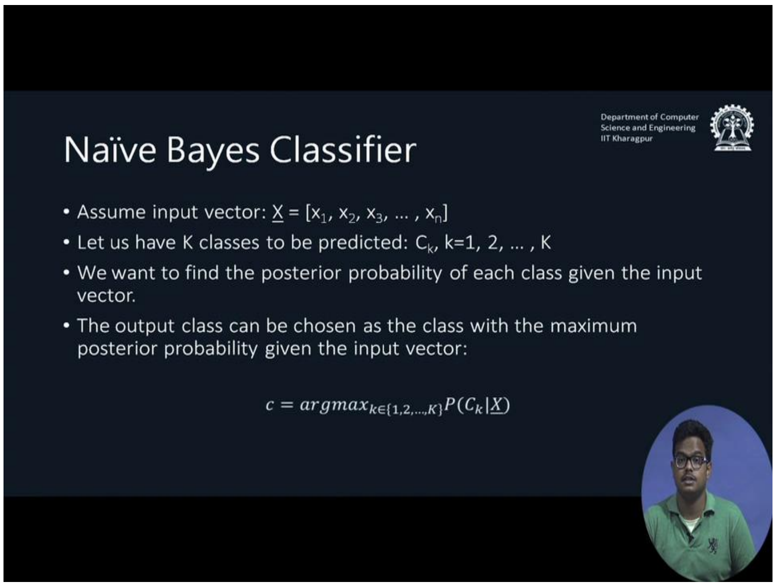
Introduction to Machine Learning

Prof. Mr. Anirban Santara
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 19
Python Exercise on Naive Bayes

Hello everyone. Welcome to the third to the hands on the session of the Introduction to Machine Learning course. I am Anirban Santhara, and I am doing my PhD in Machine Learning. In this session, we will learn how to use Naive Bayes classification algorithms for classification of emails as spam or non spam.

(Refer Slide Time: 00:37)



Naïve Bayes Classifier

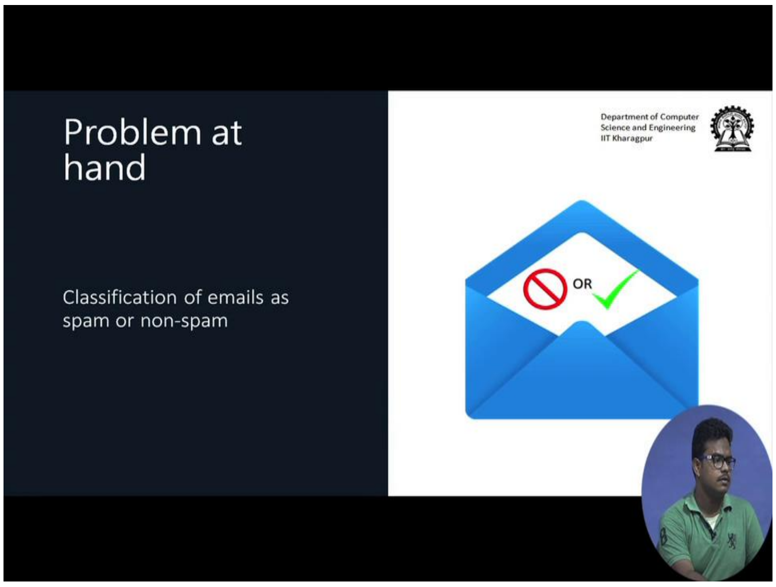
- Assume input vector: $\underline{X} = [x_1, x_2, x_3, \dots, x_n]$
- Let us have K classes to be predicted: C_k, k=1, 2, … , K
- We want to find the posterior probability of each class given the input vector.
- The output class can be chosen as the class with the maximum posterior probability given the input vector:

$$c = argmax_{k\in\{1,2,\dots,K\}} P(C_k|\underline{X})$$

Department of Computer Science and Engineering IIT Kharagpur

First, let us look back at what naive bayes classification algorithm looks like and how it works. Say we have an input vector N and it has n components and we have to predict one of k classes and we denote these classes as C k. Now, how do we go? We try to find which class has the highest posterior probability given the input feature vector X, and the class that has the highest probability is output as the target class for the example X.

(Refer Slide Time: 01:16)



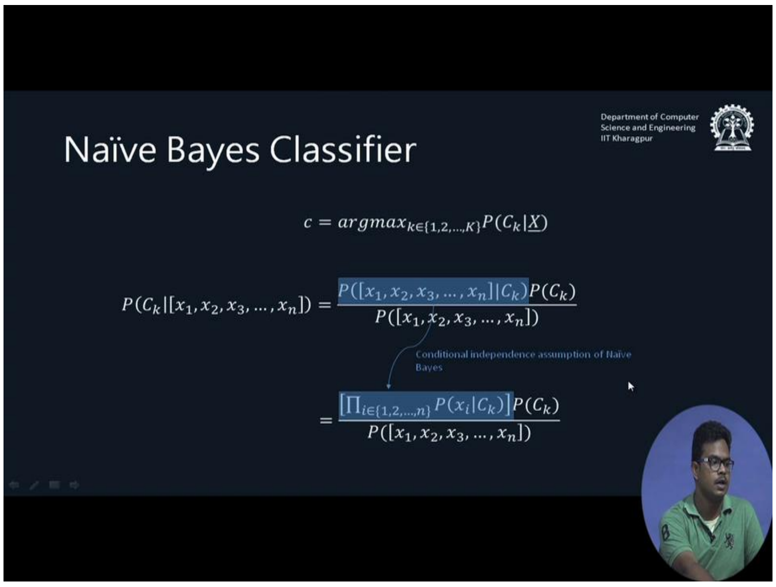
Problem at hand

Classification of emails as spam or non-spam

Department of Computer Science and Engineering IIT Kharagpur
OR

Now, the problem at hand is to classify emails as spam or non spam.

(Refer Slide Time: 01:22)



Naïve Bayes Classifier

$$c = argmax_{k \in \{1,2,\dots,K\}} P(C_k|\underline{X})$$

$$P(C_k|[x_1, x_2, x_3, \dots, x_n]) = \frac{P([x_1, x_2, x_3, \dots, x_n]|C_k)P(C_k)}{P([x_1, x_2, x_3, \dots, x_n])}$$

Conditional independence assumption of Naïve Bayes

$$= \frac{\left[\prod_{i \in \{1,2,\dots,n\}} P(x_i|C_k)\right]P(C_k)}{P([x_1, x_2, x_3, \dots, x_n])}$$

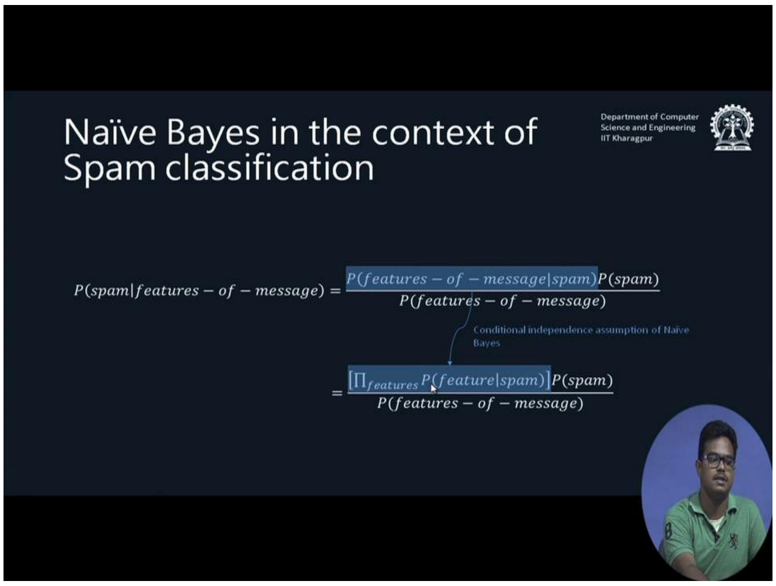
Department of Computer Science and Engineering IIT Kharagpur

And now let us see, how naive bayes classifier like how it really it would like works. So, C is the target class for the given example X, and this is going to be output as the class C k which is going to have the highest posterior probability given X, so that is given by the

Argmax function. It is going to check which value of k is giving the maximum P of C k given X. Now by base rule this can be broken up as, so I have expanded x as x 1 through x n the vector and this can be written as the likelihood of the vector x given the class C k times the prior probability of the class C k divided by the evidence of the vector X.

So, these are the terms from the base rule, and now the naive bayes classification algorithm. The naive bayes algorithm assumes that the input feature dimensions are conditionally independent given the class C k. So that is we could write this as the product over each x i given C k. This conditional independent assumption is what naive bayes algorithm is all about; naive bayes rule is all about. So this makes the classification algorithm the implementation of bit easier, the estimation a bit easier.

(Refer Slide Time: 02:56)



Naïve Bayes in the context of Spam classification

Department of Computer Science and Engineering IIT Kharagpur

$$P(spam|features-of-message) = \frac{P(features-of-message|spam)P(spam)}{P(features-of-message)}$$

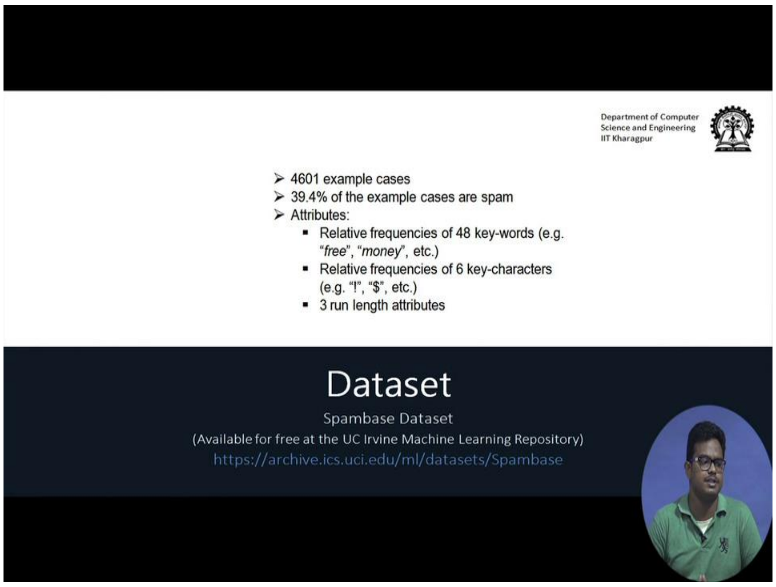
Conditional independence assumption of Naïve Bayes

$$= \frac{\left[\prod_{features} P(feature|spam)\right]P(spam)}{P(features-of-message)}$$

Now, how is this relevant in the context of spam classification? The problem at hand is to classify and email as spam or non spam given features of the email message. And you have seen a second what kind of features can be used to qualify and email which would make it distinctive for a class spam classifier.

So, the probability of spam given features of a message is what we have to estimate, and this is actually the posterior probability of the class spam. Now this can be decomposed by base rule as the likelihood of the features of the email message given then it was a spam message times the prior probability of spams divided by the features of the message, all right probability of the features of the message, and divide by naive bayes condition we can assume that the different features of the message are conditionally independent given that the class is already known to be spam. So this conditional is independence assumption is what naive bayes rule is all about.

(Refer Slide Time: 04:08)



Department of Computer Science and Engineering IIT Kharagpur

- 4601 example cases
- 39.4% of the example cases are spam
- Attributes:
  - Relative frequencies of 48 key-words (e.g. "*free*", "*money*", etc.)
  - Relative frequencies of 6 key-characters (e.g. "!", "$", etc.)
  - 3 run length attributes

Dataset

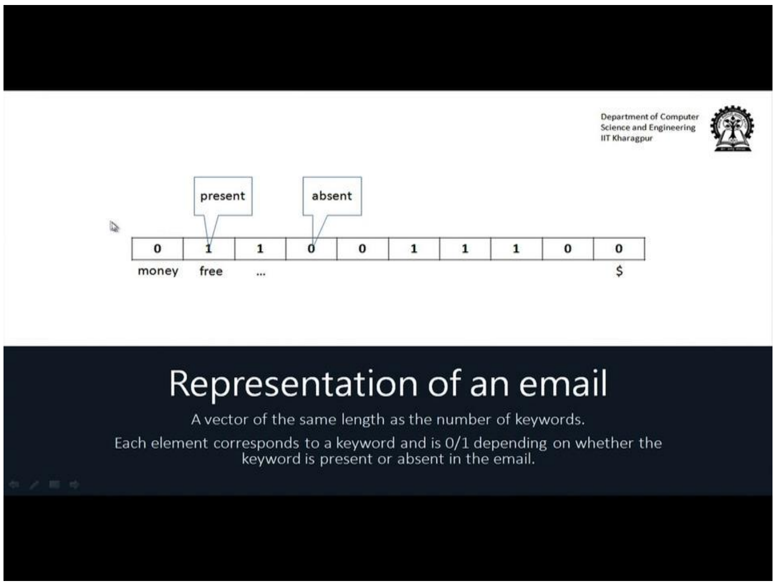
Spambase Dataset

(Available for free at the UC Irvine Machine Learning Repository)

https://archive.ics.uci.edu/ml/datasets/Spambase

On the data is that we will be using is an open sourced data set from the UC Irvine Machine Learning Repository. It is available for free in the URL that you can see on this screen. It has 4601 example cases, 39.4 percent of that a spam, the rest are non-spam. And the each email is represented as relative frequencies and see how this is quantified. The related frequencies of 48 key-words and these key words are like, "free", "money", these are the key words that appear very often in spam emails, so that increases the likelihood of the email to be as being a spam message. And also some characters like exclamation dollar these, so 6 such characters and 3 run length attributes. So we would not be consider the run in the attributes and we will be concentrating only on the first 54 features.

(Refer Slide Time: 05:09)


Department of Computer Science and Engineering IIT Kharagpur
present
absent
0 1 1 0 0 1 1 1 0 0
money
free
...
$

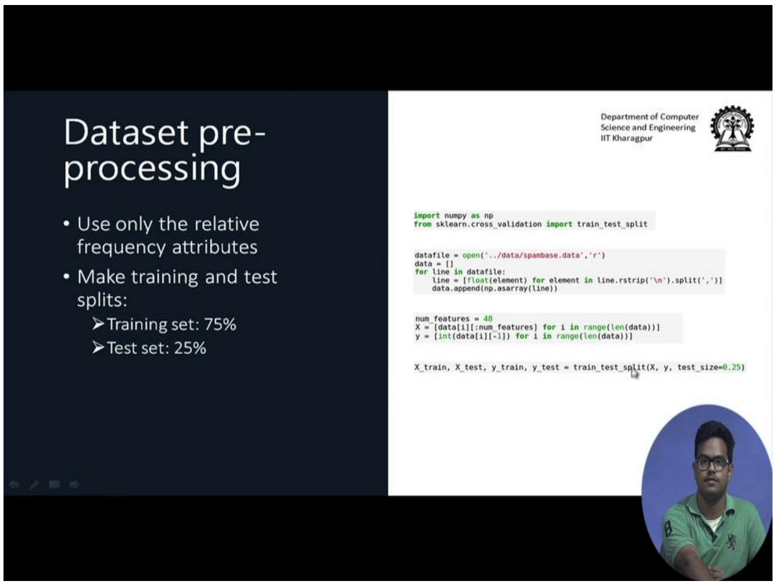
Representation of an email

A vector of the same length as the number of keywords.

Each element corresponds to a keyword and is 0/1 depending on whether the keyword is present or absent in the email.

The representation of an email is important, because that is what would make naive bayes to be applied in a proper way. So, how do we represent an email message? We represent it in a form which is called a Document Vector. It is a vector and each element of the vector is an indicator of whether a particular word is present or absent. Say for example, the first position represents money, the first element represents whether the word money is present or not. And, the second one represents whether the word free is present or not. And that is how we get a binary vector of size the number of features that we want to choose.

(Refer Slide Time: 05:57)



Dataset pre-processing

- Use only the relative frequency attributes
- Make training and test splits:
  - ➢Training set: 75%
  - ➢Test set: 25%

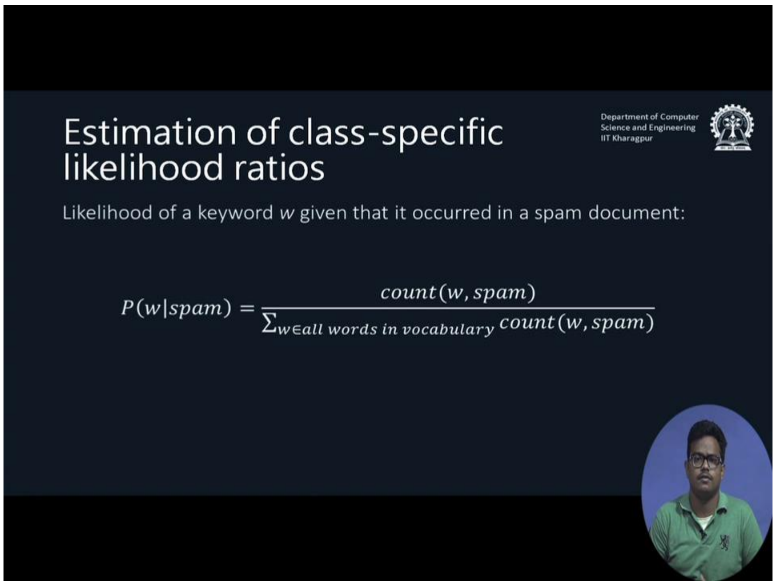
Department of Computer Science and Engineering IIT Kharagpur

```python
import numpy as np
from sklearn.cross_validation import train_test_split

datafile = open('../data/spambase.data','r')
data = []
for line in datafile:
    line = [float(element) for element in line.rstrip('\n').split(',')]
    data.append(np.asarray(line))

num_features = 48
X = [data[i][:num_features] for i in range(len(data))]
y = [int(data[i][-1]) for i in range(len(data))]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

Now, let us go ahead and make our training and test splits as we do always before taking of any machine learning study. And that the just to remind you that in this particular session unlike the others the previous ones we will not be using a scikit learns in built functions, we will be coding down the entire execution by ourselves so that we get a complete understanding of how the system works. It is really simple and it will not take a much labor. So, first we go ahead and we load our data and as you can see and then we set that the number of features that we are interested about are just 48.

So, I am sorry we are going to consider the special characters also. We just use the word frequencies in our exercise. So, the data is separated into the inputs and the labels as you can see over here, and then we make our training and test splits. The train test split function is coming from a sklearn dot cross validation. That is the only function that we use from scikit learn and nothing else, the rest we code at code down our self's.

(Refer Slide Time: 07:15)



Estimation of class-specific likelihood ratios

Likelihood of a keyword w given that it occurred in a spam document:

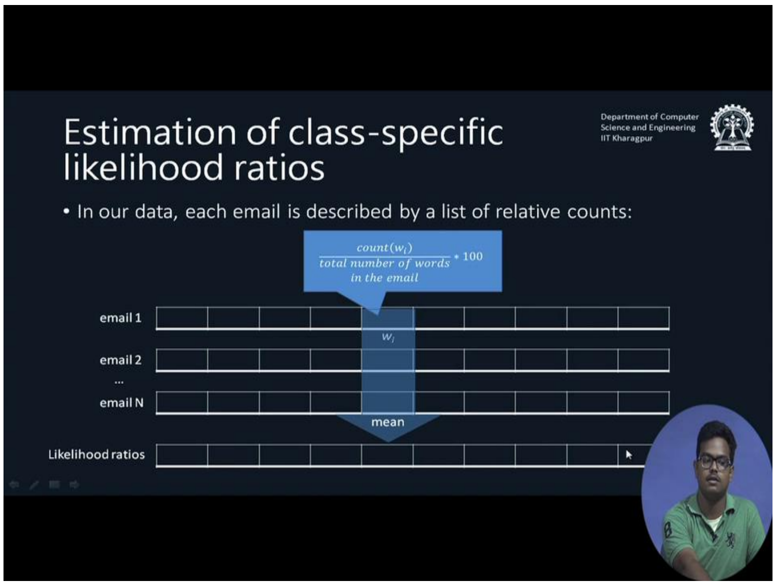
$$P(w|spam) = \frac{count(w, spam)}{\sum_{w \in all\ words\ in\ vocabulary} count(w, spam)}$$

Department of Computer Science and Engineering IIT Kharagpur

So, next we will look at how to estimate the class specific likelihood ratios. The likelihood ratios are represent that how probable is one feature given the email belong to a particular class. Say we have a spam email and we have a word w, so we want to find how likely the word w is given that the email was spam email. So, how do we estimate it? We just estimate it from the frequency definition of probability. We just count how many times the word w appeared in spam emails and then divided it by the count of like all words submit over all words in the vocabulary occurring in spam emails.

So, this is how we calculate. We find how many times this particular word appeared in the spam emails and divided by odd words like, the entire size, the entire sum over all the sizes of all the spam image that we have in our data base.

(Refer Slide Time: 08:30)



Estimation of class-specific likelihood ratios

Department of Computer Science and Engineering
IIT Kharagpur

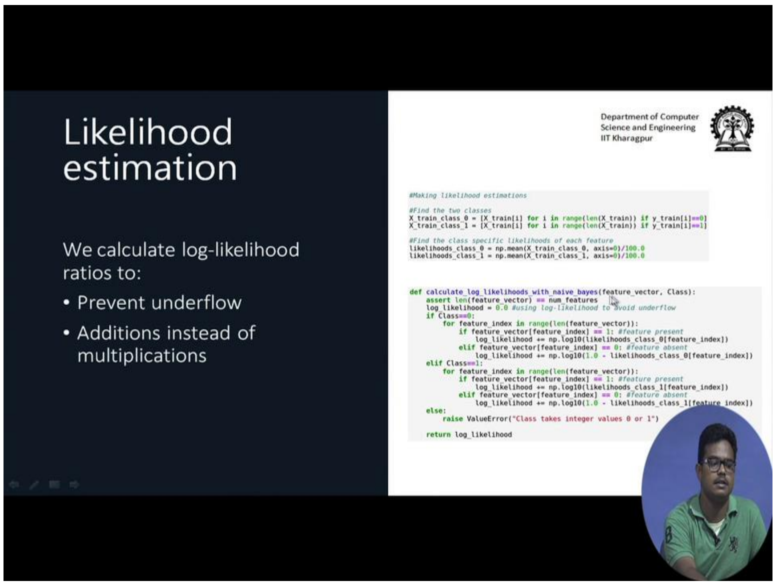
- In our data, each email is described by a list of relative counts:

$$\frac{count(w_i)}{\text{total number of words in the email}} * 100$$

Then what we do? So, how do we really implement it in our case, like the way our data appears, how do we do it? In our data set each email is represented as a vector and each position of which shows the relative count of the word in the emails. Say, we are talking about a particular word w I, so the vector will contain the number of times the word w i occurred in the email divided by the total number of words in the total number and it multiplies it by 100 to make a percentage. So whatever number it appears over here this shows that if 100 words that represent in the email then how many of the words would be the word in question that is w i. How do we get use this information to calculate this is.

So, as you can see that this is supposed to be summed over all the spam emails in the data base right and we get email y's the ratios same thing in a percentage form in each vector for each email. So what we do is, we look at all the spamming emails in our data base, and then average these counts. And of course, we divide the number which we get after averaging by 100 so that we get a fraction between 0 and 1 of probability value, and what we have other likelihood ratios.

(Refer Slide Time: 10:01)



Likelihood estimation

We calculate log-likelihood ratios to:

- Prevent underflow
- Additions instead of multiplications

Department of Computer Science and Engineering IIT Kharagpur

```
#Making likelihood estimations

#Find the two classes
X_train_class_0 = [X_train[i] for i in range(len(X_train)) if y_train[i]==0]
X_train_class_1 = [X_train[i] for i in range(len(X_train)) if y_train[i]==1]

#Find the class specific likelihoods of each feature
likelihoods_class_0 = np.mean(X_train_class_0, axis=0)/100.0
likelihoods_class_1 = np.mean(X_train_class_1, axis=0)/100.0
```

```
def calculate_log_likelihoods_with_naive_bayes(feature_vector, Class):
    assert len(feature_vector) == num_features
    log_likelihood = 0.0 #using log-likelihood to avoid underflow
    if Class==0:
        for feature_index in range(len(feature_vector)):
            if feature_vector[feature_index] == 1: #feature present
                log_likelihood += np.log10(likelihoods_class_0[feature_index])
            elif feature_vector[feature_index] == 0: #feature absent
                log_likelihood += np.log10(1.0 - likelihoods_class_0[feature_index])
    elif Class==1:
        for feature_index in range(len(feature_vector)):
            if feature_vector[feature_index] == 1: #feature present
                log_likelihood += np.log10(likelihoods_class_1[feature_index])
            elif feature_vector[feature_index] == 0: #feature absent
                log_likelihood += np.log10(1.0 - likelihoods_class_1[feature_index])
    else:
        raise ValueError("Class takes integer values 0 or 1")

    return log_likelihood
```

So how do we do it in code? First, we segregate the two classes. So, first we separate the class 0 and class 1; so class 0 represents that the email is not spam, class 1 represents the email is spam. And then we calculate the mean along the columns. As we shown in the last slide,we are calculating the means over each column. So that is why access equal to 0 and divided by 100 to convert the percentage into a probability.

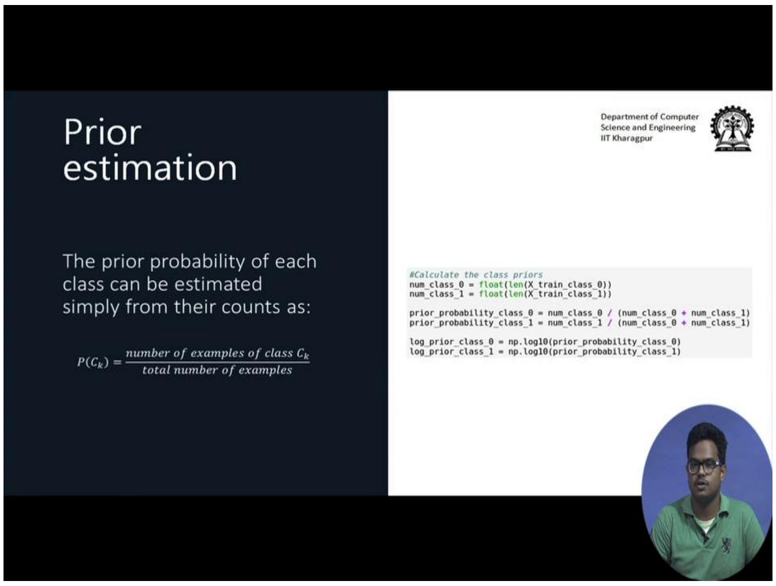
Then, we would like to find the log likely hood ratio instead of using the likely hood, because as you saw over here in this slide. You can see that we are doing a sum a product over the likelihoods of the different feature dimensions. So, each of these number is a number of which is much less than 1 in most cases and you multiplies a small numbers a lot of them together and this may result in underflow, you might end up with a very very small number of which is not possible to represent a in your computer properly. So, that leads to errors and other problems down the line.

So, that is why we avoid multiplications. We convert all probabilities into log probabilities, that is all likely hood into log likelihoods and sum the likelihoods instead of multiplying them. This is just a hack, popular hack in machine learning. We just calculate the likelihoods, so log likelihoods and this function does the same. If class 0 it will use the likelihoods of class 0. And what it does is, it takes a feature vectors. This feature vector is in the form that we discussed before over here. It is a binary vector representing which vector is present or not which word is present, which of the key words is present.

So that is the feature vector and you specify that yeah we want to find the likelihood of these feature vector given a particular class. This class is going to take values either in 0 or 1. And, what we do if the class is equal to 0, then for each feature index what we do we calculate the likelihood of that particular feature given that class. And if that feature is absent we calculate one minus likelihood, so the feature is absent.

So, you must takes some time and carefully look at the code and tries to understand that why things are happening the wave it is written there, so that is the good exercise. After we have found the likelihood ratios, the log likelihood values what we do.

(Refer Slide Time: 13:06)



Prior estimation

The prior probability of each class can be estimated simply from their counts as:

$$P(C_k) = \frac{number\ of\ examples\ of\ class\ C_k}{total\ number\ of\ examples}$$

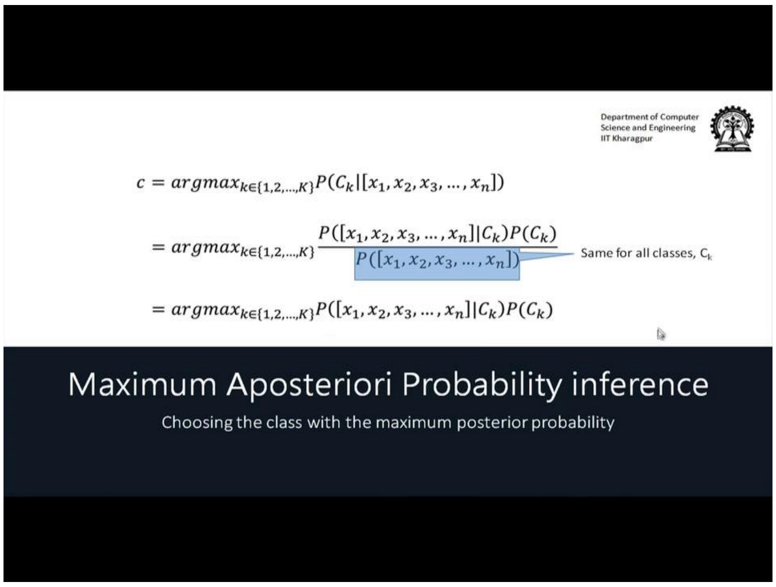
Department of Computer Science and Engineering IIT Kharagpur

```
#Calculate the class priors
num_class_0 = float(len(X_train_class_0))
num_class_1 = float(len(X_train_class_1))

prior_probability_class_0 = num_class_0 / (num_class_0 + num_class_1)
prior_probability_class_1 = num_class_1 / (num_class_0 + num_class_1)

log_prior_class_0 = np.log10(prior_probability_class_0)
log_prior_class_1 = np.log10(prior_probability_class_1)
```

We find the prior probabilities of the two different classes. So, how do we calculate the prior probabilities? We just calculate by the frequency definition of probability. We calculate how many emails in the data set where spam, and how many have non spam, and then we just simply find the ratios. For example the say C k is spam and probability of spam mails will be number of spam mail in the data set divided by the total number of emails in the data set.

We also perform log priors; we take the log of the prior probabilities just to make the job easy for us because we are just going to add them to the likelihood ratios.

(Refer Slide Time: 13:49)



Department of Computer Science and Engineering IIT Kharagpur

$$c = argmax_{k\in\{1,2,\ldots,K\}} P(C_k | [x_1, x_2, x_3, \ldots, x_n])$$

$$= argmax_{k\in\{1,2,\ldots,K\}} \frac{P([x_1, x_2, x_3, \ldots, x_n] | C_k) P(C_k)}{P([x_1, x_2, x_3, \ldots, x_n])}$$

Same for all classes, C_k

$$= argmax_{k\in\{1,2,\ldots,K\}} P([x_1, x_2, x_3, \ldots, x_n] | C_k) P(C_k)$$

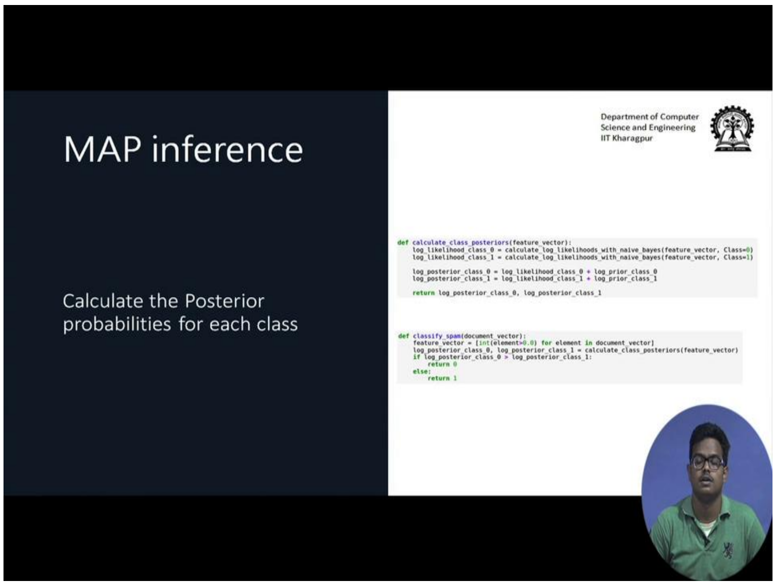
Maximum Aposteriori Probability inference

Choosing the class with the maximum posterior probability

And then the log likelihood values rather, sorry Next, after doing this we will go ahead and see how to do map inference. Map stands for Maximum Aposteriori Probability inference. This just (Refer Time: 14:03) means that we are going to choose the class that has the maximum aposteriori probability given the input value. So, we are going to choose the argmax over k's, we going to choose the class that has the maximum aposteriori probability given the input vector and so we decompose that in using the base rule.

As you can see that, this k does not appear in the denominator, so this term is going to be the same for all classes C k. This thing is actually can be removed, because this is same for all classes C k. So as we are finding one which is the class which is looking for class has the maximum value of this term we can as well look for that particular class C k that has the maximum value for just for this product. And when we use like log likelihoods then this product becomes a sum and that makes our life simpler.

(Refer Slide Time: 15:05)



MAP inference

Calculate the Posterior probabilities for each class

Department of Computer Science and Engineering IIT Kharagpur

```python
def calculate_class_posteriors(feature_vector):
    log_likelihood_class_0 = calculate_log_likelihoods_with_naive_bayes(feature_vector, Class=0)
    log_likelihood_class_1 = calculate_log_likelihoods_with_naive_bayes(feature_vector, Class=1)

    log_posterior_class_0 = log_likelihood_class_0 + log_prior_class_0
    log_posterior_class_1 = log_likelihood_class_1 + log_prior_class_1

    return log_posterior_class_0, log_posterior_class_1
```

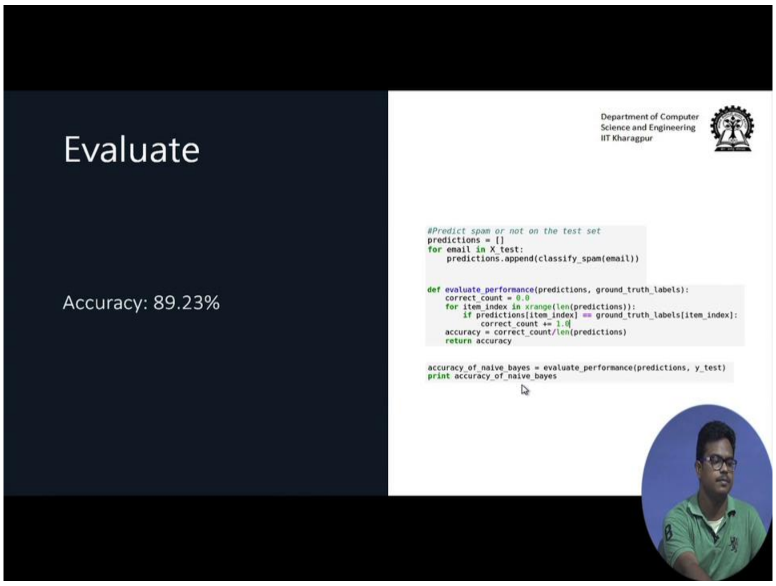
```python
def classify_spam(document_vector):
    feature_vector = [int(element>0.0) for element in document_vector]
    log_posterior_class_0, log_posterior_class_1 = calculate_class_posteriors(feature_vector)
    if log_posterior_class_0 > log_posterior_class_1:
        return 0
    else:
        return 1
```

So, now we go ahead and calculate the class posteriors. For calculating the class posteriors we again show the feature vector, the binary vector that we talked about representing which particular feature is present or absent or which particular word is present or absent. And then the log posterior of each class will be the sum of the log prior and the log likelihood, and we return them. Another function classifies spam it takes document vectors, so the document vector is the one that was presented originally in the data set which give the relative frequencies of the words.

So, we just going to check whether the element was greater than 0 in that particular vector, and if it is going to be greater than 0 this means the word appeared in the particular email and that is why we just said that element will equal to 1. This is what it does. It converted into a feature vector and then we calculate the log posteriors using this function.

And we return class 0, that is not a spam if the aposteriori probability of not a spam be greater than aposteriori probability of it being a spam email otherwise we classify it as a spam email. So, this is how we solve the problem, how we use naive bayes classifier for detection of spam.

(Refer Slide Time: 16:43)



Evaluate

Accuracy: 89.23%

Department of Computer Science and Engineering IIT Kharagpur

```
#Predict spam or not on the test set
predictions = []
for email in X_test:
    predictions.append(classify_spam(email))


def evaluate_performance(predictions, ground_truth_labels):
    correct_count = 0.0
    for item_index in xrange(len(predictions)):
        if predictions[item_index] == ground_truth_labels[item_index]:
            correct_count += 1.0
    accuracy = correct_count/len(predictions)
    return accuracy


accuracy_of_naive_bayes = evaluate_performance(predictions, y_test)
print accuracy_of_naive_bayes
```

And, evaluation shows that it gives 89.23 percent classification accuracy and that is quite compiling. And this function will check how many of the predictions match the ground truth labels. So, it is checking whether the predictions are same as the ground truth and it calculates the accuracy. You just print the value, and thus you can use naive bayes algorithm for classifying emails as spam on non spam.

So, I hope that was enjoyable. See you in the next video, bye-bye.