46.1 Calibration of Models: Need for Calibration

Let us consider a 2-class classification problem (ie., $y_i \in \{0,1\}$). Let $D_{Tr} = \{x_i, y_i\}$ be the training data. Let us now train a model on this training data, and let the model function be denoted as f(x).

For any given query point 'x_q', the output is given by $y_q = f(x_q)$. So in the task of classification, the output 'y_q' sometimes may or may not be the probabilistic estimate (ie., $P(y_q=1|x_q)$).

For example, in Naive Bayes,

$$\mathbf{P(y_q=1|x_q) \propto P(y_q=1)*\pi_{i=1}^{d} P(x_{qi}|y=1)} \text{ --- (1)}$$

$$\mathbf{P(y_q=0|x_q) \propto P(y_q=1)*\pi_{i=1}^{d} P(x_{qi}|y=1)} \text{ --- (2)}$$

We can say,

a) $y_q = 1$, if $P(y_q=1|x_q) > P(y_q=0|x_q)$
b) $y_q = 0$, if $P(y_q=1|x_q) < P(y_q=0|x_q)$

So when we look at (1) and (2), we can say the terms on the left side of the '$\propto$' symbol are proportional to the terms on the right side, but not exactly equal. So when an input 'x_q' is given, and if the output is 'y_q', then this 'y_q' may or may not be the actual $\mathbf{P(y_q=1|x_q)}$ (or) $\mathbf{P(y_q=0|x_q)}$.

Need for Calibration

Let us assume we are working on a binary classification task, and are using Log-Loss as the metric to assess the model performance. The log-loss formula for a binary classification task is given as

$\mathbf{Log\text{-}Loss = (-1/n) * \Sigma_{i=1}^{n} [y_i*log(p_i) + (1-y_i)*log(1-p_i)]}$, where

n → total number of points

$p_i = P(y_i=1|x_i)$

We need the actual probability(p_i) value, as the log-loss is determined using the actual probability(p_i) value. If the actual probability(p_i) value is wrong, then the whole log-loss value goes wrong.

The process looks like as shown below

$\mathbf{x_q \rightarrow f(x_q) = y_q \rightarrow y_q \rightarrow}$ **calibration model** $\mathbf{\rightarrow P(y_q=1|x_q)}$

All the models do not provide probabilistic estimates. Some models give poor estimates of the class probabilities, and some of them do not support the probability predictions (SVM is the best example, which gives

only the class labels, but not the probabilistic estimates).

The calibration module allows us to better calibrate the probabilities of a given model (or) to add support for probability prediction.

Calibration is just building one more model on top of the existing model to correct the class probabilities. In calibration, we are simply modifying the outputs of the previous model slightly, to convert them into the class probabilities.

The probability outputs by the models like logistic Regression, Decision Trees, Naive Bayes, etc are often not well-calibrated which can be observed by plotting the calibration plot. Hence we use calibration as a post-processing step to ensure that the final class probabilities are well-calibrated. If the calibration plot looks like a 45-degree straight line, then it means there is no need to apply calibration to the model. In such a case, we can skip the calibration step.

Calibration plots are used only in binary classification tasks. For a multi-class classification task, we first have to convert the task into multiple binary classification tasks(using the one-vs-rest approach), and then apply calibration on top of each binary classification model.

Calibration is a mandatory step, if we want probabilistic estimates as the output, as calibration corrects these probabilities. Also if we are using log-loss as the performance metric, which needs the values of $\mathbf{P(y_i|x_i)}$, the calibration is a mandatory step to be followed.

46.2 Calibration Plots

Let us assume we have a model 'f' that was trained on the training dataset 'D_{Train}'. Let us consider our cross-validation dataset is given as $D_{cv} = \{x_i, y_i\}$.

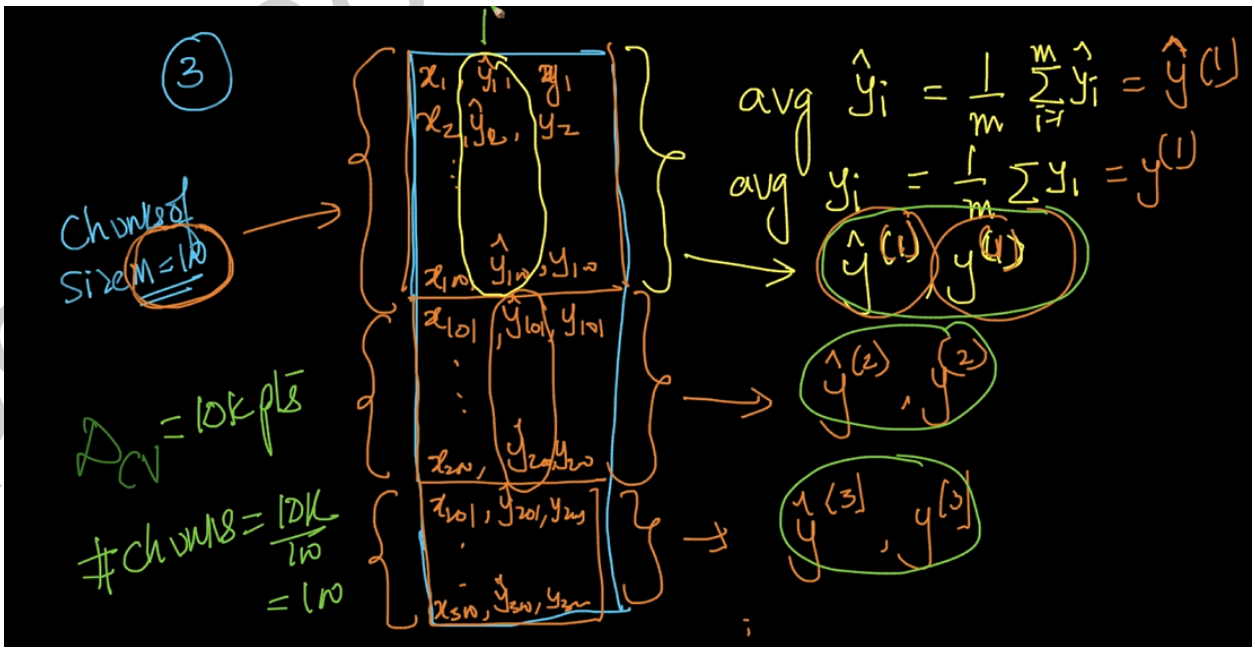
For each data point 'x_i' in 'D_{cv}', we have to make the class label prediction using the model 'f', and let the predicted class label be denoted as 'y_i^'. The actual class labels are denoted as 'y_i', and $y_i \in \{0, 1\}$.

Steps of Construction

1) For each point (x_i, y_i) in 'D_{cv}', we have to predict 'y_i^' = $f(x_i)$, and tabulate the values as

x_i	y_i	y_i^

2) Sort this table in the increasing order of 'y_i^'.
3) After sorting, break the table into chunks of size 'm' each. For each chunk, we have to calculate the average 'y_i^' and average 'y_i'.



Average y_i^ = (1/m) * $\Sigma_{i=1}^{m}$ y_i^ -- (1)

Average y_i = (1/m) * $\Sigma_{i=1}^{m}$ y_i -- (2)

The averages for these chunks are denoted as ($y^{\wedge(1)}$, $y^{(1)}$), ($y^{\wedge(2)}$, $y^{(2)}$), ($y^{\wedge(3)}$, $y^{(3)}$), ….

Let us assume there are 'n' data points in 'D_{cv}' and the size of each chunk is 'm', then

Number of chunks = n/m.

Now the data used for calibration is denoted as 'D_{calib}'.

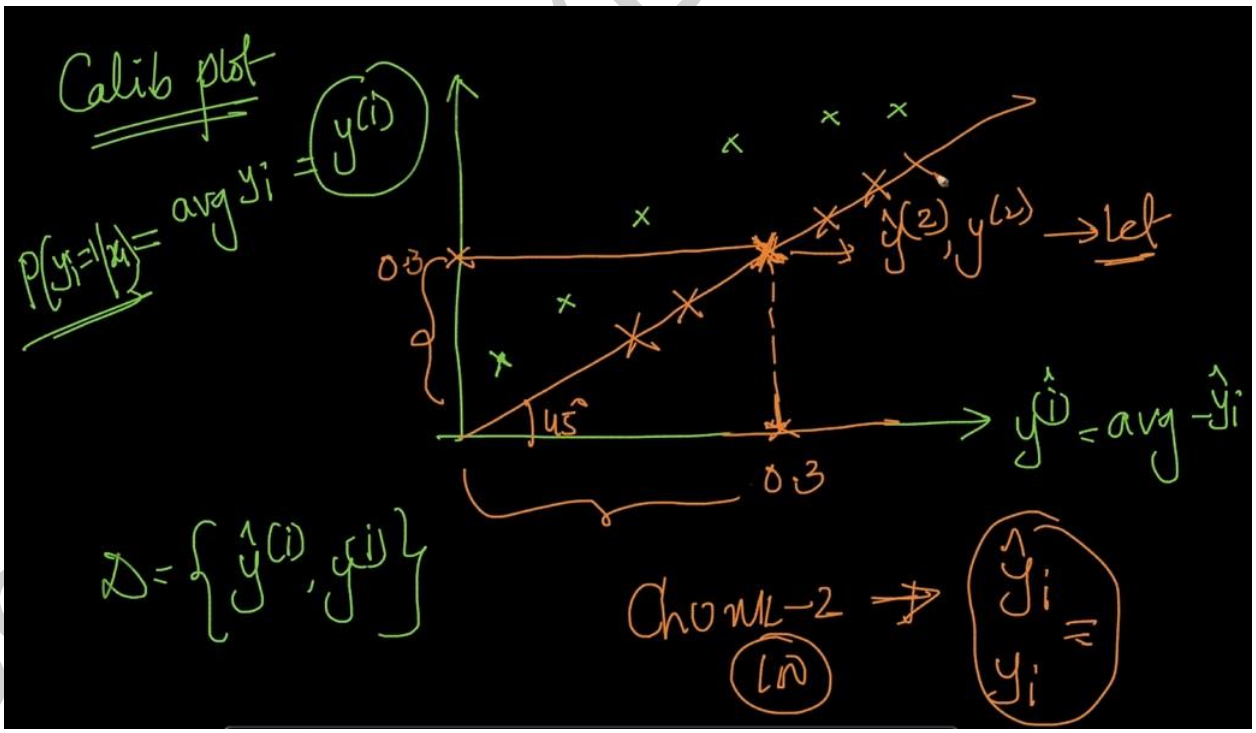
D_{calib} = {$y^{\wedge(i)}$, $y^{(i)}$}

Here $y^{\wedge(i)}$, $y^{(i)}$ are the averages obtained in (1) and (2) for each chunk.

In the calibration dataset 'D_{calib}',

y(i) represents **$P(y_i=1|x_i)$**, and **y^(i)** represents **average of $f(x_i)$'s**

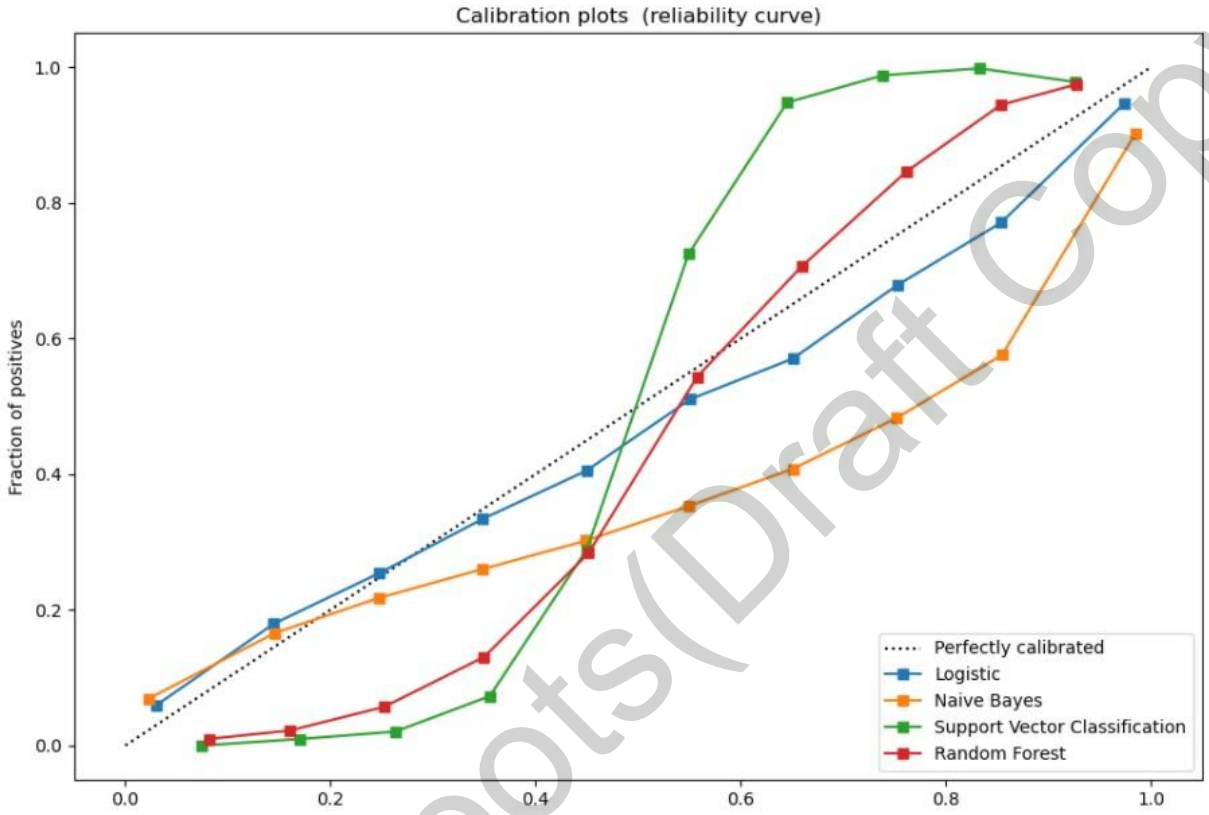
4) Now we have to build a 2D scatter plot using the calibration dataset 'D_{calib}', with $y^{\wedge(i)}$ values on the 'X' axis, and $y^{(i)}$ values on the 'Y' axis. This plot is called the **calibration plot**.



If the plot looks like a 45-degree straight line, then it means **avg($y^{(i)}$) = avg($y^{\wedge(i)}$)** for all the chunks. An ideal model that doesn't require calibration will have all the points on a 45-degree straight line. But in reality, we do not get 45-degree straight lines. We mostly get

the plot in the form of curves, and all these curves are monotonic in nature.

We can see the comparison of the calibration curves of various ML models below



Note

The calibration plots can not be used to decide which model is better. Calibration plots only tell us how well $\mathbf{y_i}$**'s** are calibrated for different ML models. In order to decide which model is better, we have to take any one of the performance metrics into consideration, depending on the problem we are solving.

46.3 Platt's Calibration/Scaling

One of the simplest techniques to perform Calibration is **Platt's Scaling** or **Sigmoidal Scaling**.

We have a calibration dataset **'D_{calib}' = {$y\hat{}^{(i)}$, $y^{(i)}$}**. Now the task is to predict $y^{(i)}$, when $y\hat{}^{(i)}$ is given.

$y\hat{}^{(i)} = f(x_i)$

$y^{(i)} = P(y_i=1|x_i)$

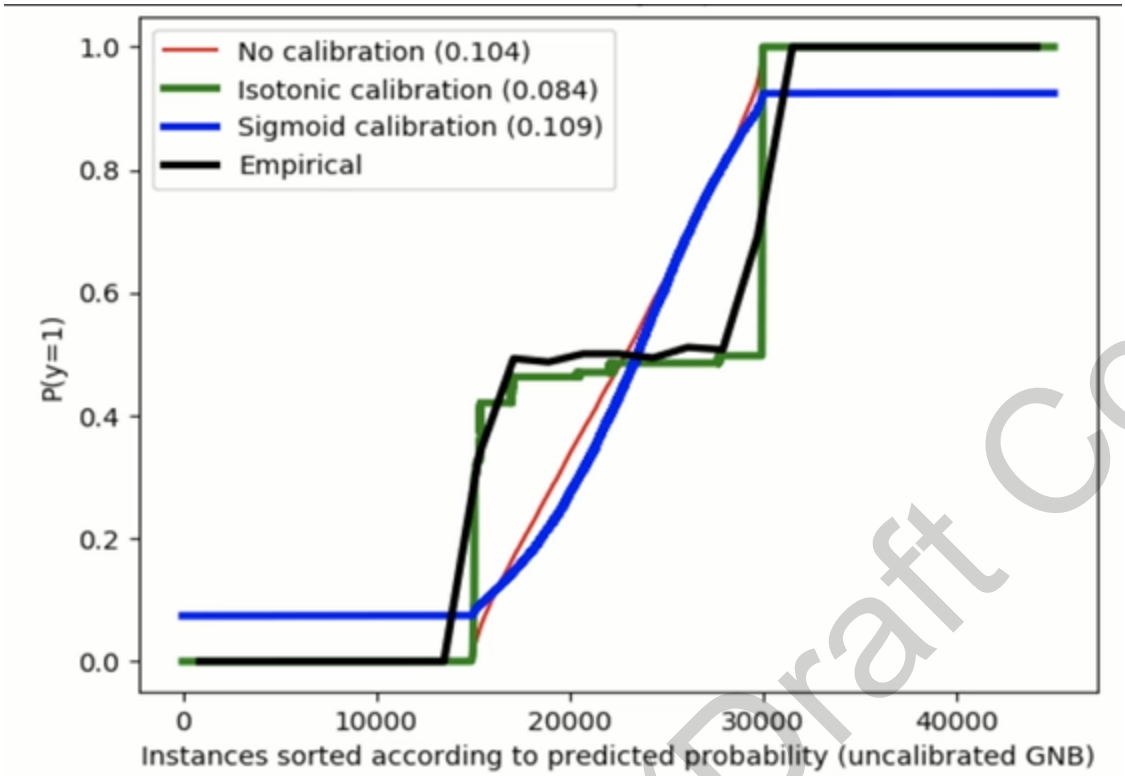
When we look at the comparison of the calibration plots for different ML models in the previous video lecture, we can observe that the calibration curves look similar to sigmoid functions.

If **y = sigmoid(x)**, then we can say that 'y' is defined as sigmoid of 'x'. Similarly, when we look at the comparison of the calibration plots, for different ML models, we can observe that the curves look similar to sigmoid curves, and $y^{(i)}$ can be obtained by applying a sigmoid function over $y\hat{}^{(i)}$. Or else we can say that $y^{(i)}$ can be defined as a sigmoid function in terms of $y\hat{}^{(i)}$. So Platt's scaling says

$P(y_q=1|x_q) = 1/(1+\exp((A*f(x_q))+B))$

In the above formula, given 'D_{calib}', we shall replace $P(y_q=1|x_q)$ with $y^{(i)}$ and $f(x_q)$ with y^., and find the best values of 'A' and 'B' by solving an optimization problem.

We have a disadvantage with Platt's scaling. Let us look at the plot below



The black curve represents the empirical data (ie., the points in 'D_{calib}'). The blue curve represents Platt's scaling calibrated data. This empirical data curve (ie., calibration curve) is nowhere similar to a sigmoid function. Platt's scaling works properly when the calibration curve is similar to a sigmoid curve. Platt's scaling fails when the calibration curve doesn't look similar to a sigmoid curve.

In case, if the calibration curve doesn't look similar to a sigmoid curve, then we have an alternative type of calibration, and that is called **Isotonic Regression**.

In the above plot, the green curve data represents isotonic regression, which is looking almost similar to the empirical data.

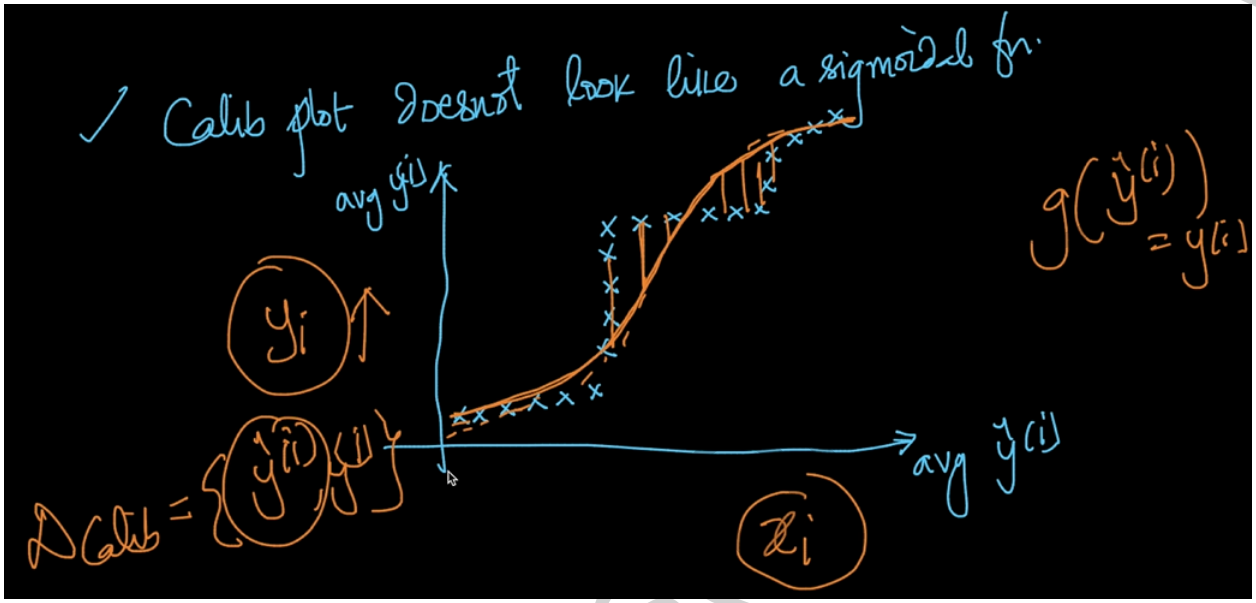
Note

If the calibration plot looks similar to a sigmoid curve or S-shape, then Platt's/Sigmoidal Scaling calibrates the data much better. Whereas if the calibration plot doesn't look similar to a sigmoid curve(if it is in different shapes), then Isotonic Regression calibrates the data better.

Platt's scaling is simpler, whereas Isotonic Regression is complex and requires a lot more data(otherwise it may overfit), but can support reliability diagrams with different shapes. Isotonic Regression is non-parametric, whereas Platt's scaling is a parametric approach.

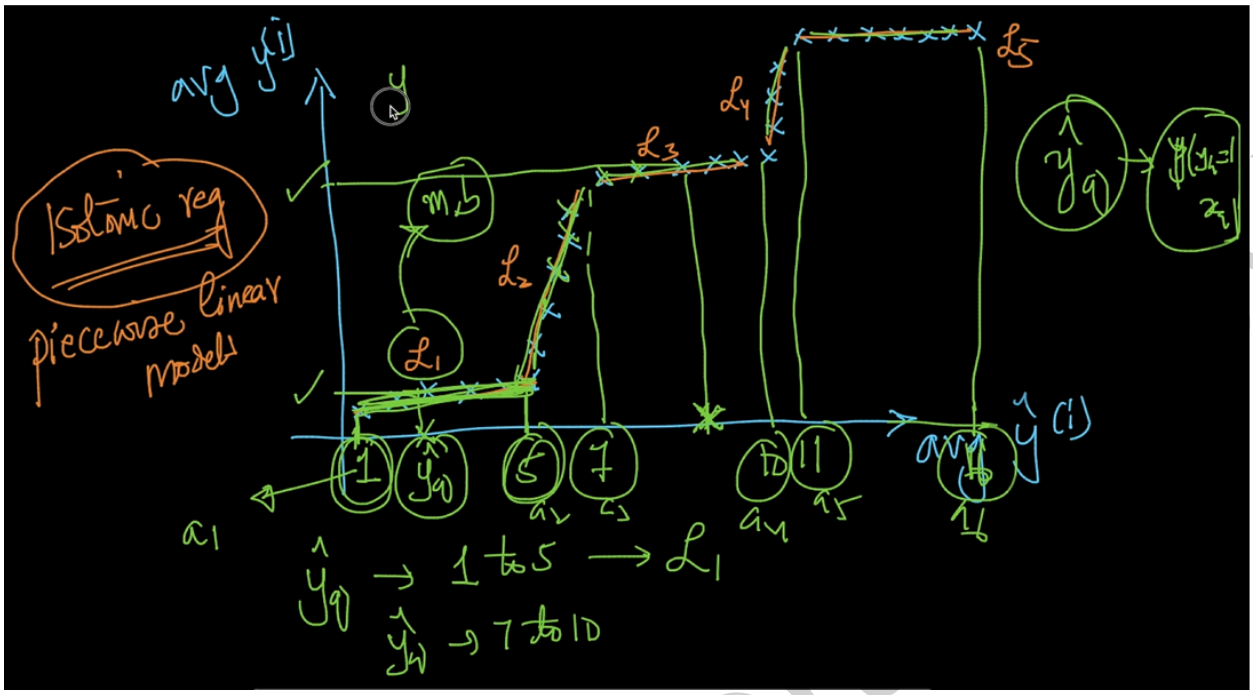
46.4 Isotonic Regression

Isotonic Regression is a very popular technique that works even if the calibration plot is not of sigmoidal shape. Let us assume the calibration plot is given to us as shown in the below figure in cross symbols.



For this kind of calibration plot, a simple sigmoidal function doesn't work as it is impossible to fit this data to a simple sigmoidal function. If we try to fit a sigmoidal function on this data, the errors will be very high.

The core concept of Isotonic regression is to break up this whole curve into multiple linear models (L_1, L_2, L_3, …..). These individual models are called piecewise linear models.



If y_q^ → 1 to 5 → Use L_1
If y_q^ → 5 to 7 → Use L_2
If y_q^ → 7 to 10 → Use L_3
If y_q^ → 10 to 11 → Use L_4
If y_q^ → 11 to 16 → Use L_5

The core idea here again is to solve an optimization problem to minimize the gaps between the predicted values and the observed values.

The whole optimization problem is about finding the values of thresholds, slopes, and intercepts of the linear lines used as the linear models. The points of calibration lie on these lines as closely as possible. This optimization works by enforcing a constraint that there should be no multiple lines. This optimization problem looks like an extended linear regression problem.

But with the Isotonic Regression, we need many more points than with Platt's Scaling. It is because, in Platt's scaling, we have only two parameters 'A' and 'B' whereas in Isotonic Regression, we have many parameters like the slopes and intercepts of each line. Hence we need more data points to estimate these parameters.

If we have a large dataset, when we split it, we can have a large amount of data points in 'D_{cv}' and thereby in 'D_{calib}', and then we can go for Isotonic regression.

If we have a smaller number of points in 'D_{calib}', it is best to go with Platt's scaling.

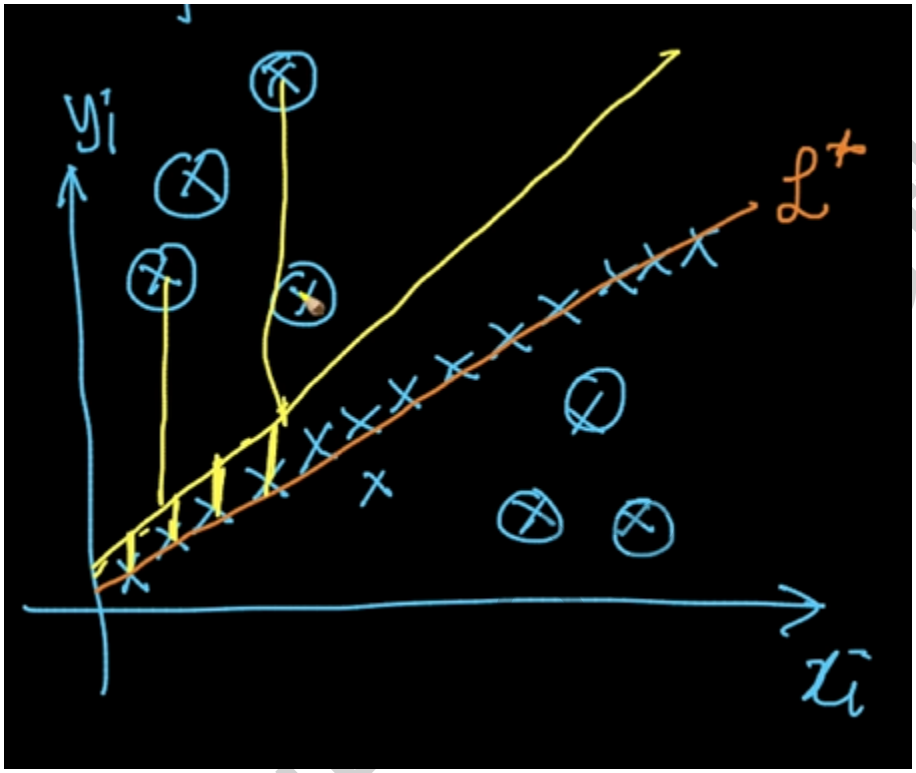
Note

As the video lecture 46.5 is all about the discussion on the code samples, we aren't providing any notes for it.

46.6 Modeling in the presence of Outliers: RANSAC

RANSAC explains how we can build a robust model in the presence of outliers. A robust model is a model that is not impacted by the outliers much.

Let us take linear regression as an example. The goal in linear regression is to find a line/plane that fits most of the points.



Let the 'D_{Train}' be the training dataset for this problem. Here $D_{Train} = \{x_i, y_i\}$ where $x_i \in R^d$, $y_i \in R^d$

So if we train a model using 'D_{Train}', then we will get a line as 'L_1'. Linear Regression tries to minimize the distance of the points from the regression line 'L_1'. (ie., $\mathbf{min\ \Sigma_{i=1}^{n} (y_i - y_i\hat{})^2}$).

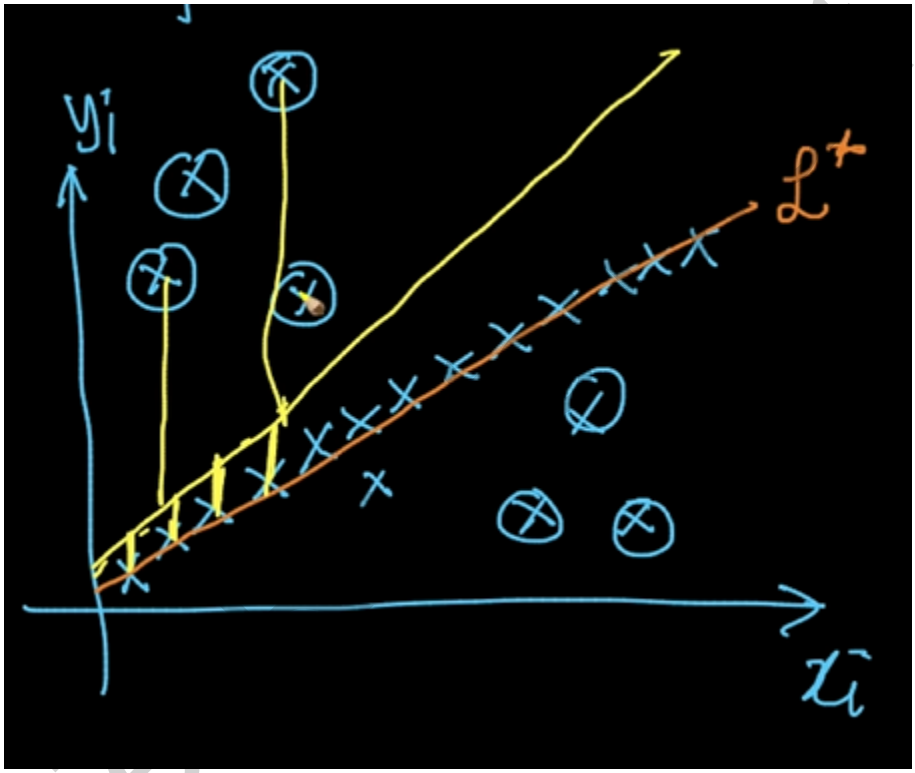
The more the extreme outliers are, the more the model pulls the regression line towards them. The regression line is pulled in that direction in which we have more extreme outliers. As this model is getting impacted by the outliers, we call it a non-robust model.

So in order to avoid the impact of outliers on the model, we have to go for the RANSAC technique.

RANSAC Procedure

1) Let us pick a sample dataset 'D_0' from 'D_{Train}' through random sampling. Let us build a model (linear regression in this case) using the 'D_0' dataset and let it be 'L_0'.

   When we sample the points randomly, the probability of an outlier being a part of 'D_0', as the probability of an outlier being a part of 'D_0' reduces, the regression line is less impacted by the outliers. In this plot, the regression line has been moved to 'L_0' and is slightly impacted as we are building this model on a sample of data points from 'D_{Train}'.



2) Compute outliers based on 'L_0' by computing (y_i-y_i^). Those points with higher values of (y_i-y_i^) are considered outliers. Let these outliers be denoted as O_0 = {outliers}.

   After computing the outliers, let's compute the next set of data $\mathbf{D_{Tr}^{|} = D_{Tr}-O_0}$.

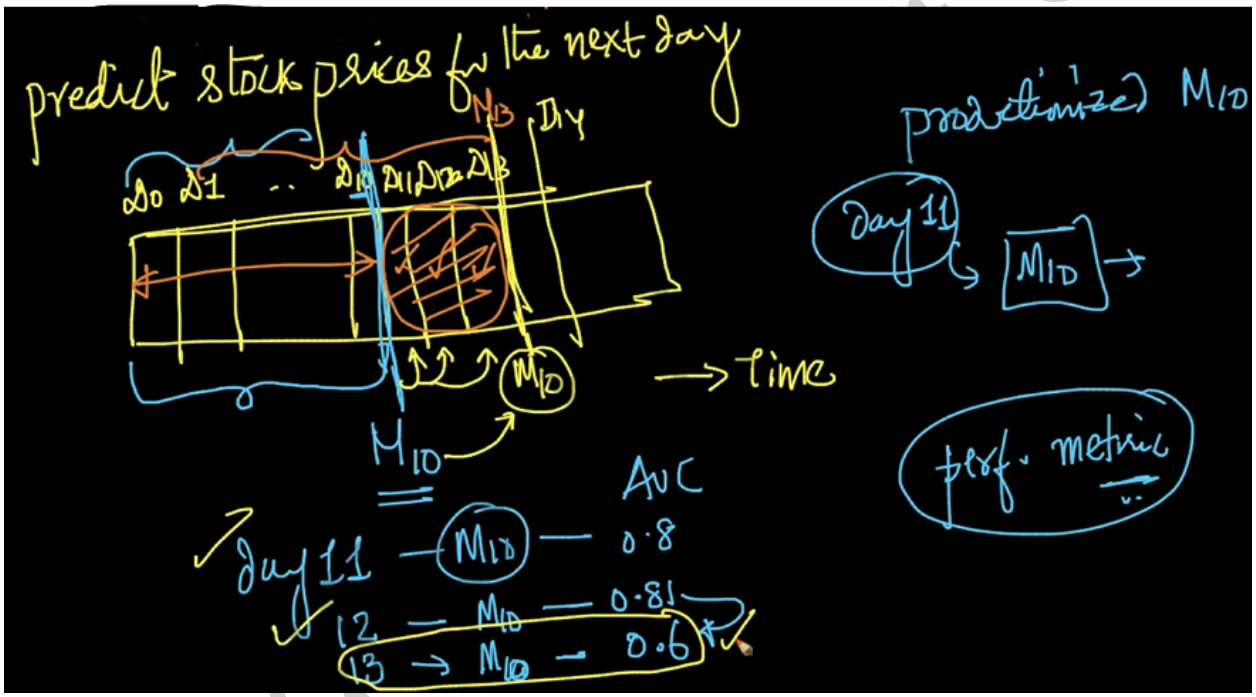
3) Now prepare a dataset 'D_1' by randomly sampling ($D_{Tr}^{|}$). Fit the model on 'D_1' and this time let the regression line be 'L_1', and the outliers are 'O_1'.

Let us find $D_{Tr}^2 = D_{Tr}^1 - O_1$. We have to repeat the process for L2, L3, L4, ….

In this way, at some point say 'L_i' and 'L_{i+1}' will look the same and there will be no outliers. Then this model is called the robust model, and this model is built using 'D_{Tr}^{i+1}', and this model is free of all the outliers. This final model is denoted as 'L*'.

46.7 Retraining Models Periodically

Let us assume we are working on the problem of predicting the stock prices for the next day. Every day we keep getting the fresh data, and the model is trained at the end of the day on the data so far we have (including the latest obtained data), and that trained data will be used in the production on the next day. For example, by the end of day 10, we'll have the model 'M_{10}' trained on the data(let us denote it as D_{10}) we had till that day(including the latest data we got on the 10^{th} day), and this model 'M_{10}' will be in production on the 11^{th} day.



On the 11th day, we get the new data and the predictions are made on the data we had till then including the new data obtained on the 11^{th} day, using the model 'M_{10}'. Let us denote this data as 'D_{11}'. We get some value for the performance score. Again on the 12^{th} day, we make predictions using the same model 'M_{10}' on the 'D_{12}', and we record the performance score. But on the 13th day, if we make predictions on 'D_{13}', we observe a drop in the performance score. This drop in the performance score is because the nature/patterns in the newly obtained data (ie., the data obtained on the 13^{th} day) is different from that of the existing data so far. In such a case, we see huge errors in the predictions made on 'D_{13}' by the

model 'M_{10}'. In such a case, in order to minimize such huge errors, we retrain the model on the data 'D_{13}'.

Q) How to determine when to retrain the model?
Ans)

a) If the cost of retraining is not so high, then we have to retrain the model on regular basis. If the cost of retraining is high, then day by day (or) periodically, if we notice the performance of the model to be dropping, then we have to retrain the model.
b) If the dataset itself is changing regularly (such data is called **non-stationary data**), then we have to check if both the train and the test data follow the same distribution. If both the distributions are different, then we have to immediately retrain the model.

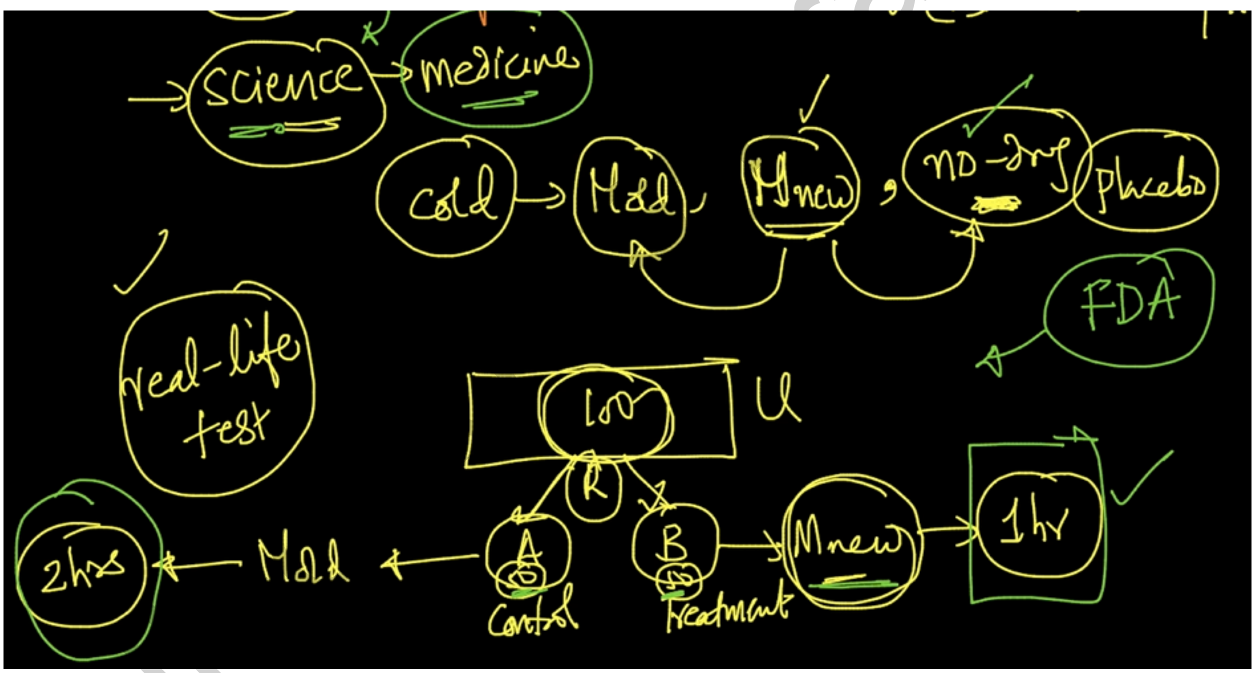
46.8 A/B Testing

A/B testing is an important phase in Machine Learning. It is also referred to as **Bucket Testing** (or) **Split run** (or) **controlled experiments**.

Example 1

Let us assume we have a medicine for a disease and let this medicine be denoted as 'M_{old}'. Let us assume a new medicine 'M_{new}' has been introduced.

Let us assume we have a bunch of users, and these users are divided into two categories ('A' and 'B'). Let the users in category 'A' be given the old medicine 'M_{old}', and the users in category 'B' be given the new medicine 'M_{new}'.



In case, if the medicine 'M_{old}' takes 2 hours to cure the disease, and 'M_{new}' takes 1 hour to cure, then the company that has manufactured 'M_{new}' can claim the medicine 'M_{new}' is working better, and it gets approved. This is the test process followed for approving a medicine.

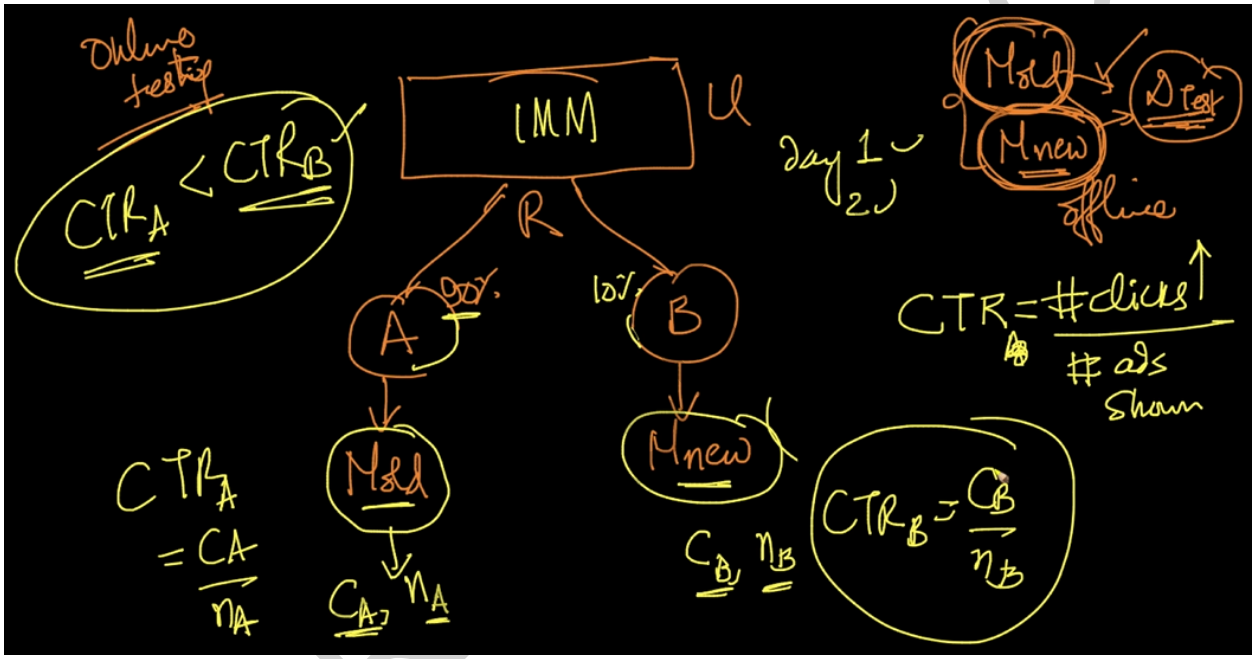
Let us now check how A/B testing is used in machine learning.

Example 2

We have to build a model for Google to predict which ad to show for a search query.

Procedure for A/B testing

There are a bunch of users and are randomly split into two groups 'A' and 'B'. Let's say 90% of the users are in 'A' and the remaining 10% of them are in 'B'. Let us also assume that we already have a model existing for this task, and let us denote it as 'M_{old}', and we have a newly trained model and let us denote it as 'M_{new}'.



Let

c_A = Number of clicks from group 'A'

c_B = Number of clicks from group 'B'

n_A = Number of ads from group 'A'

n_B = Number of ads from group 'B'

The metric to compare these two models is the **click-through rate (CTR)**.

CTR_A = (Number of clicks we got from 'A')/(Number of ads shown in 'A') = c_A/n_A

CTR_B = (Number of clicks we got from 'B')/(Number of ads shown in 'B') = c_B/n_B

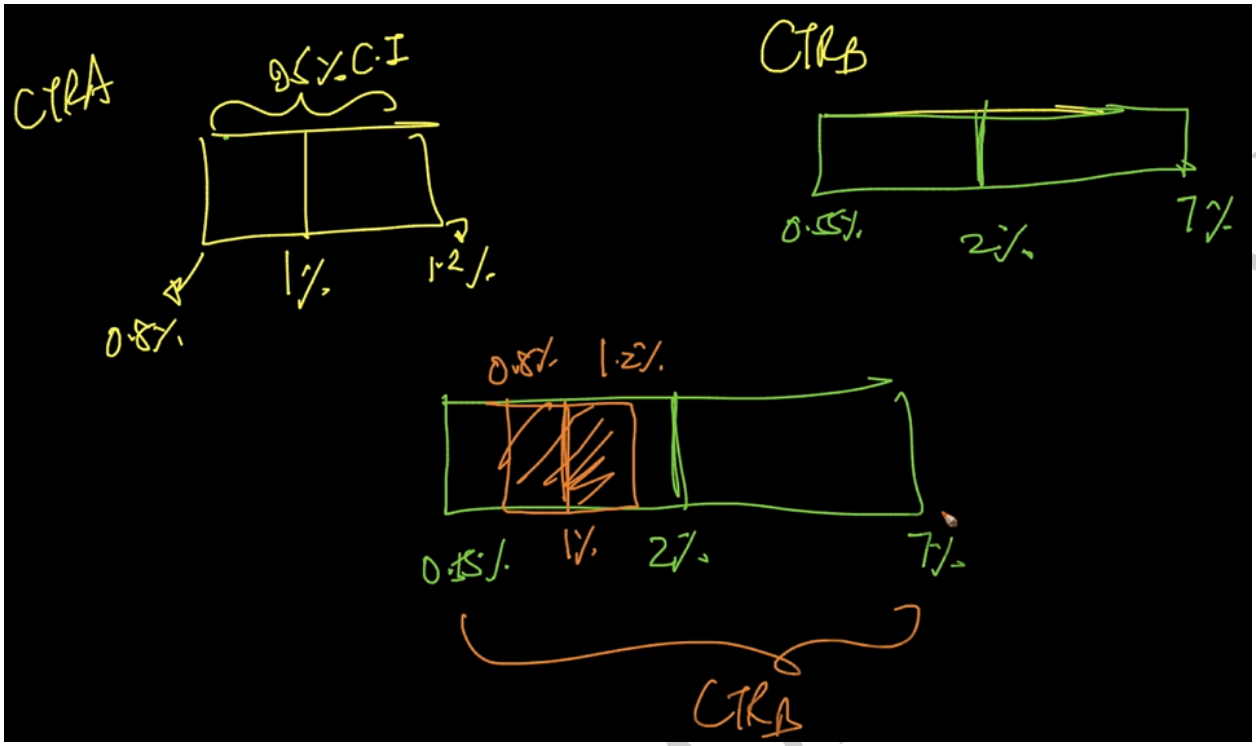
After running the models for a few days, if we find $\mathbf{CTR_A<CTR_B}$ in most of the cases, then 'M_{new}' that is deployed on group 'B' seems to perform better.

Initially, we have to split the users in a 90-10 ratio of old and new. If the performance of 'M_{new}' is better at predicting which ads to show, when compared to that of 'M_{old}', then we have to keep changing the ratios as 80-20, 70-30, 60-40, 50-50,..., 0-100.

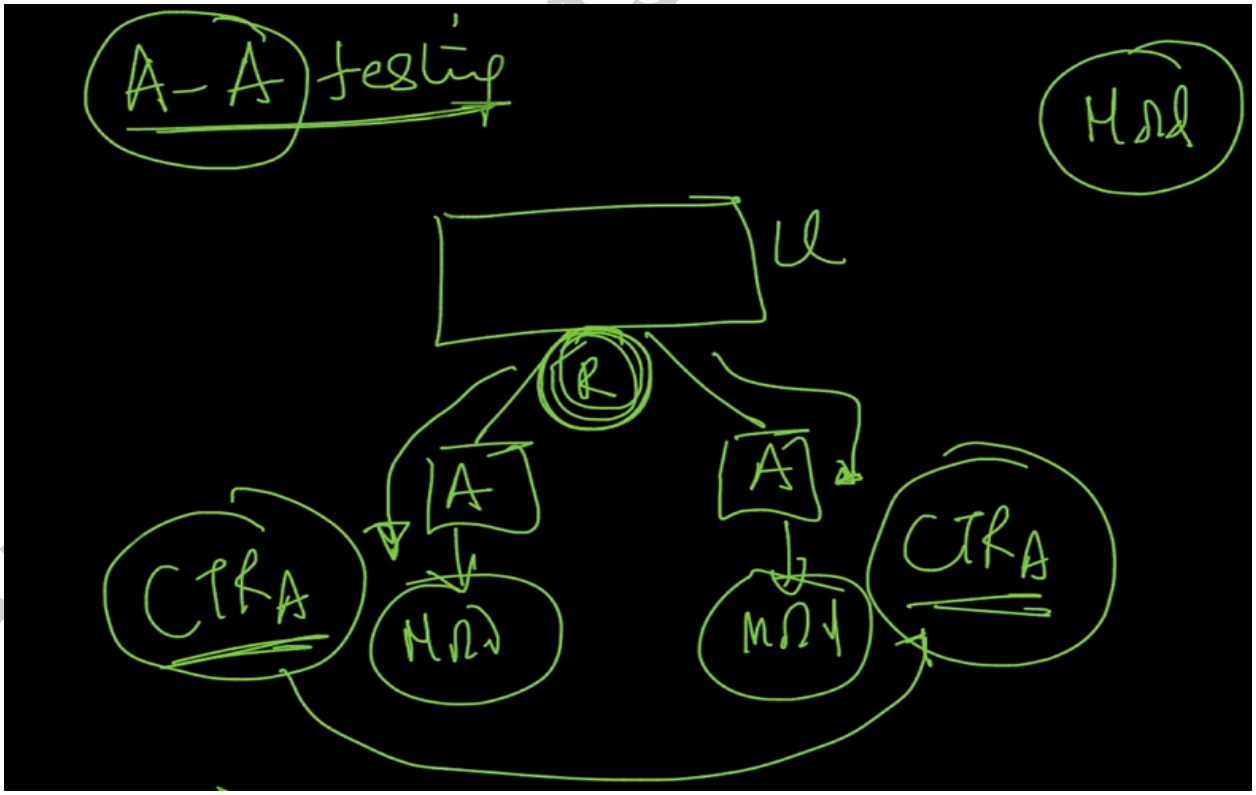
Let us assume we have got the values as given below
$c_A = 100$, $c_B = 2$, $n_A = 10000$, $n_B = 100$, now
$CTR_A = c_A/n_A = 100/10000 = 0.01$ (1%)
$CTR_B = c_B/n_B = 2/100 = 0.02$ (2%)
Here just by looking at the values of CTR_A and CTR_B, we generally make a conclusion that 'M_{new}' is performing better than 'M_{old}'. But we are not supposed to make a conclusion like that because in '$\mathbf{CTR_A}$', the value of '$\mathbf{n_A}$' is very much larger when compared to '$\mathbf{n_B}$' in '$\mathbf{CTR_B}$'. Those two values are not even nearby. Hence in such a case, we have to compute the confidence interval of '$\mathbf{CTR_A}$'.

For the above example (ie., with the values $c_A = 100$, $c_B = 2$, $n_A = 10000$, $n_B = 100$), let us say the
95% confidence interval for CTR_A = [0.8%, 1.2%]
95% confidence interval for CTR_B = [0.55%, 7%]

We have to check if both confidence intervals are overlapping. If they both overlap, then we couldn't say which model is performing better, and if they both do not overlap, then whichever confidence interval is wider, the model associated with it is considered to be the better performer.


CTRA
95% C.I
1%
1.2%
0.8%
CTRB
0.15%
2%
7%
0.8%
1.2%
0.15%
1%
2%
7%
CTRB

A-A Testing

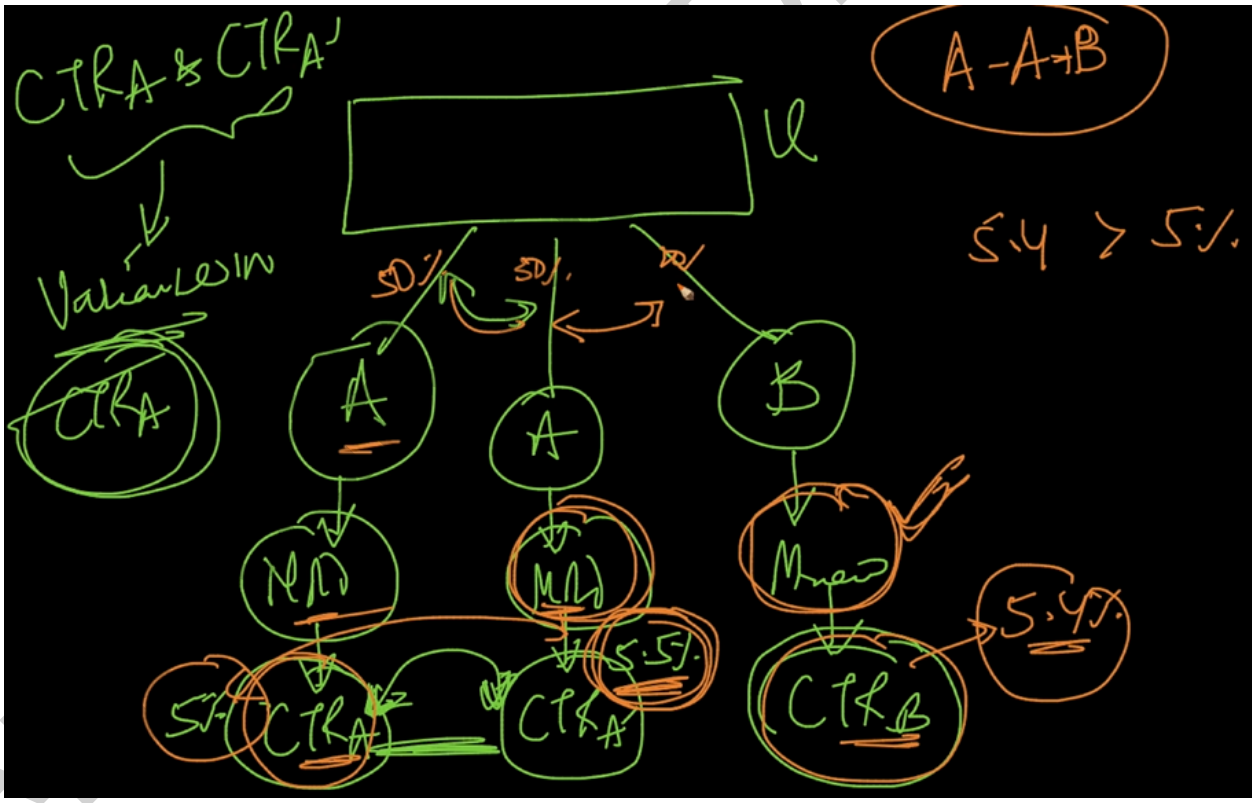

A-A testing
UI
R
A
A
CTRA
CTRA
CTRA

Let us assume we have a bunch of users and we are splitting them into 2 groups ‘A’ and ‘A’. We have to run the old model ‘M_{old}’ on both the groups and let the obtained CTR scores be denoted as CTR_A and $CTR_A{}^{|}$ respectively. Here CTR_A and $CTR_A{}^{|}$ are not the same because the data/users in one group are different from the data/users in the other group.

Typically used A-A Testing working procedure

Let the whole universe of users be divided into 3 groups. They are A, A, and B. Let us now train the model ‘M_{old}’ separately on the users present in ‘A’ and the other ‘A’, and train the model ‘M_{new}’ on the users present in ‘B’.

Let the CTRs obtained be denoted as CTR_A, $CTR_A{}^{|}$, and CTR_B. Even though the same model was used 3 times, the data present in each of the groups is different. The difference between CTR_A and $CTR_A{}^{|}$ will tell us how differently the same model performs if there is a slight change in the data.



When we look at the CTR scores of these 3 models, we blindly should not conclude that the new model ‘M_{new}’ performs better when compared to the old model ‘M_{old}’, just because of $CTR_A < CTR_B$. This

difference in performance is due to randomness while splitting the users. So as the randomness changes, the CTR also changes. This is called **A-A-B testing**.

46.9 Data Science Life Cycle

1) **Understand Business Requirements**

Define the problem, understand the customer and their business for whom we are building the project.

The prime focus should be on working on what adds much value to the business and helps in making decisions to move the business in the best way.

Working on something that doesn't add much value to the business is of no use.

2) **Data Acquisition**

The process of data acquisition is mostly ETL. The processing is done mostly using SQL.

The major sources/storages of data used are the databases, data warehouses, log files, and Hadoop/spark systems.

3) **Data Preparation**

It involves data cleaning and data preprocessing. The first 3 phases are the time-consuming phases.

4) **Exploratory Data Analysis**

After the data preparation, we start plotting the data, visualizing the data using various visualization techniques. We also apply various statistical techniques such as Hypothesis testing, etc and at the end, we slice and dice the data by splitting the data and trying to understand what's happening, which feature is helping more to achieve our goal.

5) **Modeling, Evaluation, and Interpretation**

Modeling deals with building various models such as regression, classification, clustering, etc that perfectly suits the given business problem.

Evaluation deals with defining KPI(ie., all the performance

metrics for our models are defined in this phase for the best results) and adding huge business value.

6) Communicate Results

The results should be communicated in a clear and simple way to all the stakeholders, business customers, and top-level managers(ie., non-technical people)

The communication of results should be in such a way that should convince the business customers and make them believe, this is the best model for the given business problem.

7) Deployment

After getting approval from the higher-level management for the results that were communicated, we should go ahead with the deployment of the model.

Deployment deals with software engineering. This phase is worked out sometimes by the Machine Learning Engineers, sometimes by the software development engineers, and sometimes by the distributed system engineers.

8) Real-world Testing (A/B Testing)

The real-world testing deals with verifying the results of the data analysis, models whether all the results we have obtained are really meaningful in the production environment.

At the end of A/B testing, we have to measure the true business impact. The result of all the work done in the previous phases of this life cycle will become useful/useless on the basis of the measure of the true business impact.

9) Customer/Business

After achieving a good business impact, we have to go back to the business/customer and convince them with the experimental data and the solution we have. At the end of the day, it is our solution that has to add huge value to the business and satisfy the customer.

10) Operations

This phase deals with retraining the models, handling failures in the model, and defining an entire process of how to retrain the models and handle failures. This is called the **operationalization of models**.

11) Optimization

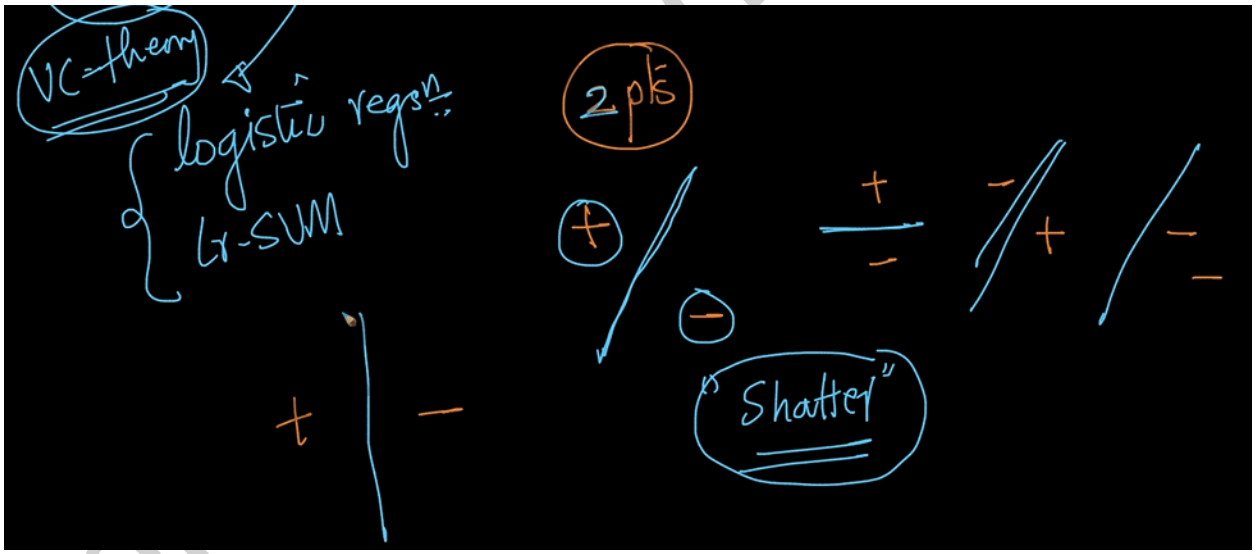
The model that was built so far is the first-cut model. In this phase, we go back and try to improve the models (by moving from one model type to another), acquiring more data, adding more features, and optimizing the production code. This phase is a continuously running phase.

46.10 VC Dimension (Vapnik Chervonenkis Dimension)

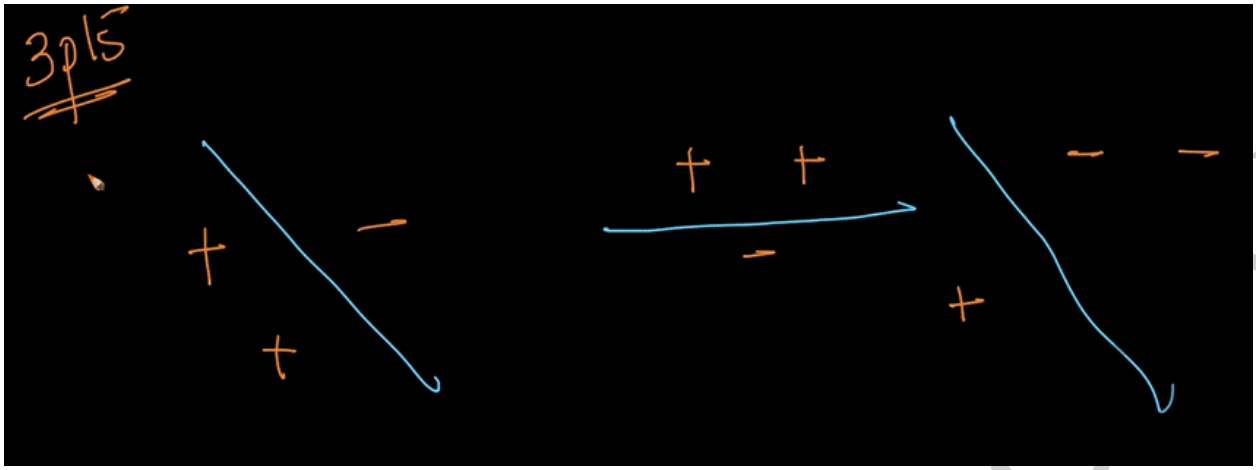
Is there a numeric value to find out how powerful is a class of models? (like how powerful the linear models are, how powerful RBF-SBM models are, how powerful the boosting models are, etc). This is an important aspect of theoretical machine learning.

The practical approach to find out which model is powerful is by taking a bunch of datasets, and building the models on them, and finding out the performances. The theoretical approach is by giving a numerical value to each of the models which indicates how powerful each model is.

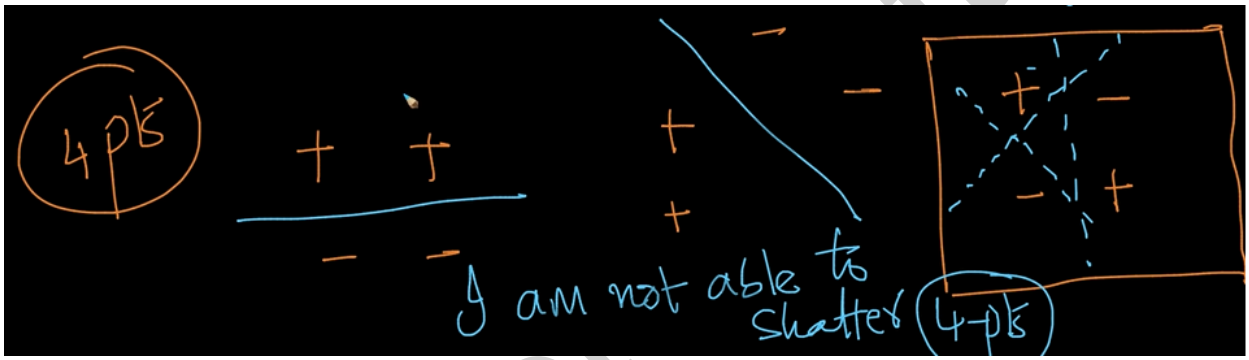
For example, let us consider the linear models (where line/plane/hyper plane are the decision surfaces). If we have only 2 points (1 +ve and 1 -ve), then irrespective of the way the points are shattered, we can create a decision surface easily.



Similarly when we have 3 points, we still can create a decision surface, irrespective of the way the points are shattered.



In case of 4 points, sometimes, it is possible and sometimes it is not possible to create a decision surface that perfectly classifies all the points.



There exists at least one alignment where these points cannot be separated perfectly.
So the VC-Dimension value for a linear model = 3
(ie., maximum of upto 3 points can be shattered by a linear model, for all possible alignments)
VC Dimension is a measure of the capacity of a space of functions that can be learned by a statistical classification algorithm. It is defined as the cardinality of the largest set of points that the algorithm can shatter.

In case of a Boosting classifier, if we have
Number of Base classifiers = T
Class of each of these Base classifiers = B
VC Dimension of each of these base classifiers = D

Then the VC Dimension of the set of all such classifiers = **T*(D+1)*(3*log(T*(D+1))+2)**

In RBF Kernel SVM,
VC Dimension = ∞

If VC-Dimension(M_1)>VC-Dimension(M_2), then theoritically we can say that the model 'M_1' is power than 'M_2'.
By the VC Dimension value, we can say RBF-Kernel SVM is so powerful, but it doesn't imply that RBF Kernel SVM is the best model in practice.

Use of VC Dimension

Th VC dimension can predict a probabilistic upper bound on the test error of a classification model.

P(test error <= train error + sqrt((1/N)*[(D*(log((2*N)/D)+1))-log(η/4)])) = 1-η

(0<=η<=1)
Where

D → VC Dimension of the classification model
N → Number of points in the training dataset

Th above formula is valid only if D<<N. If 'D' is larger, then the test error may be higher than the training error, and this happens due to overfitting.