram function. It would be respect to one feature. This figure would look like a curve (if bell, then the distribution might be normal / gaussian). The Bell curve is called the Probability density function (PDF). Then , we would be converting to PDF function - at this point of time what percentage of distribution is within that particular range.

III. Z-Score statistics

In a Gaussian or normal curve, when we go to the right of the mean, we get 1 standard deviation, then 2 standard deviation. On the left of the mean, we see -1 standard deviation, -2 standard deviation and so on. Within the first standard deviation of a Gaussian distribution, we have around 68% of our information, and within the second standard deviation we have around 95% information. And, if we want to convert this entire information to our standard normal distribution where mean is 0 and standard deviation is 1, we use Z-Score.

Z-Score = (x(i) - mean)/ s.d.

If I have {1,2,3,4,5} in our distribution, then mean = 3, s.d. = 1
Z-score = 3 - 3 / 1 = 0 for x(i) = 3
And so on.

If I want to find out 1.5 s.d. Away from the mean, then we use standard normal distribution and for getting that we use Z-score and Z-score table.

Example (Student data):
Mean = 75, s.d. 10, P(x > 60) = ?
Now, when we convert it to standard normal distribution, we see that our mean 75 gets converted to mean = 0; 65 gets converted to -1 and 60 gets converted to -.15. And, we need to find the region under the standard normal curve where s.d. > -1.5

For this we would use the Z-score table, which would give us the left hand side value of area under the curve for s.d. < -1.5.

Now, our curve has 3 regions, Z-score basically gives region 3 where s.d. < -1.5. Region 1 is for the region between s.d >= -1.5 and s.d. <= 0. Region 2 is the s.d. > 0.

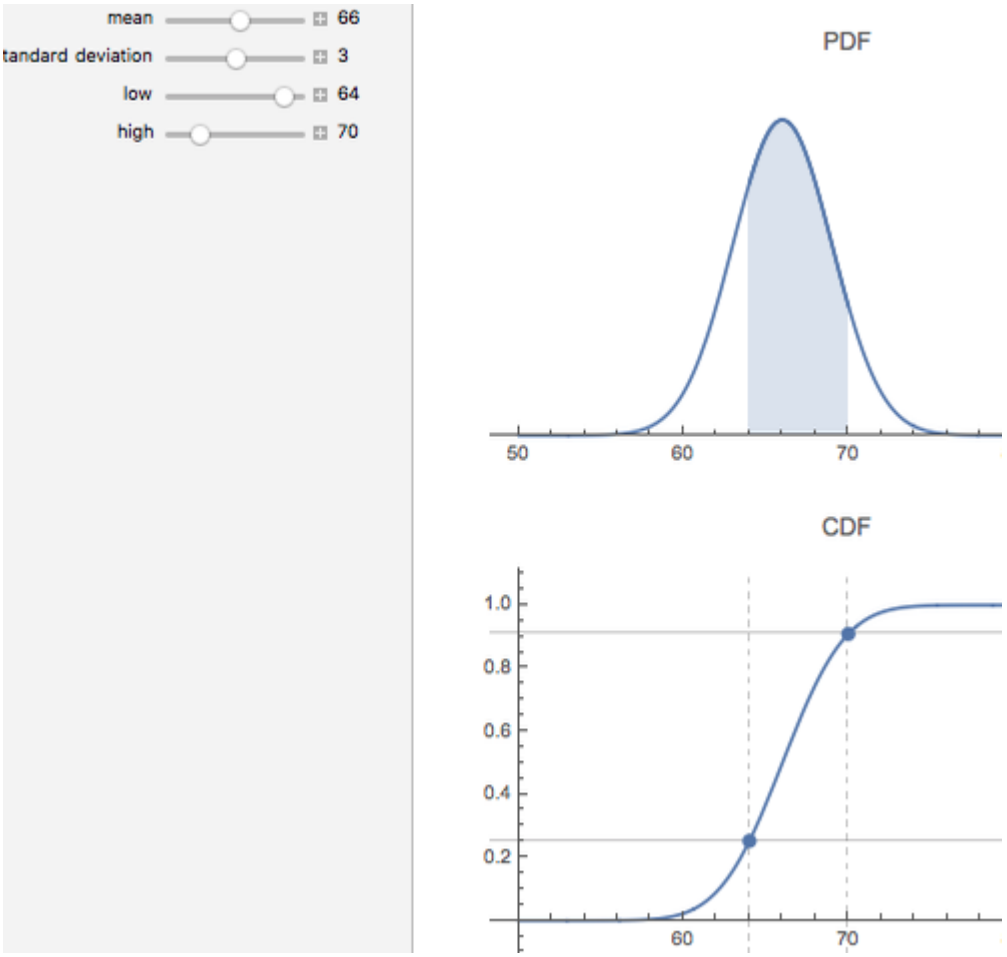
Region 3 value from the Z-score table is 0.0668 (6.68%). Now, we know that for a standard normal distribution, the curve is symmetrical around the mean, so region 2 is 50% of the overall value under the curve. Then, region 1 is (50 - 6.668)% i.e. 43.37% approximately. Then, the value of Region 1 and Region 2 combined would be around 94% approx.

IV. Probability Density Function:

Plotting all the points in the form of histograms shows on y-axis how many points are within a particular bin / range. If I want to convert it into a PDF or smoothen the histogram, it resembles a bell curve. As soon as we draw a bell curve, the count value gets replaced by % of the distribution on the y-axis.

It basically says that within this range, what is the % of the distribution present in the particular region. If that particular value is 0.2, which means 20% of the points distribution are in that particular region.

Cumulative density function (CDF) is different from PDF. In this , we basically add the percentage distribution for these points and the curve looks different from PDF.



Suppose, we take a point x(j) and we get 90% on the y-axis corresponding to it. This indicates that 90% of the distribution is less than ….x(j)…kgs. That also indicates that 10% of the distribution is greater than x(j) kgs.

V. Ridge and Lasso Regression

Regularization hypertuning techniques.
Sum of residuals in linear regression: $\sum(y' - y)^2$

Suppose there are two parameters: Experience and Salary. We try to create a best fit line using linear regression.

Suppose my training data has only two points which are far and fall almost on a line. We create a line and then calculate the sum of residuals to reduce the cost function. Now, in this case, for the training dataset, the points lie exactly on the line we made, the residuals would be zero. But, we wish to create a generalized model. Now, the test data might show high errors due to the huge distance from this line. So, this case is of overfitting.

For my training dataset, we have got a low bias / low error result. But, for test data, we see that the model is less generalized and shows high errors.

Now, we can use Ridge and Lasso regression to convert this high variance condition to a low variance condition.

How does Ridge and Lasso regression solve these problems?

Usually, in a linear regression model, Sum of residuals is given by $\sum(y' - y)^2$. But, in ridge regression, we add one more parameter. Cost function for ridge regression = $\sum(y' - y)^2 + \lambda*(slope)^2$.
So, if we see a steep curve in our linear regression model, we can deduce that it might lead to overfitting as with a unit increase in the experience, there is a quite high increase in the salary. In Ridge regression, we add $\lambda*(slope)^2$ to this model where we already see that the slope is quite high. We can assign 0 to any positive value to λ. Let us take it as 1 for now. Let us assume that the slope is 1.3 at present. If we do the calculation $\lambda*(slope)^2 = 1 *(1.3)(1.3) = 1.69$. Now, we will see the same value for other lines possible and see if they can reduce it.

Suppose residuals for this line might be a smaller value and slope might also be a smaller value as steepness has gone, thus, $\lambda*(slope)^2$ would also be a smaller number. Hence, cost reduces on the whole. So, we can use this line as the new best fit line.

We are basically penalizing features which have higher slopes to make the best fit line less steeper. If there is just one feature slope would be m. If we have 2 features, $\lambda*(slope)^2 = \lambda*(m1^2 + m2^2)$.

This will lead to less variance and better generalization. There would be slightly higher bias for the training dataset now but the test accuracy improves.

Cost function for Lasso regression = $\sum(y' - y)^2 + \lambda*|slope|$. It not only helps in regularization, but also helps in feature selection.

Suppose, we have $y = m1x1 + m2x2 + m3x3 + m4x4 + c1$. Now, as per Lasso regression, $|slope| = |m1 + m2 + m3 + m4|$. Wherever, the slope is very less, those features would get removed as they are not important / significant in making the prediction. This leads to feature selection as well. For Ridge regression, on the other hand, the $(slope)^2$ shrinks but never reduces to zero.

Curse of Dimensionality:

Dimensions, features, attributes are all terms used for features.

Suppose, for M1, I am taking 2 features; for M2 , I am taking 5 features;
For M3, I am taking 10 features; for M4 100 features;
For M5, I am taking 200 features; for M6 1000 features and for M7, 10000 features.

In M1, We will calculate Price of the house w.r.t Size of the house and no. of bedrooms.

In M2, additional features are state, bed size and area schools.
This model will definitely give better accuracy than the previous one as this provides better information to predict.

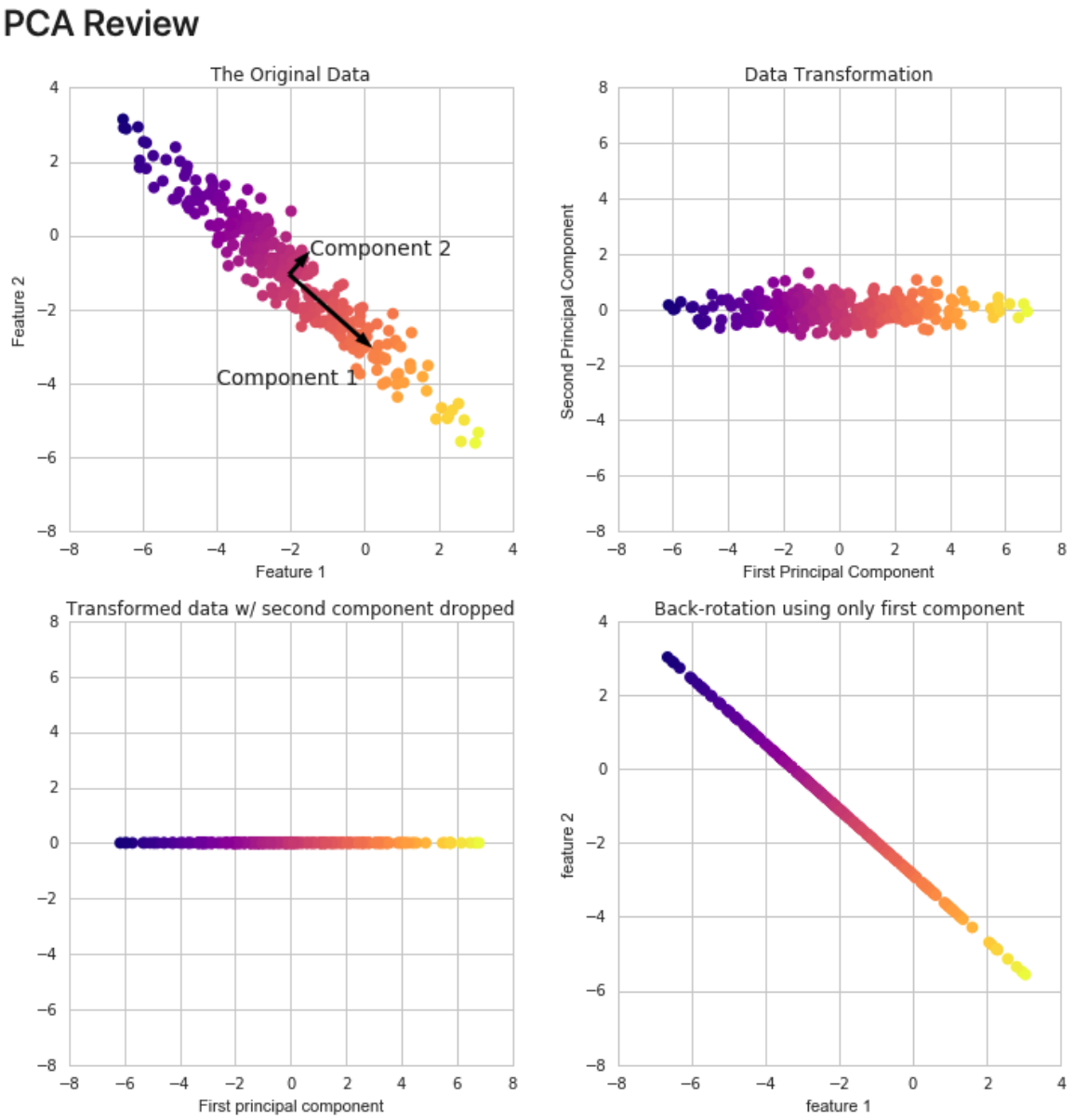
Now, in M3, we are taking 10 independent features. This model also gives greater accuracy than M2 and M1.

As we increase the number of features, our accuracy and performance is also increasing. But after a certain threshold value, our model will not give increased accuracy even by increasing the number of independent features / dimensions.

This is called the curse of dimensionality. With the increase of dimensions, it is not necessary that the accuracy improves after a certain threshold value. It is possible that this accuracy might decrease after a certain threshold. Until the threshold is achieved, the model learns new information from the independent datasets and features. With the exponential increase in the number of features, the model gets confused and the accuracy drops.

Then how to select the correct number of features / dimensions?
We use Chi square test, correlation coefficient etc. to determine this number of features.

Dimensionality Reduction - Principal Component Analysis



Suppose, I have features in a 2-dimensional space and we need to convert these features into 1-dimensional space.

Step 1: Consider the best vector space (1-d line) and project all the points to it. This line is called Principal Component 1 (PCA1).
Step 2: The next line (PCA2) would be drawn orthogonal to the PCA1 line. Now, when I try to project all the points to this line PCA2, there is a lot of information / variance that is lost during this process. So, we try to reduce this information loss. The reason we consider these two orthogonal lines is that they cause lesser information loss.

VI. Hypothesis Testing

P Values, T Test, Anova Test, Z Test, Type I and Type II Error

What exactly is hypothesis testing?

Any data in statistics is useful only if you analyze it and deduce conclusions or inferences. In hypothesis testing, we actually evaluate two or more exclusive statements on a population using a sample of data.

- ***Statement 1***: The person is guilty
- ***Statement 2***: The person is innocent

Steps of hypothesis testing:

a. Make an initial assumption (Ho - null hypothesis - The person is innocent)
b. Collect data / evidences
c. Gather evidence to reject or not reject the null hypothesis

Alternate hypothesis, H1 - Opposite of null hypothesis. If we are able to prove Ho, then we consider it as true else due to lack of evidence, we consider H1 as true.

Suppose if a null hypothesis is actually true (defendant is innocent) but I do not have enough evidence. Then Ho gets rejected and H1 would be considered as true (the defendant will be treated as guilty). If we look at the confusion matrix as follows:

	Ho	**H1**
Do not reject	Ok	***Type 2 Error***: we took H1 as true as per evidence, but actually Ho was true
Reject	***Type I Error***: Due to the lack of evidence, I have to reject Ho even though it might be true.	Ok

Suppose. Ho: the market is going to crash and H1: the market is not going to crash.
I collected various pieces of evidence and found that H1 is true but actually Ho was true : the market crashed. This is type II error.

If $P < 0.05$ (significance value), then we reject Ho and consider H1.

VII. P Value

Based upon the number of touches in the center of the spacebar key, we can make a normal bell curve. It is also called a 2-tailed test.

Suppose for a particular point, P value is 0.01. This means that if we repeat this experiment 100 times, then the number of times we will be touching the space key in this area will be 1. Suppose the P value at a point is 0.8, then if we repeat this experiment 100 times, then the number of times we will be touching the space key in this area will be 80 times.

P-value is the probability for the null hypothesis to be true.
Null hypothesis: It treats everything same or equal

Consider I have a fair coin. If I am performing a specific toss experiment 100 times, based on fair nature at least 50 times head or tail should be coming up.
Null hypothesis, Ho - the coin is fair
H1: the coin is not fair
Now, when we perform an experiment of toss (100 times). Out of that we achieve: 67 times head, 33 times tail. 50 would be the mean value of the bell curve and 67 would be on its right. It would be good to get this value nearer to the mean. We see that the P-value is falling in the tail region , away from the mean. SO, we need to reject the null hypothesis and we cannot say that it is treating all the same. The coin is not fair as Ho is rejected.

5% of data is usually divided in two parts under the tail region and the rest 95% is there within the two tails.

Now, suppose, P value is 0.05 (significance). If my experiment comes with a P value and it lies in the tail region, then we will reject Ho definitely because it is likely not possible to be true.

If suppose , we get 55% head in 100 tosses, we notice that P value does not come in the extreme tail regions, so we accept the null hypothesis and say that it is really a fair coin.

VIII. T Test, Chi Square test, ANOVA Test

Given the age, gender, weight and height data of males and females.

Null Hypothesis, Ho: There is no difference.
Alternate Hypothesis, H1: There is a difference between male and female proportion.
Test: {Considering Ho is true, where is the likelihood that H1 is true?}

a. ***One sample proportion test:*** (if we are considering just one-categorical feature such as gender). P - value needs to be selected before we start this test.
   If P value is less than 0.05, then we reject Ho. There is actually a difference between male and female ratio / proportion.
   P<= 0.05 → significance value alpha

b. ***Chi Square Test***: (if we are considering two categorical features; gender given the age group).

c. **T test**: (If we are considering One Continuous numerical variable e.g. height - mean height)

d. ***T test or Correlation Test***: (If we are considering Two continuous numerical variables e.g. weight and height). According to Pearson correlation, the value ranges from -1 to 1. The correlation near to 0 signifies that there is no relationship.

e. ***ANOVA Test***: If we are considering one categorical and one numerical variable and a categorical variable with more than two categories. If it has two categories only, then we apply the T test.

IX. Metrics in Classification Model Performance

a. Confusion matrix
b. FPR (False Positive Rate - Type I Error)
c. FNR (False Negative Rate - Type II Error)
d. Recall (TPR i.e. True Positive Rate, Sensitivity)
e. Precision (+ive pred val)
f. Accuracy
g. F Beta
h. Cohen Kappa
i. ROC Curve, AUC Score
j. PR Curve

Any classification problem can be solved either through Class Labels (A, B = 0.5) or Probabilities (We have to select this threshold probability very carefully).

Suppose in a problem statement of binary classification we have 1000 records, out of which 500 are Yes and 500 are No. OR 600 Yes and 400 No, then we can say that it is a balanced dataset. Even 700 Yes, 300 No → balanced dataset. My machine learning will not get biased based on the output (as more or less the dataset is balanced).

But if this ratio is 80: 20 (Yes: No), some of the machine learning algorithms will get biased based on the output.

If we have a balanced dataset, we use the metrics such as Accuracy. If we have an imbalance dataset, we use Precision, recall or F Beta score.

Confusion is a 2X2 matrix where the top values are the actual values and on the left hand side are my predicted values.

Actual values →	**1**	**0**
Predicted values → 1	True Positives	Type I Error (FPR): False Positives

0	Type II Error (FNR): False Negatives	True Negatives

Type I Error - FPR = FP / FP + TN
Type II Error - FNR = FN / FN + TN
Accuracy = TP + TN / TP + FP + TN + FN

What if my data is imbalanced? Suppose we have 900 Yes and 100 No. My model might predict everything belonging to the Yes category based upon output ratio. It basically is blindly suggesting that it belongs to this particular category. For this we use precision and recall.

True Positivity Rate / Sensitivity / Recall: TP/(TP + FN) - Out of the total actual positive values, how many did we predict correctly

Positive prediction value / Precision: TP/(TP + FP) - Out of the total predicted positive results, how many results were actually positive.

Use cases of Recall and Precision:

Spam detection - Precision (the mail is not a spam but it has been predicted as a spam - the customer is going to miss that email)

Cancer or not - Recall (the person is told that he does not have cancer but he actually has cancer)

Whenever, your false positive is much more important, then use precision.
Whenever, your false negative is much more important, then use recall.

F-Beta: Sometimes in an imbalanced dataset , both false positives and false negatives are important, then we use F-Beta score.

F Beta = $(1 + \beta^2)$ Precision * Recall / β^2 (Precision + Recall)

If β value is 1, then it becomes F1 - score
If β value is 0.5, then it becomes F0.5 - score
If β value is 2, then it becomes F2 - score

If β value is 1, then F1 Score: 2*Precision * Recall / (Precision + Recall): Harmonic mean

When do we select Beta as 1?
When FP and FN are equally important, then we select Beta as 1.

When FP is having more impact than FN, at that time we select Beta as 0.5. (Range: 0-1)
When FN is more impactful, at that time we select Beta as 2. (Range: 1-9)

Now, let us explore performance metrics where we have to decide a probability threshold:

Use case: Disease data in healthcare

For our model, we would decide some threshold value, [0, 0.2, 0.4, 0.6, 0.8, 1]
Suppose it is 0, then anything which has a predicted value greater than 0 will have a value of class label as 1. Then we calculate TPR and FPR. Here, TPR = 1, FPR = 1, Then we plot TPR versus FPR for the ROC Curve.

Now, when we connect all the points on the ROC curve, the area under this curve is called AUC score. The greater the area under the curve, the better the model is. This area should be more than the diagonal line connected in the graph. If it is less than that, it is a dumb model.

If we plot this curve and show it to a domain expert, he would tell us whether they require higher TPR, we can select one particular value of threshold where FPR is zero for higher TPR. If the domain expert says that we do not care about FPR but we just need the highest TPR, we can select the top most point in the curve.

X. Logistic Regression

Logistic regression is basically used for Binary classification. But why is it called regression? On X-axis if we take weights and on Y-axis we classify them as Obese or not obese. Can we solve this problem with the help of linear regression?

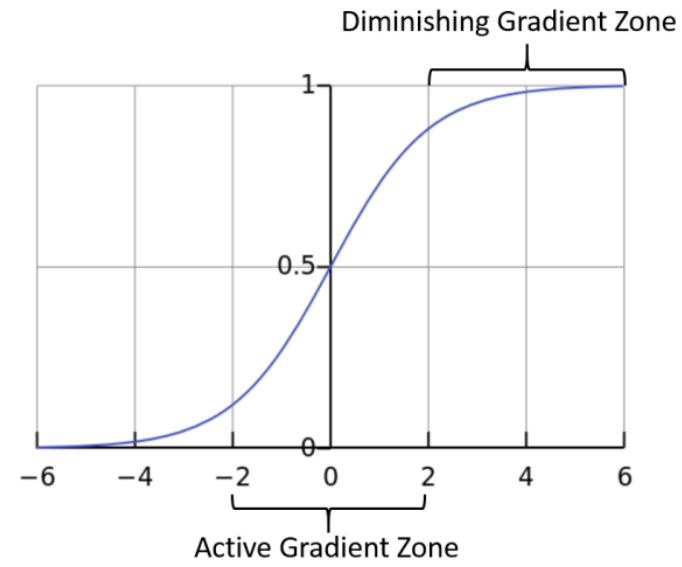
Now, linear regression states that the distance between predicted values and actual values should be minimal and predicted values are given by: $y' = mx + c$ or $h(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5$; here, β_0 is the y-intercept; β_1, β_2 and β_3 are changes in y with unit changes in x_1 , x_2 and x_3 respectively.

Now, suppose we consider the weight in our example of weights → 75; at this point, the output would be exactly in the middle (0 and 1) = 0.5. $h(x) >= 0.5$ {obese} and $h(x) < 0.5$ {Not obese}. With the help of this line, we are able to classify the points (weights).

Just by considering a straight line, we can solve the classification problem, then why do we need Logistic regression? To answer this, suppose, we just have one new outlier in our data, this changes our best fit line in linear regression and classification for all the points as the straight line gets deviated.

Because of this drawback of the need of creating a best fit line each time a datapoint is added, we use logistic regression for classification. Another drawback of linear regression, is the best fit line might give outputs greater than 1 or lesser than 0 as well. This problem is solved by using sigmoid function in logistic regression as it squashes the values greater than 1 or lesser than 0.

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



Logistic Regression:

It is usually applied to those problems where the two classification points can be classified using linear regression. We need to find the exact coefficients of the best fit line but we do not use linear regression. Assumptions we make are:

- For positive points are denoted as + 1 and negative points are denoted as -1.
- 'Y-intercept - b' value is considered as 0. Y = w'x. The distance between any point x and the best fit plane is given as w'x + b / ||w||. If we consider w as a unit vector, ||w|| becomes 1 and b has been assumed as 0. Then we can say that the distance of a point x and a plane is w'x. Above the best fit plane, if we have a point, then its distance from the plane comes out as positive whereas below it comes as negative.
- Now, the distance of point x1 above the plane would be positive and would be w'x1.
- y(w'x) > 0 (Case 1)
- Now, consider y = w'x2 below the best fit line for x2 point. For it y would be negative. Then we can say: y(w'x) > 0 (Case 2) for this point as well.
- Case 3: Consider an outlier above the best fit line. Y = -1 for the outlier and w'x > 0. Thus, y(w'x) < 0 for the outlier. This point is incorrectly classified.

Our cost function should be maximized: max∑yw'x for i = 1 to n . Here, y and x are given and the only thing we need to find is the parameter - w_i.

What if we have outliers in logistic regression?

For a positive point lying in the negative region i.e. below the best fit line, we get a negative cost: yw'x Then, the best fit line deviates to make the cost positive. How do we rectify it?

For tacking it, we modify the cost function by applying sigmoid function on the earlier cost function.

Max∑f (yw'x) for i = 1 to n and f() is the sigmoid function. This sigmoid function makes sure that the entire value is transformed between 0 and 1. By doing that, it is removing the effect of outlier.

How can we solve a multiclass classification problem using Logistic Regression?

Logistic Regression (One vs. Rest): First M1 model is created segregating one category from the rest of the categories. Next, iteration, model M2 is created, treating one category and the rest as another. Similarly, in the next iteration, another model M3 is created.

Suppose, the three categories we have are O1, O2 and O3. In the first iteration, O1 might be + 1 and O2, O3 might be -1. Similarly, in the next iteration, O1, O2 might be + 1 and O3 might be -1. This way we combine values of O1 to create a model M1; values of O2 to create a model O2 and values of O3 to create a model M3. If I give a new test data, then the whole feature will go to M1 model, then M1 model will give some probability. Then, M2 will give some probability when a feature is given to it. And, then M3 gives another probability. Now, the model that gives the highest probability, say M3 , then that point belongs to the O3 category. While coding, there is a multi-class parameter which needs to be set as ‘ovr’ to implement multiclass Logistic Regression.

XI. Cross-validation Types

When we do a train-test-split with 30% split, 70% of the random dataset is selected. There is a random state variable. There would be certain data points which were important but were not taken. This might lead to reduced accuracy. Suppose, we change the random state and again do train-test-split, our accuracy fluctuates. This way our accuracy changes as the random state changes. This shows fluctuating output.

To handle this problem, we use cross validation. Different types of cross-validation:

a. ***Leave One Out CV (LOOCV)***: Suppose I have 1000 records, the LOOCV rule is that from all these records, I will take one dataset at a time as my test and remaining all would be my training data points. Based on this model will be trained on all the training data and would be tested on that one dataset. This way the next experiment is repeated with different test samples. I have to perform many iterations to make the model ready for generalization. It will lead to low bias. For the training dataset and one test sample, we would get quite good results but very poor results on unseen data points.
b. ***K-Fold CV***: Suppose my dataset is 1000 records, we select a K value. This K basically means K experiments. For each experiment, it will decide the test data. For example we can take a K value of 5 and a split of 20% for train and test data. Find out the mean accuracy of all K experiments. Sometimes, one can communicate the minimum and maximum accuracy of the model as well to the stakeholders. One of the drawbacks of K Fold accuracy is that 205 split might lead to an imbalance dataset wherein the test data might have only one type of points and training data might have another.
c. ***Stratified K-Fold CV***: Suppose, K = 5, it is made sure that every time test data is selected, the number of instances of each class is taken in a proper way so that the data does not get imbalanced. Please note that if the entire dataset is imbalanced, we need to handle it first at the time of feature engineering.
d. ***Time series CV***: Suppose we have data for Day 1to Day 7 and we need to predict some results for Day 6 and 7. We need to rely on the previous days to predict for the

desired days. For Day 6, suppose my inputs would be Day 1 to Day 5, for Day 7, my inputs would be Day 2 to Day 6. For Day 8, the inputs would be Day 3 to Day 7. That is how time series CV works.

XII. Naive Bayes Classifier

Theoretical Understanding:

Independent Events - Suppose, if i am tossing one coin, then the probability of heads and tails are ½ and ½ respectively. Now, if we toss these coins again. The probability of heads or tails in the second toss is not dependent on the outcome of the first toss. Thus, this is an example of independent events.

Dependent Events - Suppose, in a bag I have marbles (3R 2B). Suppose, we pick up 1 blue marble, the probability of getting a blue marble is ⅗. Now in the next event if i want to pick up one marble out of the remaining 4 marbles. Now the probability of the second marble being blue is dependent on the outcome of the first marble. Thus, this is an example of dependent events.

Conditional Probability - P(A|B) = P (A∩B)/P(B) Suppose, in event A, I pick up a blue marble, the probability would be ⅗. Now, suppose, in event B, I take out second blue marble given A event is already performed i.e. P(B|A) = ¼ as there is just one blue marble remaining. When we are considering P(B|A), if we multiply P(A) and P(B|A), we get 1/10. This is equal to P (A∩B). This is actually our conditional probability formula only.

Bayes theorem: P(A|B) = P (A∩B)/P(B) and P(B) = P (B∩A)/P(A). Now, P (A∩B) = P (B∩A).

Thus, P(A|B) * P(B) = P(B|A) * P(A)

This implies that P(A|B) = (P(B|A) * P(A)) / P(B)

Here, P(A|B) is the posterior probability. P(B|A) is the likelihood. P(A) is the prior probability. P(B) is the marginal probability.

How is the Bayes theorem used in the Naive Bayes ALgorithm?

Suppose I have x = {x1, x2, x3, x4,.....xn} features and output as {y}. Let us consider that it is a classification problem.

P(y|x1,x2,x3,x4…..xn) = P(x1|y)P(x2|y)P(x3|y).....P(xn|y) * P(y) / P(x1) P(x2) P(x3)....P(xn)

P(y|x1,x2,x3,x4…..xn) = P(y) $\prod$P(xi/y) / P(x1) P(x2) P(x3)....P(xn)

for i ranging from 1 to n

P(y|x1,x2,x3,x4…..xn) $\propto$ P(y) $\prod$P(xi/y) for i ranging from 1 to n

y = ARGMAX(P(y) $\prod$P(xi/y)) for i ranging from 1 to n

Practical Example:

Outlook feature:

	Yes	No	P(Y)	P(N)
Sunny	2	3	2/9	3/5
Overcast	4	0	4/9	0
Rainy	3	2	3/9	2/5
Total	9	5	100%	100%

Temperature feature:

	Yes	No	P(Y)	P(N)
Hot	2	2	2/9	2/5
Mild	4	2	4/9	2/5
Cold	3	1	3/9	1/5
Total	9	5	100%	100%

Play Outcome:

		P(Y) & P(N)
Yes	9	9/14
No	5	5/14
Total	14	

Now, suppose for today the day is sunny and hot:

P(Yes | Today) = P(Sunny | Yes) * P(Hot | Yes) * P(Yes) / P(Today)

P(Yes | Today) = 2/9 * 2/9 *9/14 = 2/63 = 0.031

P(No | Today) = ⅗ * ⅖ * 5/14 = 3/35 = 0.08571

P(Yes) = 0.031 / 0.031 + 0.08571 (normalizing it) = 0.27 approx.

P(No) = 1 - 0.27 = 0.73

I.e. the output would be No as it has higher probability

Another practical example:

NLP: *Judge whether the statement review is good or bad*

f1	f2	f3	f4	o/p
The	food	Delicious	Bad	
1	1	1	0	1
1	1	0	1	0
0	1	0	1	0
0	1	1	0	1
0	0	0	1	0

Sentence 1: The food is delicious

Sentence 2: The food is bad

Sentence 3: Food is bad

After using Stop words, stemming, Bag of words, TFIDF → we utilize the final features.

P(y = Yes | Sentences) = P(y = Yes | x1, x2, x3,....xn) $\propto$ P(y) ΠP(xi/y) for i ranging from 1 to n

X1 is nothing but the words in the sentences after pre-processing.

P(y = Yes) = ⅖ (from o/p column)

P(x1/y = Yes) = 1/2 (where The is present and o/p is 1)

P(x2/y = yes) = 2/4 (where food is present and o/p is 1)

P(x3/y = yes) = 2/2 (where delicious is present and o/p is 1)

P(y = Yes | Sentences) = P(y = Yes) * ΠP(xi/y) for i ranging from 1 to n = ⅖ * ½ * 2/4 * 2/2 = 1/10 = 0.01

Similarly P(y = No | Sentences) = ⅗ * ½ * ⅔ * 3/3 = ⅕ = 0.2

Thus, normalizing , P(y = Yes) = 0.01 / 0/01 + 0.02 = 0.33

P(No) ~ 0.67 , we get that finally the output is No i.e 0 for the max inputs

Where does this fail?

Suppose in a new sentence: The food is tasty. Tasty is a word which has not been seen before. By default, it would be treated as a negative scenario. Also, it might fail when there is an imbalanced dataset. We would see how to rectify this in the NLP section.

1. What Are the Basic Assumptions?

Features Are Independent

2. Advantages

1. Work Very well with many number of features

   Explanation: Because of the class independence assumption, naive Bayes classifiers can quickly learn to use high dimensional features with limited training data compared to more sophisticated methods. This can be useful in situations where the dataset is small compared to the number of features, such as images or texts. Naive Bayes implicitly treats all features as being independent of one another, and therefore the sorts of curse-of-dimensionality problems which typically rear their head when dealing with high-dimensional data do not apply.

   If your data has k dimensions, then a fully general ML algorithm which attempts to learn all possible correlations between these features has to deal with 2^k possible feature interactions, and therefore needs on the order of 2^k many data points to be performant. However because Naive Bayes assumes independence between features, it only needs on the order of k many data points, exponentially fewer.

   However this comes at the cost of only being able to capture much simpler mappings between the input variables and the output class, and as such Naive Bayes could never compete with something like a large neural network trained on a large dataset when it comes to tasks like image recognition, although it might perform better on *very* small datasets.

2. Works Well with Large training Dataset
3. It converges faster when we are training the model

   Explanation: Unlike other machine learning models, naive bayes require little to no training. When trying to make a prediction that involves multiple features, we simply use the math by making the *naive* assumption that the features are independent.

4. It also performs well with categorical features

   Explanation: For a Naive Bayes classifier, categorical values are the easiest to deal with. All you are really after is P(Feature | Class). This should be easy for the days of the week. Compute P(Monday | Class=Yes) and so on.

3. Disadvantages

1. Correlated features affects performance

4. Whether Feature Scaling is required?

No. *In fact, any Algorithm which is NOT distance based, is not affected by Feature Scaling.* As Naive Bayes algorithm is based on probability not on distance, so it doesn't require feature scaling.

5. Impact of Missing Values?

Naive Bayes can handle missing data. Attributes are handled separately by the algorithm at both model construction time and prediction time. As such, if a data instance has a missing value for an attribute, it can be ignored while preparing the model, and ignored when a probability is calculated for a class value tutorial : https://www.youtube.com/watch?v=EqjyLfpv5oA

6. Impact of outliers?

It is usually robust to outliers.

One potential issue with outliers is that unseen observations can lead to 0 probabilities. And we know in naive bayes we multiply probab of words lying in that particular class and results zero. For example,

Bernoulli Naive Bayes applied to word features will always produce 0 probabilities when it encounters a word that wasn't seen in the training data. Outliers in this sense can be a problem.

However, all these and similar issues of Naive Bayes have well-known solutions (like Laplace smoothing, i.e. adding an artificial count for every word) and are routinely implemented. In Gaussian Naive Bayes, outliers will affect the shape of the Gaussian distribution and have the usual effects on the mean etc. So depending on your use case, it still makes sense to remove outliers.

Different Problem statement you can solve using Naive Bayes

1. Sentiment Analysis
2. Spam classification
3. twitter sentiment analysis
4. document categorization

XIII. Linear Regression Algorithm

Theoretical Concepts:

y=mx + c; m = slope, c = intercept

We need to find the best fit line. If my x is 0, then y is c i.e. Y-intercept. Within a unit change in X-axis, the change in the y-value is called slope or m. m = (y2 - y1) / (x2 - x1)

The summation of squared error should be minimized by the requisite values of m and c obtained. Here, cost function = distance b/w the best fit point and the actual point should be minimum. Thus, cost function = $1/2n \sum(y' - y)^2$ where i ranges from 1 to m. Here, m is the number of points.

Predicted points: y' and actual points: y. But, if we just use the above equation, we might get several best fit lines, and choosing just one might be time consuming.

So, what can we do?

Example: y = x; y' = mx + c; here, we can assume the best fit line passes through origin and c = 0; thus y' = mx

Now, we can substitute x = 1, and m = 1; then y' = 1;

For x = 2, y' = 2 and y = 2 and so on. This is actually my best fit line when my slope is 1. After getting this equation, we calculate our cost function and try to reduce it.

Here, cost function, $J(m) = 1/2n((1-1)^2 + (2-2)^2 + (3-3)^2) = 0$

With respect to every m value, we can plot our cost function. For m = 1, J(m) = 0.

For m = 0.5, x = 1; y = 0.5,

For x= 2, y = 1,

For x = 3, y = 1.5

Then our cost function, $J(m) = 1/2n((1-0.5)^2 + (2-1)^2 + (3-1.5)^2) = 0.58$. Thus, a curvature of J(m) versus m can be plotted and we can see the gradient descent. How do we arrive at the global minimum?

Based on some m value, we get some initial J(m) value. In order to move downwards, we should use the convergence theorem.

Convergence theorem: $m = m - \alpha \partial J / \partial m$ where α is the learning rate

If the slope ($\partial J / \partial m$) is negative, the curve points downwards. When we get a negative slope, then m = m + (+ive smaller value). This step would be very very small and it would move slowly towards global minimum. If we take a larger alpha, then the jumps might be bigger and it might not converge after several iterations and keep oscillating.

At the global minima, the slope will be zero. This would be the slope of the best fit line. Until then, we would keep following the convergence theorem.

If I have multiple independent features, then each of the features will try to reach a global minimum.

For multiple linear regression, $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5$; here, β_0 is the y-intercept; β_1, β_2 and β_3 are changes in y with unit changes in x_1 , x_2 and x_3 respectively.

Multicollinearity: In regression, "multicollinearity" refers to predictors that are correlated with other predictors. Multicollinearity occurs when your model includes multiple factors that are correlated not just to your response variable, but also to each other. In other words, it results when you have factors that are a bit redundant. To handle such a multicollinearity situation, one solution is to remove highly correlated feature (check for P value in model summary and remove the one for which it is higher).

R Square and Adjusted R Square:

$R^2 = 1 - SSres/SStot$

Here, $SSres$ is the sum of squares of residuals or errors = $\sum(y' - y)^2$

$SStot$ is the sum of average totals = $\sum(y' - y_{mean})^2$

The closer the value of R^2 to 1, the better the model is.

It will be less than zero only when the best fit line is worse than the average value.

As we go on adding new independent features, our R^2 value usually increases. This is because the model then tries to apply some coefficient value such that our $SSres$ value decreases. Then, resultantly, our R^2 value increases. We should also note that this value will never decrease when we keep on adding independent features to it. But, there might be a scenario that the independent feature being added might not be correlated to the target output. Even though R^2 value might show an increase in such a case, but it is basically not penalizing the newly added features which do not have any correlation with the target. For this reason, we use adjusted R square.

$$Adjusted\ R^2 = 1 - \frac{(1 - R^2)(N - 1)}{N - p - 1}$$

Where

R^2 Sample R-Squared

N Total Sample Size

p Number of independent variable

From the above formula, we can note that as the value of p increases when independent features which are not correlated to the target variable are added, N - p - 1 value decreases, leading to an overall decrease in the Adjusted R square value as a higher term is subtracted from 1 to achieve it. In case, the added features have correlation with the target variable, then R^2 value would be higher such that the rest of the multiplication factor will become overwhelmed and would not make much of a difference to the overall R^2 value.

Differences between R^2 value and Adjusted R^2 value:

Every time an independent variable is added to a model, R^2 value increases, even if the independent variable is insignificant. It never declines, whereas adjusted R^2 value increases only when the independent variable is significant and affects the dependent variable.

Adjusted R^2 value is always less than or equal to R^2 value.

Removing the highly correlated feature would work when we have quite fewer numbers of features and a smaller dataset. We also lose certain information when we remove some data. In case, we have a large dataset, we would go for Ridge or Lasso correlation.

1. What Are the Basic Assumptions?(favorite)

There are four assumptions associated with a linear regression model:

1. ***Linearity***: The relationship between X and the mean of Y is linear.
2. ***Homoscedasticity***: The variance of residual is the same for any value of X.
3. ***Independence***: Observations are independent of each other.
4. ***Normality***: For any fixed value of X, Y is normally distributed.

2. Advantages

1. Linear regression performs exceptionally well for linearly separable data
2. Easy to implement and train the model
3. It can handle overfitting using dimensionality reduction techniques and cross validation and regularization

3. Disadvantages

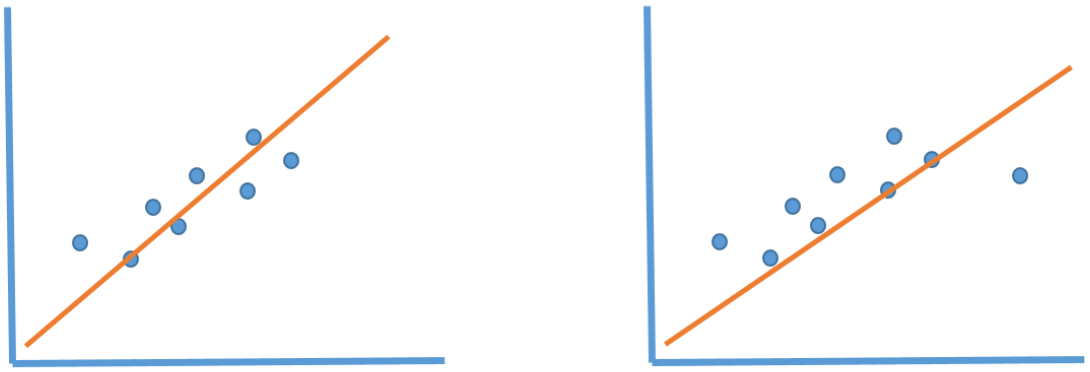
1. Sometimes Lot of Feature Engineering Is required
2. If the independent features are correlated it may affect performance
3. It is often quite prone to noise and overfitting

4. Whether Feature Scaling is required?

Yes (Whenever we talk about gradient descent, we need to do feature scaling because that will help us to reach the optimum solution / global minima very quickly.)

5. Impact of Missing Values?
It is sensitive to missing values (feature engineering is used to handle it)

6. Impact of outliers?
Linear regression needs the relationship between the independent and dependent variables to be linear. It is also important to check for outliers since linear regression is sensitive to outlier effects. Just to reduce the mean squared error (MSE), the line is changing in the below figure with the appearance of an outlier. To handle it, ridge and lasso regression are used.



Types of Problems it can solve(Supervised)

1. Regression

Overfitting And Underfitting
We will use polynomial linear regression to explain bias and variance tradeoff as well as overfitting and underfitting. Please note that if the degree of the polynomial is one, then the curve is a straight line. The sum of mean square error (cost function) is higher in that scenario when compared to a polynomial curve with degree greater than one. When error is very high for a training dataset, then the particular scenario is called *underfitting*. Suppose, we use a quite higher order polynomial such that almost all the training data points are fitted quite well by the model, this scenario is called *overfitting*. This is due to the fact that the accuracy would go down for the test data even though it is very high for the training data.

Our main aim is to achieve an optimum solution such that accuracy is high for both training and test data. That model will give us low bias and low variance.

In an underfitting scenario, we basically have high bias (error of the training data) and high variance (generalizability - error on the test data). For overfitting scenario, we have low bias but high variance.

Different Problem statement you can solve using Linear Regression

1. Advance House Price Prediction

2. Flight Price Prediction

XIV. Support vector Machines (SVM)

Theoretical Concepts:

The main aim of SVM is to create a hyperplane and two separating margin lines such that there are two more hyperplanes in parallel to the particular hyperplane and these hyperplanes pass through the nearest neighboring point in the two clusters.

This whole distance between the r=two parallel hyperplanes - d+ive and d-ive is called margin. We aim to get a generalized model so that it is easily applicable on unseen points. This margin and hyperplane helps in such generalization.

Now, we can create multiple hyperplanes separating the two clusters apart from this one hyperplane. Then, why choose this hyperplane? Our main aim is to select the hyperplane which gives us the maximum marginal distance. This linear hyperplane is for linearly separable points.

What if the points are non-linearly separable?
A linearly separable hyperplane cannot be made for such a dataset. For tackling this we use kernels which convert 2 -dimension to a higher dimension. Then, between the higher dimensional dataset, we can construct linearly separable hyperplanes

What is a support vector?
The nearest positive point and nearest negative point that passes through the parallel marginal hyperplanes are called support vectors. It may have multiple support vectors.

SVM Math Intuition:

Let us consider a simple example of logistic regression where we have two points (4,4) and (-4,0) in a 2-D plane. We wish to separate them using a linear hyperplane. Now, suppose the slope of such a linear plane is -1. The equation of this line is given as
$w'x + b = 0\ where\ '\ denotes\ the\ transpose.$
We also know this equation: y = mx + c; m = -1 (assumption made).
c = b = 0 as the line has been assumed to be passing through origin.Thus, y = $w'x$
$y = [-1\ 0]'[-4\ 0] = 4\ (this\ value\ is\ always\ going\ to\ be\ positive)$

Anytime, we calculate y for points below the linear separator, y is going to be positive. And, when we calculate it for points above the linear separator, y is going to be negative. Now, we can take this value as + 1 and -1 for two clusters respectively.

In SVM, this hyperplane equation can also be given similar to logistic regression:
$w'x + b = 0.$
I am going to find the nearest points for the hyperplane in the two clusters. Suppose, it's at a distance -1 and + 1 respectively as assumed above.
So, the equations can be $w'x + b = -1\ and\ w'x + b = 1$ respectively.

Now, here b will not be zero as the hyperplane is not passing through zero. Now, we basically wish to compute the distance between two points x1 and x2 lying on these two parallel hyperplanes. We can write the equations as follows:
$w'x1 + b = -1 \, and \, w'x2 + b = 1$. So, we can subtract the two equations:
$w'(x2 - x1) = 2$.
To remove w' , we are going to divide by ||w|| on both sides. So, 2/||w|| would be our optimization function and we need to maximize this.
We need to update (w,b) to maximize the optimization function such that y = 1 when $w'x + b$ >= 1 and y = -1 when $w'x + b$ <= -1.

We can also write it as: y*$(w'x + b)$ >= 1

If it is not greater than equal to 1, then it means that it is a misclassification.

One point to note is that in a real world scenario, we will not have such separable data points. There would be a lot of overlap.

We have to change (w*, b*) such that min(||w||/2) + c∑ζi where i ranges from 1 to n. 'c' represents how many errors the model can consider; **ζ** is the value of the error. This value 'c' is called regularization and is obtained by hyperparameter tuning. This particular SVM is called linear SVM and the margin is a ***hard margin***.
(w*, b*) = min(||w||/2) + c∑ζi where i ranges from 1 to n

SVM Kernels:

1. Polynomial Kernels
2. RBF Kernels
3. Sigmoid Kernels

In case of soft margin, there would be certain consideration for error. Now, non-linearly separable data can be handled using the SVM kernel which will convert a lower dimensional non-linearly separable dataset into a linearly separable higher dimensional dataset. This transformation is done by SVM Kernel from lower dimension to higher dimension. It does it using some mathematical formulas.

Let us consider one-dimensional points for example. —-------*****-----****-------******------
To classify these points, in order to use SVM, we first need to draw a hyperplane but they are 1-dimensional. So, we would use a kernel to make this dataset 2-dimensional. Let us suppose, we are using a Polynomial kernel (y = f(x) = x²). So, these points get projected as a parabola. Then, we can draw a hyperplane to separate these points.

Polynomial Kernel: Let us consider we have elliptical data points which are non-linearly separable in x1 and x2 plane. We will use a polynomial kernel to make the data points linearly separable in a higher dimension.

The formula for such a polynomial kernel would be: f(x1, x2) = (x1'.x2 + 1)^d where d is the higher order dimension.

The unique elements in x1 and x2 multiplication would be $x1^2$ and $x2^2$ whereas the repetitive parts would be x1x2. So, we see that our 2-dimensional points are becoming higher-dimensional - x1, x2, $x1^2$, $x2^2$, x1x2 (axes: x1, x2 and x1x2) such that we can obtain linearly separable data points and hyperplane.

1. What Are the Basic Assumptions?

There are no such assumptions

2. Advantages

1. SVM is more effective in high dimensional spaces. : the reason that SVMs work well with high-dimensional data is that **they are automatically regularized**, and regularization is a way to prevent overfitting with high-dimensional data.
2. SVM is relatively memory efficient. (It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.)
3. SVM's are very good when we have no idea on the data.
4. Works well with even unstructured and semi structured data like text, Images and trees.
5. The kernel trick is the real strength of SVM. With an appropriate kernel function, we can solve any complex problem.
6. SVM models have generalization in practice, the risk of overfitting is less in SVM.

3. Disadvantages

1. More Training Time is required for larger dataset
2. It is difficult to choose a good kernel function https://www.youtube.com/watch?v=mTyT-oHoivA
3. The SVM hyperparameters are Cost -C and gamma. It is not that easy to fine-tune these hyper-parameters. It is hard to visualize their impact

4. Whether Feature Scaling is required?

Yes - Feature scaling is crucial for some machine learning algorithms, which **consider distances between observations because the distance between two observations differs for non-scaled and scaled cases**. As we've already stated, the decision boundary maximizes the distance to the nearest data points from different classes.

5. Impact of Missing Values?

Although SVMs are an attractive option when constructing a classifier, SVMs do not easily accommodate missing covariate information. Similar to other prediction and classification methods, in-attention to missing data when constructing an SVM can impact the accuracy and utility of the resulting classifier.

6. Impact of outliers?

It is usually sensitive to outliers. The penalty on misclassification is defined by a convex loss called the hinge loss, and the unboundedness of the convex loss causes sensitivity to outliers.

Types of Problems it can solve(Supervised)

1. Classification
2. Regression

Overfitting And Underfitting

In SVM, to avoid overfitting, we choose a Soft Margin, instead of a Hard one i.e. we let some data points enter our margin intentionally (but we still penalize it) so that our classifier don't overfit on our training sample

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Different Problem statement you can solve using SVM

1. We can use SVM with every ANN use cases
2. Intrusion Detection
3. Handwriting Recognition

Practical Implementation

1. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
2. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

Performance Metrics

Classification

1. Confusion Matrix
2. Precision,Recall, F1 score

Regression

1. R2,Adjusted R2
2. MSE,RMSE,MAE

Which all algorithms are impacted by an imbalance dataset?

Machine Learning Algorithms such as Logistic Regression, KNN,SVM (basically which includes Gradient Descent and Euclidean Distance Computation) are affected by Imbalanced Datasets.

XV. Decision Tree Classifier And Regressor

Interview Questions:

1. Decision Tree
2. Entropy, Information Gain, Gini Impurity
3. Decision Tree Working For Categorical and Numerical Features
4. What are the scenarios where Decision Tree works well
5. Decision Tree ***Low Bias And High Variance***- Overfitting
6. Hyperparameter Techniques

7. Library used for constructing decision tree
8. Impact of Outliers Of Decision Tree
9. Impact of missing values on Decision Tree
10. Does Decision Tree require Feature Scaling

Theoretical Understanding:

1. ***Entropy***: f1 , f2, f3 etc. features; o/p is yes or no

   ID3 Algorithm → which node to select for split up

   Here, is when entropy comes into picture to select the right attribute for splitting purposes.

   Entropy helps us to measure the purity of the split.

   Suppose we split f1. Initially, we had 9 yes and 5 No's. After the split, at f2 we had 3 yes, 2 No and at f3 we had 6 yes / 3 No. Ideally, after the split we should have all the Yes on one side and all the No on another.

   Again the split happens, this time we get 3 Yes, 0 No ; 2 No and 0 Yes on one side. We get 6 Yes, 0 No; 3 No, 0 Yes on the other side after the split. In order to get the correct leaf nodes fast, we need to select the right parameters / features of the dataset for split.

   We have to go and calculate the purity split at each split.

   Entropy: $H(S) = -P_{+}\log_2(P_{+}) - P_{-}\log_2(P_{-})$

   P_{+}/P_{-} : % of +ive cases / % of -ive cases

   S: subset of training sample

   Entropy for above example:

   $H(S) = -3/5\log_2(⅗) -2/5\log_2(2/5)$

   H(S) = 0.78 bits APPROX.

   Note: If we have a completely impure subset (3 Yes, 3 No), then its entropy is 1 bit. And, for a pure split (4 yes, 0 no) , entropy is 0 bits.

   So, we would check for entropy at f2 and f3 for above example and select the one for which entropy is better. But, this calculation is just for one node. We should also consider other attributes which are needed for reaching pure leaf nodes. For that, we use Information gain. Basically, it calculates total entropy value from top to the leaf node to decide which path is better. Entropy ranges between 0 and 1.

2. ***Information Gain***: In short, we can say that the average of all the entropy is based on a specific split. Suppose, based on f2 we wish to split into f1 and f3; or based on f1 into f2 and f3. We need to decide which split would be better to reach leaf nodes earlier. For that we need information gain.

   $Gain(S,A) = H(S) - \sum|S_v| / |S| (H(S_v))$ where v belongs to val

Here, S_v is the subset after the split, S is the total subset, H(S) is the entropy; $H(S_v)$ is the entropy after the splitting for the subset

Suppose, f1 (9Y, 5N) is splitting into f2 and f3. For f2, we get (6Y, 2N) and for f3 (3Y, 3N).

Here, H(f1) = H(S) i.e. entropy before the split = 0.94

$H(S_v)$ is the entropy after the split

H(f2) = 0.81 ; H(f3) = 1

Thus, gain = 0.91 - (8/14 * 0.81) - (6/14 * 1) = 0.049

Instead, I can also go from f2 splitting into f1 and f3. So, we will check which one gives a higher information gain and we will select that particular way of splitting.

3. ***Gini Impurity***: Both entropy and gini impurity do the same task which is to calculate the purity of the split. But, in most cases Gini impurity is preferred. Let us see why?

   This is computationally more efficient and takes a lesser amount of time. For entropy, logarithmic calculations take some amount of time whereas in GI , we do not have any logarithmic calculations.

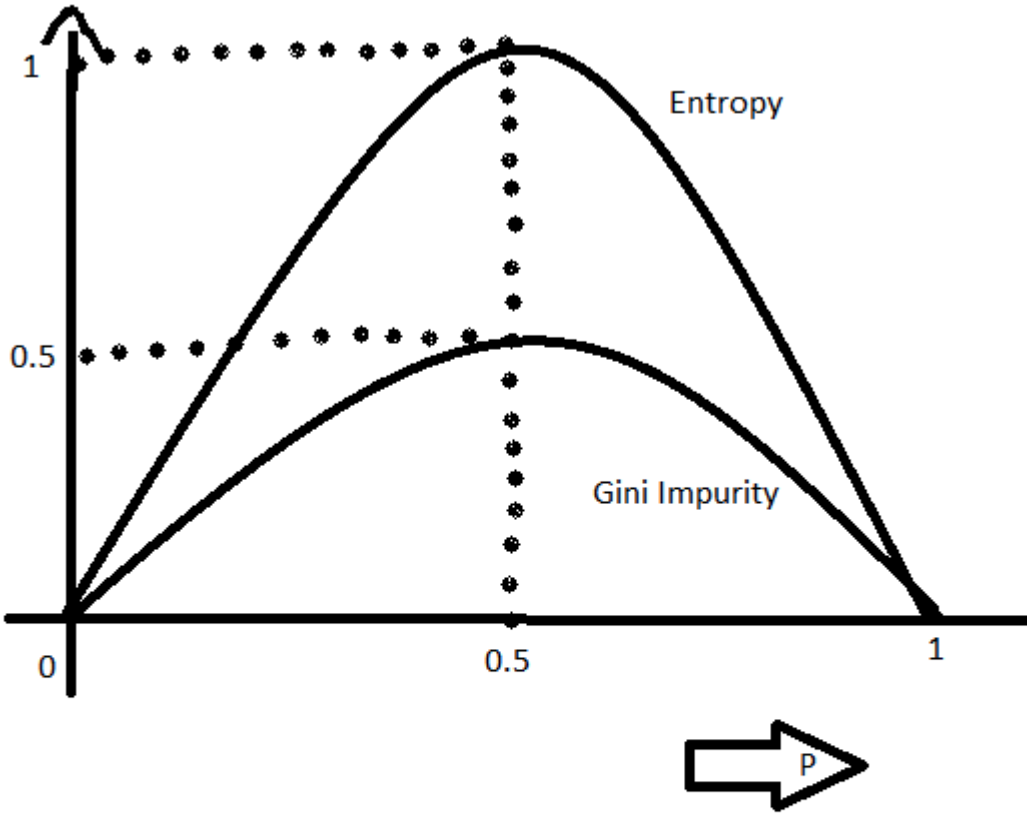
   F1 (6Y/3N) → f2 (3Y/3N) and f3 (3Y/0N)

   Entropy → 0.5 → 1 and 0 respectively

   Now, Gini Impurity, GI = 1 - $\sum(P^2)$ where i ranges from 1 to n

   Where P^2 is the summation of squares of P_+ and P_-

   GI = 1 - [(3/6)^2 + (3/6)^2] = 1 - [0.25 + 0.25] = 0.5 , whereas entropy for f2 was 1. So, we are getting less GI when compared to entropy.



$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

$$Gini(E) = 1 - \sum_{j=1}^{c} p_j^2$$

4. If your feature is a numerical variable, the first step decision tree algorithm does is sorting all the values. It will try to consider some threshold values and is going to check for each and every record for a particular feature. If that xi is less than or equal the threshold, it will create a branch for it

   In the given example, f1 (4yes, 4 no) , threshold is 2.3; and it is split; on one side there is just 1 Yes. On the other side (>2.3) we have 3Yes / 4No. The next threshold would be considered the next value, say 3.6. Again the split will happen for <= 3.6 (2Yes / 0No) and > 3.6 (1Yes/4 No). For this entropy and information gain is calculated and then decided based on which has the better value and that split is taken.

   There is a disadvantage here. Suppose, we have millions of records. The time complexity would keep on increasing with so many samples. Decision trees with numerical features will take more time for training. The same disadvantage is true for ensemble methods.

1. What Are the Basic Assumptions?

There are no such assumptions

2. Advantages

Advantages of Decision Tree

1. ***Clear Visualization*:** The algorithm is simple to understand, interpret and visualize as the idea is mostly used in our daily lives. Output of a Decision Tree can be easily interpreted by humans.
2. ***Simple and easy to understand***: Decision Tree looks like simple if-else statements which are very easy to understand.
3. Decision Tree can be used for ***both classification and regression problems.***
4. The Decision Tree can handle ***both continuous and categorical variables***.
5. ***No feature scaling required***: No feature scaling (standardization and normalization) required in case of Decision Tree as it ***uses rule based approach instead of distance calculation.***
6. ***Handles non-linear parameters*** efficiently: Non linear parameters don't affect the performance of a Decision Tree unlike curve based algorithms. So, if there is high non-linearity between the independent variables, Decision Trees may outperform as compared to other curve based algorithms.
7. Decision Tree can ***automatically handle missing values***.
8. The Decision Tree is usually ***robust to outliers*** and can handle them automatically.
9. ***Less Training Period***: Training period is less as compared to Random Forest because it generates only one tree unlike forest of trees in the Random Forest.

3. Disadvantages

Disadvantages of Decision Tree

1. ***Overfitting***: This is the main problem of the Decision Tree. It generally leads to overfitting of the data which ultimately leads to ***wrong predictions***. In order to fit the data (even noisy data), it keeps generating new nodes and ultimately the ***tree becomes too complex to interpret.*** In this way, it ***loses its generalization capabilities***. It performs very well on the trained data but starts making a lot of mistakes on the unseen data.
2. ***High variance***: As mentioned in point 1, Decision Tree generally leads to the overfitting of data. Due to the overfitting, there are very high chances of ***high variance*** in the output which leads to many errors in the final estimation and shows high inaccuracy in the results. In order to achieve zero bias (overfitting), it leads to high variance.
3. ***Unstable***: Adding a new data point can lead to re-generation of the overall tree and all nodes need to be recalculated and recreated.
4. ***Not suitable for large datasets***: If data size is large, then one single tree may grow complex and lead to overfitting. So in this case, we should use Random Forest instead of a single Decision Tree.

4. Whether Feature Scaling is required?

No

6. Impact of outliers?

It is ***not sensitive to outliers***.Since, extreme values or outliers, never cause much reduction in RSS, they are never involved in split. Hence, tree based methods are insensitive to outliers.

Types of Problems it can solve(Supervised)

1. Classification

2. Regression

Overfitting And Underfitting

How to avoid overfitting

Practical Implementation

1. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
2. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html

Performance Metrics

Classification

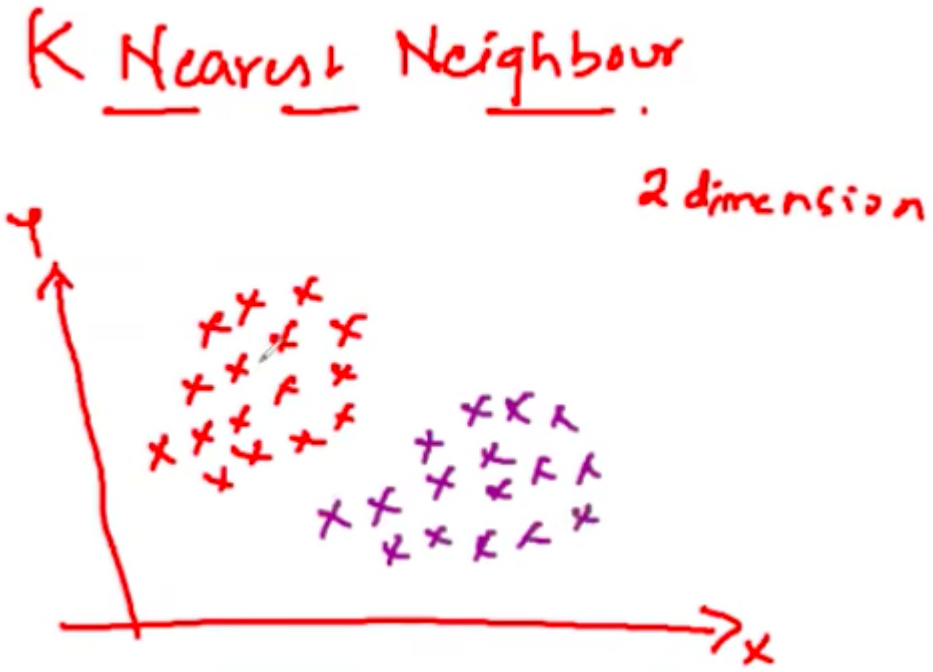
1. Confusion Matrix
2. Precision,Recall, F1 score

Regression

1. R2,Adjusted R2
2. MSE,RMSE,MAE

XVI. K Nearest Neighbor (KNN) Classification

Suppose I have a two-dimensional dataset as shown below:

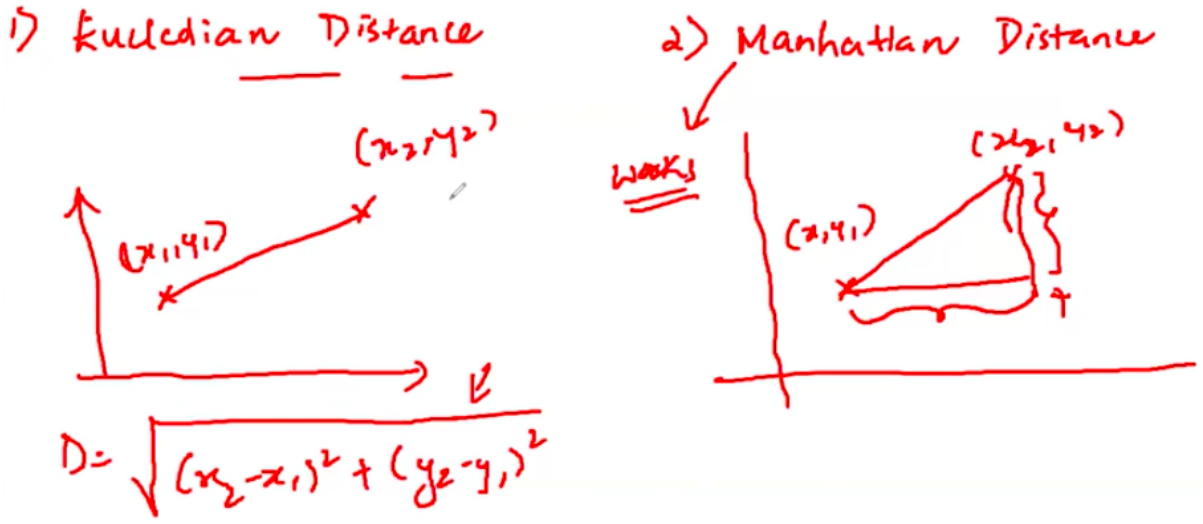


If a new point comes, then how do we determine whether this point belongs to Category 1 or Category 2. KNN helps us to determine the answer to this question.

In the first step, we select the K value which means how many nearest neighbors would be considered. After K is decided, the nearest K points from the new point are

identified for the two categories. So, we basically calculate this distance of the K nearest neighbors. The third step would be to identify how many nearest neighbors are of Category 1 and how many of Category 2 in the total K nearest neighbors chosen. If there is a maximum number of Category 1 neighbors, then that point belongs to Category 1 or else it would belong to Category 2. Once the model is able to classify points in this manner, the model is ready.

Now, how is this distance calculated?
Euclidean Distance: This formula says the distance between two points (x1 1 , y1 1) and (x2 2 , y2 2) is $\mathbf{d = \sqrt{[(x_2 - x_1)^2 + (y_2 - y_1)^2]}}$.
Manhattan Distance: Manhattan distance is calculated as **the sum of the absolute differences between the two vectors**. The Manhattan distance is defined by $\mathbf{Dm(x,y)=\sum i=1D|xi-yi| = |x2 - x1| + |y2 - y1|}$, which is its L1-norm.



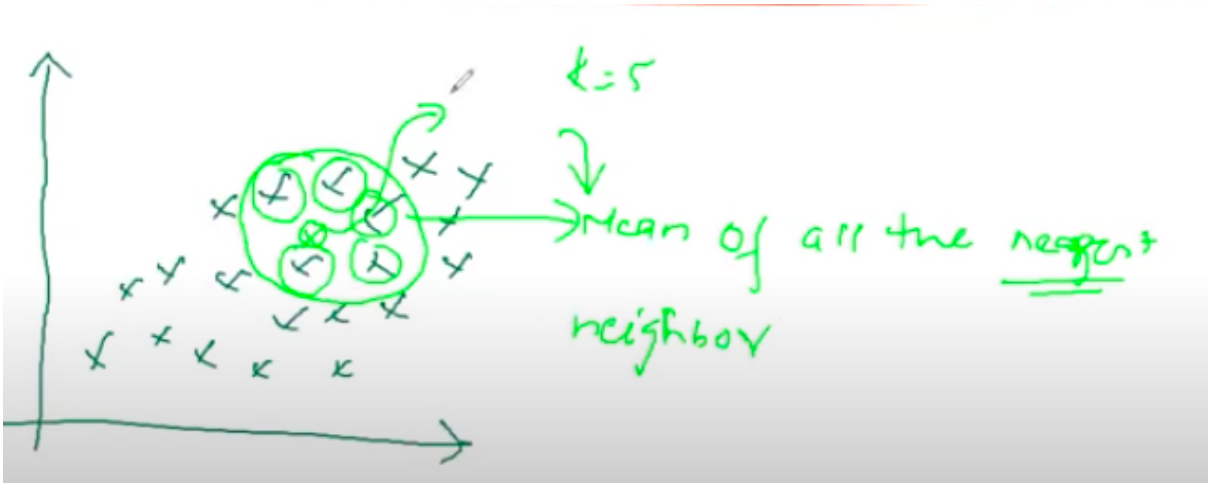
Suppose we have a highly imbalanced dataset with 900 Yes and 100 No, will the KNN classification model be biased?
Now, suppose a new point comes over in this dataset. Then, will the KNN be biased?

It will definitely get impacted. Suppose we take K as 150 and take 150 nearest neighbors. At that time, the maximum points present are of the majority category and this leads to bias inherent in the output.

What if we have outliers? Will the result of the classification get impacted?
The kNN may get biased or impacted because the bias cluster of neighbors might get selected in the K nearest neighbors.

If we get an equal number of nearest neighbors on both sides, we should then select the value of k as odd.

Regression use case of KNN: We take the mean value of the nearest neighbors to predict the regression value of a particular new input.



XVII. Ensemble Techniques

Two techniques:

Bagging (Bootstrap aggregation)

One of the examples is RandomForest.

Boosting

Examples: ADABOOST, Gradient Boosting, XgBoost

Bagging Technique:

Combining multiple models - for each and every model, we will provide a sample of records D' such that the dataset D' is always less than the overall dataset D. The dataset is resampled again and provided to the other model M2. This is called Row sampling with replacement. The models will get trained for the respective datasets simultaneously. For M1, another D'' data goes for testing and a certain output is generated. For a binary classifier model, the output would be of the form 0 and 1. Then, the output of all the models is applied with a voting classifier - majority of the output is taken as the final output.

If the majority vote is for 1 , then the output is 1 else it is 0. The step where row sampling with replacement is done is called bootstrap. The step where voting is done is called aggregation. Thus, the technique is called bootstrap aggregation.

Now, in Random forests, the models M1, M2 etc. get replaced with decision trees.

Random Forests:

In Random Forest , the models M1, M2, M3 etc. are decision trees. So, we also do row sampling with replacement and we also do feature sampling with replacement along with it. For aggregating, we use majority vote.
A decision tree model has low bias and high variance - it basically is prone to overfitting. The training error is very less but the test error might be high. In Random Forests, we are using multiple decision trees, and each might have high variance, but when we combine the result of these decision trees , it leads to low variance for the Random Forest model.

Suppose we have 1000 records in the beginning, and suppose at a later stage we change 200 records, will the result change for the Random Forest?
This data change will not make much impact on the model output because the data gets split in various decision trees using row sampling with replacement.

Instead of classification, for a regression problem, the random forest takes the average or median of the outputs from the various decision trees.

What are the hyper parameters used for Random Forest?
Number of decision trees to be used for a specific model

Boosting Techniques:
Base Learner models created sequentially to work on the datasets incorrectly classified by each previous base learner.
It will go on unless we provide the number of base learners to be used.

ADABoost (Boosting Technique):

Suppose for a dataset, we have 7 records, 3 features and one output column. We also assign a sample weight in ADABOOST. Initially, all the records are basically assigned the same weights. In Step 2, we create our first base learner with the help of decision trees in ADABoost. It is created with the help of only one depth unlike random forests. These decision trees are called Stomps. When we consider f1 , we create one stomp. For f2, we create another stomp and for f3, we consider another stomp. From these stomps, we need to select the first sequential base learning model based on which model has least Gini impurity or entropy.

Suppose the first option model classifies 4 correct and 1 incorrect records. Total error is calculated by summing up all sample weights which are misclassified. Thus, just one record is misclassified, then total error would be 1/7.

Now, performance of stomp is given by the following formula:
½ loge ((1 - total error)/total error).

Thus, performance of stomp = ½ loge(1 - 1/7 / 1/7) = 1/2loge6 = 0.896

Now, we need to update this weight for the records. We are going to increase the weights of incorrectly classified records whereas decrease the weights for others.

Now, new sample weight for incorrectly classified record = old weight * e^(performance)

New sample weight for incorrectly classified = 1/ 7 * (e^0.896) = 0.349

new sample weight for correctly classified record = old weight * e^(-performance)

New sample weight for correctly classified = 1/ 7 * (e^-0.896) = 0.05

Summation of all the weight values is done and then the weights are normalized by dividing by this sum. This will lead to a total sum value of weights equalizing to 1.

Now, for the second sequential model, based on the normalized weights, buckets are made. Our algorithm will run 8 different iterations to select the record randomly. Based on this new dataset, a new stomp model is prepared based on which model has least entropy or gini impurity.

Finally, after passing through all the stomps, we would get quite less error than at the initial stage.

Then, the majority vote would happen as it happens in random forests. Basically, we are combining weak learners to make a strong learner.

Gradient Boosting (Boosting Technique):

Suppose, we have two independent features: Experience and Degree; dependent feature is salary.

Step 1: Devise a base model to compute the average salary of all the outputs. We can take it as y', let's say we got it as 75.

Step 2: Compute residual errors, pseudo residuals:
Residual formula for example is (actual - prediction). There are various loss functions. This is just an example.

Step 3: Construct a decision tree sequentially. Inputs would be experience and degree. Output would be residual errors, R1.

Step 4: If we pass the independent features again from the decision tree, we get R2 values.

This leads to overfitting after calculating y' + R2 (i.e. predicted output + residuals). To prevent this, we will be adding a learning rate , α multiplied by R2. This results in a very higher salary as residuals get reduced by multiplying with alpha.

Thus, in the next step we add another decision tree and so on.

$F(x) = h_o(x) + \alpha_1 h_1(x) + \alpha_2 h_2(x) + \ldots\ldots + \alpha_n h_n(x)$

The main aim is to reduce this residual error. Here, alpha ranges from 0 to 1.

This is called a sequential tree or a boosting tree.

Mathematical intuition and Pseudo algorithm:

Wikipedia Link: https://en.wikipedia.org/wiki/Gradient_boosting

Inputs:

- $\{x_i, y_i\}$,
- Loss function (differentiable) - log loss, hinge loss etc. $L(y, FC_x)$,
- Number of Trees

Pseudo algorithm:

1. Initialize the model with constant value

$$F_0(x) = \arg\min_{\gamma} \sum_{i-1}^{n} L(y_i, \gamma)$$

Gamma is nothing but the predicted value. The loss function would basically be 1/2 $\sum(y' - y)^2$ where i ranges from 1 to n and y' is the predicted value.

We need to find a y' value such that the loss is minimized. Take the first order derivative of the loss function with respect to y' and find y'

2. For m = 1 to M:
M is the number of trees.

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

Fit a base learner (or weak learner, e.g. tree) closed under scaling $h_m(x)$ to pseudo-residuals, i.e. train it using the training $\{(x_i, r_{im})\}$

Compute multiplier γ_m by solving the following one-dimensional optimization problem

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

Output $F_M(x)$.

Extreme Gradient - XgBoost (Boosting Technique):

In XgBoost, we always go for binary splits even if we have more than two classes.

Suppose, we have features, such as Salary, creditScore (Bad, Good, Normal) and the output is Loan approval(1/0).

Salary	Credit Score	Approval	Res	
<=50K	B	0	-0.5	
<=50K	G	1	0.5	
<=50K	G	1	0.5	
>50K	B	0	-0.5	
>50K	G	1	0.5	
>50K	N	1	0.5	
<=50K	N	0	-0.5	

We will start constructing the tree for base model:

[-0.5,0.5,0.5,-0.5,0.5,0.5,-0.5]
Salary → <50K and >50K i.e. [-0.5,0.5,0.5,-0.5] and [-0.5,0.5,0.5] respectively.

Step 1: Construct a base model

Step 2: Calculate the similarity score:
∑(Residuals)^2 / (∑Probability(1 - Probability) + **λ**)

Thus, similarity score for first split in our example: [-0.5 + 0.5 + 0.5 - 0.5]^2 /[0.5(1-0.5) + 0.5 (1 - 0.5) + 0.5 (1 - 0.5) + 0.5 (1 - 0.5)] = 0

Similarity score for second split: (-0.5 + 0.5 + 0.5)^2 / /[0.5(1-0.5) + 0.5 (1 - 0.5) + 0.5 (1 - 0.5)] = 0.25/0.75 = 0.33

Similarity weight for the initial salary (root) = 0.25 / 1.75 = 1/7 = 0.142

Step 3: Compute the gain

Gain = 0 + 0.33 - 0.14 = 0.21

Now, we will go for a credit split (binary): Bad vs. Good & Normal: [-0.5] vs.[0.5, 0.5, -0.5] for the first salary split in the previous step. Their similarity scores would come out to be 1 and 0.33 respectively.

The gain would be 1 + 0.33 - 0 = 1.33

Similarly, we would calculate the gain for the right side split by considering the categories as Bad and Good vs. Normal. The split which gives better information gain is what we would choose. The similarity scores in this case would be 0.33 and 1 respectively. So, we are getting the same gain. Then we can go ahead with either of them.

To decide whether we need to do further split or not - we use post pruning. It is basically calculated by a cover value = Probability(1 - Probability). We can find the value and cut the tree based on this or branch it.

Whenever I get new data, I need to find the base model output using log(odds) = log(P/1-P). For Probability 0.5, log(odds) = 1. The learning rate is 0.01. Then, applying sigmoid function σ(0 + 0.01*1) = σ(0.1) = 1 / 1 + e^ -0.1 = 0.6

Then, my new probability will get computed. Based on this new probability, new residuals are calculated. We will compute the decision trees by taking the residual independent features and take $\sigma(0 + \alpha(T_1) + \alpha(T_2) + \alpha(T_3) + \ldots\ldots \alpha(T_n)$. Probability would be good when residuals would be very less.

We can select this alpha value as a hyper parameter.

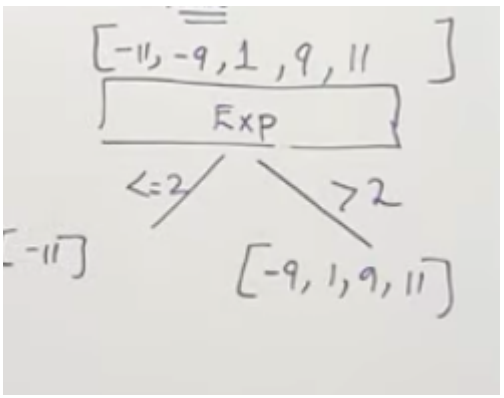
XgBoost Regression - In-depth Intuition

Experience	**Gap**	**Salary**	**Res1(w.r.t. 51K)**
2	Yes	40K	-11K
2.5	Yes	42K	-9K
3	No	52K	1K
4	No	60K	9K
4.5	Yes	62K	11K

Let us try to create a Base Model by taking the average of all the salaries → 51K.

Based on this average salary, we would find Residual, Res1.

Now, we would take Experience, Gap and Res1 and try to find the root node on which we would split.



We calculate similarity weight.

Now, if we recall that in XgBoost Classifier, the similarity weight was calculated by the following formula: ∑(Residuals)^2 / (∑Probability(1 - Probability))

Here, for XgBoost Regressor, this formula would change, and the similarity weight would be calculated by the following formula: ∑(Residuals)^2 / (No. of Residuals + **λ**)

To check if the split around 2 years experience was a good split or not, we calculate the similarity weight followed by Information gain and then decide.

Thus, similarity weight for the left side of current split using the above formula = 121 / 1 + 1) if we take **λ** = 1;

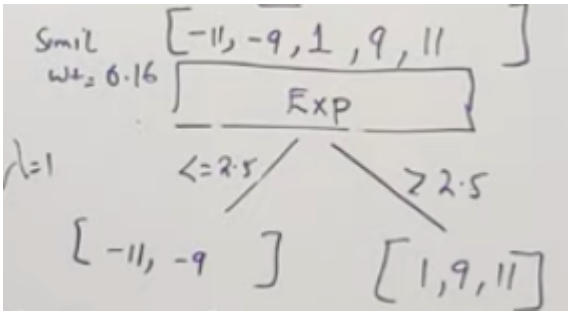
Thus, the similarity weight here for the left side of current split around 2 years experience comes out to be 60.5

Similarity weight for the right side of the split is calculated the same way: $(-9 + 1 + 9 + 11)^2/4 + 1 = 144/5 = 28.8$

Now, the similarity weight of the root: $(-11 + 1 - 9 + 9 + 11)^2/5 + 1 = 0.16$

Thus, our **information gain for first split** = 60.5 + 28.8 - 0.16 = **89.14**

Now, I will go with the next split. Let us split around 2.5 years of experience instead of 2 years as done earlier.



The left side would include [-11, -9] and the right side would include [1, 9, 11].

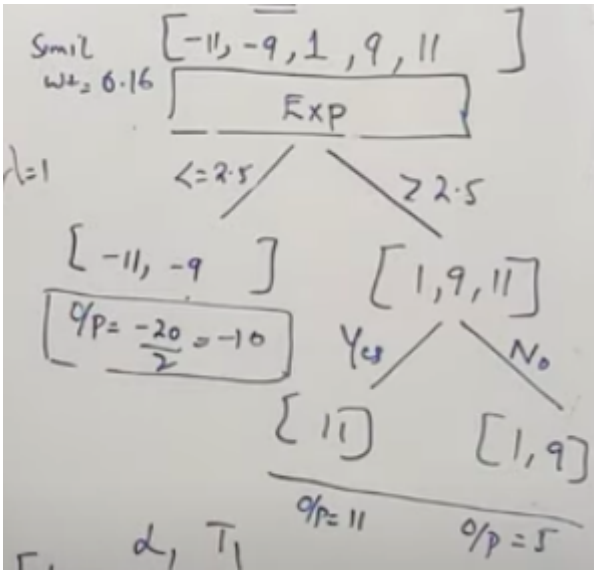
In this case, the similarity score for the left side would come out to be 133.33 and the similarity weight for the right side would be 110.25.

Thus, our **information gain for this split** = 133.33 + 110.25 - 0.16 = **143.42**

We deduce that this gain is better than the earlier split, thus we would go with this split rather than the first one. Similarly, we check for other possible splits and choose the one with the best information gain.

The second split after deciding the first split would be around Gap (Yes, No) and would try to compare which has the highest information gain.

Finally, the output of each node would be the average value.



After this base model, we pass the records through this base model,

Let us consider our learning parameter, alpha = 0.5. Suppose the first record is being passed through this base model, the result would be: 51 + (0.5)[-10] = 46K

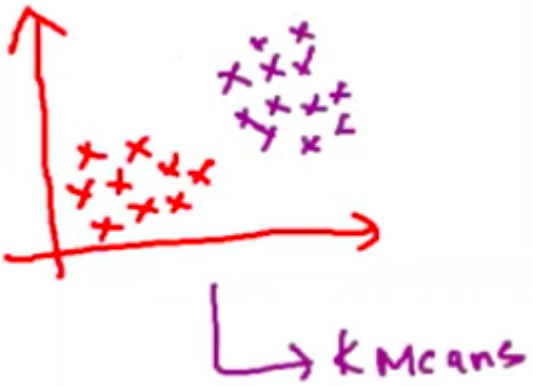
Like this, we consider multiple decision trees, output = base salary + $\alpha(T_1)$ + $\alpha(T_2)$ + $\alpha(T_3)$ +..... $\alpha(T_n)$

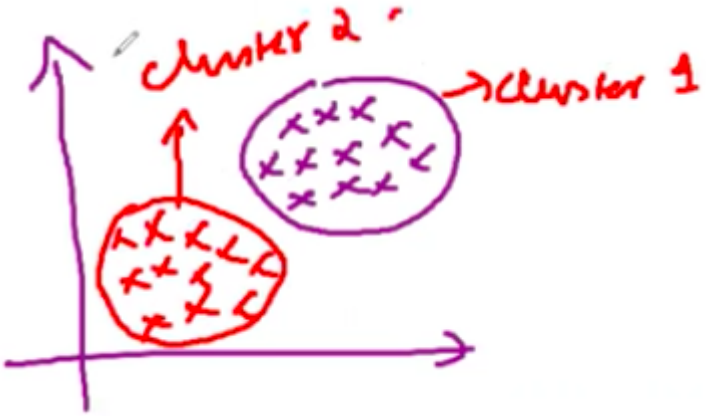
There is also one more parameter, γ. For example, we can take γ as 150.5. Now, we subtract information gain after each split from γ and see if the value is negative or positive. If this value is negative, we can prune the tree at this point whereas we can continue if the value is positive. This γ is a hyperparameter that is used in post pruning.

XVIII. K Means Clustering (Unsupervised technique)

Note: Elbow method is used to find K

Math Intuition:

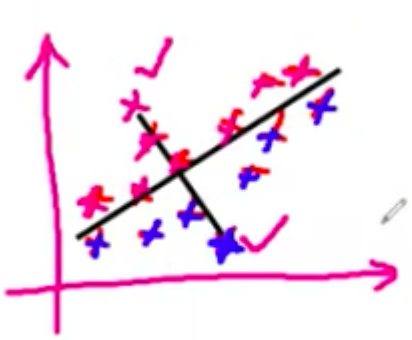




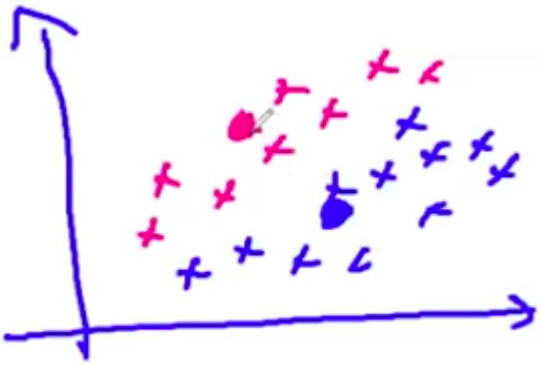
How does this algorithm work?
What metrics should we use?
Which distance do we consider - Euclidean or Manhattan?
What is this elbow method?

This K value is basically centroids - which means those many clusters.

Step 1: Let us suppose K = 2
Step 2: Initialize K centroids randomly in the plane
Step 3: We will use Euclidean distance and find points which are nearer to these centroids respectively. We select the groups and find the mean value for each group.



Step 4: Move the centroids taken initially to these mean positions.



Step 5: Again, we find out which all points are nearer to the new centroids and redefine the groups and calculate their means. Update the centroids and continue so on.

This will continue unless we get two fixed constant groups with less changes. No movement of points happens from one group to another.

But, how do we select this K value initially?

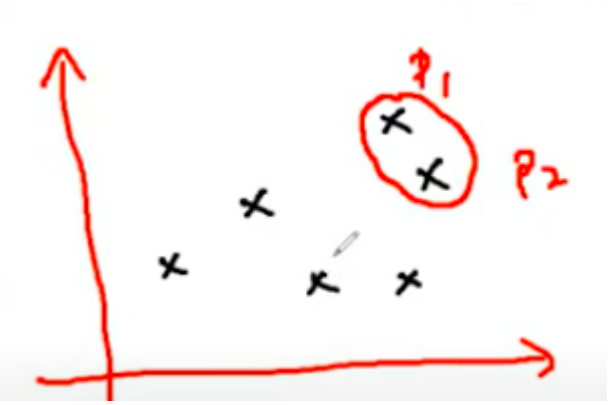
Using Elbow method: We run a loop from K = 1 to say some n value (suppose 20)

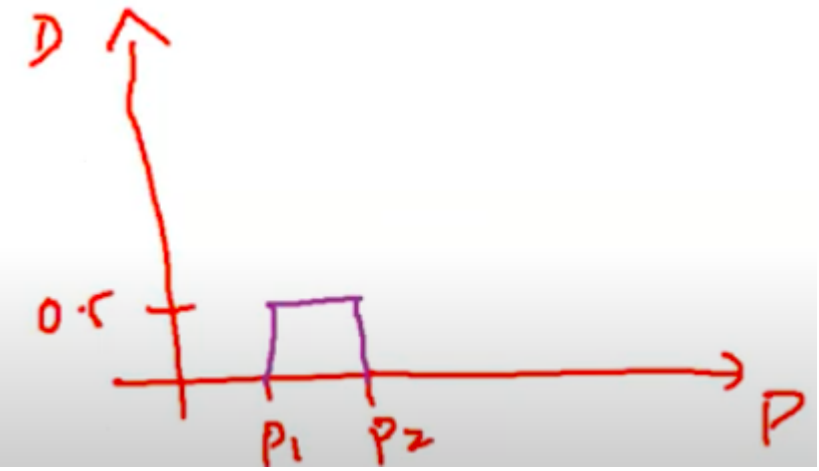
Whole process of K means it is run for K= 1. Then, we find out within cluster sum of squares, Wcss = $\sum(c_i + x_i)^2$ where i ranges from 1 to n. For K = 1, this Wcss would be quite high.

Then, we plot Wcss vs. K. Initially, with K = 1, Wcss would be quite high. As we increase K = 2, this Wcss value decreases because we have two centroids now. At a certain point, this Wcss value becomes almost constant. We have to select this last K value that had an abrupt decrease. After selecting a particular optimal K value, we implement the K-Means clustering algorithm using the above steps.

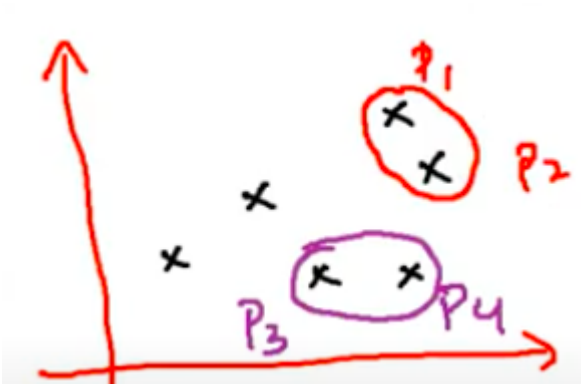
XIX. Hierarchical Clustering (Unsupervised technique)

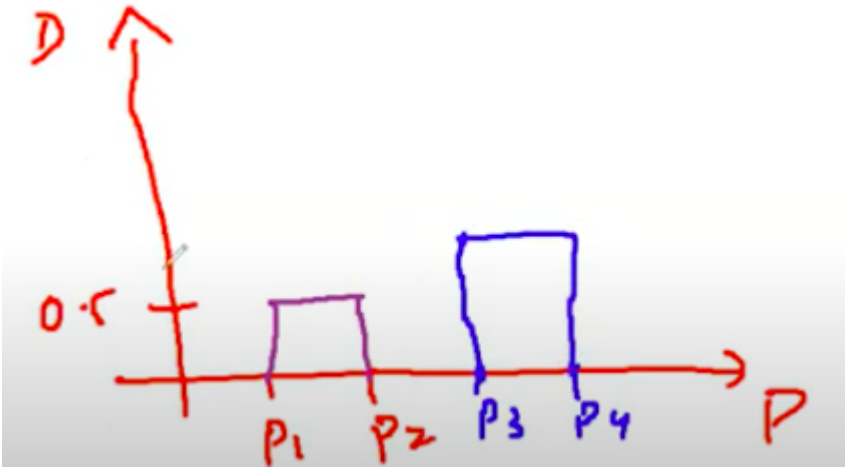
Initially, we would have some points. Each and every point is specified as a distinct cluster initially. Suppose, two points appear to be nearer and another three points appear nearer than to the rest of the points. We would combine these points in a dendogram and find the average value.



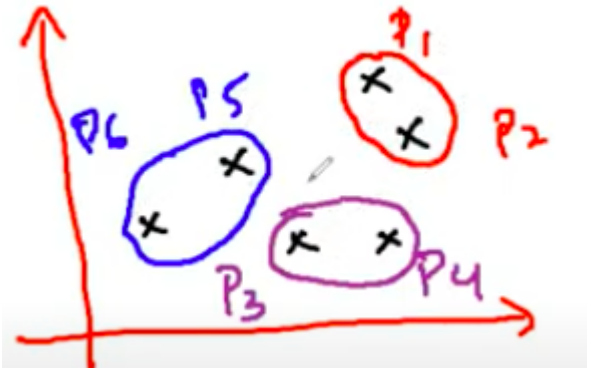


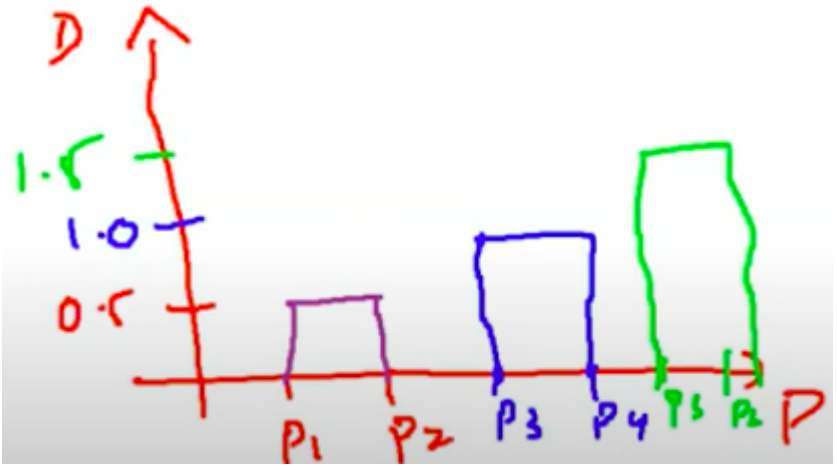
Then, we find the next two or more closer points and define another dendogram.



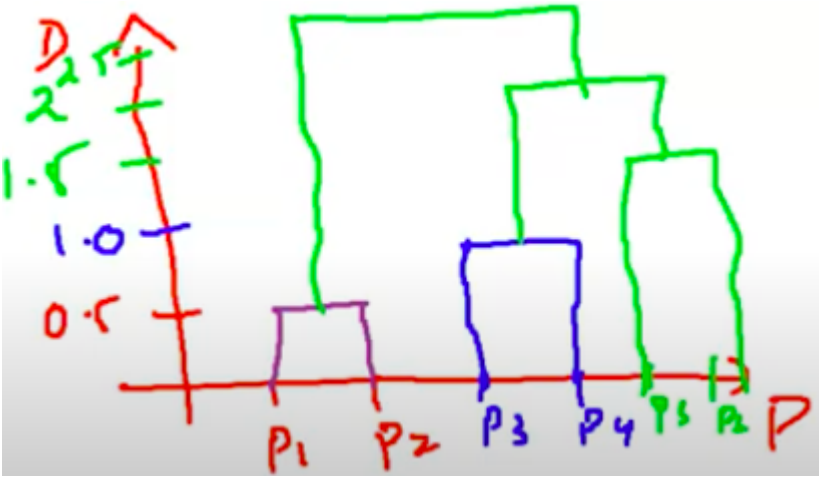


Then, we proceed to find the next nearest points and define another dendogram.



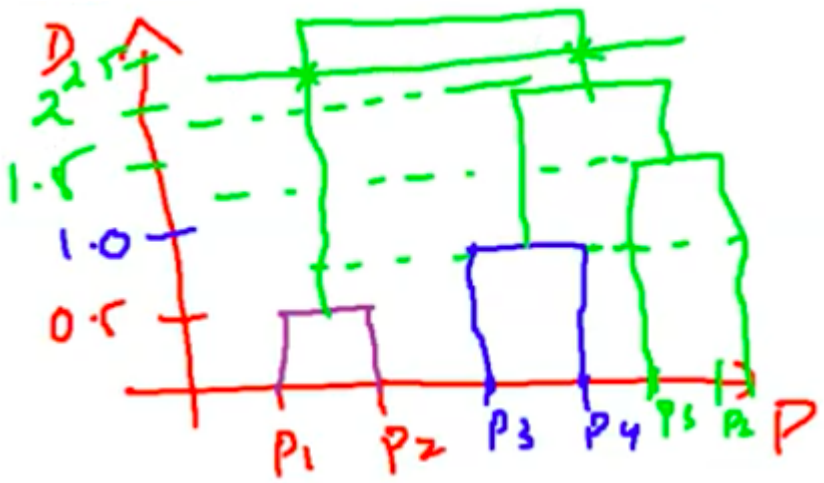


Then, we try to find out which two clusters are closer to each other. Then, we combine those two clusters. And finally we combine the entire groups to form a single cluster.

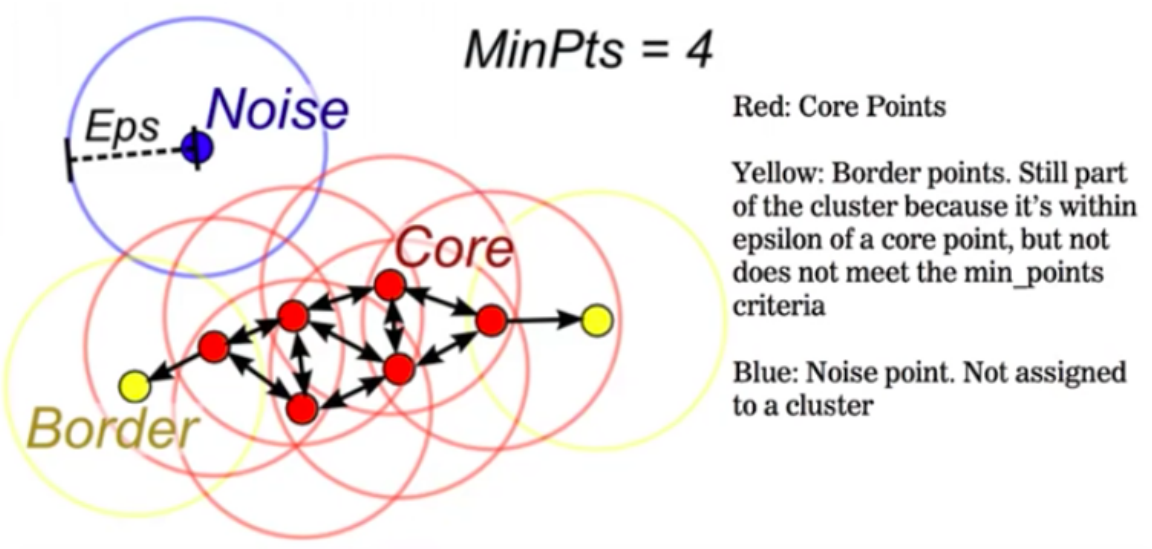


Our main aim is to find out basically what should be the exact number of clusters we should use. This distance is measured by the Euclidean distance formula. To find out the exact number of clusters we should form, we should select the longest vertical line such that no horizontal line passes through it.

Now, once we find this line, we draw a horizontal line and find how many intersecting points are there. Those many clusters need to be formed.



XX. DBScan algorithm (Density-Based Spatial Clustering of Applications with Noise) {Unsupervised Technique}



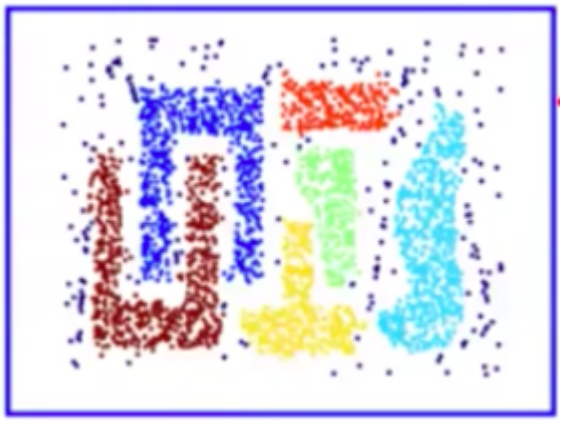
Five components:

1. Epsilon
2. Minimum points
3. Core points
4. Border points
5. Noise points

We initially consider some Epsilon and minimum points values. Epsilon value indicates the radius to create a circle. Within this circle, I have to consider at least certain points given by minPoints. The point within this circle is Core point. Suppose, a specific point does not satisfy the condition of minimum points, even though it's within epsilon of a core point, then this is called a border point. Suppose, I have another point which does not satisfy epsilon and minimum number of points condition, then this point becomes a noise point or an outlier. DBScan treats these noise points very nicely and does not treat them as a separate cluster.

Example of K-Means clustering with noisy data:

Example of DBScan clustering with noisy data:



Advantages of DBScan:

- Is good at separating clusters of high density versus clusters of low density within a given dataset
- Good with handling outliers

Disadvantages of DBScan:

- Not works well with clusters of varying densities. It struggles with clusters of similar densities.
- Struggles with high dimensionality data.

Library used: sklearn.cluster.DBSCAN

XXI. Validation Techniques for Clustering algorithms

1. ***Silhouette (Clustering)*** :
   The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from −1 to +1, where a high value indicates that the object is well matched to

its own cluster and poorly matched to neighboring clusters. If most objects have a high value, then the clustering configuration is appropriate. If many points have a low or negative value, then the clustering configuration may have too many or too few clusters.

The silhouette can be calculated with any distance metric, such as the Euclidean distance or the Manhattan distance.

Three steps are involved:
Step 1: We will take one datapoint and will calculate the distance within the cluster. We will calculate this distance for each point with respect to the other points in the same cluster. This average of distance is denoted by a_i and is given by the following formula:

For data point $i \in C_I$ (data point i in the cluster C_I),

$$a(i) = \frac{1}{|C_I| - 1} \sum_{j \in C_I, i \neq j} d(i, j)$$

Where, a(i) is the mean distance in cluster C_I for data point i. $|C_I|$ is the number of points belonging to cluster I and d(i,j) is the distance of point i from j. We have taken $|C_I|$ - 1 in the denominator because we are not considering the distance of point i from itself.

2. We then define the mean dissimilarity of point i from a cluster C_J as the mean of the distance from i to all the points in C_J (J is not equal to I).

For each data point $i \in C_I$, we now define

$$b(i) = \min_{J \neq I} \frac{1}{|C_J|} \sum_{j \in C_J} d(i, j)$$

3.

We now define a *silhouette* (value) of one data point i

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \text{ if } |C_I| > 1$$

and

$$s(i) = 0, \text{ if } |C_I| = 1$$

Which can be also written as:

$$s(i) = \begin{cases} 1 - a(i)/b(i), & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ b(i)/a(i) - 1, & \text{if } a(i) > b(i) \end{cases}$$

From the above definition it is clear that

$$-1 \leq s(i) \leq 1$$

If a(i) >> b(i), then clustering is not done properly. If a(i) << b(i), then clustering is done appropriately.

A high value of s(i) indicates that the points are correctly matched to the right clusters and poorly matched to the neighboring clusters.

XXII. Natural Language Processing(NLP)

Machine Learning Libraries: SpaCy, NLTK
Deep learning Libraries: PyTorch, Keras, TensorFlow
BERT, Transformers: Hugging Face Libraries