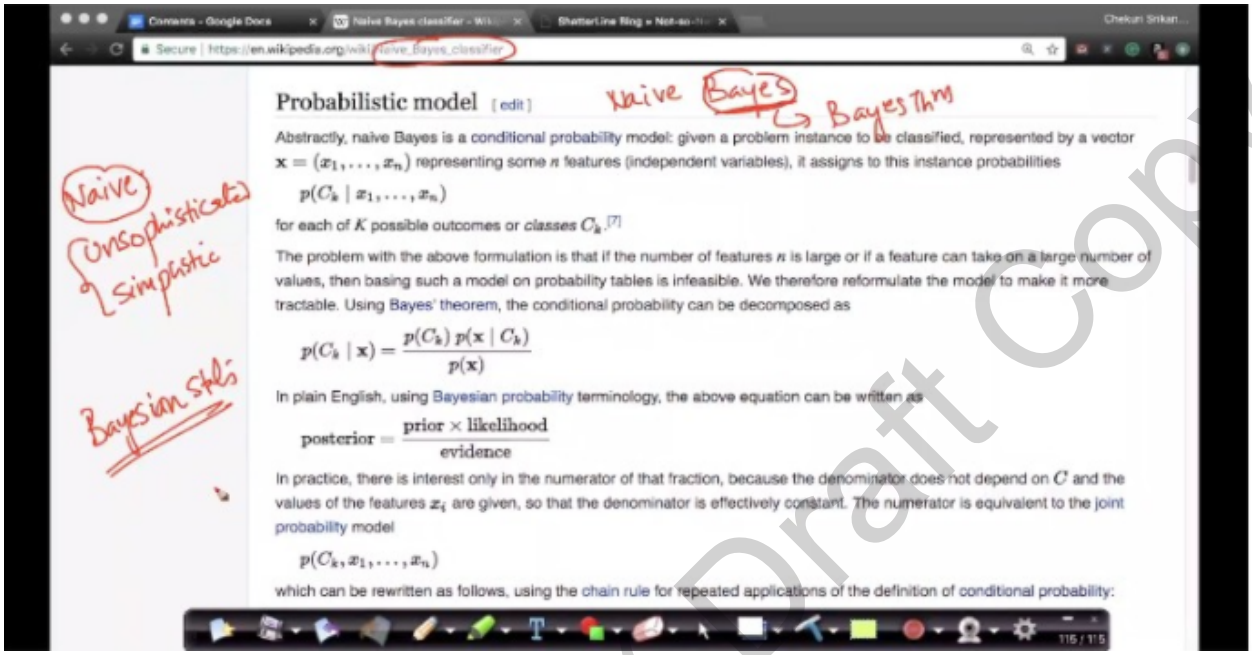
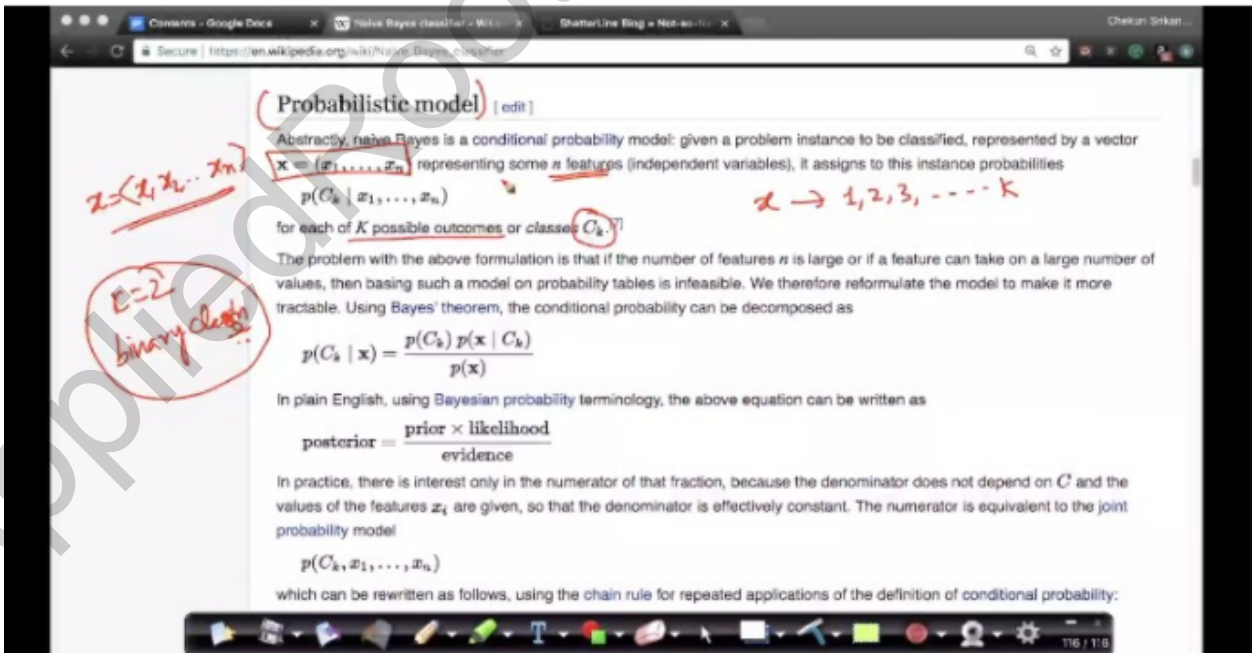
In this chapter we will look at an algorithm called Naive Bayes.

32.1 Naive Bayes Algorithm



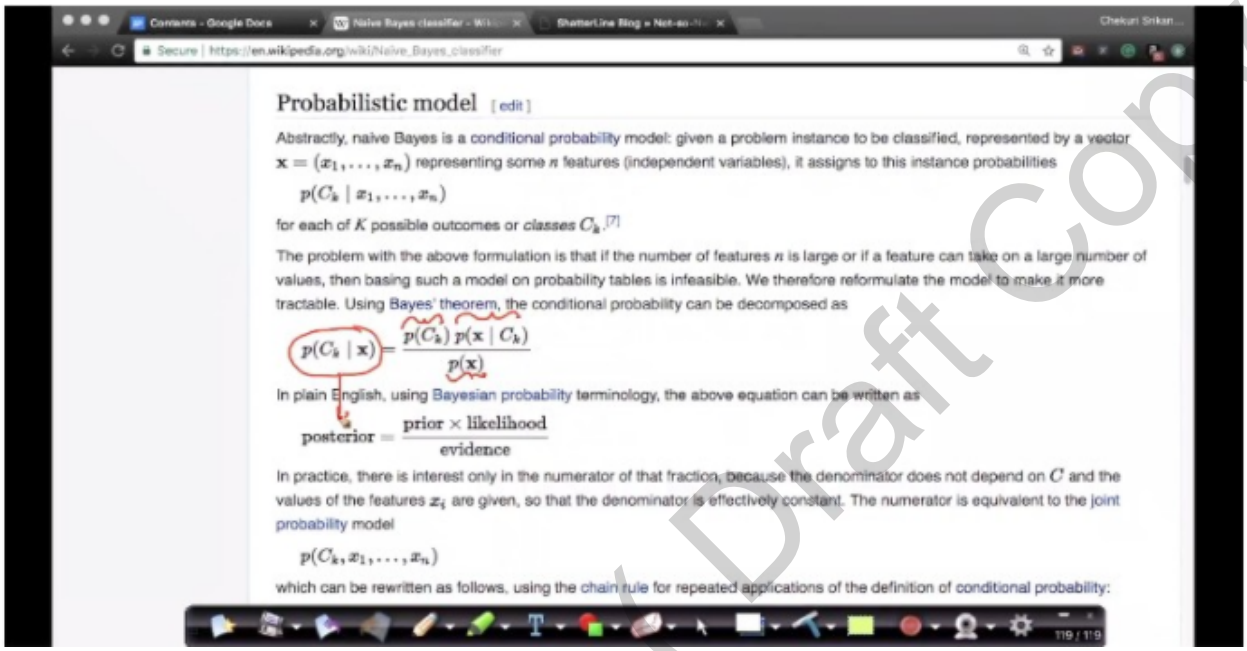
Timestamp 0:57

Naive Bayes algorithm is based on the Bayes Theorem. Bayes theorem plays a very crucial rule in the world of statistics, in fact there is one entire branch called Bayesian Statistics.



Timestamp 2:28

In this chapter we will discuss the theory behind this algorithm.
Say we have a datapoint x, we can represent it in vector form as => <$x_1, x_2, ..., x_n$ >, here x contains n features, each datapoint can belong to one of the K classes. So, we are dealing with a multi class classification problem, if K = 2 we have a binary classification problem.



Timestamp 4:25

Now we want to find out p(C_k| x), it means that given a feature x , what is the probability that the feature x belongs to class(C) k, denoted by subscript. So, we can find this probability for all classes from 1 to k, and say that the point belongs to that class which has the maximum probability.

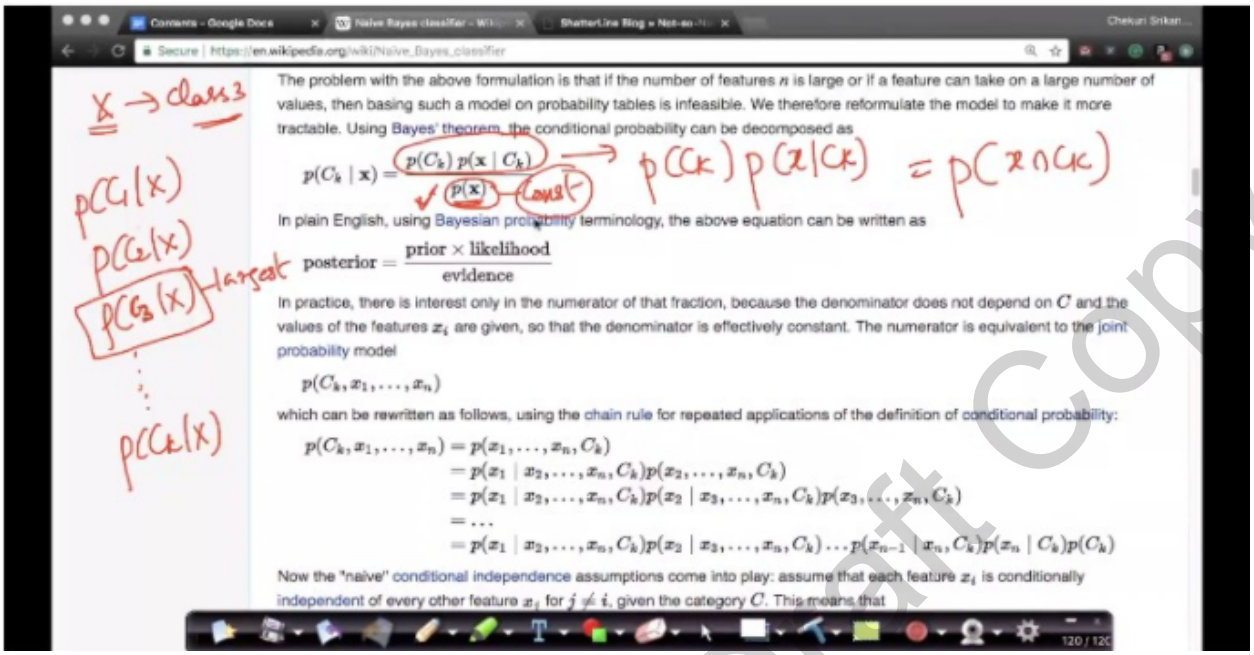
Now, using Bayes theorem the conditional probability can be written as

p(C_k| x) = (p(C_k) × p(x | C_k)) / p(x)

In Bayesian terminology we have,

Posterior = (prior × likelihood) / (evidence)

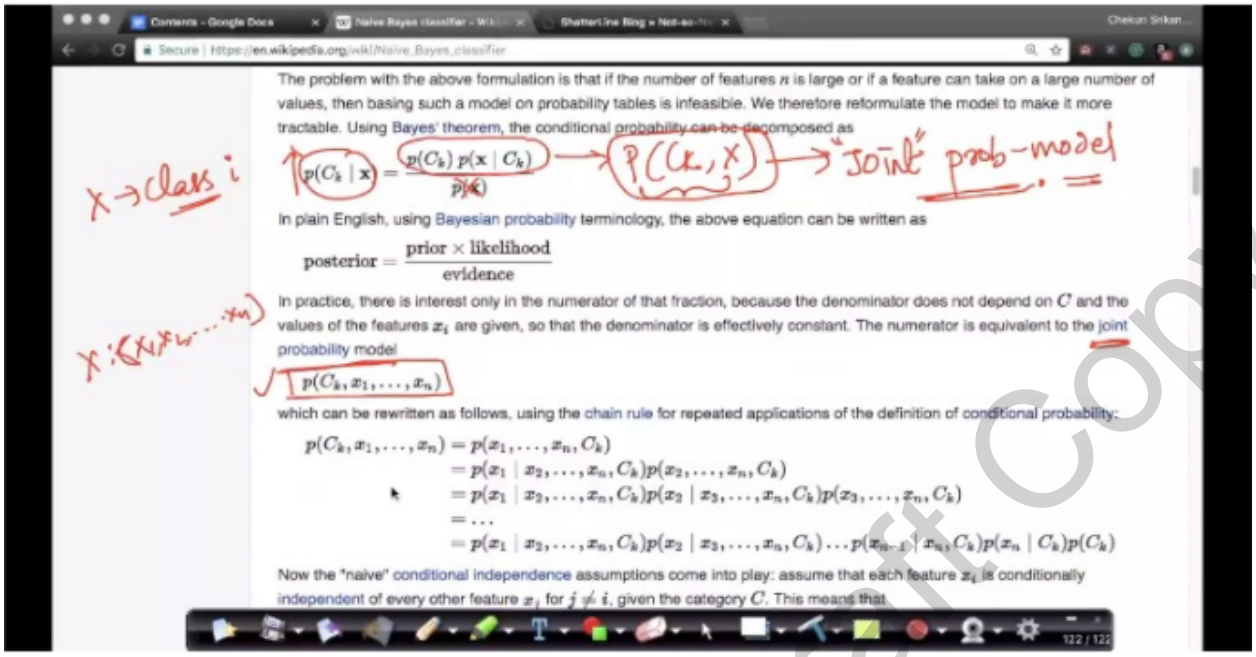
Where,
Posterior = p(C_k| x)
Prior = p(C_k)
Likelihood = p(x | C_k)
Evidence = p(x)



Timestamp 6:18

So our main task is to calculate the probabilities for each class and determine which one is the maximum. Now, if we carefully notice the denominator, it is the same for all the classes as shown in the image, so for determining the maximum we don't need the denominator, we can only focus on the numerator.

The numerator part is => $p(C_k) \times p(x \mid C_k)$, this term calculates the probability of an event where both C_k and x occurs. So, we can simply write this as $p(x \cap C_k)$.

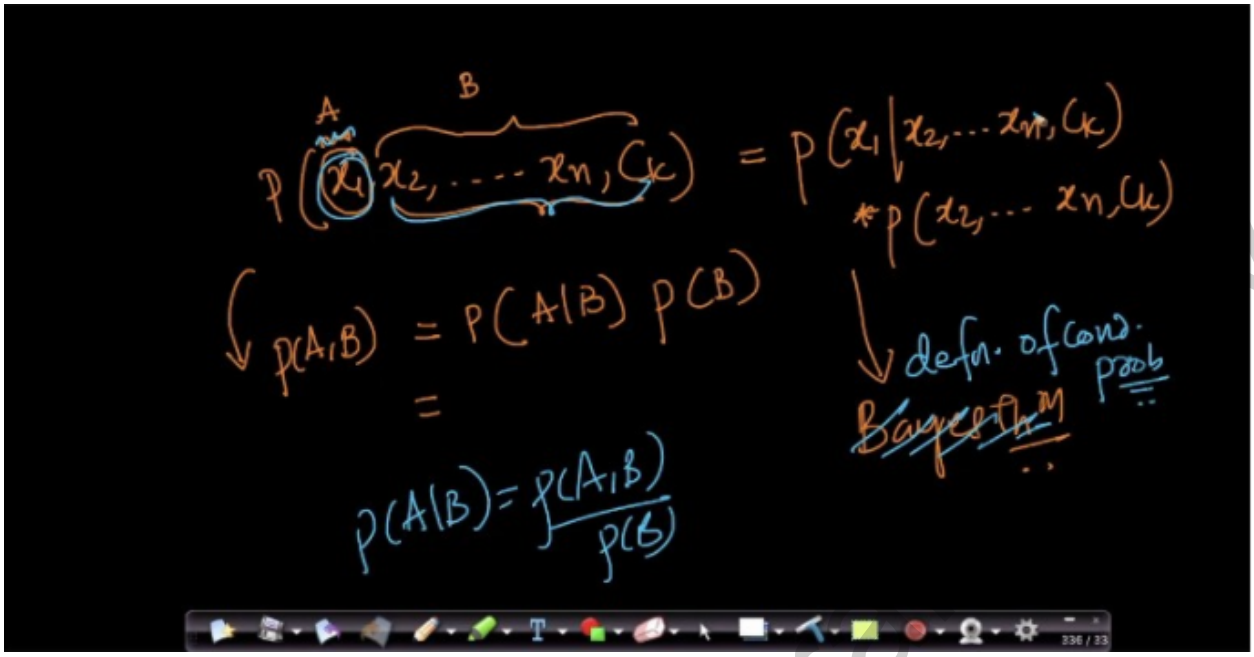


Timestamp 9:29

Now, p(C_k| x) ∝ p (x∩ C_k). The proportionality comes because we have ignored the denominator.

p (x∩ C_k) can also be written as p (C_k, x) . So whichever class possesses the maximum value we can select that class as our prediction. Here we are calculating the probability of two events, this is often called a joint probability model.

p (C_k, x) can be written as p(C_k, x_1, x_2,....., x_n), here we have just written x in its components of n - features/dimensions.



Timestamp 12:21

Our task has boiled down to finding this => p(C_k, x_1, x_2,....., x_n).

Here we will use something known as the chain rule of conditional probability, where we use the conditional probability recursively.

Conditional probability for a event A given B, is given by => p(A | B) = p(A∩B)/p(B). Also p(A∩B) = p(A | B) ×p(B)

We will apply this simple formula recursively on this term => p(C_k, x_1, x_2,....., x_n).

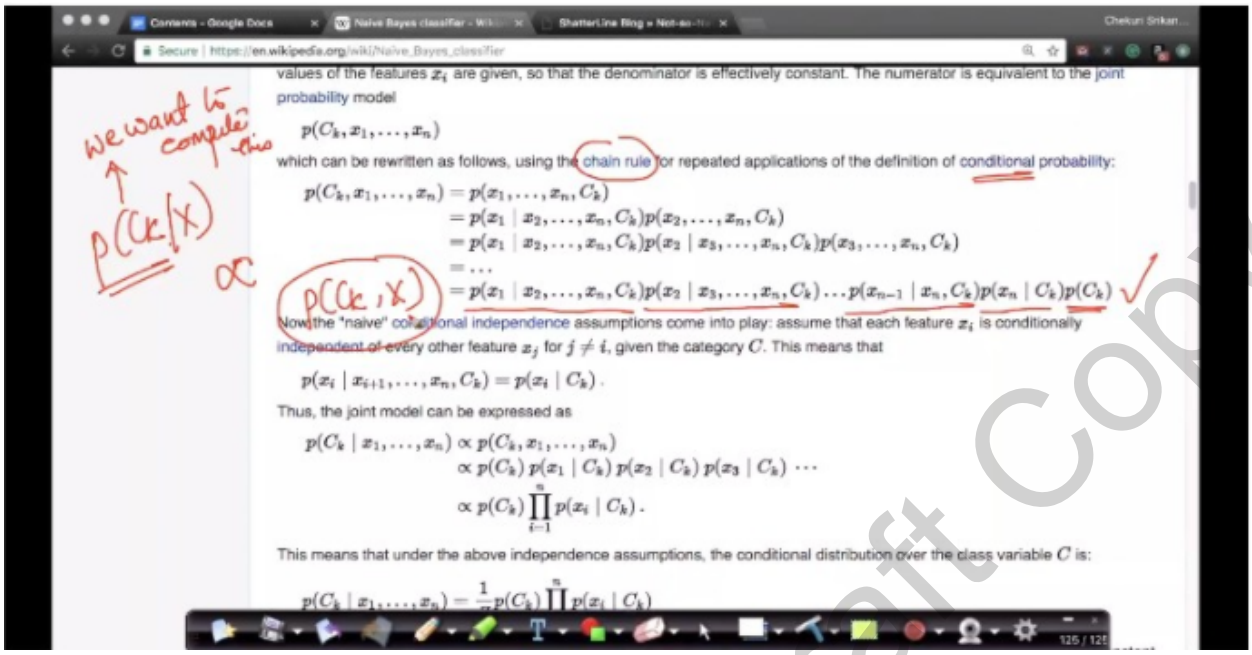
Now,

p(C_k, x_1, x_2,....., x_n) = p(x_1, x_2,....., x_n,C_k) [we can exchange terms because A∩B = B∩A]

Now, we can break our elements into two components say x_1 = A and x_1, x_2,....., x_n,C_k = B, and apply conditional probability.

So , we have

p(x_1, x_2,....., x_n,C_k) = p(x_1| x_2,....., x_n,C_k) ×p(x_2,....., x_n,C_k)



Timestamp 15:45

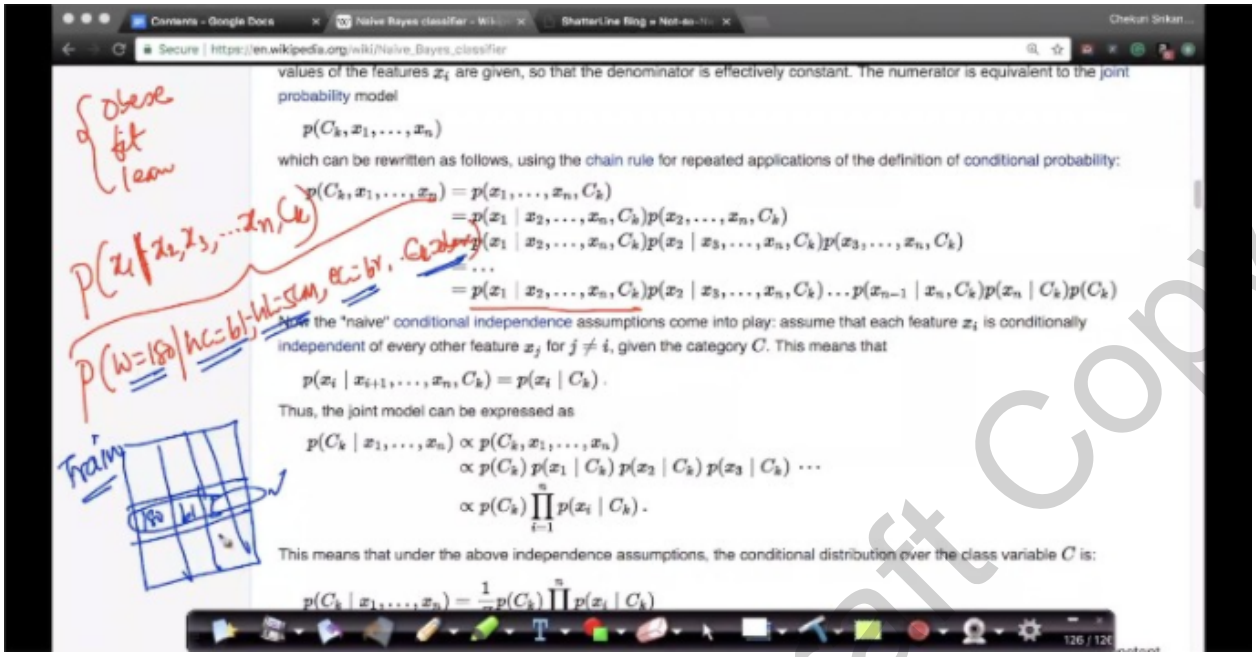
We will continue applying conditional probability recursively.

This is what we get after the first expansion.
$p(x_1, x_2, \ldots, x_n, C_k) = p(x_1 \mid x_2, \ldots, x_n, C_k) \times p(x_2, \ldots, x_n, C_k)$

Now we will keep the first part => $p(x_1 \mid x_2, \ldots, x_n, C_k)$ as it is and expand the second one => $p(x_2, \ldots, x_n, C_k)$, as shown in the image above.

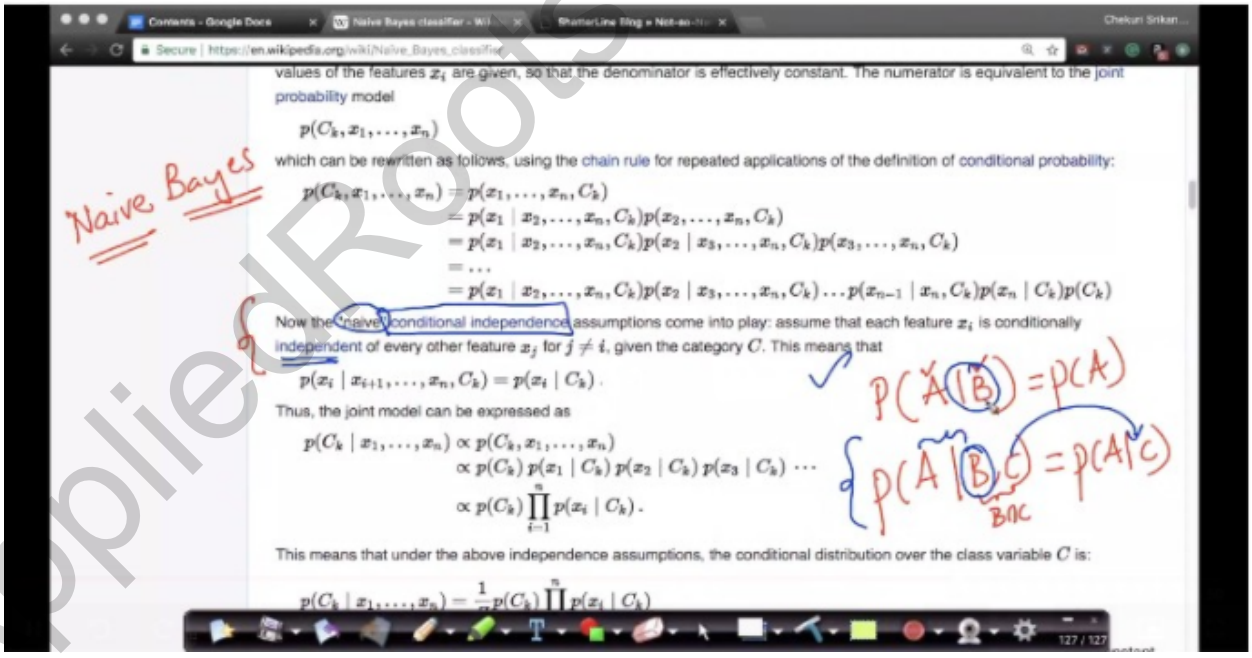
Continuing like this, the final term that we get is,
$= p(x_1 \mid x_2, \ldots, x_n, C_k) \times p(x_2 \mid x_3, \ldots, x_n, C_k) \ldots p(x_{n-1} \mid x_n, C_k) \times p(x_n \mid C_k) \times pC_k$

Now we need to solve for components of this probability terms from the data.



Timestamp 18:00

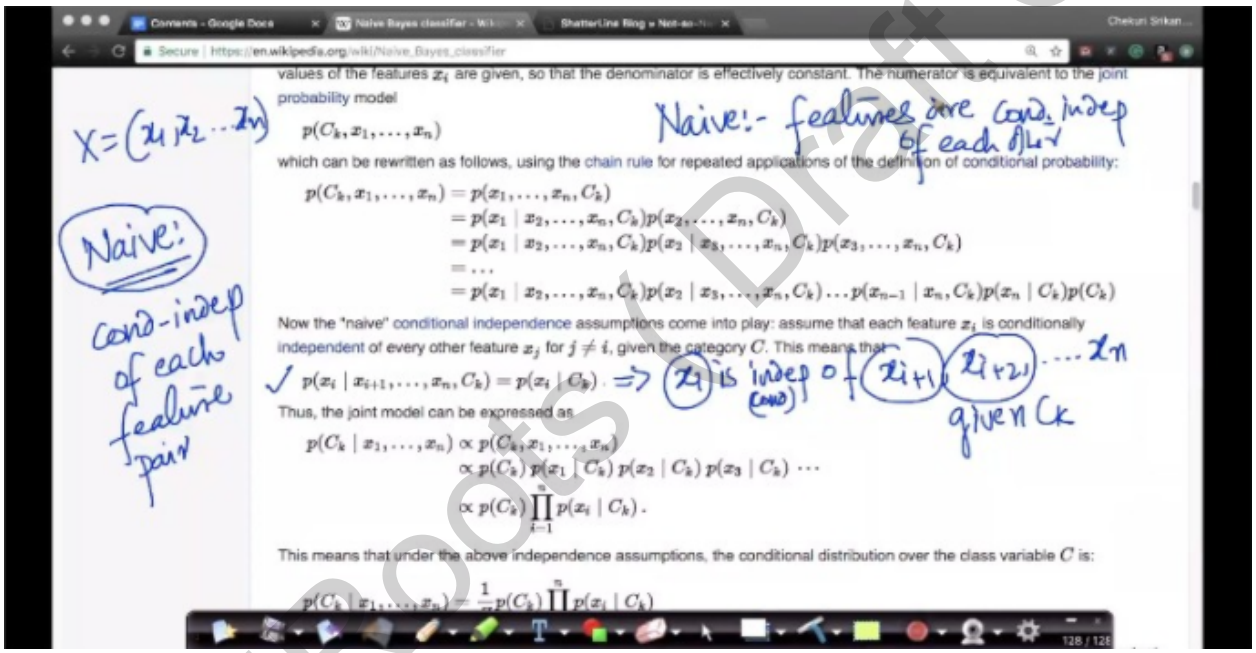
Now, finding these individual terms is very difficult because we have to count those data points for which our condition matches exactly. For that we need lots of data, rather lots of combinations of data, which may not be always possible.



Timestamp 20:05

In order to overcome this problem of finding an exact match we make a naive assumption. That's why this is called Naive Bayes. The assumption is that of conditional independence of the features.
Two events are said to be independent if the occurrence of one doesn't affect the other.
p(A|B) = p(A)

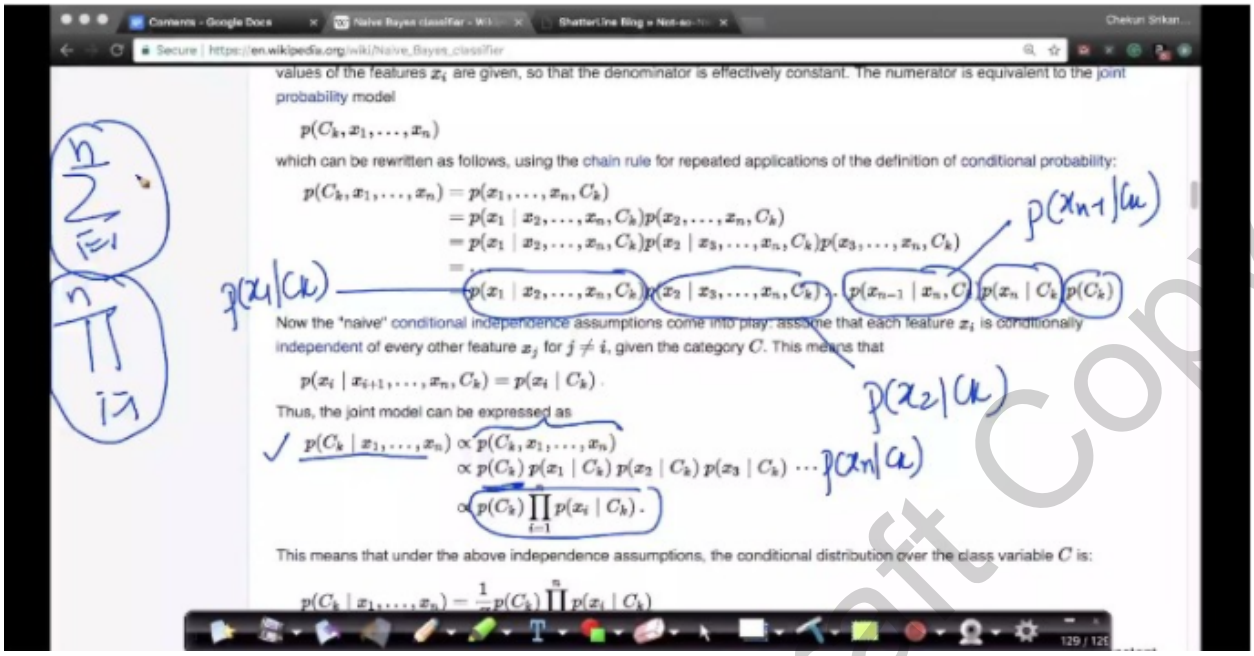
Similarly conditional independence can be written as
p(A| B, C) = p(A|C),
where p(A| B, C) is the probability of A given both B and C. Since the probability of A given C is the same as the probability of A given both B and C, this equality expresses that B contributes nothing to the certainty of A. In this case, A and B are said to be conditionally independent given C.



Timestamp 21:44

So our naive bayes assumption is that feature x_i is conditionally independent of every other feature x_j for j ≠ i, given the class k.

Now this idea will be used for every term in our probability. In general,
p(x_i| x_{i+1},....., x_n,C_k) = p(x_i|C_k)
Here x_i is conditionally independent of x_{i+1}, x_{i+2},..., x_n given C_k. This vastly simplifies our calculation.

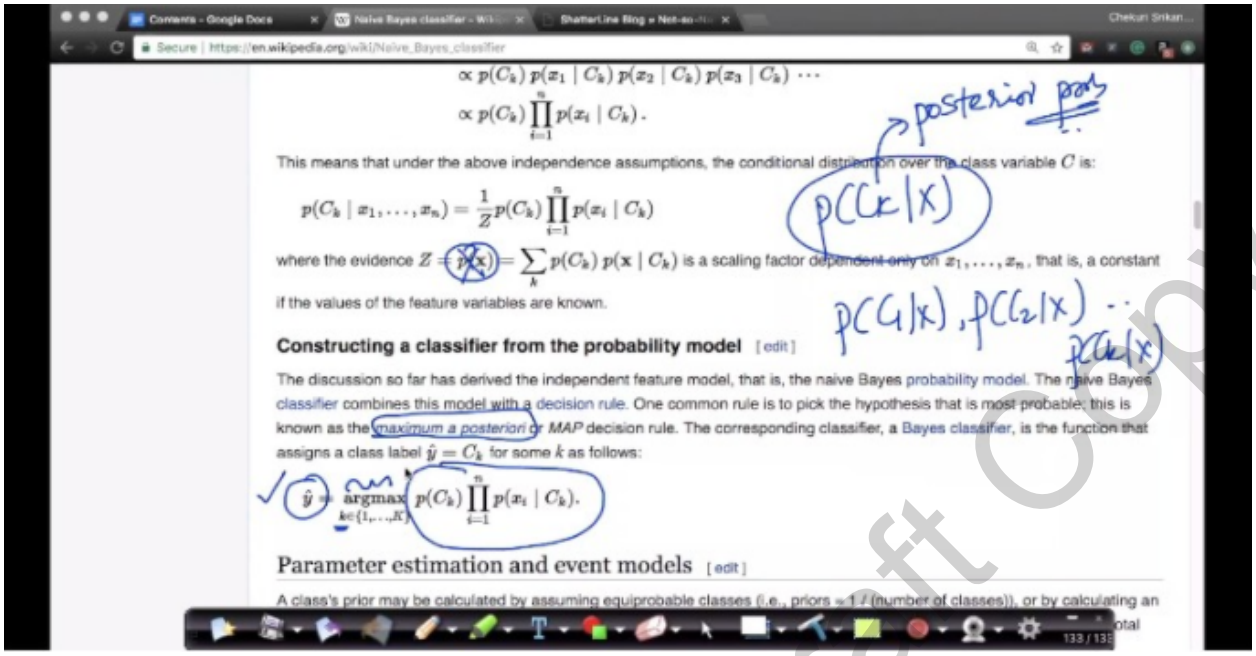


Timestamp 23:00

Now the conditional independence can be applied to each of the terms. After following the steps as shown in the image we reach to our final probability, which is

$$p(C_k, x_1, x_2, \ldots, x_n) \propto p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)$$

The proportionality comes because we are ignoring the denominator. It is a product of individual terms along with the probability of the class.



Timestamp 25:28

If we want an exact term, we can remove the proportionality and bring in the denominator as shown in the image.

Now for a classifier we will find the probability for each of the classes, and select that class which gives the maximum probability.

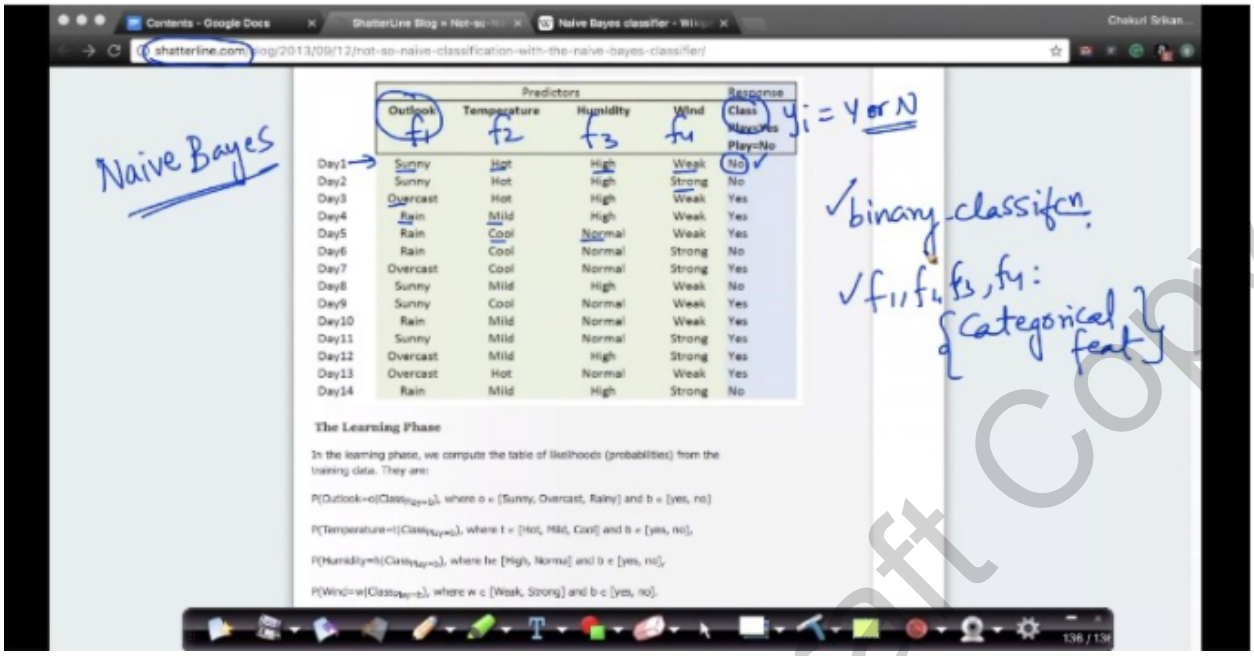
$\hat{y} = argmax_{k \in \{1,\ldots,K\}} p(C_k) \prod_{i=1}^{n} p(x_i|C_k)$

This rule is known as *maximum a posteriori* or MAP decision rule, as we are selecting the argmax of the maximum posterior.

https://en.wikipedia.org/wiki/Naive_Bayes_classifier

32.2 Toy example: Train and test Stages

http://shatterline.com/blog/2013/09/12/not-so-naive-classification-with-the-naive-bayes-classifier/

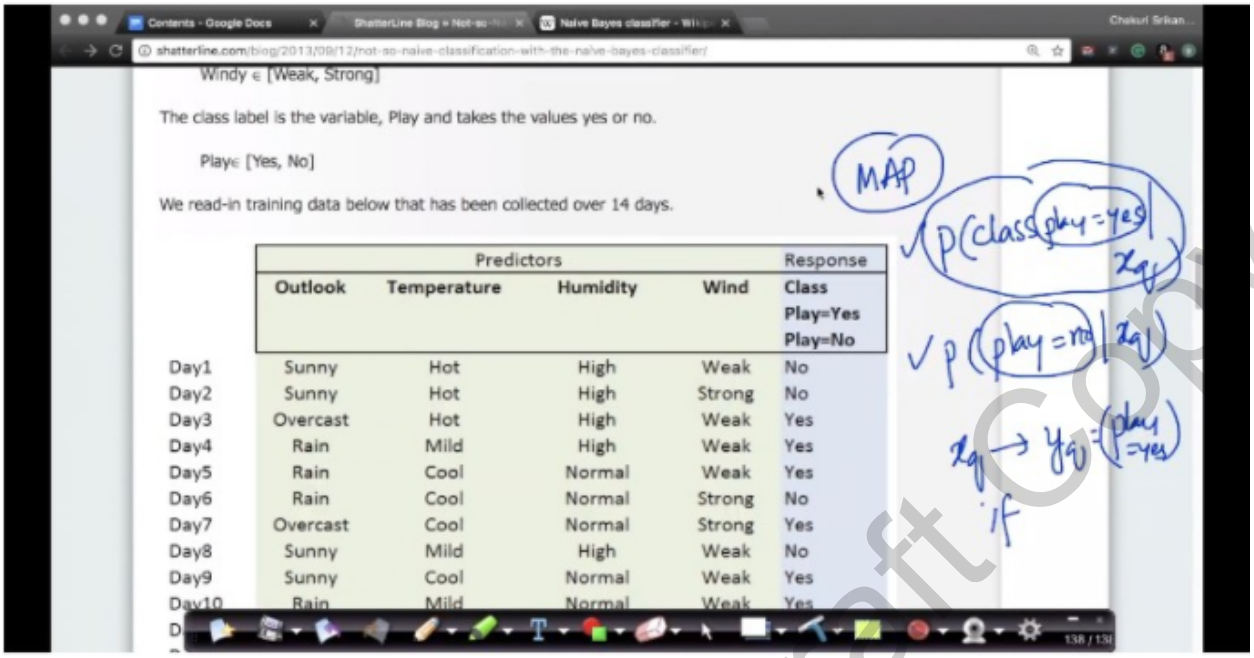


Timestamp 2:28

In this chapter we will look at a toy dataset and apply naive bayes algorithm on this dataset. It's a binary classification problem with target variable play, $y_i \epsilon$ {Yes(Y), No(N)}. Our dataset contains 4 features => Outlook(f_1), Temperature(f_2), Humidity(f_3), Wind(f_4). All these features are categorical features, naive bayes can also be used for continuous features but it is easy to visualize for categorical data. So we will stick to categorical data.

Outlook(f_1) can have 3 categories: sunny, overcast, rain. Temperature(f_2) can have 3 categories: hot, mild, cool. Humidity(f_3) can have 2 categories: high, normal. Wind(f_4)can have 2 categories: strong, weak.

Here we are trying to predict whether it is suitable to play tennis outside given various weather parameters.

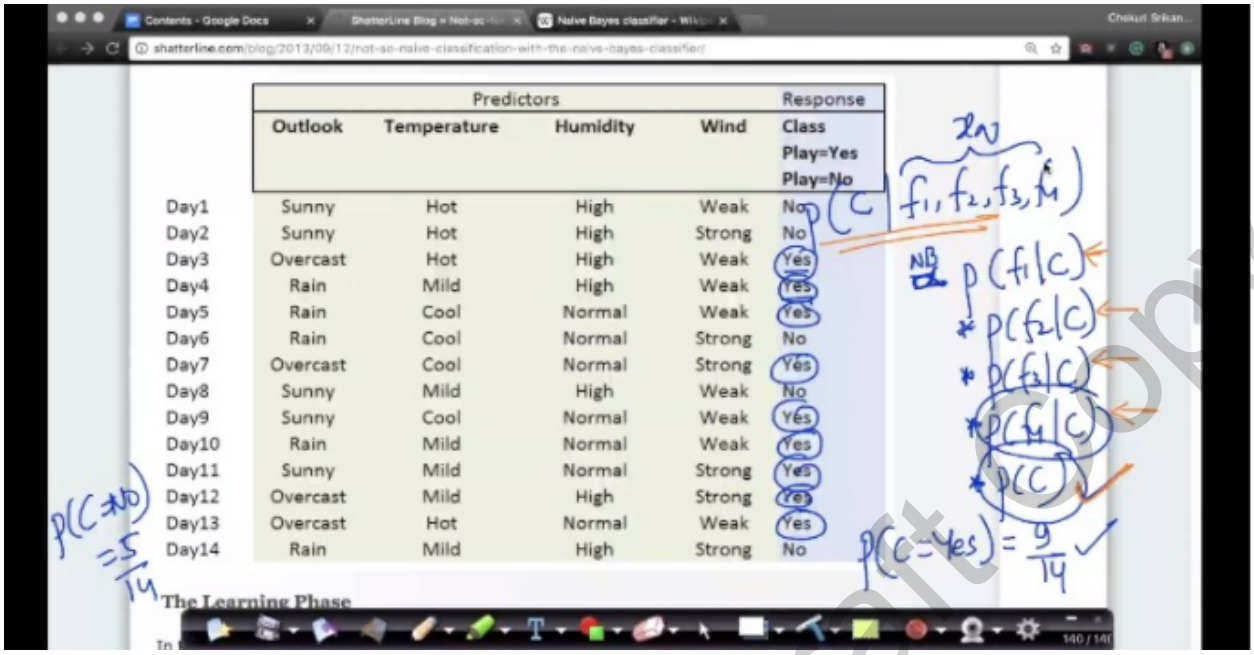


	Predictors				Response
	Outlook	Temperature	Humidity	Wind	Class Play=Yes Play=No
Day1	Sunny	Hot	High	Weak	No
Day2	Sunny	Hot	High	Strong	No
Day3	Overcast	Hot	High	Weak	Yes
Day4	Rain	Mild	High	Weak	Yes
Day5	Rain	Cool	Normal	Weak	Yes
Day6	Rain	Cool	Normal	Strong	No
Day7	Overcast	Cool	Normal	Strong	Yes
Day8	Sunny	Mild	High	Weak	No
Day9	Sunny	Cool	Normal	Weak	Yes
Day10	Rain	Mild	Normal	Weak	Yes

Timestamp 4:32

So, here we need to calculate two probabilities,
p(Play = Yes | x_q), it says what is the probability that we should outside play outside given weather params x_q. Similarly p(Play = No | x_q), what is the probability that we should not play outside given weather params?

Out of both these terms whichever term possesses the highest value we declare that to be our class label, MAP rule.



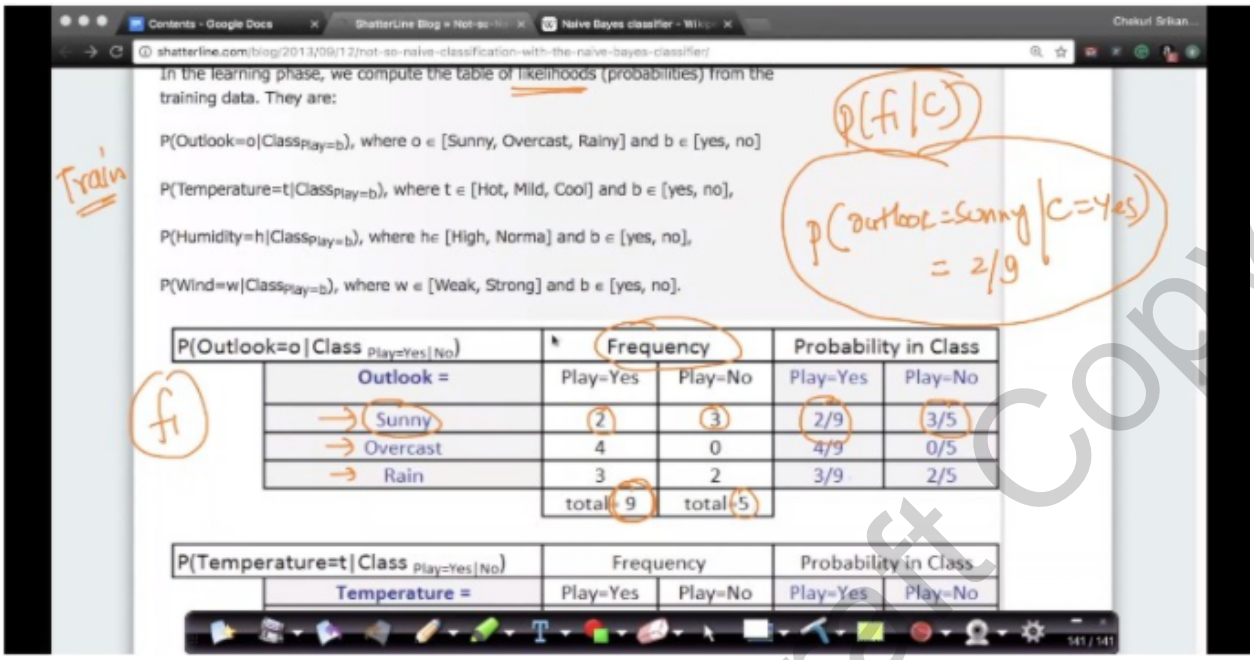
Timestamp 8:02

As per Naive Bayes,
$p(C \mid f_1, f_2, f_3, f_4) \propto p(f_1 \mid C) * p(f_2 \mid C) * p(f_3 \mid C) * p(f_4 \mid C) * p(C)$

We have written the above term directly from Naive Bayes where f_i's are our features and C is our class.

p(C = Yes) = 9/14, we have 9 yes out of 14 data points.
p(C = No) = 5/14, we have 5 no out of 14 data points.

This are called prior probabilities.



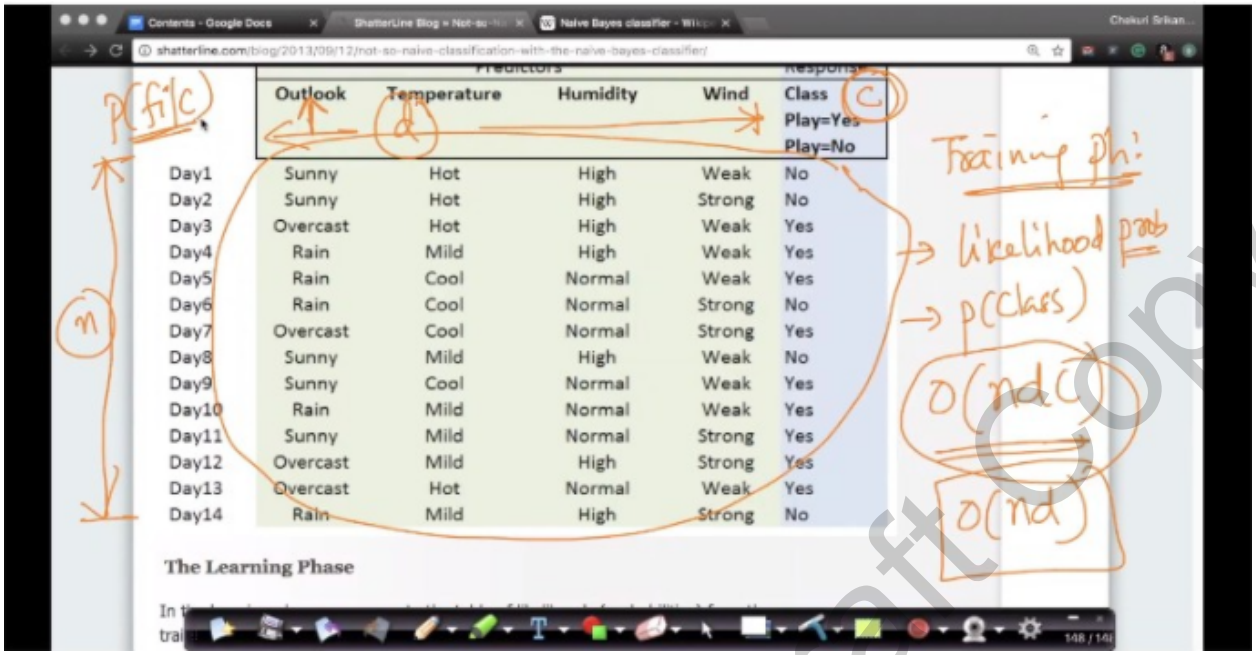
Timestamp 12:33

Now we need to calculate the feature probabilities(likelihood), for calculating these we will use the train data. This phase is called the learning/training phase.
Let's calculate one of the terms => p(Outlook = sunny| C = Yes)

Now here we need to find the probability of an event where outlook = sunny given C = yes. So, at first we will count all the data points which have class Yes and out of those data points we need to count those features where outlook = sunny.

So, after counting we got p(Outlook = sunny| C = Yes) = 2/9, Out of 9 data points which have class label = yes, 2 of them had an outlook = sunny.
SImilarly p(Outlook = sunny| C = No) = 3/5, Out of 5 data points which have class label = No, 3 of them had an outlook = sunny.

We need to calculate the likelihood of each of the features, and store it in some data structure. As shown in the image above. Please follow the shatterline link to get a complete list of all the probabilities.



Timestamp 15:19

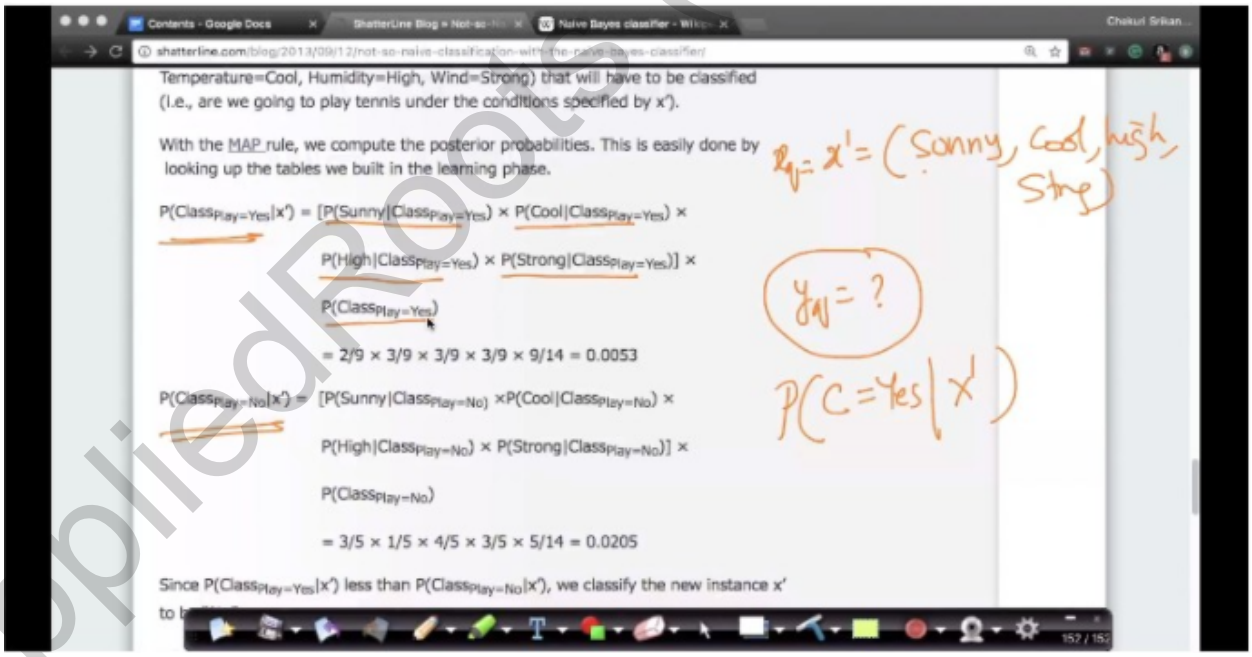
Let's digress for a minute and calculate the complexities.

Time Complexity
Say we have n data points in our dataset and each data point has d dimension. Also say we have c classes.Then if we use a brute force algorithm our time complexity is O(ndc).
This is because for calculating the individual probabilities we need to go through each feature i.e. d, n times because we have n data points. This we need to do for c classes.

If we use some optimization, time complexity can be reduced to O(n). if d is small. Otherwise it is O(nd).

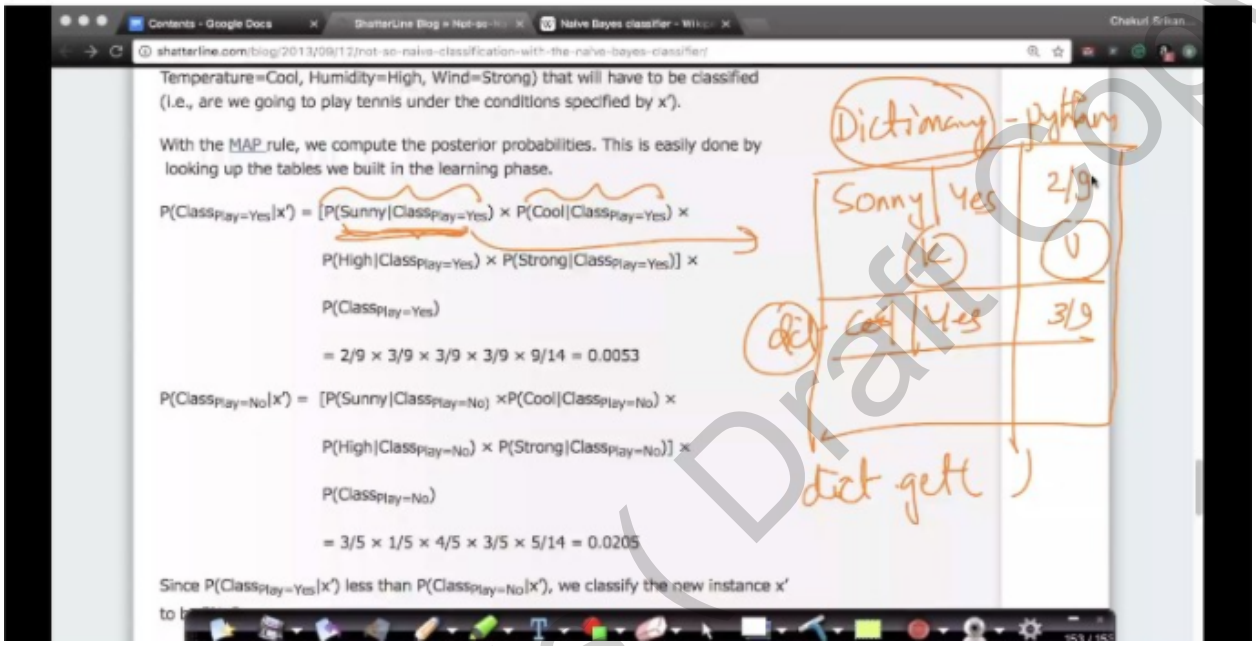
Timestamp 17:00

Space Complexity is O(dc), because we need to store the probabilities of each feature i.e. d, c times i.e. no of classes. Also we need to store class probabilities.



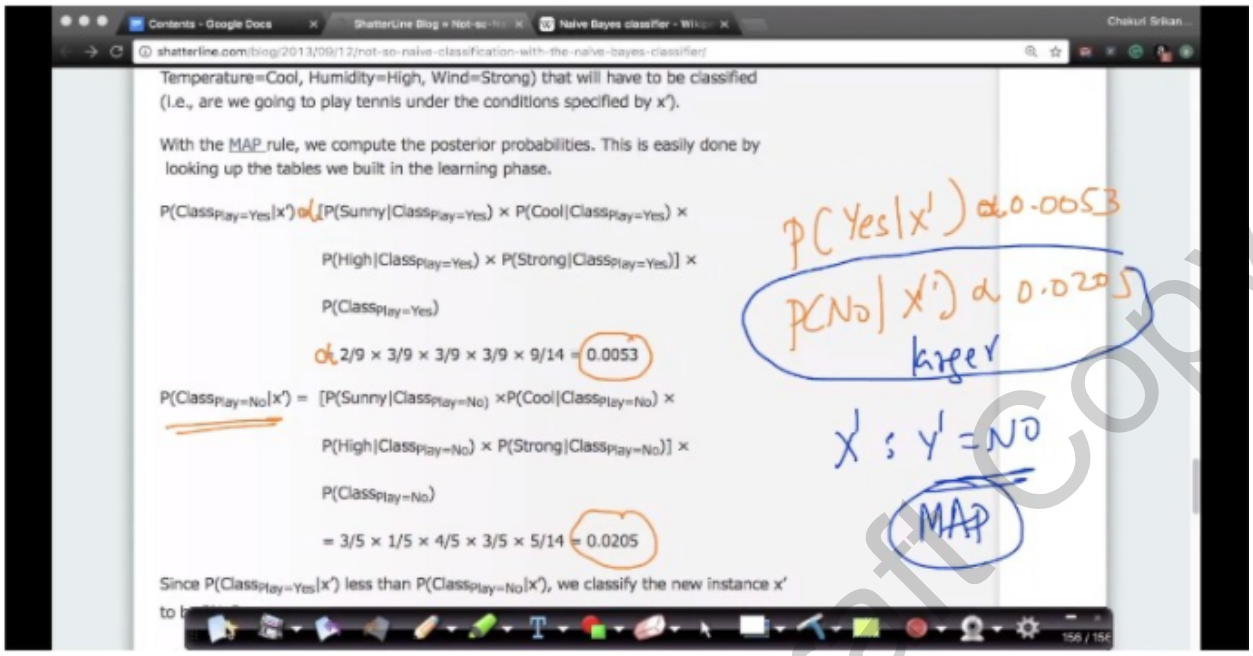
Timestamp 18:30

Now, given a data point during test time we need to classify the data point. Say we have a data point x_q=(sunny, cool, high, strong), here we are given values for the 4 features. We need to calculate two posteriors, because we have two classes.
p(class = Yes | x_q) and p(class = No | x_q).
We can write these posteriors in terms of their features as shown in the image.



Timestamp 20:02

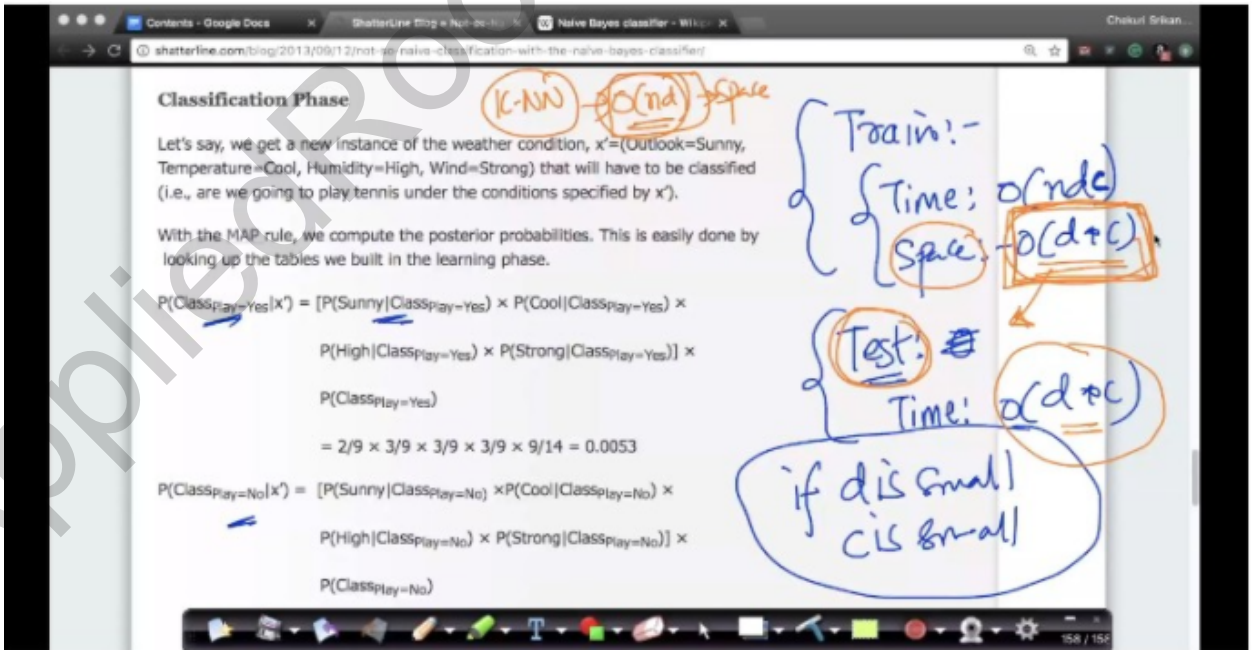
For individual terms we need to use the probabilities that we have calculated during the training phase. For example, one of the feature is p(Outlook = Sunny | class = Yes), so during training we can build a dictionary with feature combinations as key and the probabilities as value. We can have a key <Sunny,Yes> and its probability i.e. 2/9 as its value. We can use python's inbuilt dictionary methods to implement these.



Timestamp 21:24

After putting all the likelihood values and priors in we get,
p(class = Yes | x_q) ∝ 0.0053
p(class = No | x_q) ∝ 0.0205

So as per MAP rule we can say that our class instance is No.

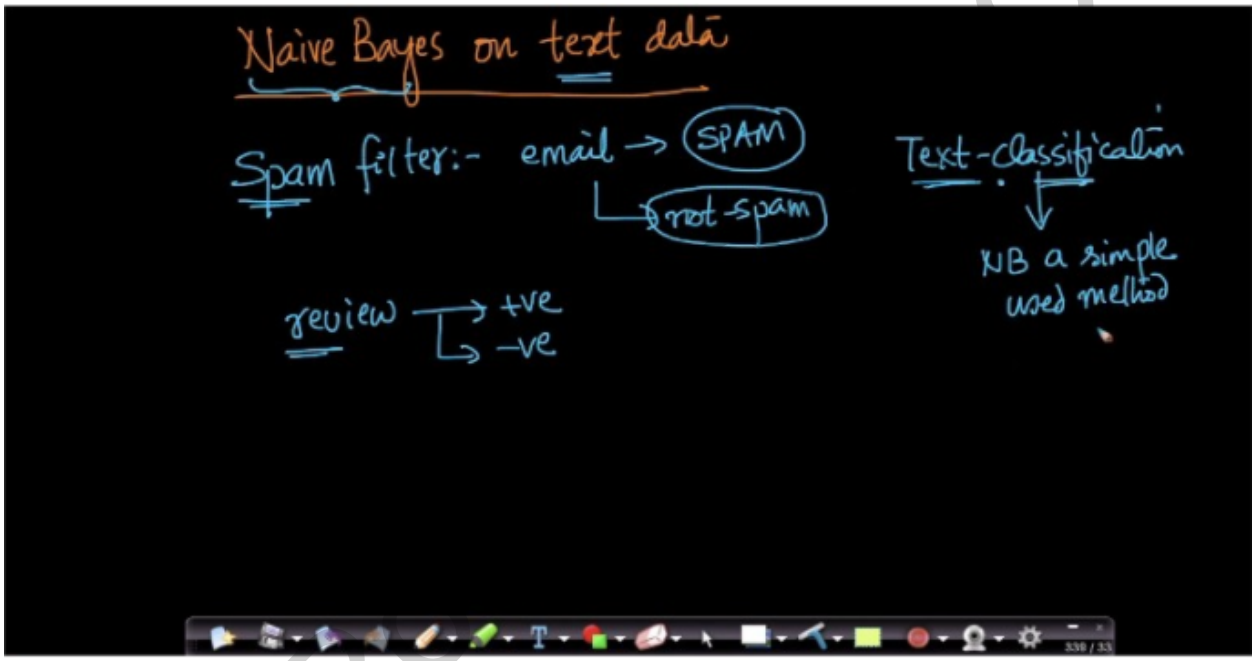


Timestamp 24:17

Now let's look at complexity during test time.
Time Complexity = O(d*c), where d is the dimension of the data, c is the no of classes. This is because we need to look up d times i.e. for each feature, for each class i.e. c times. We are also considering our lookup time to be constant.

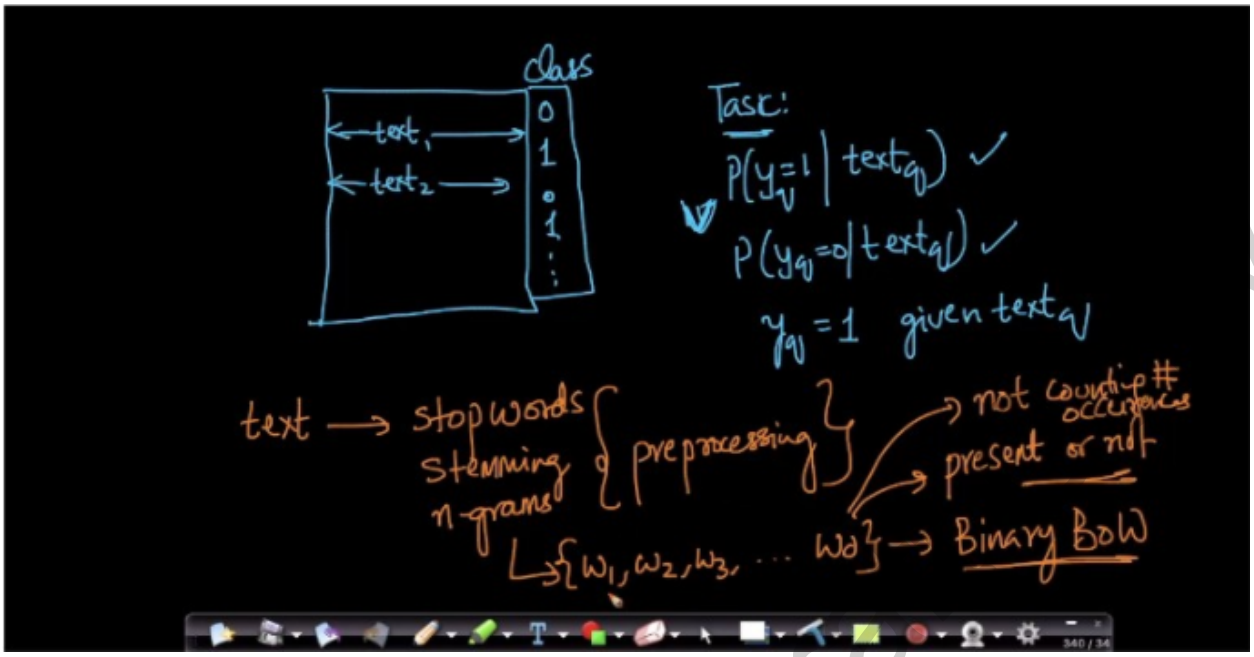
If we compare NB with KNN, we see that NB is extremely space efficient at runtime because in KNN we need all the data points in memory, whereas NB requires only the look up tables to be in the memory.

32.3 Naive Bayes on Text Data



Timestamp 1:31

NB is extremely useful when it comes to text data. One of the earliest spam filters was built using NB. A spam filter classifies a mail into ϵ{Spam, Not Spam}. NB is very efficient in classifying text data into various classes.

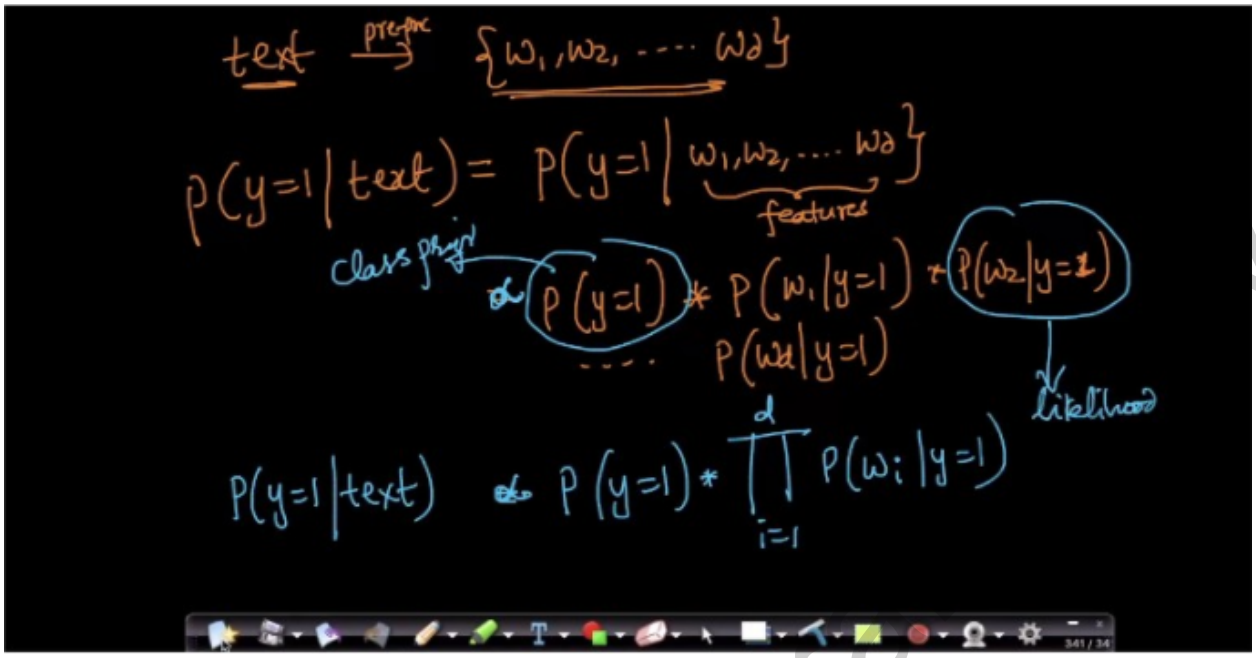


Timestamp 4:36

Suppose we are given a dataset where we have textual data; each text belonging to a certain class. Let's keep it to binary classification; it can be easily extended to multi-class classification problems. So our class labels are ϵ{0, 1}. As shown in the image above.
Now we want to calculate,
p(y_q = 1 | $text_q$)and p(y_q = 0 | $text_q$) i.e. probability that the class is 0 or 1, given text $text_q$.

For handling text data we need to preprocess it, we can remove stopwords, do stemming, calculate n-grams.etc. All these steps are application dependent and not concrete.

After doing the preprocessing we need to convert this text data into numerical vectors. In NB we typically use binary bag of words(BOW). Binary BOW contains cells for each unique word in the corpus. Each cell can have a value of 0 or 1, representing whether that particular word is present in the current text or not. We will consider only those cells where its value is 1, i.e. the words present in the text.



Timestamp 7:00

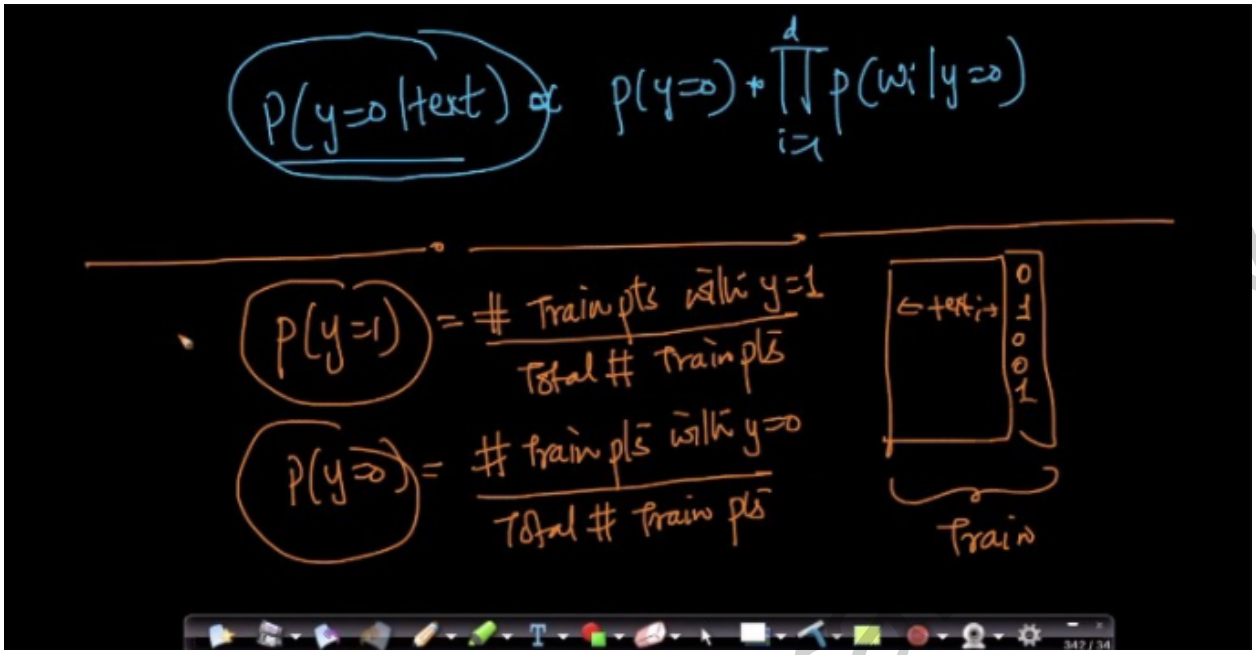
Say we have a text $text_q$, after preprocessing we get => {w_1,w_2,…,w_d}. These individual words are features.

Now we need to calculate, p(y = 1| $text_q$)

So,

$$p(y = 1 | text_q) \propto p(y = 1) \prod_{i=1}^{d} p(w_i | y = 1)$$

The above term can be derived as shown in the image above.



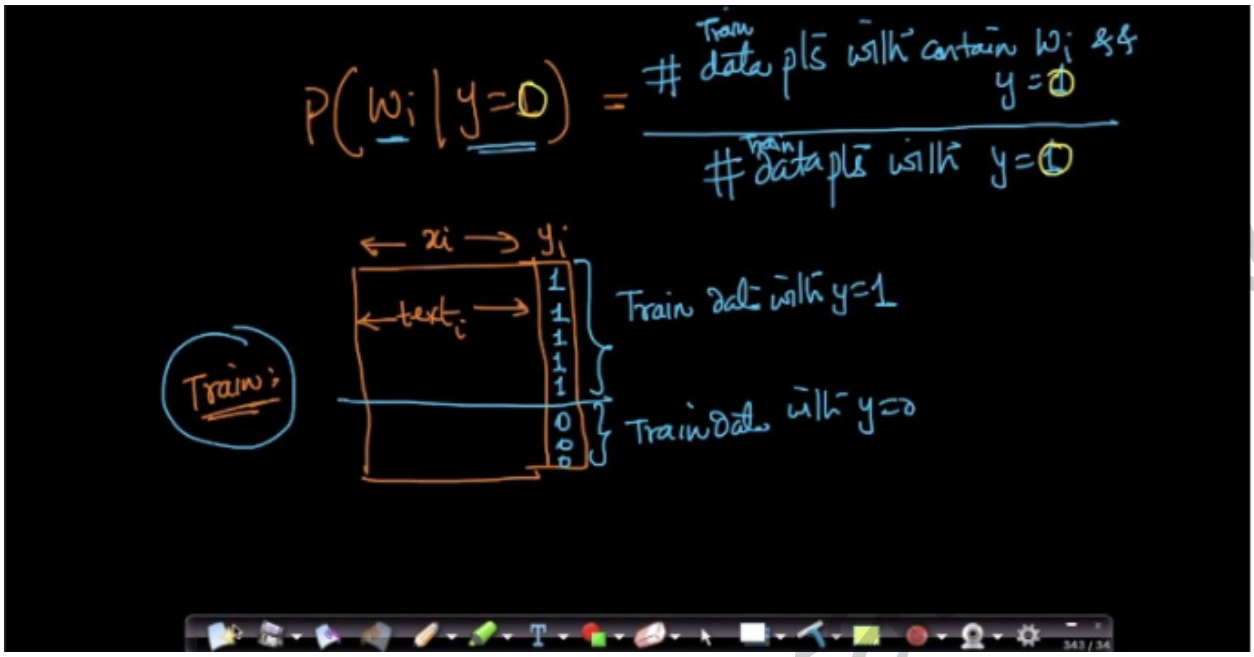
Timestamp 9:10

Similarly,

p(y = 0| $text_q$) ∝ p(y = 0) $\prod_{i=1}^{d}$ p(w_i|y = 0)

The prior probabilities i.e. p(y = 1) and p(y = 0) can be calculated as,

p(y = 1) = (#train points with y = 1) / (total # of points)
p(y = 0) = (#train points with y = 0) / (total # of points)



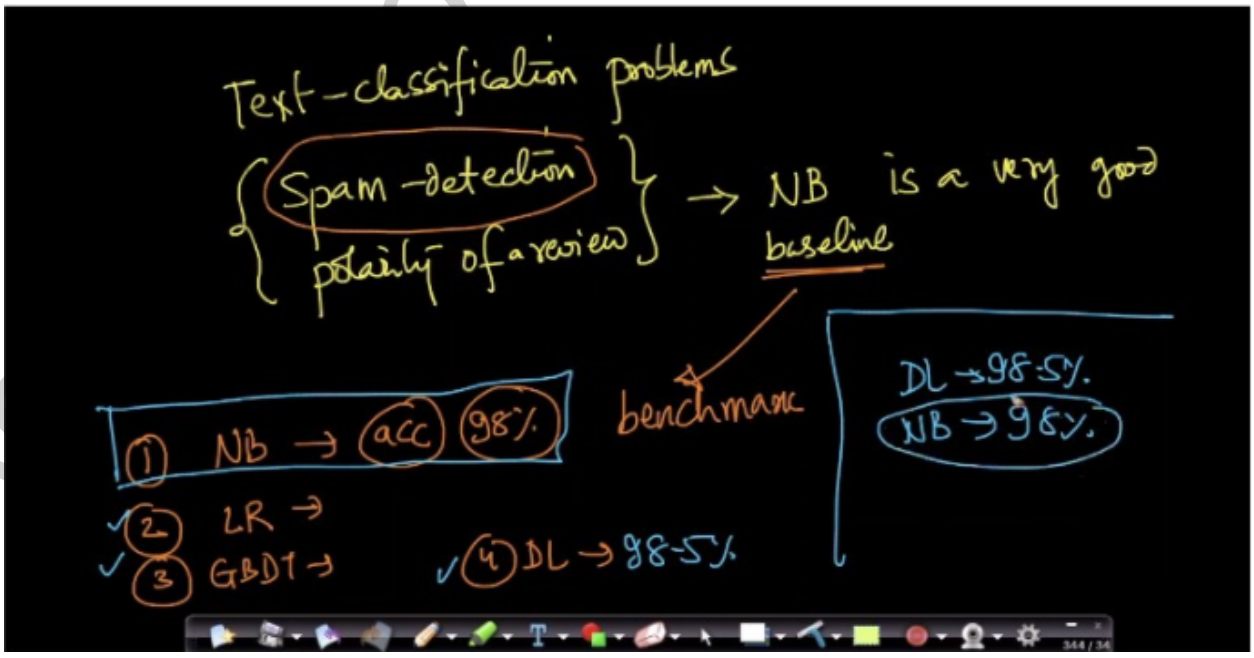
Timestamp 12:19

Now we can calculate the likelihoods, it can be calculated as

p(w_i| y = 0) = (#Train data pts which contains w_i and y = 0) / (#Train data pts with y = 0)

Similarly it can be done for y = 1.

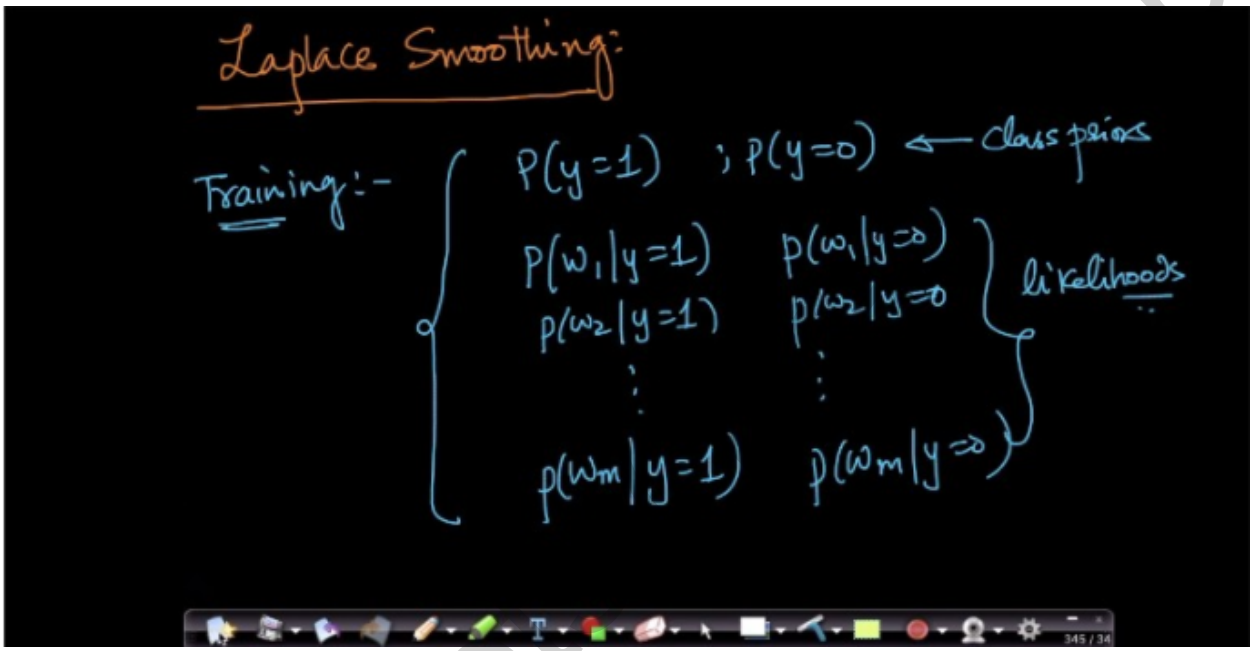
Now we have all the pieces for calculating the probabilities.



Timestamp 14:46

Naive Bayes serves as a very good baseline mode for problems like text classification, polarity of a review.etc. A baseline model is like a benchmark. We can train other models for text classification and compare with NB.

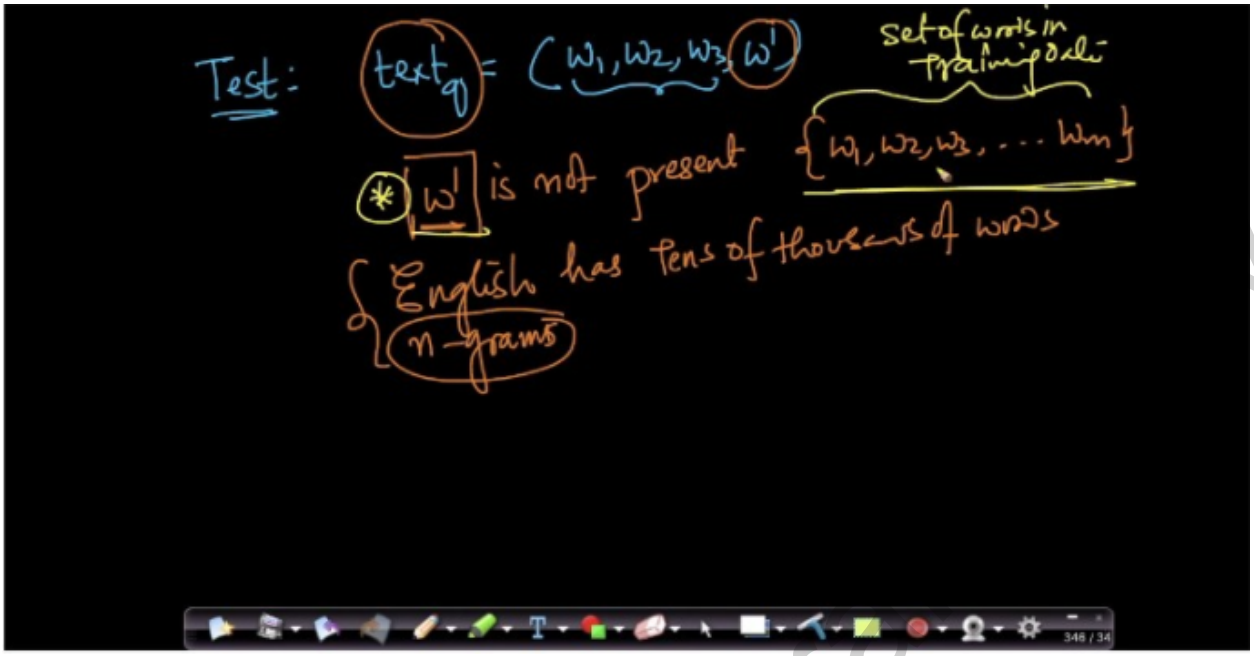
32.4 Laplace/Additive Smoothing



Timestamp 1:14

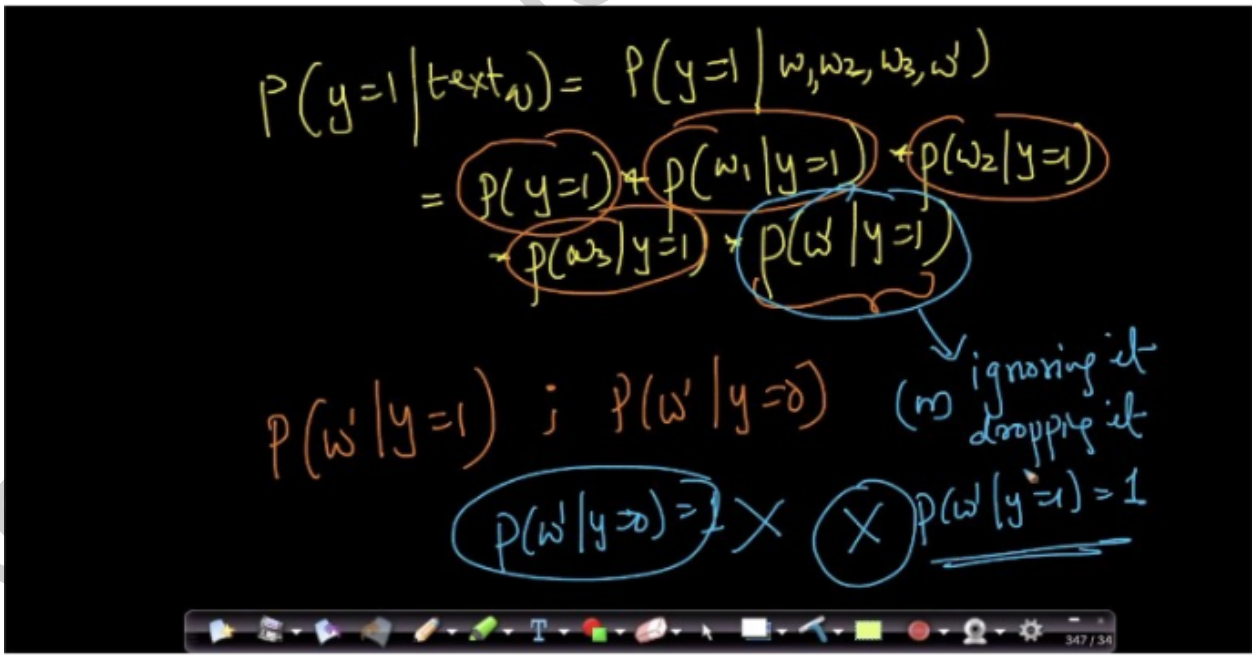
Here we will discuss something known as laplace smoothing/additive.

During the training phase we compute all the class priors and the likelihood of the features as shown in the image above. Now during train say we have a corpus of words $\{w_1, w_2, w_3, \ldots, w_m\}$



Timestamp 3:24

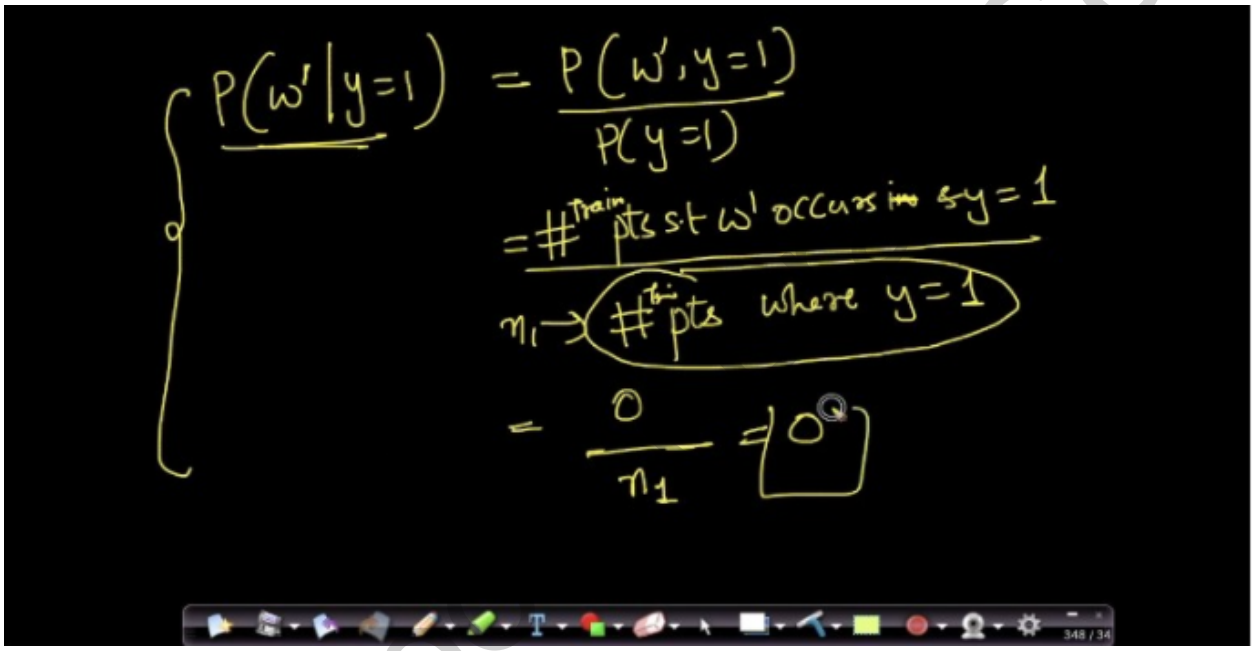
Now during test time suppose we get a text $text_q$ = {w_1,w_2,w_3,w' }, it contains word w', now this word is not present in our train dataset. English language contains lots of words and it is not always possible to get all the words in our corpus. Also if we are using n-grams, it may happen that some combination of words may not be present in our train data.



Timestamp 5:30

Now in order to calculate the probability of this $text_q$, we need to compute the likelihoods of individual words. One of the words that this $text_q$ contains is $w^{|}$,which is not present in our train dataset.
So, the question is how to deal with this situation and what value should be used for p(y = 1 | $w^{|}$) or p(y = 0 | $w^{|}$).

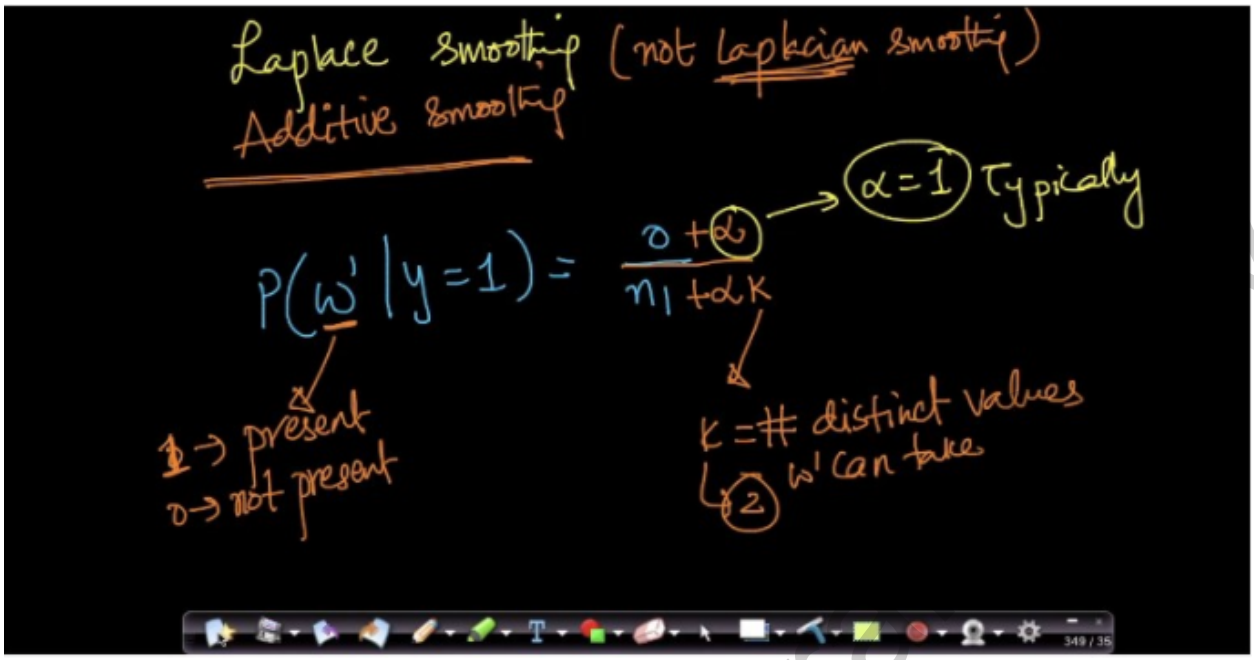
One solution is to ignore or drop these unknown words. But this is not correct, because ignoring means that we are replacing it with 1. Meaning, p(y = 1 | $w^{|}$) = 1, which is incorrect.



Timestamp 12:17

Another solution is to calculate the actual probability, here n_1 is the no of data points where y = 1.
We have to calculate the term p($w^{|}$| y = 1), here the numerator denotes no. of terms where word $w^{|}$ occurs and y = 1 , which is 0, because there are no texts in the train dataset which contains the word $w^{|}$.

If we include this term in our calculation of probability the whole term becomes 0, which is not desirable.



Timestamp 10:23

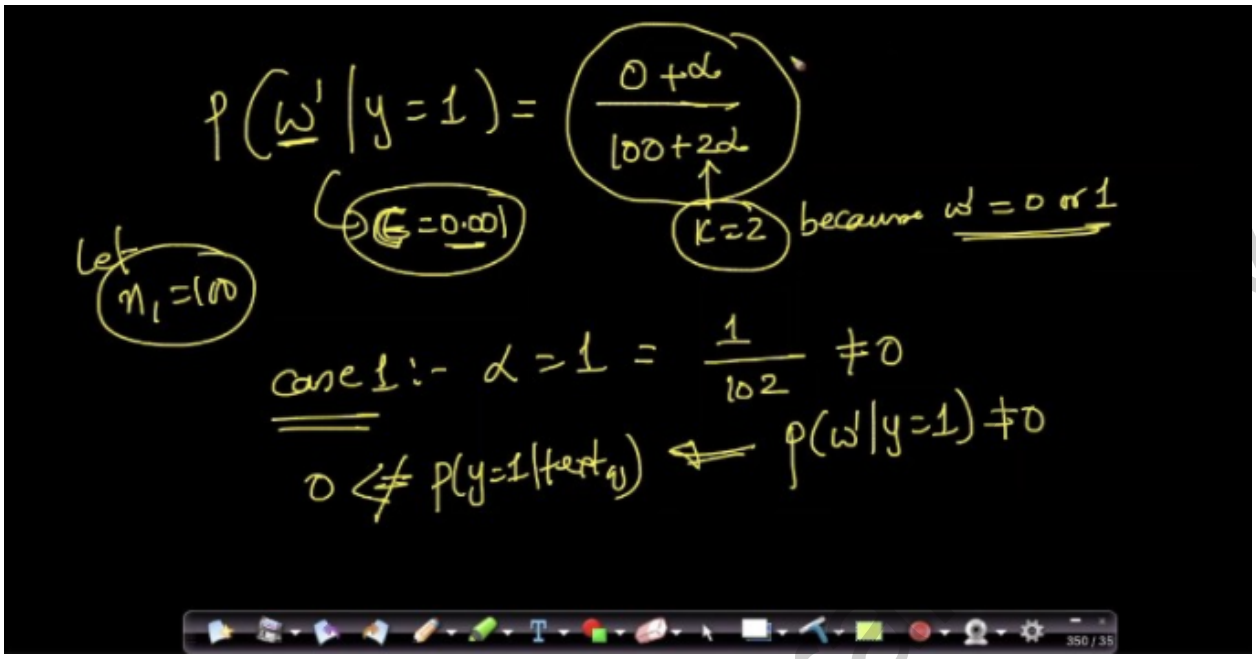
Now in order to deal with this problem we use something known as laplace/additive smoothing. Remember not laplacian smoothing, which is used in image processing.

In this technique we add constant terms to both numerator and denominator. In numerator we add a α whose value is typically 1(not always), and in the denominator we add $\alpha \times k$, where k is the number of distinct values a feature can take.

In our example k = 2 as a word w_i can be either present or not.

So our probability for $w^{|}$becomes,

$p(w^{|} | y = 1) = (0 + \alpha)/(n_1 + \alpha \times k)$

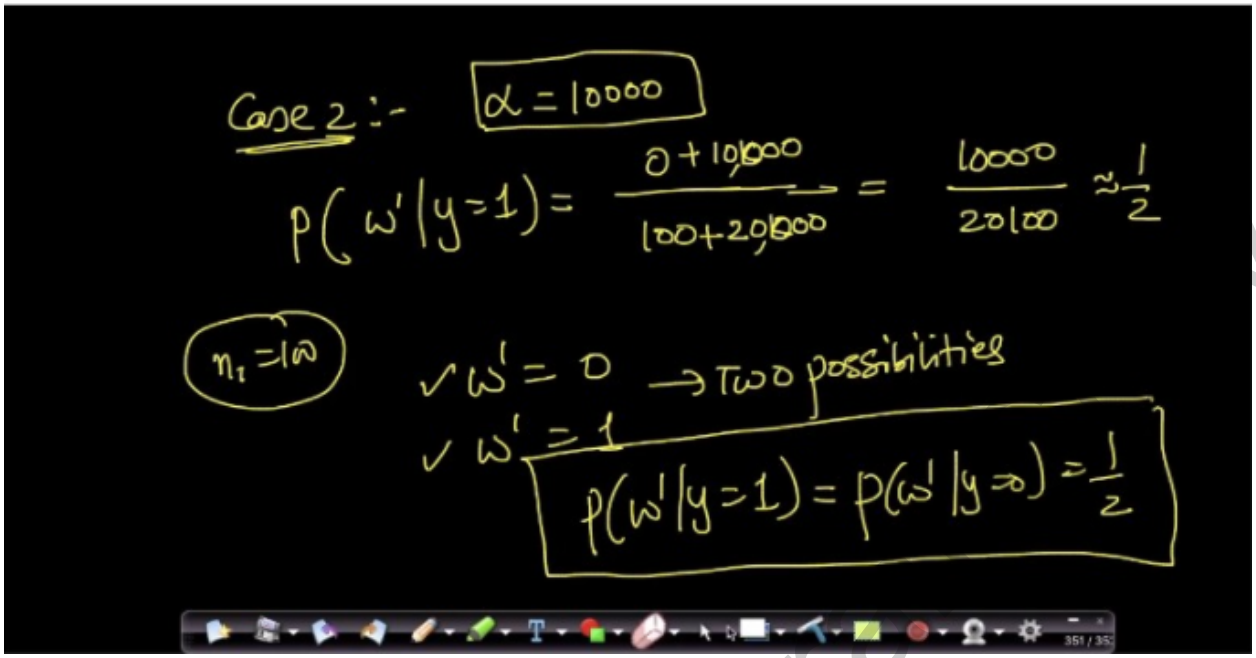


Timestamp 12:28

Now, we have gotten around the problem of getting 0 values as we are adding constants. Let's take a case where α= 1, n_1= 100 and in our example we had k = 2.

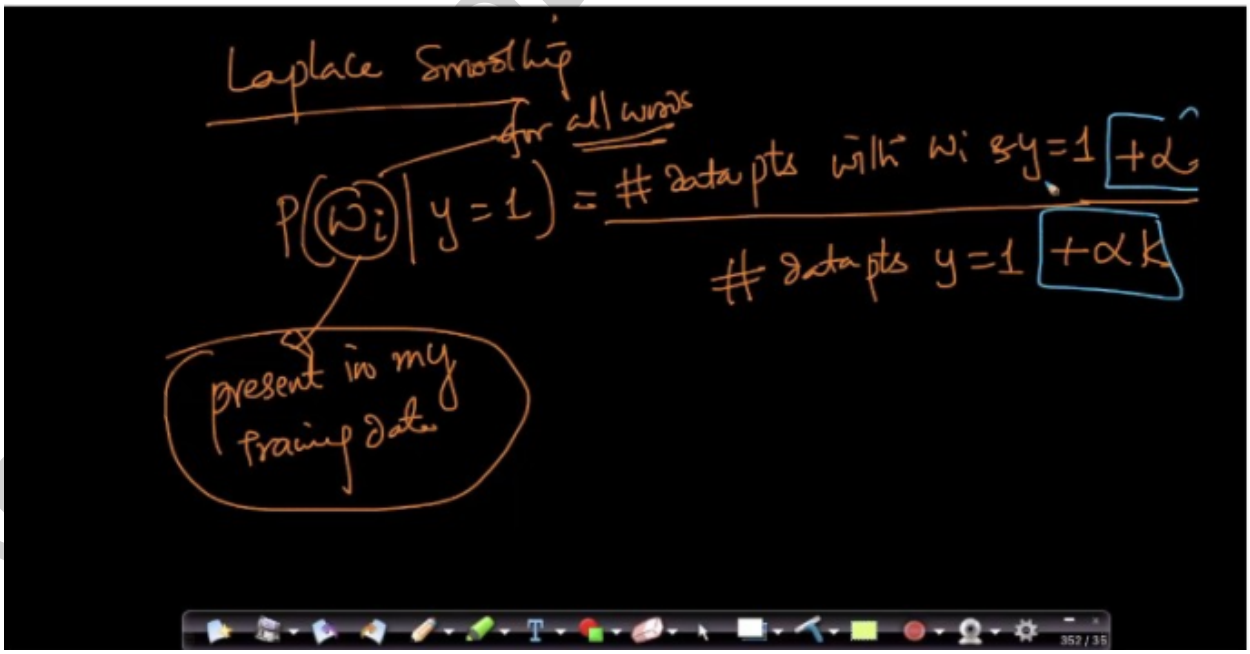
So, p($w^{|}$| y = 1) = (0 + 1)/(100+ 1 $\times$ 2) = 1/102 $\neq$0.

Now, the question arises why this is the correct way to deal with this problem, we could have simply replaced our probability for words which don't occur in our train data with some small value(ε).



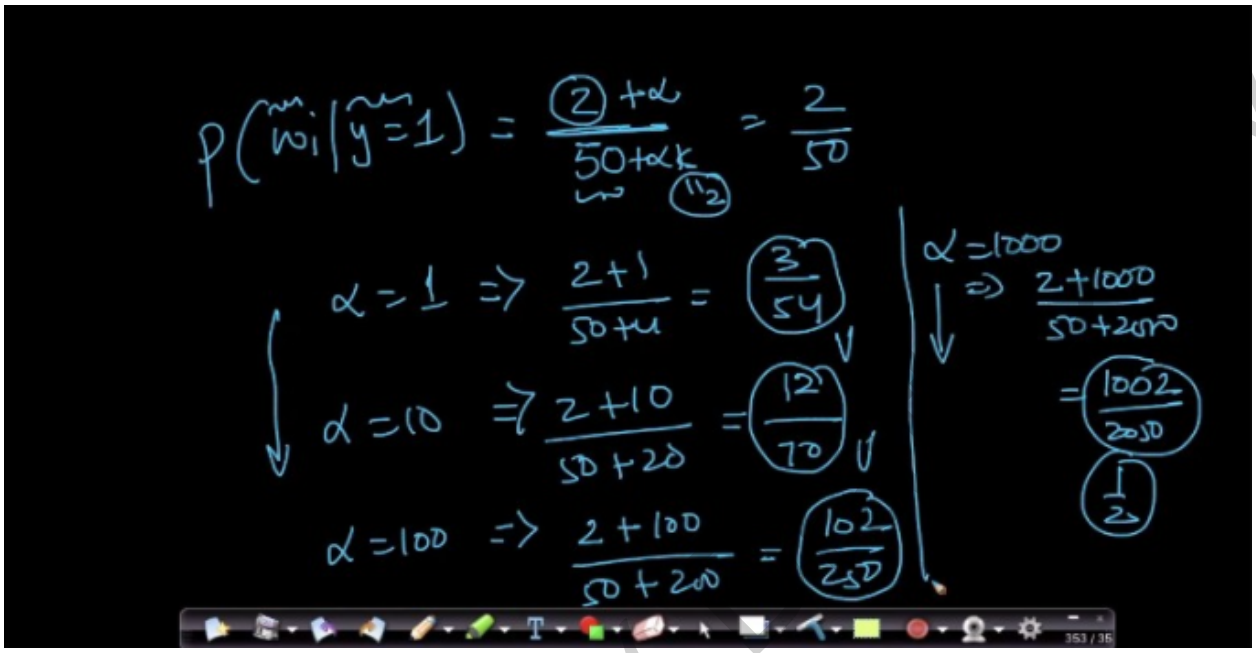
TImestamp 15:35

We will look at another case where α is large, say = 10000. In this case we get the probabilities as 1/2 for both the classes as shown in the image above which is much better than ignoring it. As we are saying that we don't know about this unknown word and how it affects class values so let's take equal probabilities for both the classes. Introducing large α's means introducing high bias.



Timestamp 17:08

So we will introduce the laplace smoothing for all the words in our training dataset, not just for unknown words. So, for every term we are adding something to the likelihoods, that's the reason it is called additive smoothing.

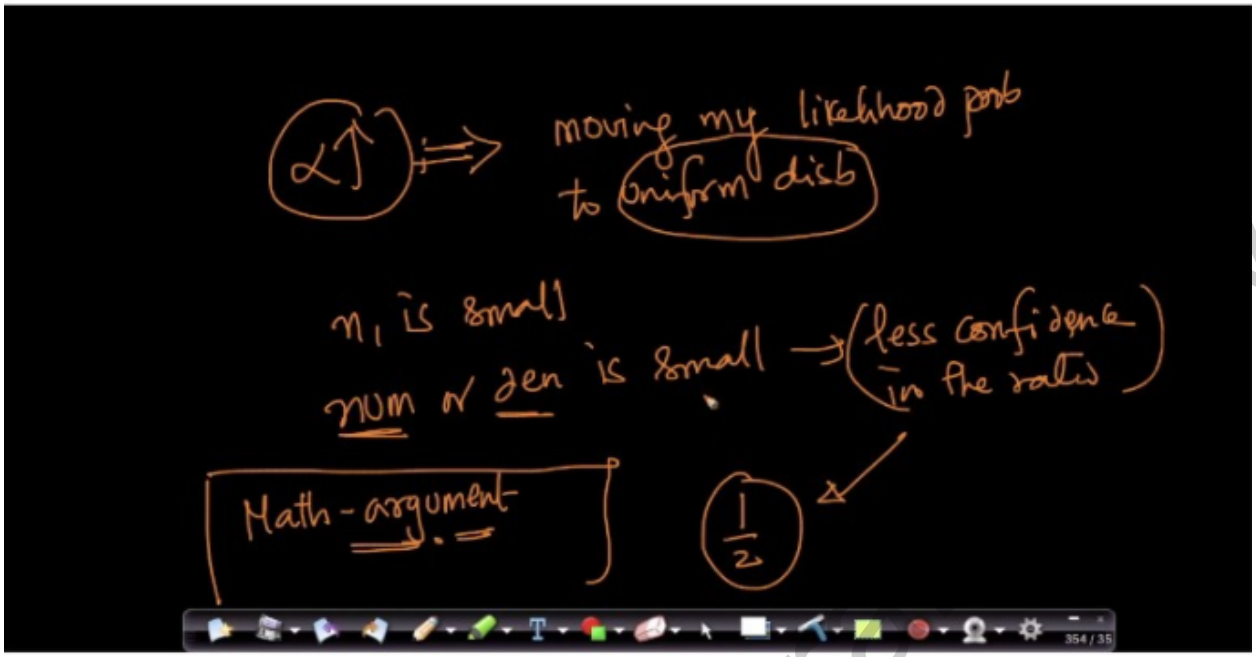


Timestamp 17:26

Here we will try to answer why it is called smoothing. For that let's take a word w_i, and calculate the probability p(w_i| y = 1),

Say p(w_i| y = 1) = 2/50.

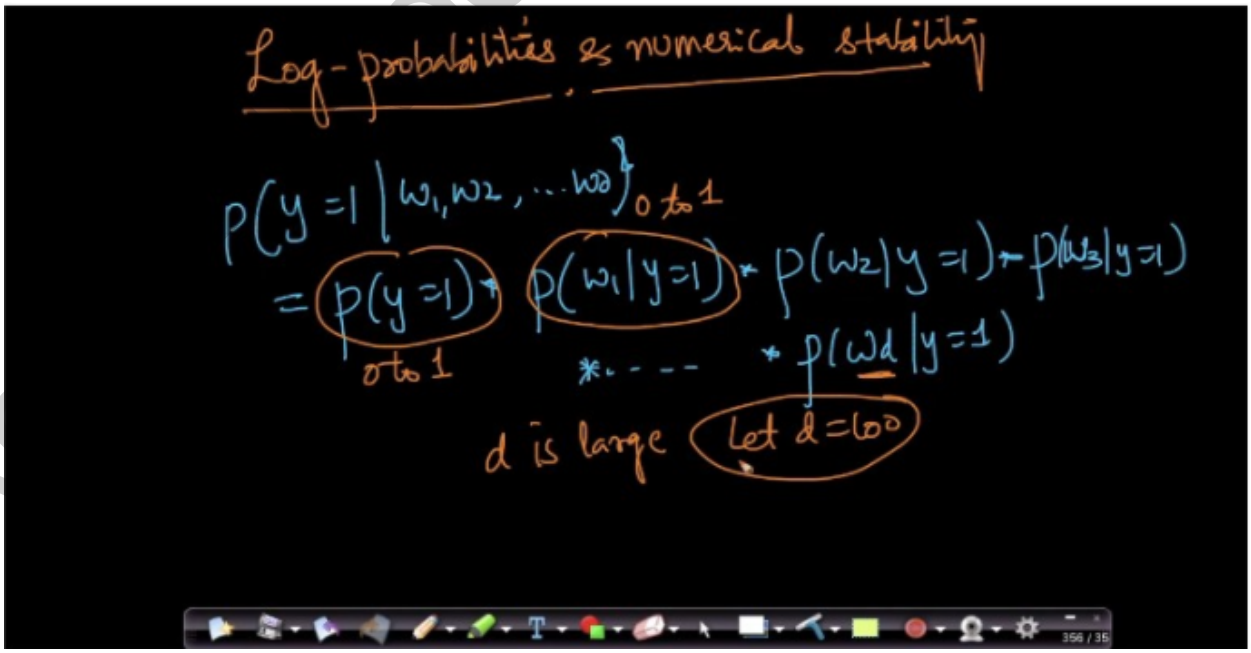
Now we will introduce laplace smoothing and calculate the probability values for different α's. We have calculated various probabilities based on different values α as it can be seen in the image above. We can observe from the values that we got that as the value of α's are increasing the probability values are moving towards 1/2.



TImestamp 21:38

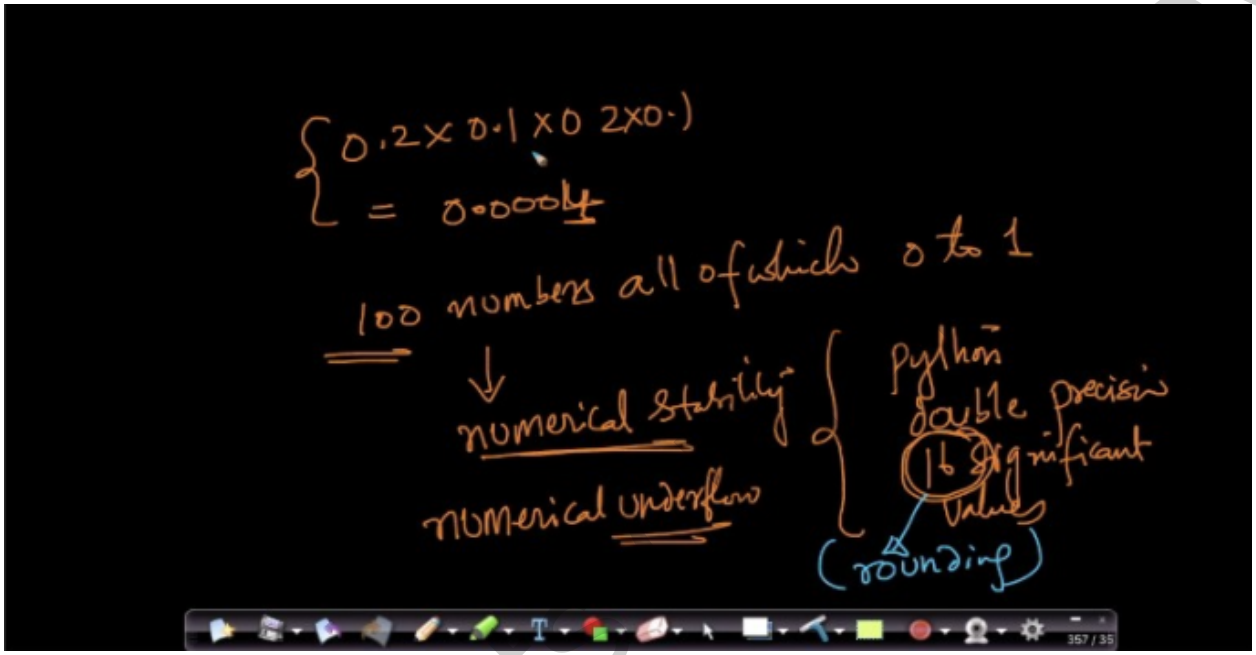
As the value of αincreases we are moving our likelihood probabilities to uniform distribution. Remember in uniform distribution all the values have equal probabilities. So it causes a smoothning of the probabilities values. In our case we had k = 2, that's why we were getting 1/2. We can also use laplace smoothing if we have very few points of a certain class. The mathematical proof of laplace smoothing is beyond the scope of this course.

32.5 Log-probabilities for numerical stability



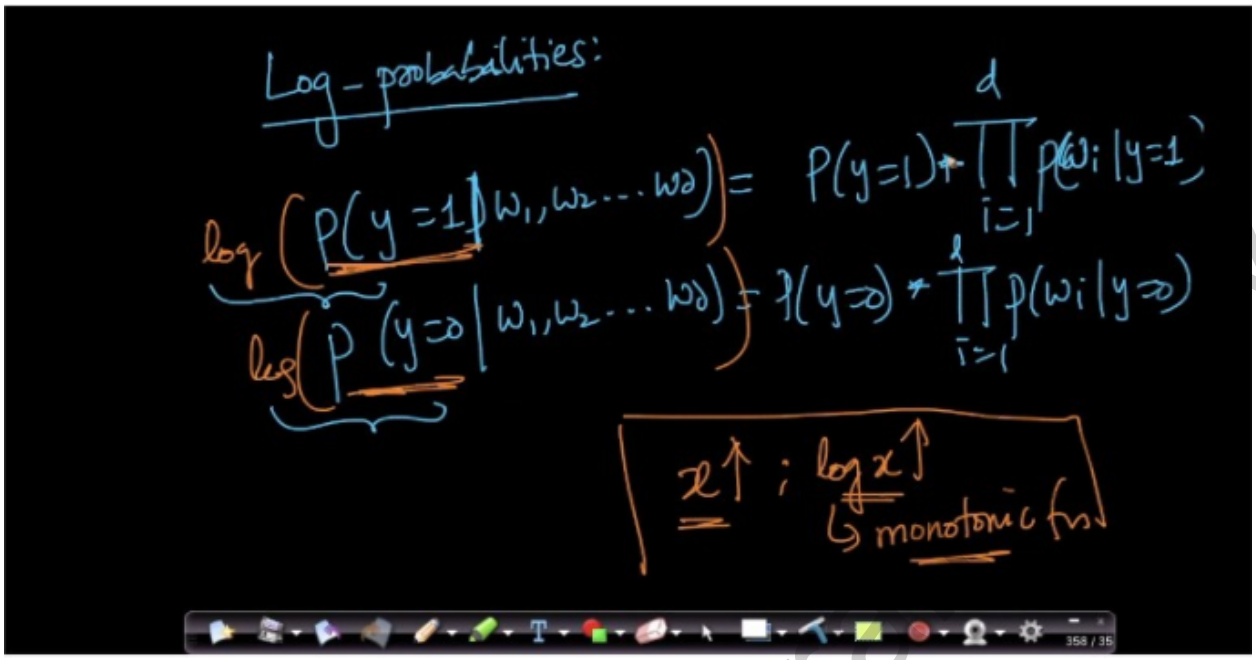
Timestamp 0:57

In order to calculate the probability we need to multiply the likelihoods of different features. Say we have a probability term p($y = 1 | w_1, w_2 ..... w_d$) as shown in the above image, where d is the number of features, which say is very large like 100. Also keep in mind that all the terms are probability terms so their value lie between 0 & 1.



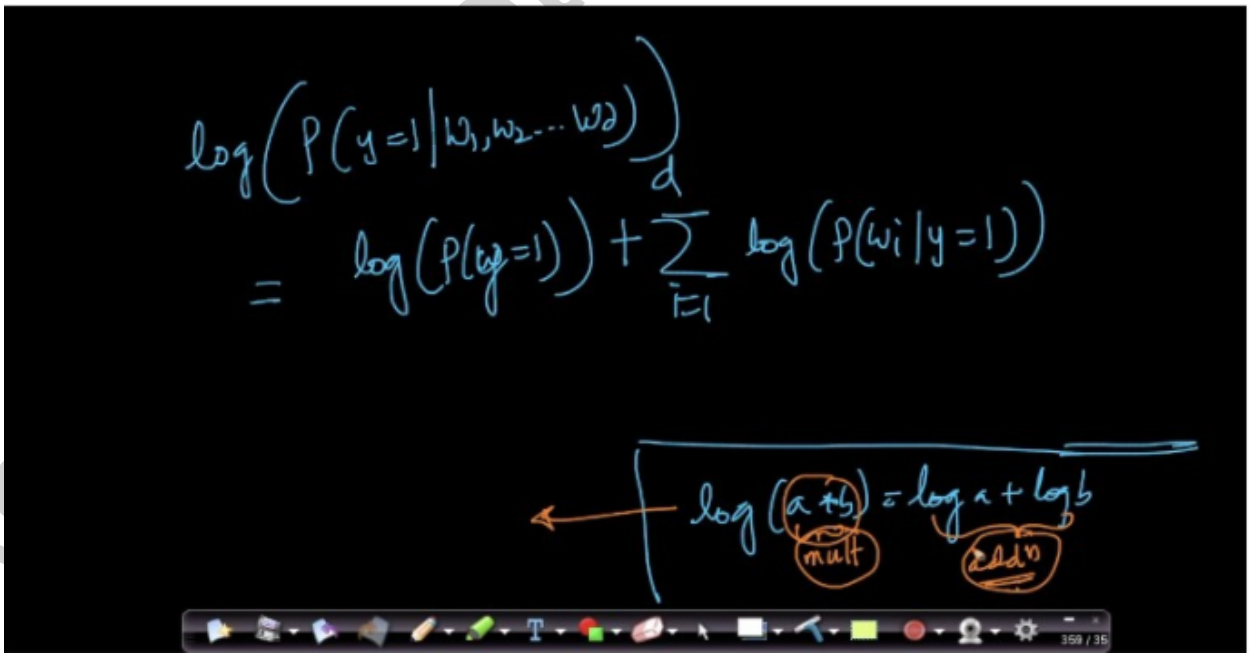
Timestamp 2:40

Suppose we take four probability values 0.2, 0.1, 0.2, 0.1. If we multiply these values we get 0.0004. This is a very small number and if we multiply 100 such numbers it can lead to issues of numerical stability. In python if we have a variable of double precision it can have upto 16 significant digits. If we get more significant digits than this, it starts rounding the number. This causes numerical underflow.



Timestamp 5:25

In order to deal with this numerical instability we use log probabilities. Log probabilities are nothing but we use log on either side of the equation. We can do this because log x is a monotonic function, as x increases log x also increases. If we take log(0.0004) = -3.39, which is better than dealing with high precision numbers. As this negative no. has lesser significant digits.



Timestamp 6:53

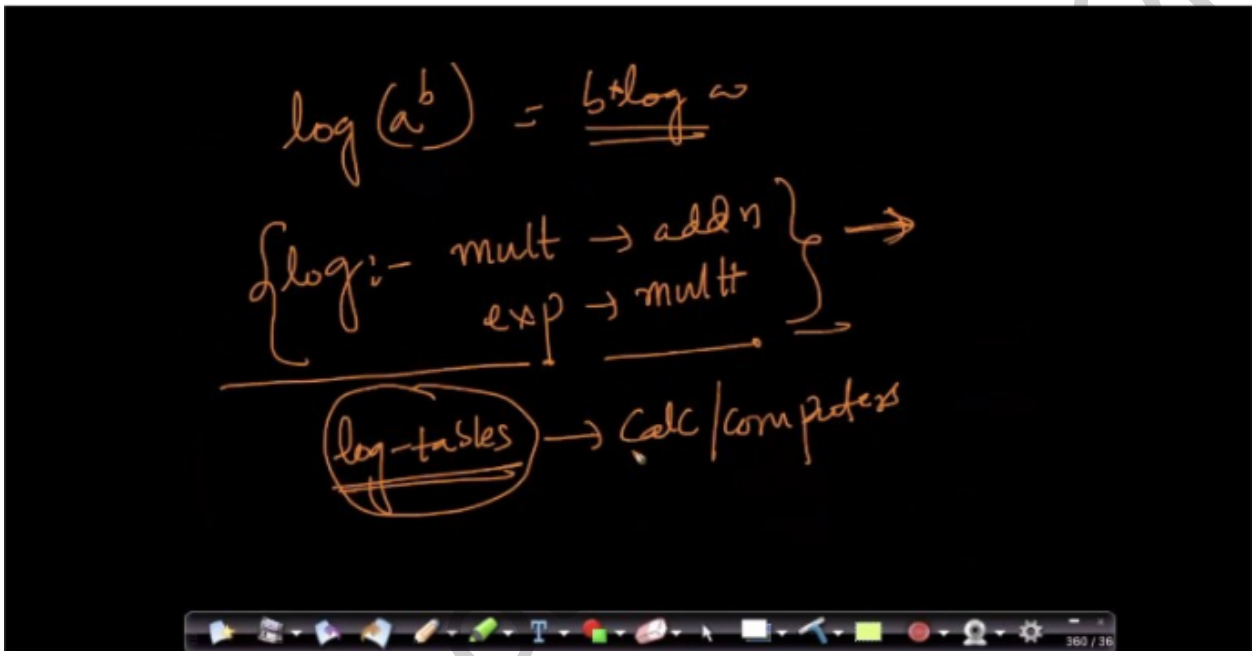
So, our probability equation is,

$p(y = 1 | w_1,w_2,w_3,\ldots,w_d) = p(y = 1) \prod_{i=1}^{d} p(w_i|y = 1)$

Taking log on both sides,

$\log(p(y = 1 | w_1,w_2,w_3,\ldots,w_d)) = \log(p(y =1)) + \sum_{i=1}^{d} p(w_i|y = 1)$

The product term is converted into a sum term because log(a*b) = log(a) + log(b). So we can now add log values without worrying about numerical instability.



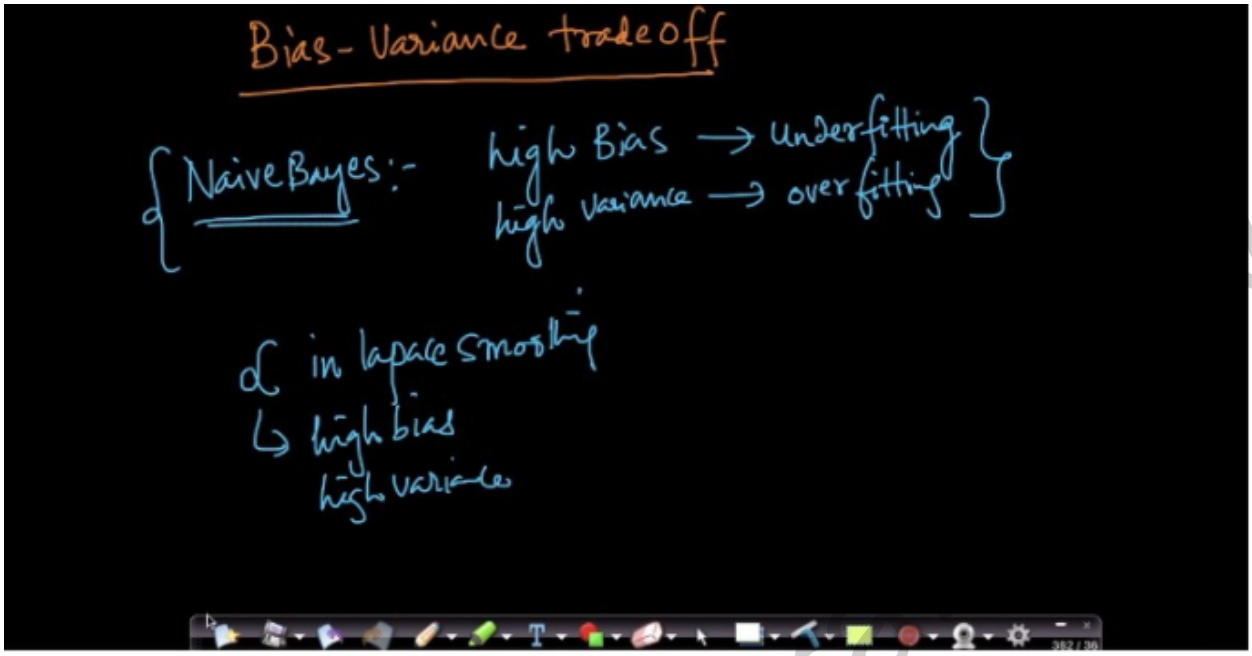
Timestamp 9:40

Logs are extremely useful because it converts multiplication in to addition
log(a*b) = log(a) + log(b),
It also converts exponentiation into multiplication
$\log(a^b)$ = b* log(a)

In earlier times we used to use log-tables for calculating these log - values.

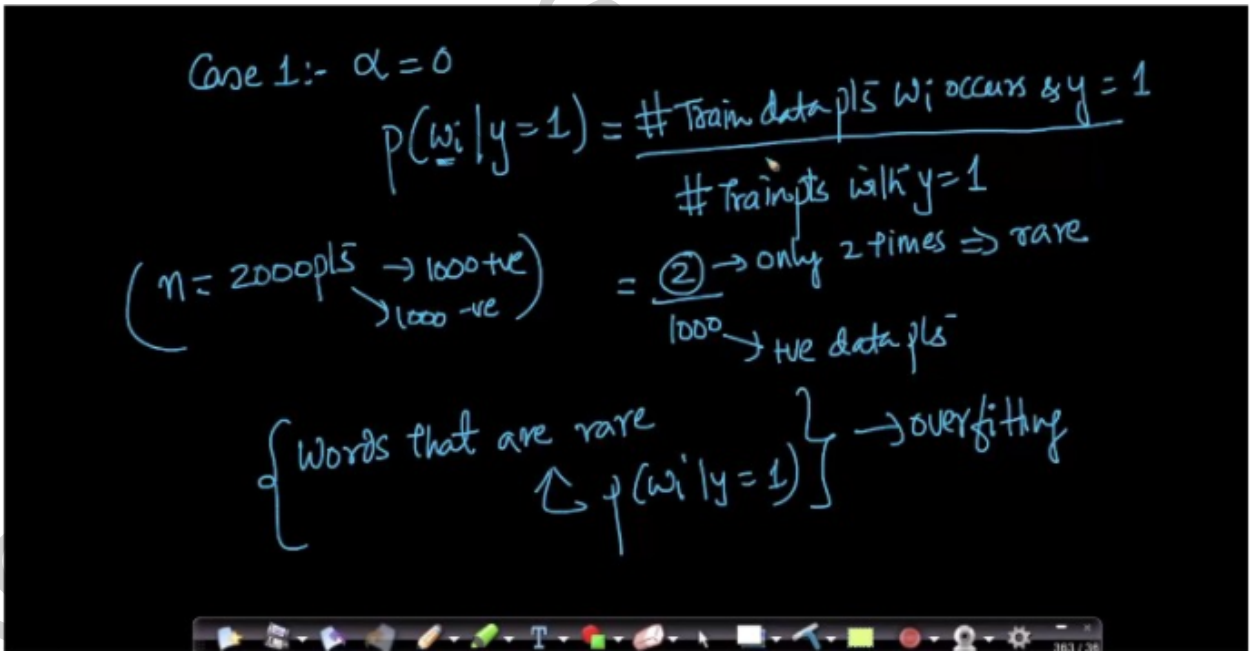
Log-probabilities help us prevent numerical instability because it's easier to deal with negative numbers than numbers of high precision.

32.6 Bias and Variance Tradeoff



Timestamp 0:50

In this chapter we will deal with the bias - variance tradeoff for NB.
Recall that high bias means underfitting and high variance means overfitting. In NB only α determines whether we have high bias or high variance.



Timestamp 3:55

Let's look at various cases indicating various values of α,

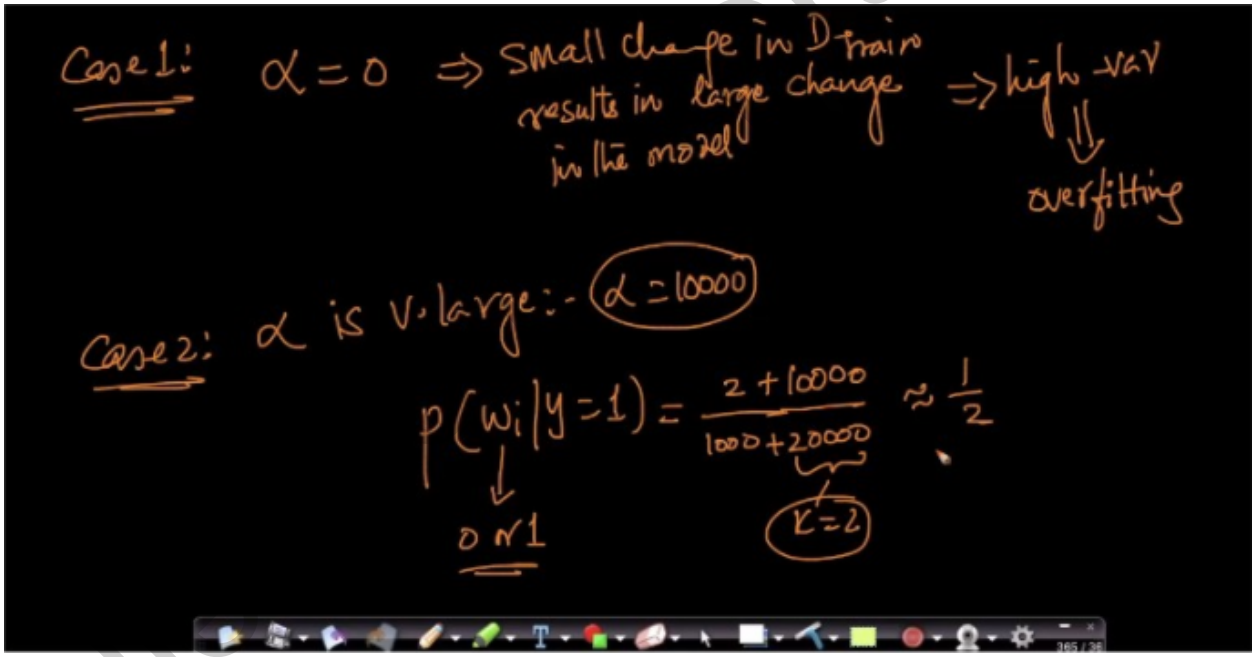
Case 1: $\alpha = 0$

Here we have considered the value of α to be 0. Let's say we want compute the likelihood of w_i; in our dataset we have 2000 data points, where 1000 = positive and 1000 = negative.

p(w_i| y = 1) = (#Train data pts which contains w_i and y = 1) / (#Train data pts with y = 1)

Say we got this probability value as = 2/1000. We can see that this particular word w_i is very rare as it occurs in only two positive documents and also note that we don't have any smoothing as α is set to 0. Now even for this word which is very rare we will have a probability value. If we change our model data i.e. train data slightly and remove these two texts where w_i occurs, these probability changes from 2/1000 to 0/1000. This means slight change in data is changing our model considerably indicating high variance. High variance means overfitting.

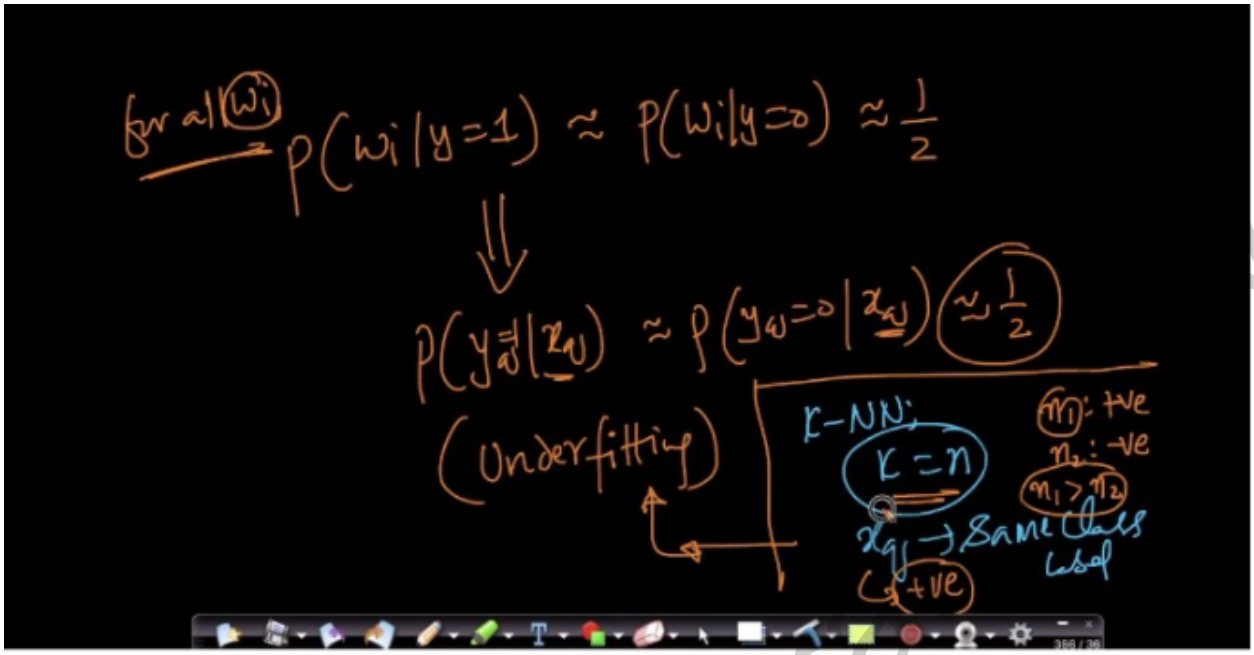
So lower α values means overfitting.



Timestamp 7:14

Case 2: α is very large say 10000

Again we calculate the probability of the word w_i, but this time with high smoothing. Also in our example k = 2.
So,
p(w_i| y = 1) = (2 + 10000) / (1000 + 20000) ≈ 1/2

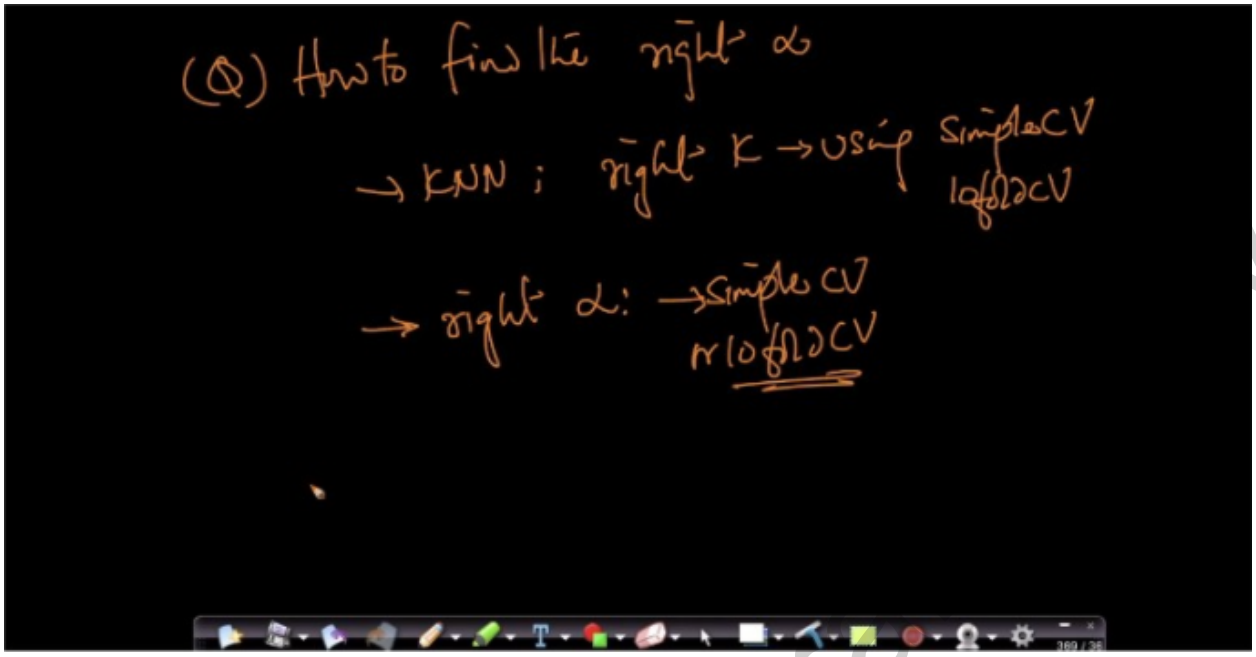


Timestamp 11:13

For every class we will get the same probability. As our α term dominates the probability values and less importance is given to actual probability values. Our actual prediction depends only only on the class priors. So, whichever class is present in the majority our model will always predict that class. This means we are underfitting.

So higher α values means underfitting.

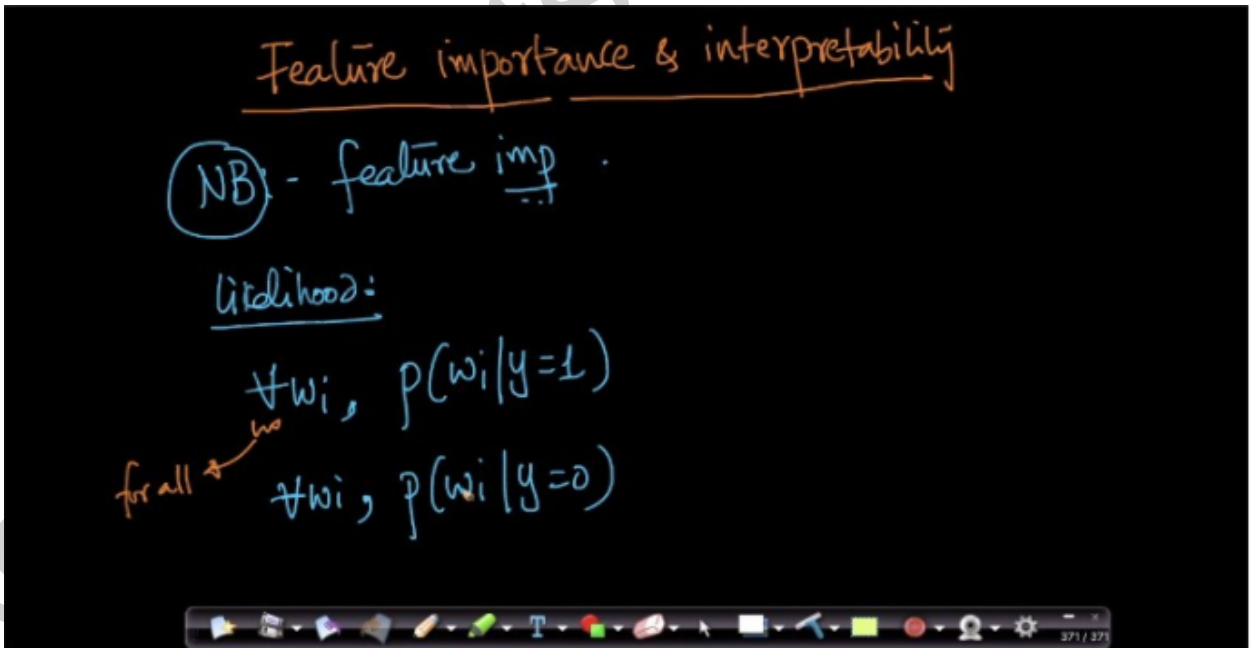
We can compare this situation with KNN, where we can set our neighbours K = n(#data points), then for every query we will get the same prediction. The prediction will be the class, which is present in majority in the train dataset. Leading to underfitting.



Timestamp 12:45

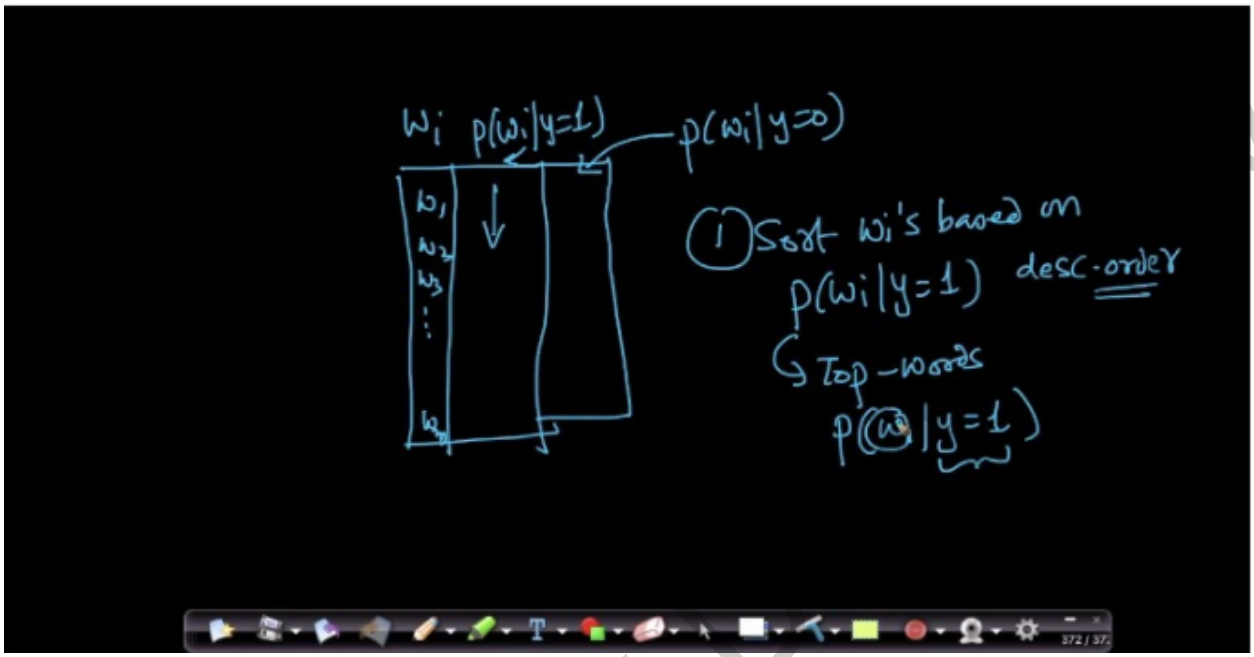
Now how to find the right α, we can find this using cross validation, like we did in KNN. α is called a hyperparameter. Similarly the value of k in KNN is also a hyperparameter.

32.7 Feature Importance and Interpretability



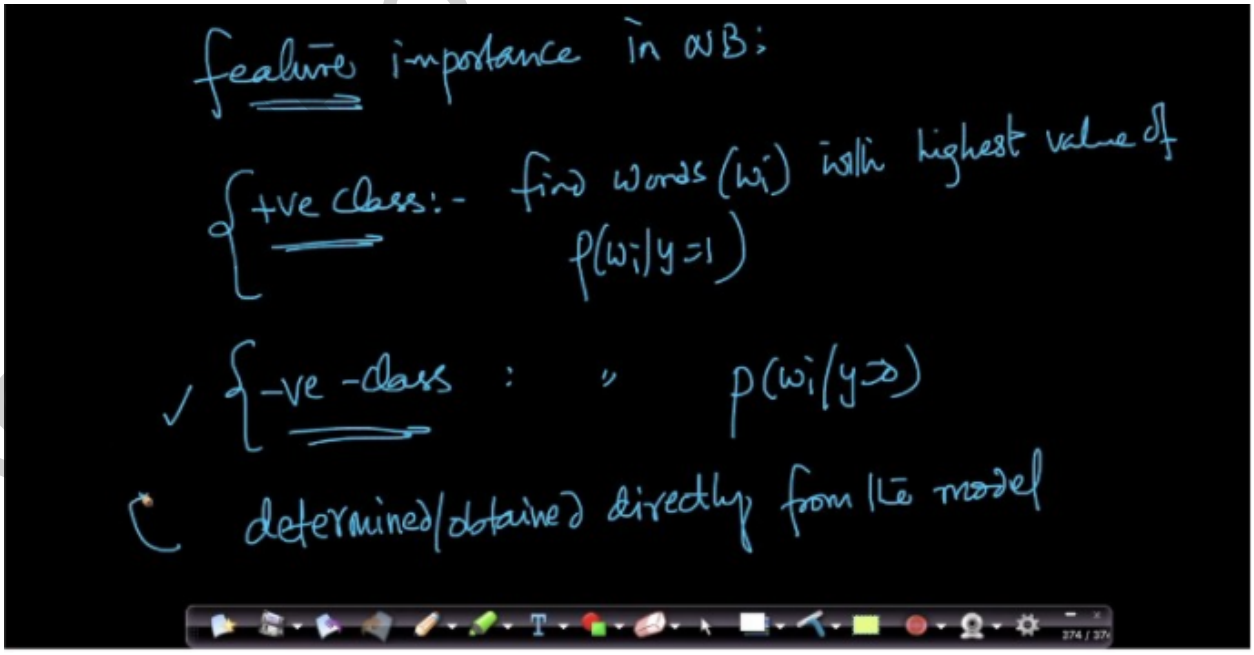
Timestamp 1:07

In this chapter we will try to find the feature importance and interpretability for a prediction. In order to find these we need to calculate the likelihood for all words.



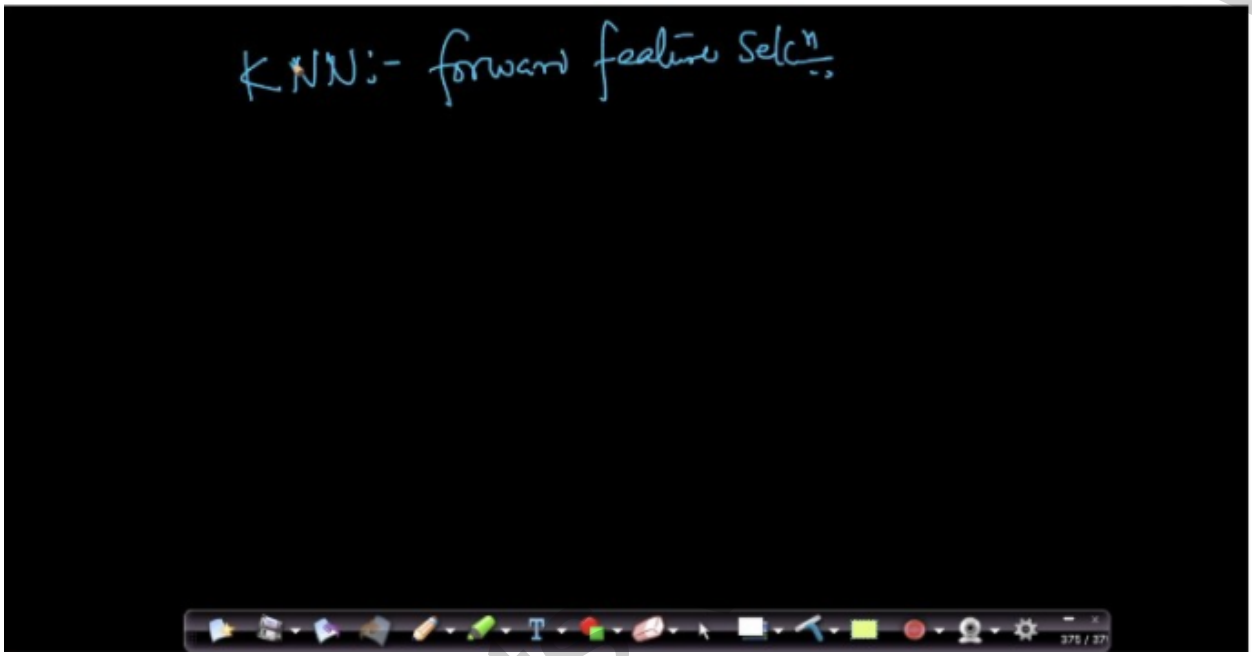
Timestamp 2:35

We will calculate the likelihood i.e. $p(w_i| y = 1)$ and $p(w_i| y = 0)$ for each word and store it into a table. Now we need to sort this table as per their probability values in descending order. Words which occur on top of this sorted table i.e. highest probabilities are the most frequent words in a particular class.



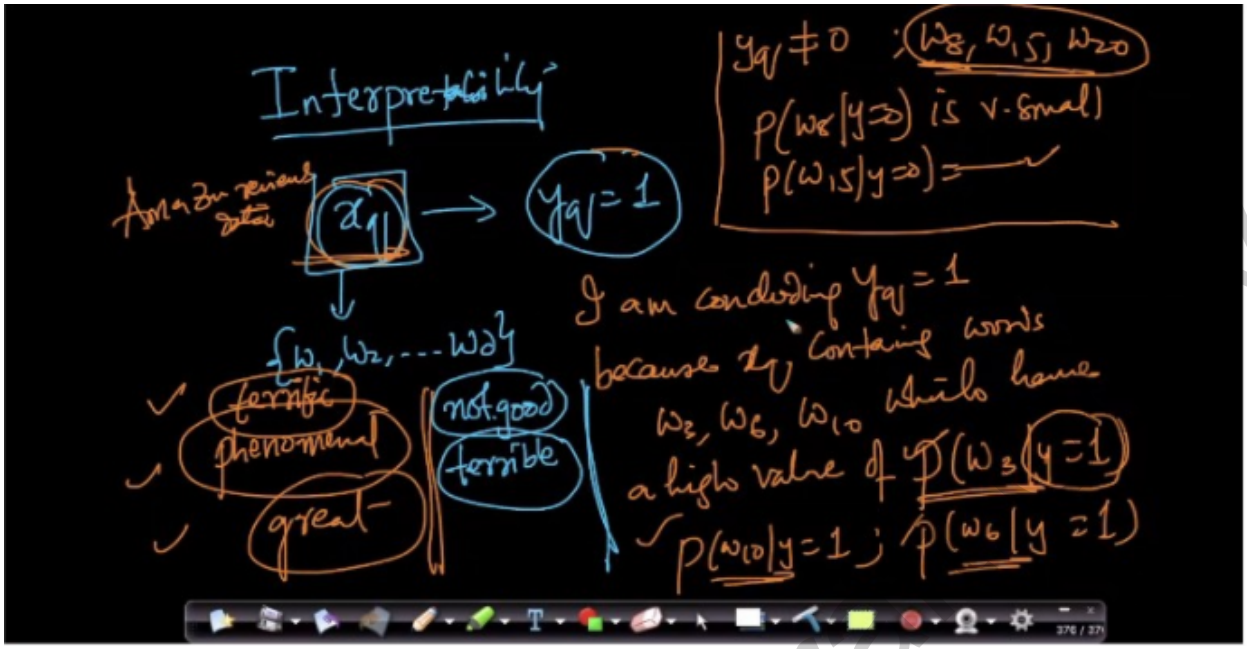
Timestamp 5:29

So features/words which possess the highest probability values are the most important features for a particular class. As when we compute the total probability, features/words which have the highest likelihood have the highest impact in total probability value.
Feature importance in case of NB can be directly computed from the model.



Timestamp 5:44

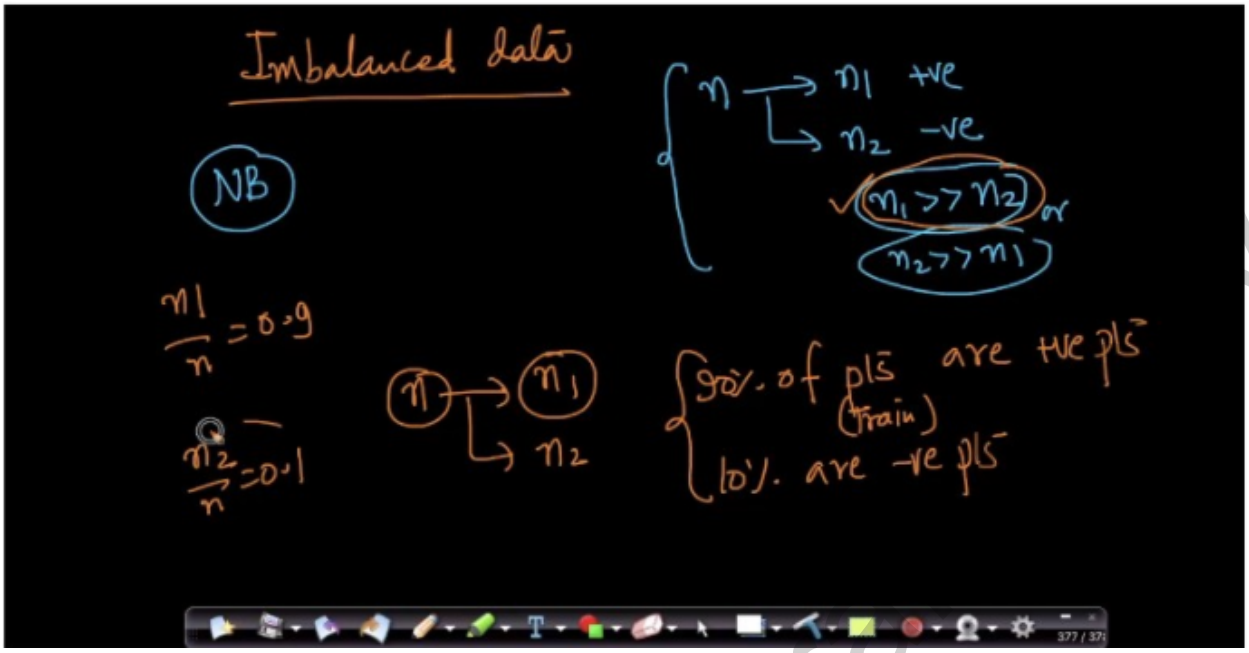
This is unlike KNN where we used techniques like forward and backward feature selection for determining the feature importance.



Timestamp 9:40

Now let's talk about interpretability, suppose we have a query point x_q,containing words { $w_1,w_2,w_3,\ldots,w_d$}, suppose we got y_q = 1. Now we can look at the individual probability terms and find those words which have the highest values, say we got {w_3,w_6,w_{10}}. From this we can conclude that we got the class label y_q = 1, because of the high likelihood of these words. These words determined our class label.

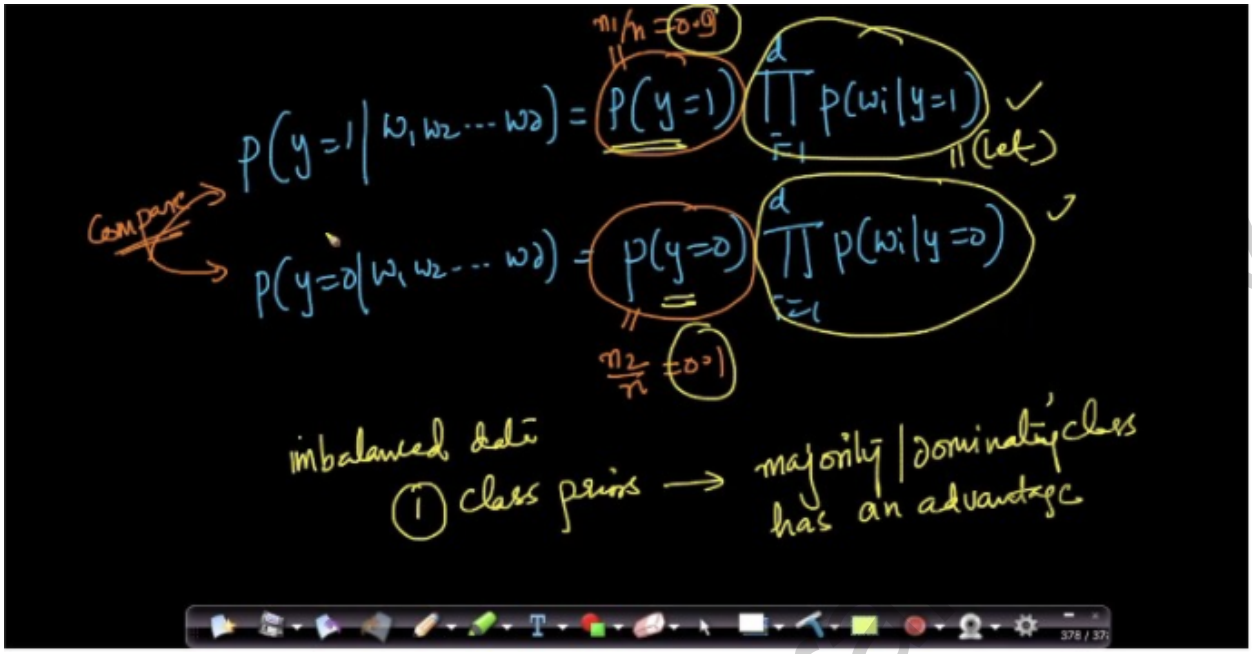
32.8 Imbalanced data



Timestamp 1:48

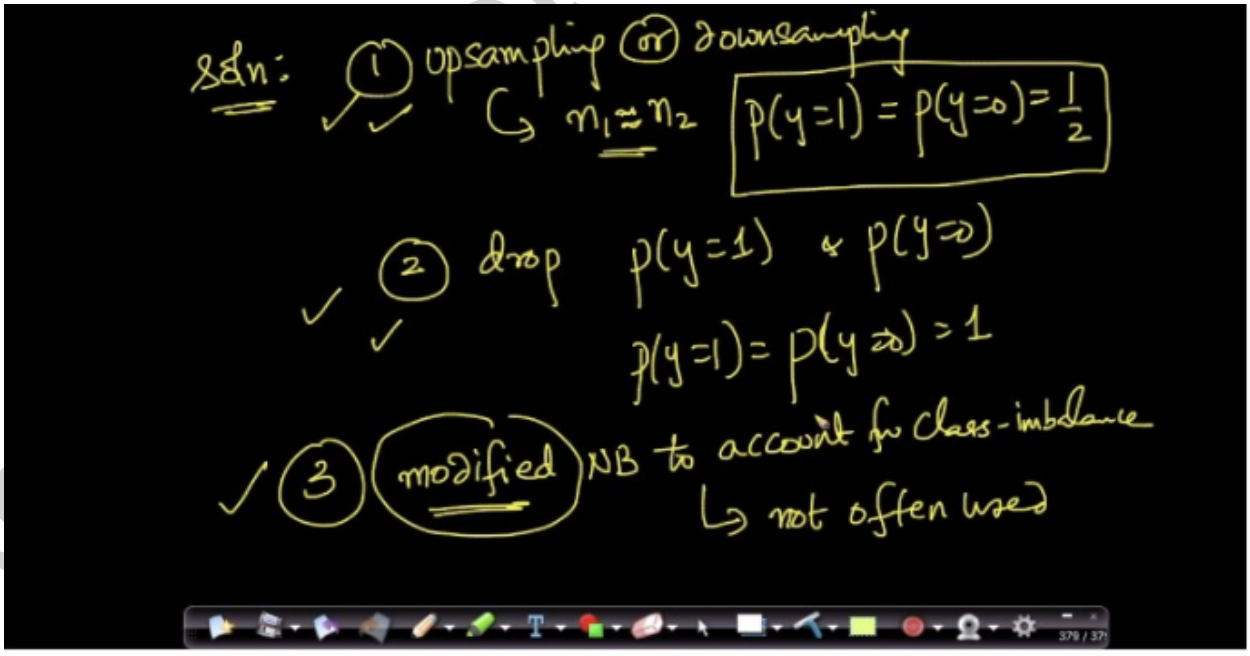
In this chapter we will look at the effect of an imbalanced dataset on Naive Bayes. Let's quickly define imbalanced dataset, say we have dataset containing n data points, where n_1 is the number of positive points and n_2 is the number of negative points. Now if n_1 is significantly greater than n_2 i.e. $n_1 \gg n_2$ or n_2 is significantly greater than n_1 i.e. $n_1 \ll n_2$, we call such a dataset as an imbalance dataset.

For our case let's assume that $n_1 \gg n_2$, say 90% of the data are positive and the remaining 10% are negative.
So, n_1/ n = 0.9 and n_2/ n = 0.1



Timestamp 3:35

If we calculate the probabilities for each of the classes, we need to calculate two terms: class priors and likelihoods. Suppose the likelihood is same for both the classes then we are left with priors. Prior for the majority class will dominate over the prior for the minority class. In our case the positive class will dominate over the minority class.
So, in case of an imbalanced dataset, the majority class has an undue advantage.



Timestamp 6:18

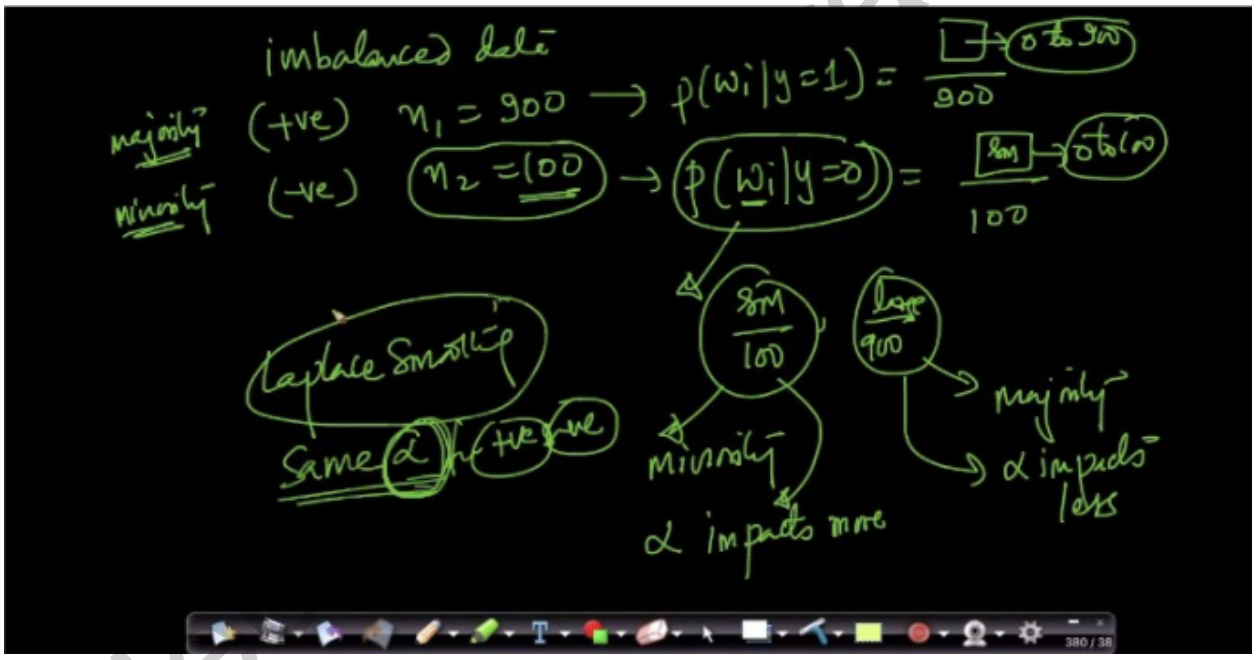
Here we will look at ways to handle imbalance,

1. Upsampling or Downsampling - We can either upsample the minority class or downsample the majority class to make both these class equal, i.e. $n_1 = n_2$.If we do this we are effectively making our class priors = 1/2

2. We can simply drop class priors. Dropping class priors effectively means setting them to 1. If we observe carefully both point 1 and 2 are equivalent as in both the cases we are setting the class priors to the same value. Since we only compare two probabilities it doesn't matter what priors we use as long as they are the same.

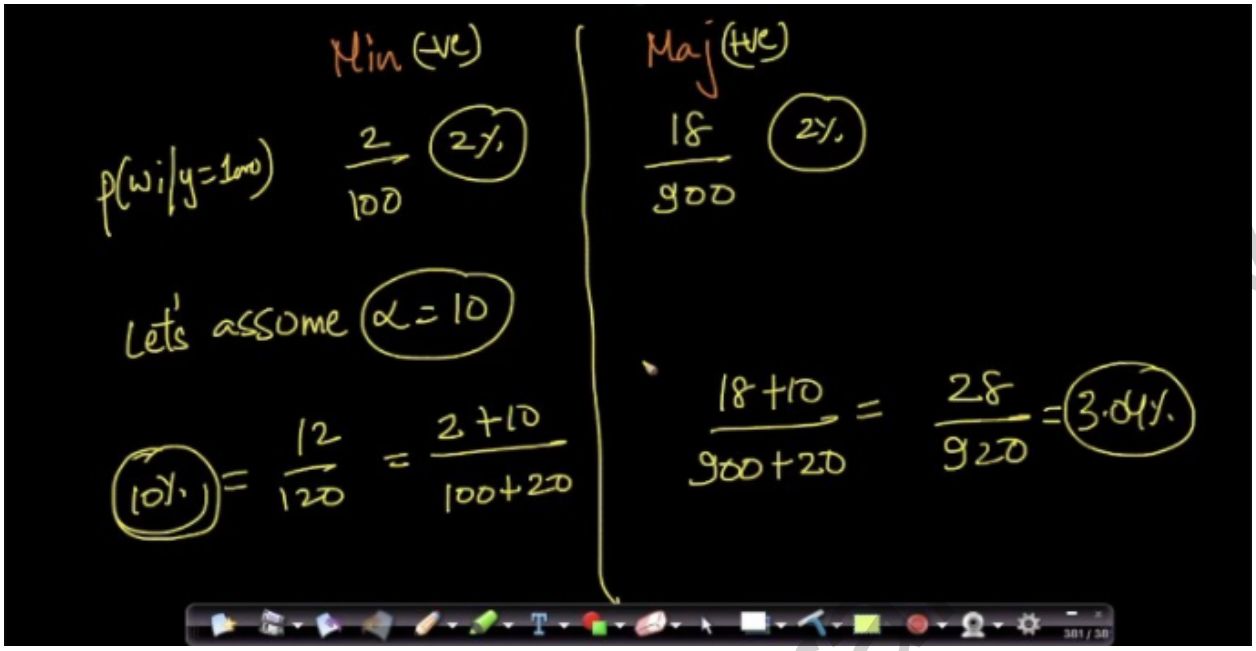
3. Modified NB, we can modify our NB to account for class imbalance, however it is seldom used.

Mostly point 1 and 2 are used in practice.



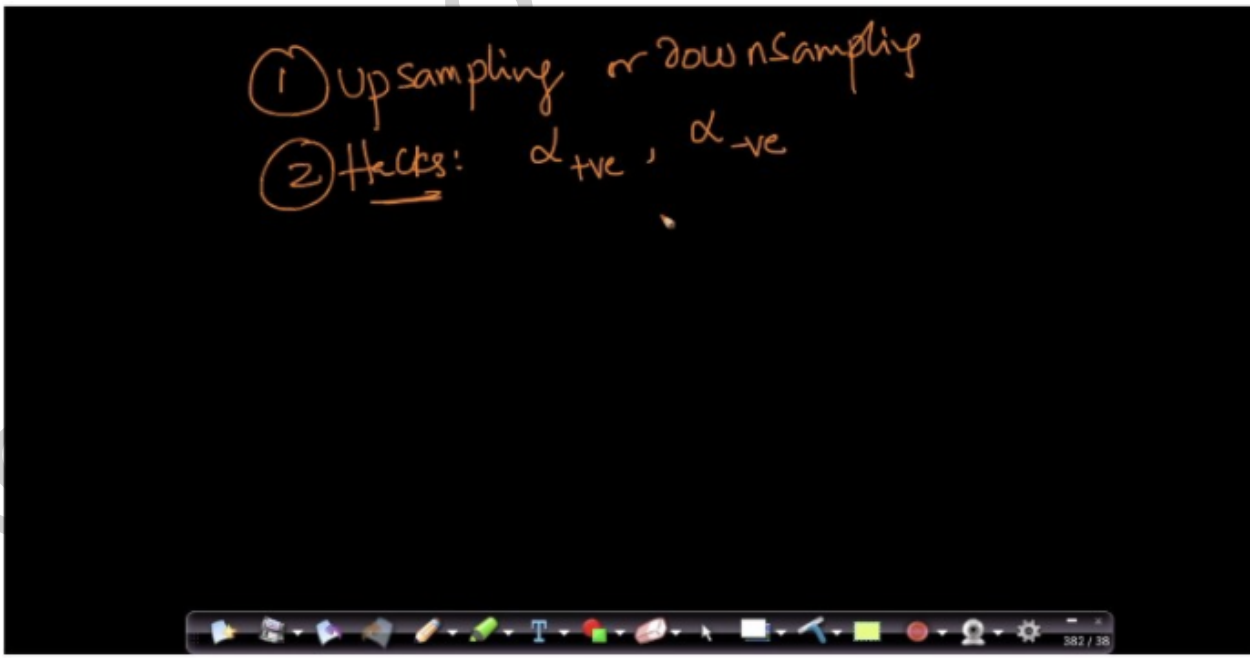
Timestamp 8:56

Here we will look at the effect of laplace smoothing in an imbalance dataset.
Let's say we have a dataset having 900 positive points and 100 negative points, here the positive class is the majority class. Now let's say we calculate the likelihood of the word w_i,for both these classes as shown in the image. If we apply laplace smoothing on both these two terms the minority class is impacted more than the majority class.



Timestamp 11:15

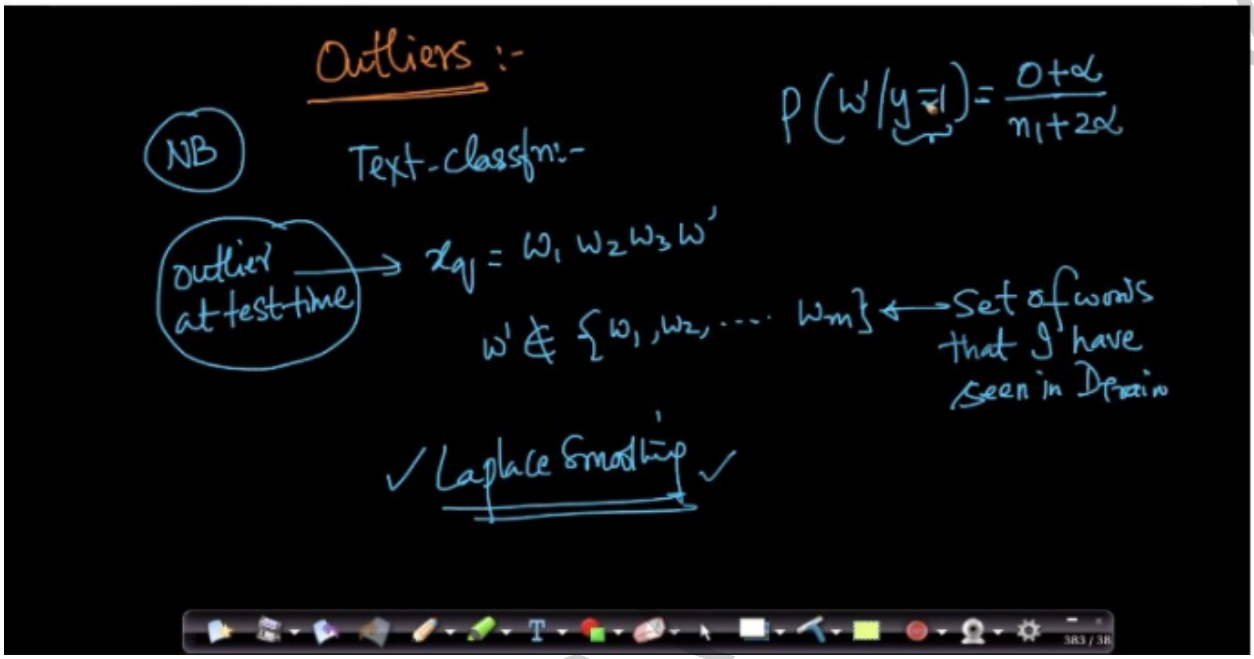
Let's see it with an example.
As we can see from the image above we calculated the probabilities for both the classes without using any smoothing. For both the majority and minority class we got the probability % to be 2%. Now after applying laplace smoothing with $\alpha = 2$, the probability % for the minority class shot up to 10% from 2% whereas for the majority class it went to 3.04%, showing that laplace smoothing affects the minority class more.



Timestamp 12:24

One of the hacks we can use to handle the impact of laplace smoothing is to use different α's for different classes. However this technique is not grounded in theory. It's better to follow techniques like upsampling/downsampling to handle class imbalance.

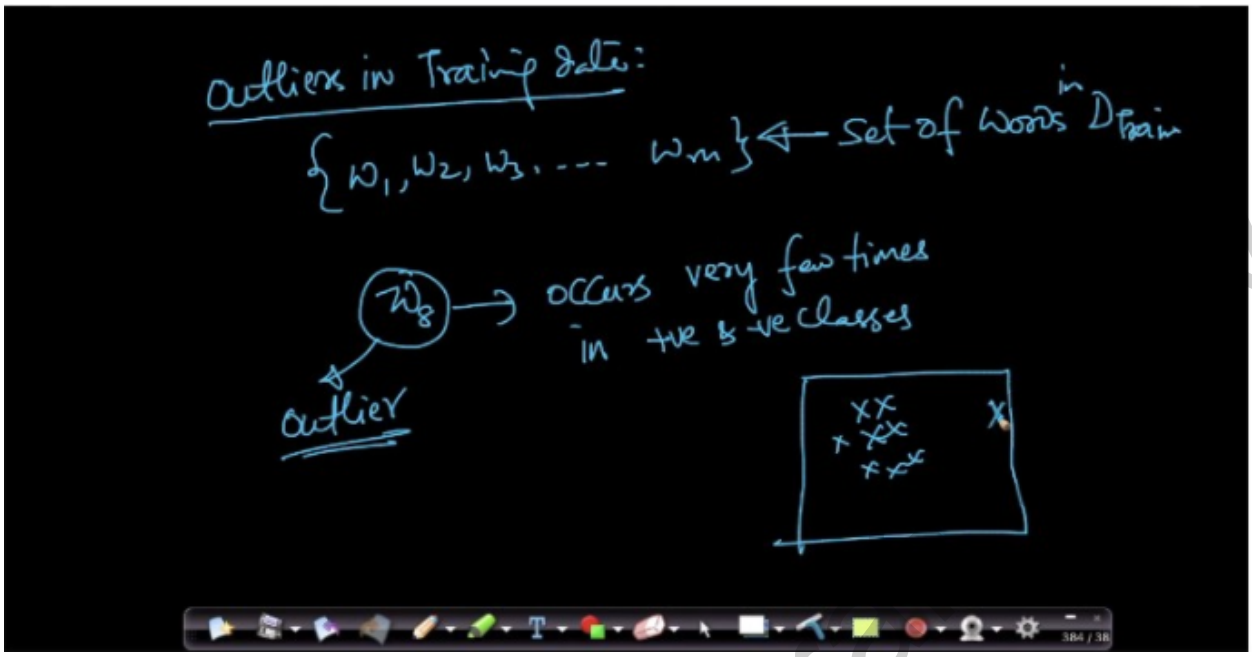
32.9 Outliers



Timestamp 1:42

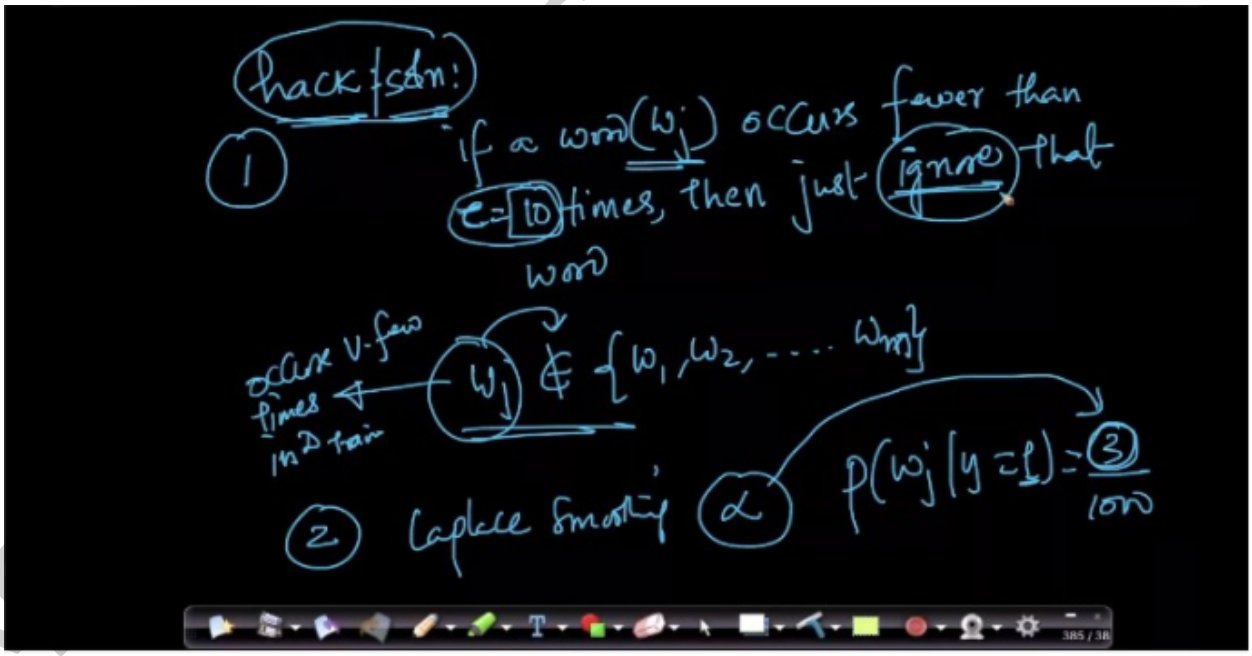
In this chapter we will see how to deal with an outlier.

Suppose we are doing text classification and during test time we see a word $w^{|}$, which was not present during train time. If we calculate the likelihood of this value we will get 0 making the whole probability 0. So in order to deal with this situation we use laplace smoothing which we learnt in earlier chapters.



Timestamp 2:54

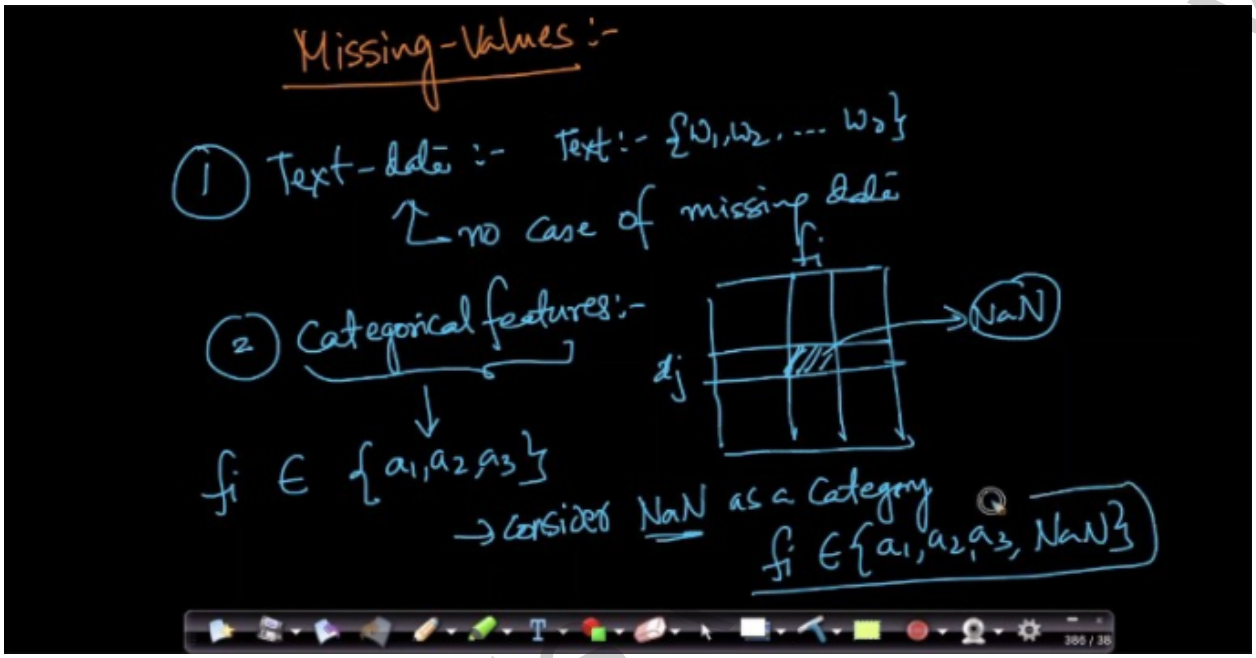
There can be outliers even in the training data. An outlier in training data can be defined as a word/feature which occurs very few times in training data. It is such a point around which there are no points.



Timestamp 5:06

We can deal with these problems in training data by simply ignoring the features/words. We can set a threshold and say if a particular feature occurs below a certain threshold we will ignore such words.
Another solution is to use laplace smoothing with an appropriate value of α.

32.10 Missing values

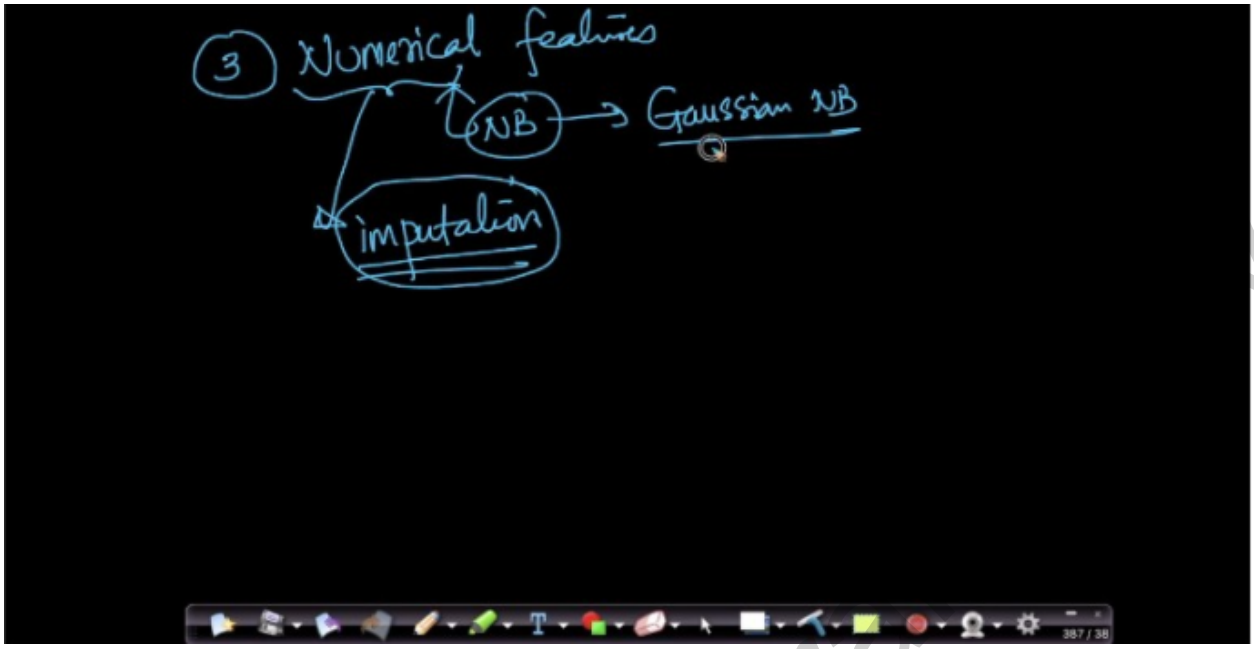


Timestamp 1:56

In this chapter we will talk about dealing with missing values.

1.Text data - If we are dealing with text data there is no case of missing data, either the word is present or not present.

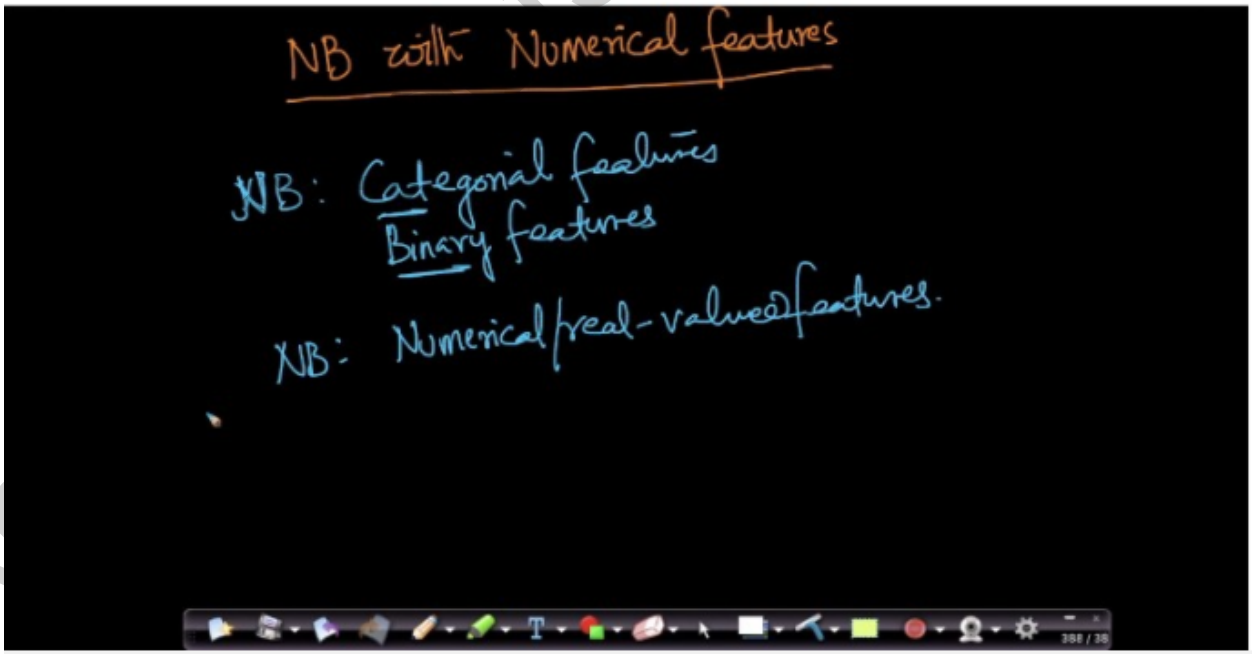
2. Categorical features - Say we have feature f_i, having categories { a_1,a_2,a_3}. Now suppose in one of the rows this feature value is missing. So we can consider the missing value NaN, itself as a feature. So, our feature set becomes { a_1,a_2,a_3, NaN}.This is one of the hacks that one can use. Also it is not limited to Naive Bayes but can be used anywhere.



Timestamp 2:55

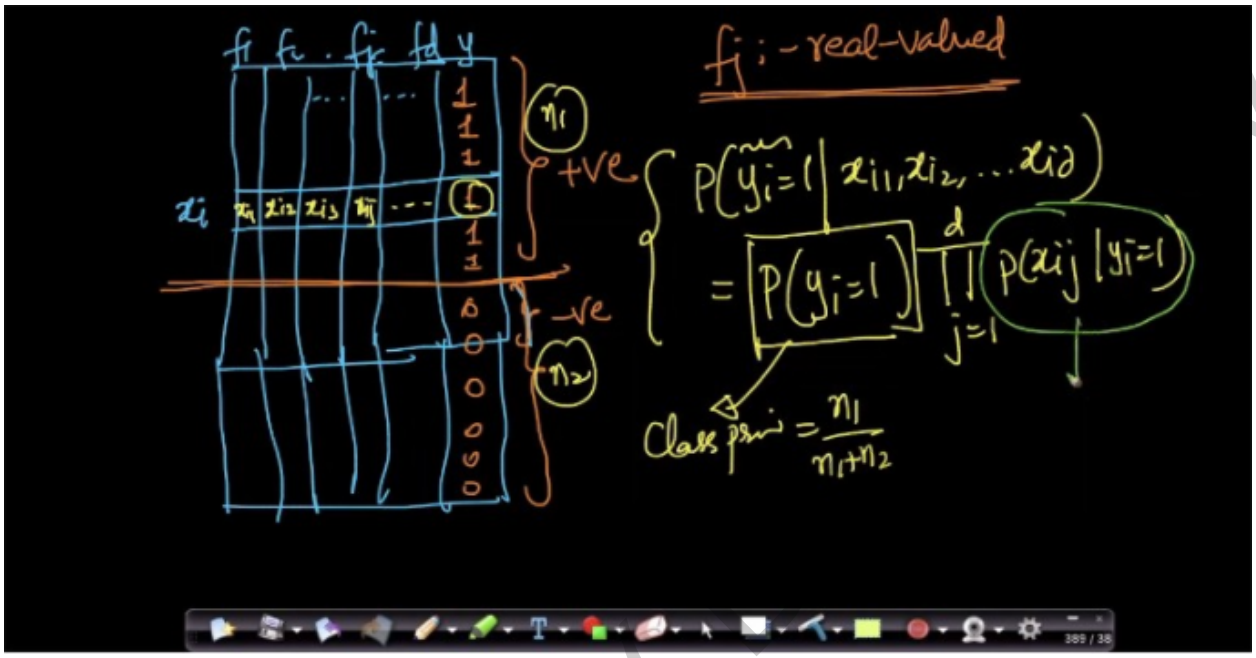
3. Numerical Features - We can handle missing numerical features by using various imputations like mean, median.etc.

32.11 Handling Numerical Features(Gaussian NB)



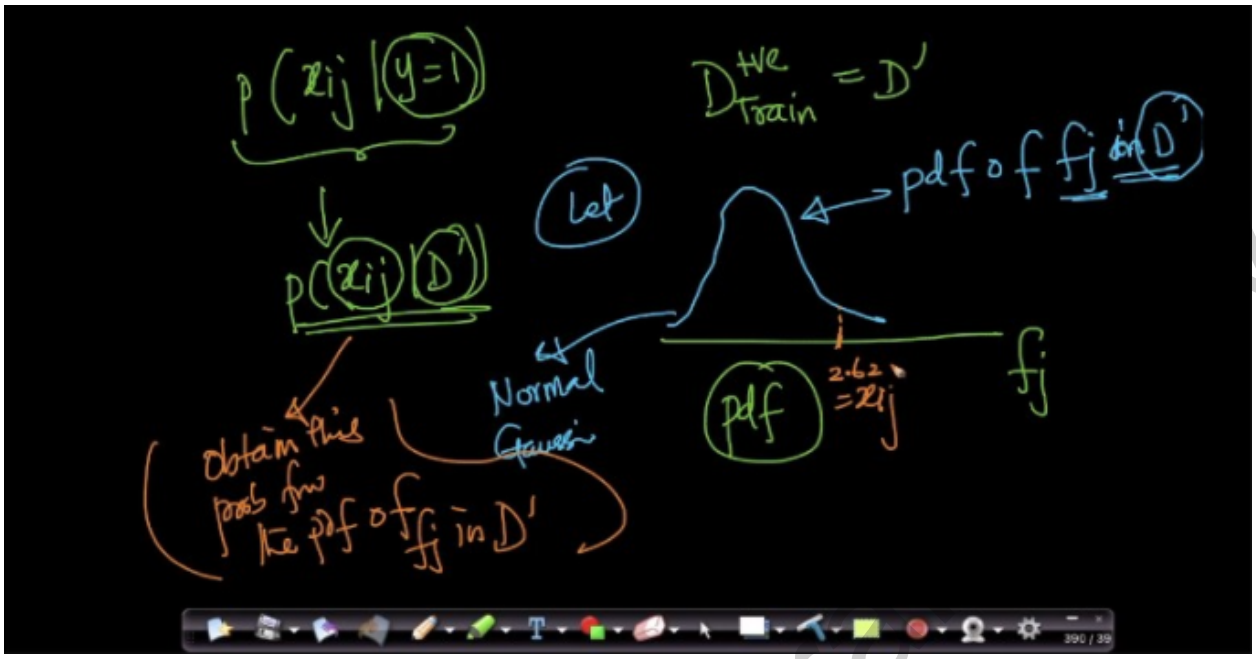
Timestamp 0:36

In this chapter we will learn how to deal with numerical(real valued) features in NB. Till now we have only worked with categorical and binary(words) features. In order to deal with numerical features we will use something known as Gaussian NB.



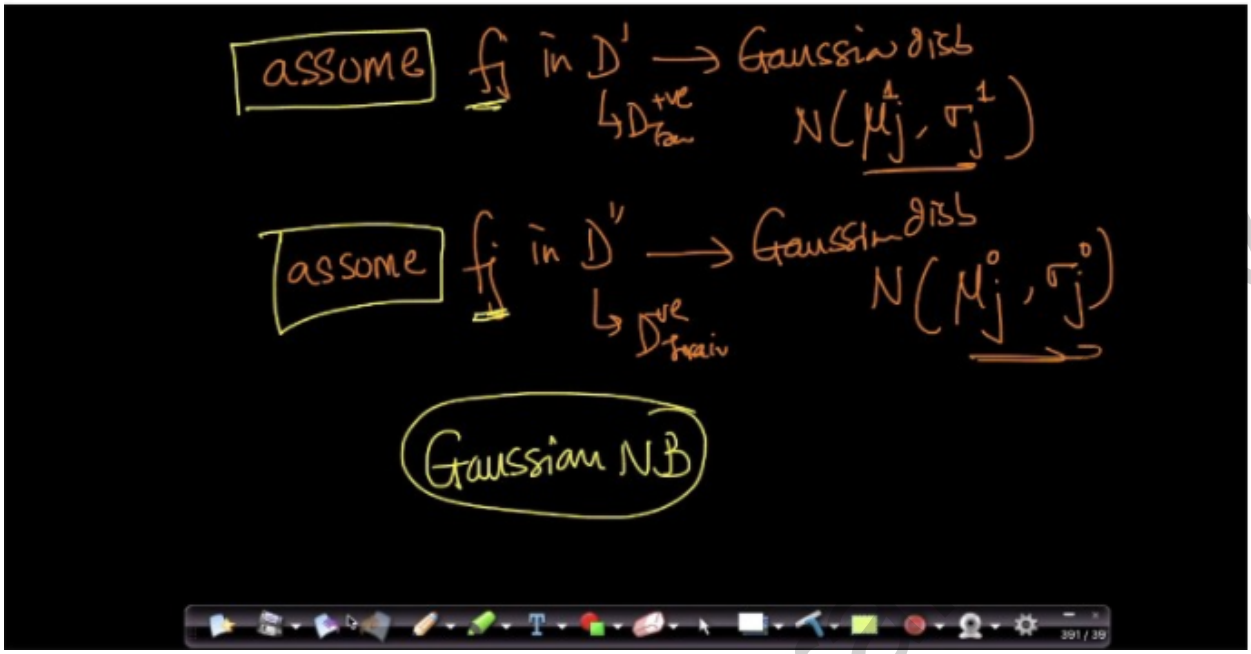
Timestamp 3:56

Consider a dataset where we have d features out of which say feature f_j is real valued. Now let's divide our dataset into groups one for all the positive labels(n_1) and another for all the negative labels(n_2). Consider a positive data point x_j, and it's jth feature i.e. x_{ij}, likelihood for this term can be written as p(x_{ij}| y_i= 1). Now how should we calculate this term?



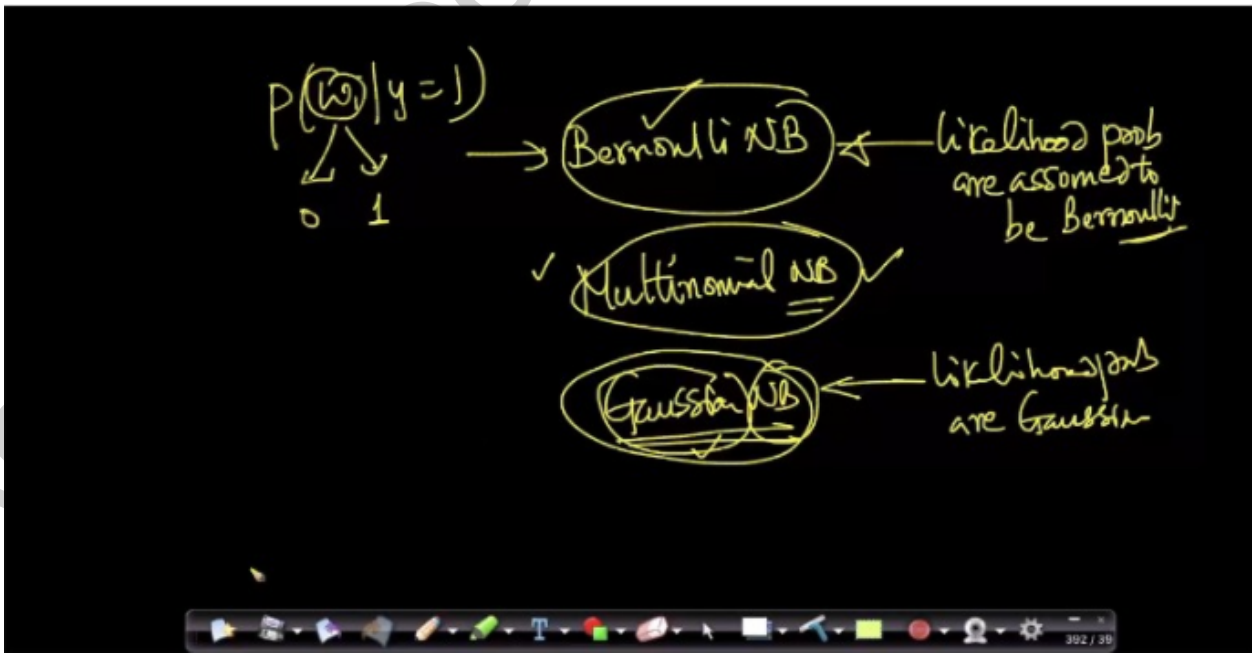
Timestamp 7:16

In order to calculate this term p(x_{ij}| y_i= 1), where x_{ij}corresponds to feature j of datapoint x_i, we will first take only the positive data points from the whole dataset, let's call this $D^{|}$. Now from all the data points of $D^{|}$, we take feature f_j,and draw a pdf corresponding to its values as shown in the image above. Now, in order to get a probability for any value say x_{ij}, we can calculate the pdf value from the pdf curve as shown in the image above.



Timestamp 10:08

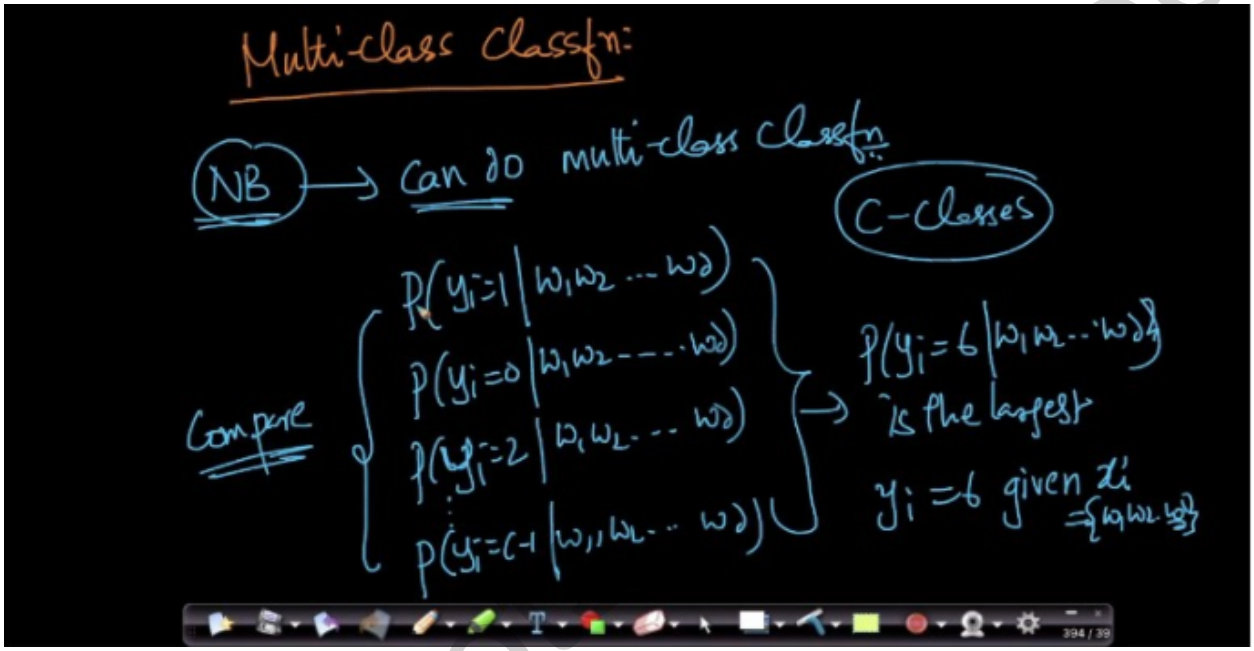
When dealing with numerical features we assume that these values follow a Gaussian Distribution. For the positive dataset $D^{|}$, we assume that the real valued feature f_j containing only the values from $D^{|}$follows a Gaussian Distribution with mean $\mu_j^1$and standard deviation σ_j^1, computed only from the values of the $D^{|}$. Similar steps can be followed for the negative dataset, as shown in the image above. This is called a Gaussian NB.



Timestamp 12:08

While dealing with words w_i, it could take two values either 0 or 1. It follows Bernoulli Distribution. It is also called Bernoulli NB. So, we can figure out what distribution a feature is following and based on the pdf/pmf of these distributions and we can calculate likelihood probabilities.

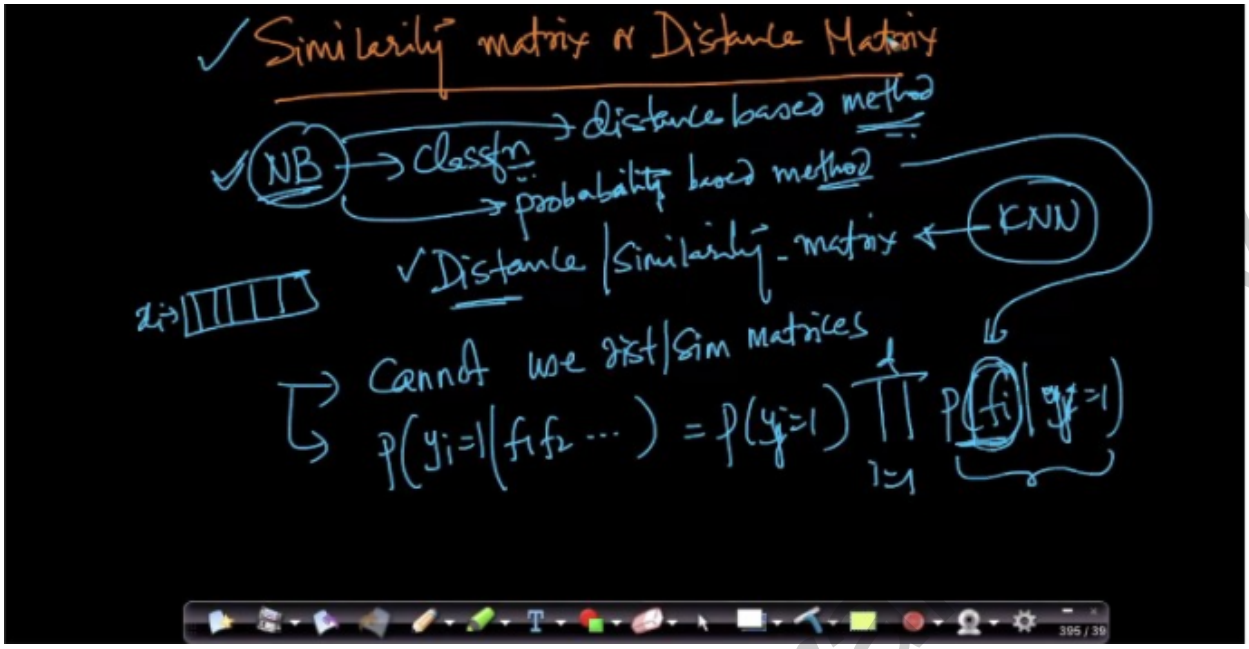
32.12 Multiclass classification



Timestamp 1:44

Naive Bayes can easily handle Multi Class Classification. We just need to calculate the probability for each of the classes, compare them and then select the class which has the highest probability.

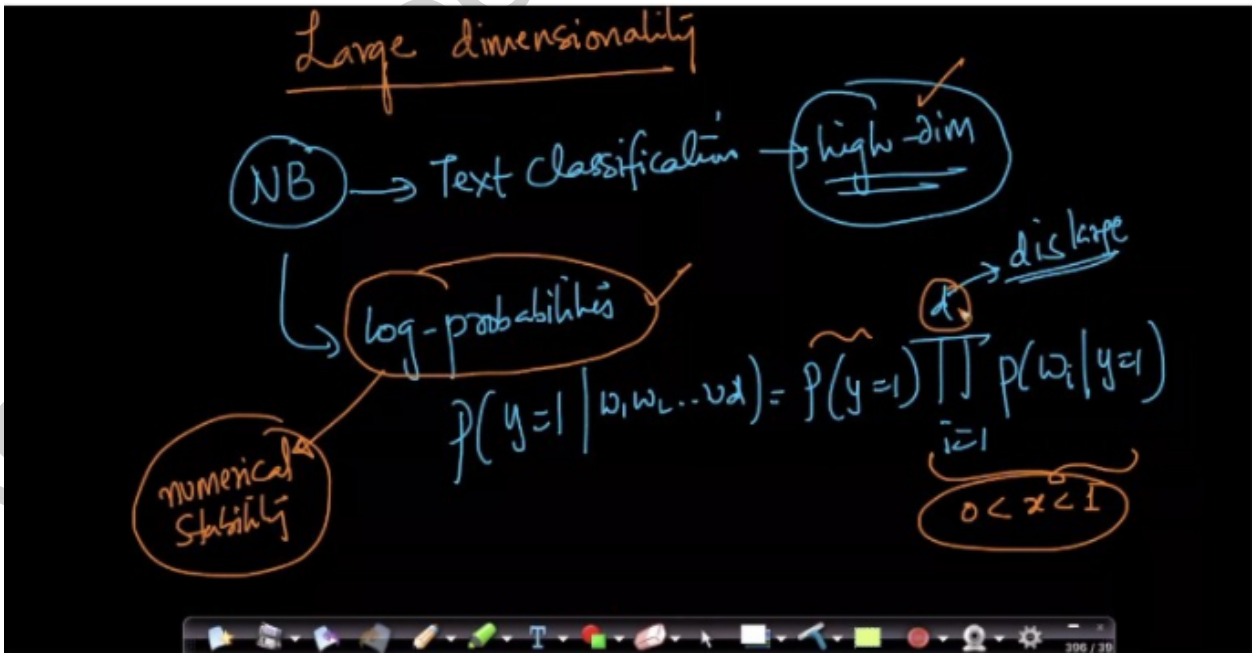
32.13 Similarity or Distance Matrix



Timestamp 2:15

Similarity or Distance matrix is usually used when it is difficult to represent a data point in the form of a vector, like we saw in KNN while doing pharma example. Naive Bayes cannot handle similarity or distance matrix in general, because it is not a distance based method rather a probability based method here we do not calculate the distance between two data points, rather we find out their likelihood probabilities.

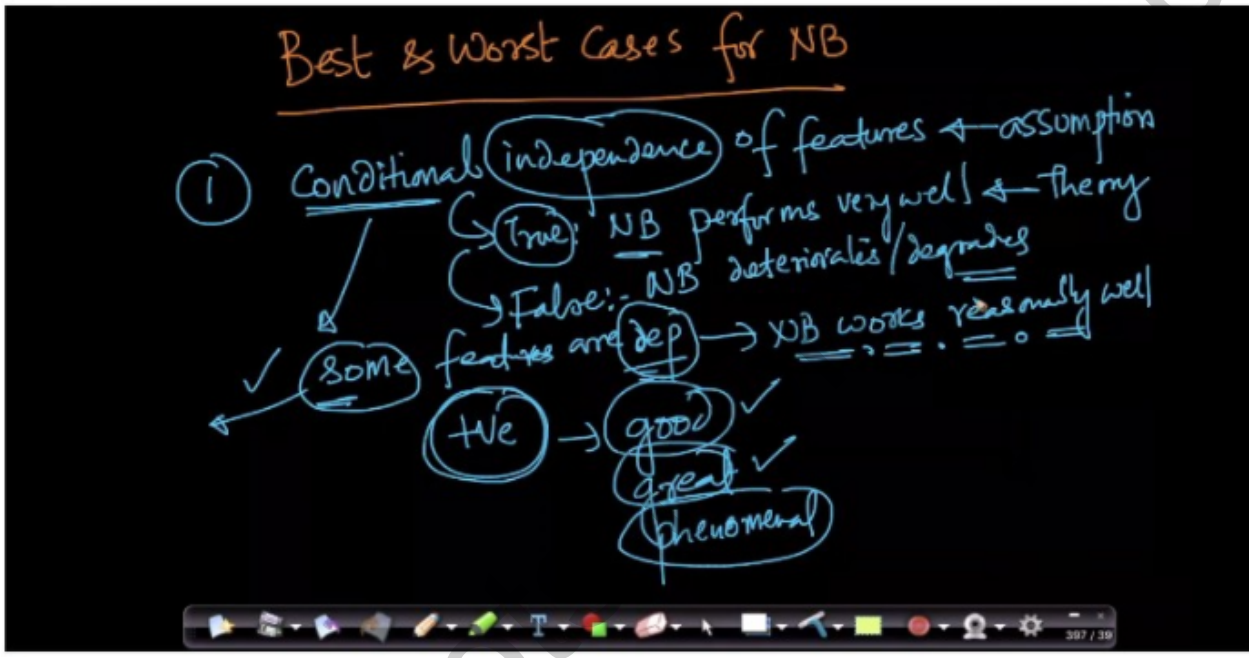
32.14 Large Dimensionality



Timestamp 1:40

Naive Bayes can easily handle data of large dimensions. While dealing with text classification we had text data which itself is of very high dimension. Only thing to keep in mind is to use log-probabilities to avoid numerical instability.

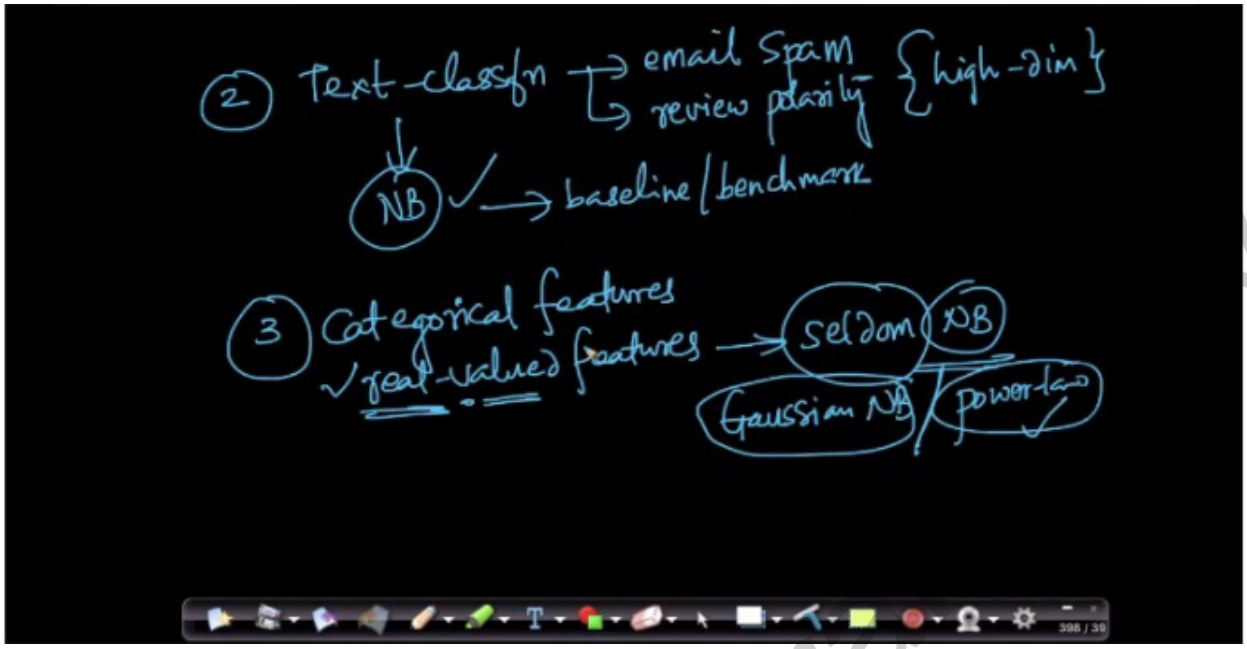
32.15 Best and worst cases



Timestamp 2:50

Now let's look at the cases where NB performs best and worst.

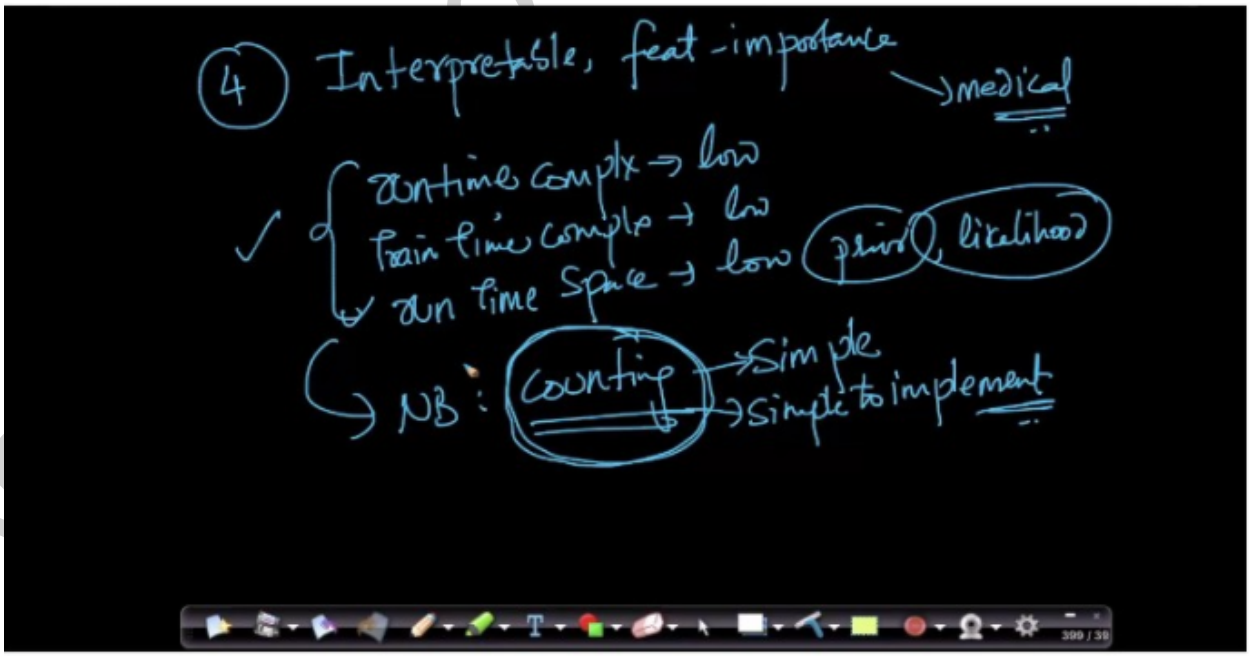
1.If the features are conditionally independent (the fundamental assumption of NB), then our model performs very well. Even if some of the features are slightly correlated then also our model performs reasonably well. Consider a positive case where words like good, great, phenomenal exist, even though these words are related our model will perform reasonably well in this situation. There are some mathematical proofs justifying this.



Timestamp 4:37

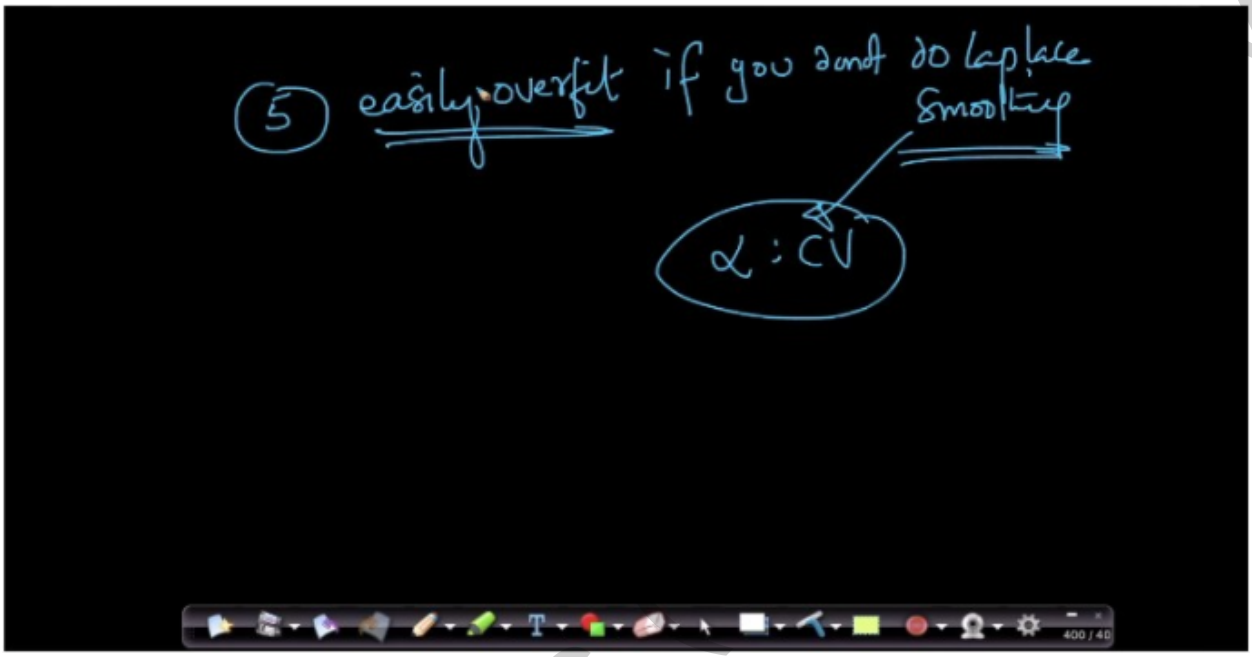
2. NB is extremely useful in dealing with high dimensional data as we saw during text classification. It serves as a good benchmark for problems like spam detection, polarity of a review,etc.

3. NB performs very well with categorical features. For real valued features we can use Gaussian NB.However it is seldom used at least in internet domains.



Timestamp 6:27

4. NB is highly interpretable. Feature importance can also be easily calculated. This makes it suitable for problems where interpretability is important like medical purposes. Also the time and space complexities for NB is very low when compared to KNN, which makes it useful in low latency solutions. It is also very easy to implement.



Timestamp 7:00

5. One of the disadvantages of NB is that it can easily overfit. In order to tackle this we need to do laplace smoothing. We can find the correct value of α using cross-validation.

32.16 Code example

We can see the scikit learn's implementation of Naive Bayes here =>

https://scikit-learn.org/stable/modules/naive_bayes.html