Text classification

Professor Eric Atwell: Hello. This is Eric Atwell. Today, I'm going to talk about text classification. These slides are our introduction to chapter 4 of the Speech and Language Processing textbook by Dan Jurafsky and James Martin. So, hopefully, if you want to find out more about this topic, then read the textbook chapter 4.

So text classification is a particular sort of classification. So you'll have been introduced to machine learning classifiers in the data science course and possibly looked at some more in machine learning and deep learning. And the difference here is the data is text.

So I'm going to look at in particular one classifier, the Naive Bayes classifier. It's not a deep learning classifier. It doesn't look particularly sophisticated. But it works quite well for text.

So we've got to look at this. Other classifiers will work too, but this is a nice example. So we've got to look at the underlying model and how it's applied in learning and then how it's applied for a particular example. Sentiment analysis is a good one. But there are others too.

So we've got to look in some more detail at sentiment classification because it's a hot topic and look at Native Bayes and how it's similar to Markov or N-gram language modeling that we saw in the previous lecture. I'm going to look at evaluation. And, in particular, you might think accuracy is a good measure, but there are other measures, precision, recall, and F measure. And Naive Bayes is particularly good for binary classification, good or bad, for example.

But you also want to work out-- sometimes there could be three or more classes. And how do you evaluate for those? And then, finally, I want to look at some sort of ethical issue using classification, avoiding harms because sometimes a classification can take into account implicit bias in the training set and then reproduce that bias in the test.

So let's start off by looking at what text classification is about. In text analysis, an obvious thing is spam identification. So here's a possible email. And you want to identify or rather Outlook needs to identify is this spam or not. You don't have to read the email.

Typically, it's in terms of the words in the spam. In America, remembering the textbook was written by Americans-- so a big issue in American history is before there was Twitter or tweets, before there was the internet, when the United States was first being set up, there were a number of independent states like New York or Florida. And they had to decide individually whether or not they should join up to become the United States. And rather than sending tweets, J. Madison and Hamilton were three politicians who wrote letters or papers to try to persuade the people to join the United States.

Authorship of 12 of these letters was in dispute. Nobody was quite sure who wrote it because they forgot to sign it. But back in the 1960s, some mathematicians, Mosteller and Wallace, used Bayesian mathematics to analyze the texts and try to predict which of the three classes, in this case the authors, applied for these 12 letters. And if you want to find out more about this, this is a classic example of doing text analytics classification back in the 1960s before there were any big corpora or deep learning methods or anything else like that.

Another topical issue at the moment is there's lots and lots of research papers in disciplines, for example, in medicine. And you want to work out what's the subject of the medical article. But in medicine, there is a

medical subject hierarchy used in medical research. Basically, librarians and medics have got together to work out the classes and subclasses in medicine. And nonmedics certainly understand what these are. But the idea is you'd like to be able to take an article and classify it according to the MeSH Subject Category Hierarchy.

This med article is about blood supply. Well, possibly, it can be about more than one thing. So this article is about blood supply and drug therapy. You can have a multi-class classifier necessary.

Very popular is the idea of sentiment analysis. Is the text positive or negative? And this, for example, started off with movie reviews. There are lots of available movie reviews. Leeds University was one of the founding places for a movie review database, which has now gone worldwide.

And the founder was a student here and is now quite well off because it's become very popular. Now, here's some examples. Zany characters and richly applied satire, and some great plot twists. Well, this is a positive review.

It was pathetic. The worst part about it was the boxing scenes as a negative review. So in reviews, there are particular words like richly, and great, and awesome, and love, which are positive. And other words are like pathetic, and worse, and awful, and ridiculous, which are negative.

So you can count up a very simple sentiment analysis. Basically, it counts up the positive words, counts up a negative words. And whichever is most of, that's the overall category.

So sentiment analysis is useful for a movie review. Is the review positive or negative? In industry, it's very popular for product reviews, and that's why Amazon invites you to, when you buy something, give a review in plain English, and also give you a star rating. And therefore, you can automatically collect positive reviews in terms of star rating, and negative reviews in terms of star rating, and also the text that goes with it.

There's other applications as well-- public opinions about consumer confidence. Whenever there's an election in politics, there's lots of positive or negative tweets about the different candidates, and also predicting the election outcome or the market trends. So in predicting whether or not stocks or shares are going to go up or down, one way of doing this is to analyze the newspaper articles about that stock and see if that predicts positive or negative movement.

Of course, sentiment isn't just positive or negative. So there's a Scherer typology of affective states that's used in psychology. So there's emotion, mood, interpersonal stance, attitude, and personality traits, different types of sentiment, and each of these have got different classes. So in terms of emotion, you can be angry, sad, joyful, fearful, and so on, and that's quite different from the attitude of liking, loving, hating, and so on.

Now, sentiment analysis, in computational terms, is pretty much just about the attitudes of people when they write things. So when you write a movie review, you like it, you love it, or you hate it. That's the sort of thing that you're capturing.

There is also personality traits, to some extent, you might want to try to work out from the review. Is the review anxious or reckless or hostile? But essentially, it's the attitude you're measuring, not these other things.

Sentiment analysis is basically the detection of attitudes, and so in this chapter, chapter 4, we just focus on simple, very simple attitude. Is it positive, or is it negative? There are further classifications, and further on

in the textbook, you will see some more sophisticated analyzes. But for now, let's just say, is it positive or negative?

So in summary, text classification has lots of applications, like sentiment analysis, positive or negative. Spam analysis-- is it spam or not spam? Many of these are binary. That's nice and simple.

But you can also have more than two classes, if there's several possible offers, and you want to identify which one it is. Or if you have a text, and you want to identify what language it belongs to, there's many languages. Or what is the topic of the medical document?

If a research paper, is it about gene therapy, or is it about-- blah, blah, blah? I don't know. So there could be many categories.

In general then, the input is a document, and also a fixed set of classes. So in advance, the classifier-- or the researcher has decided what are the classes? And the output is, for this document, which class it belongs to from this set of classes?

How do you do this? Well, actually, it's not necessarily machine learning. So spam identification has been around before there were deep learning classifiers. A very simple solution, works reasonably well, is to have some rules, like if your email address is blacklisted, or there's dollars involved, or "you have been selected."

This can work quite well. The rules have to be determined by an expert, and of course, the problem is that the spam developers know about these rules, and therefore, find ways around it. So the rules are transparent and available and work quite well, but building and maintaining the rules can be expensive. You've got to keep up with the spammers.

So the other way is to do supervised machine learning. So the input is a document, a set of classes, and also a training set of hand labeled documents. So someone has to go through, in advance, and collect a lot of emails, let's say, and label them as being spam or not spam. An expert, a human, has to decide.

And this is the difficult bit-- for supervised machine learning in general, you've got to have a training set, where each instance is labeled, and for text analytics, this often is quite expensive. Some experts have to label each of the documents. And then the output is a classifier, which given a document, can tell you what class it belongs to.

In all of this, which particular learning algorithm you use isn't that important. We're going to look at Naive Bayes, but neural networks or logistic regression or other things work equally well. So hopefully, you're going to see-- if you're going to use a Python toolkit, it probably doesn't matter so much which classifier you use.

Or rather, what you tend to do is try out several different classifiers and see, empirically, which one is best. Naive Bayes seems to work reasonably well from previous experience, so you probably want to include that. But you want to include some others as well.

So what does test classification involve? And let's look at this Naive Bayes classifier. It's naive in that it makes some very simplifying assumptions, but the simplifying assumptions seem to work reasonably well.

First of all is the "document is just a bag of words." Here's the document. "I love this movie. It's sweet, but with satirical humor."

So in real life, the order of words is important, but for the bag of words assumption, you just assume it's a set of words. And then you can count up, for each word, how many times it occurs. So sometimes, words like it and I and the are frequent, but actually, we don't even care about that too much. So the frequency of words doesn't matter too much. It's just the words that are there.

The bag of words representation is essentially a list of words that are created in a document and, possibly, how many times they occur, although as we'll see later on, the frequency doesn't matter too much. It's just basically is the word in there, and there's a whole lot of other words which are not in there. So the important frequency is really 1 or 0.

And this is used to give us a class. So a particular combination of words can predict the class-- in this case, sentiment yes or sentiment no, or good or bad.

A Naive Bayes classifier means you use Bayes' rule. Bayes' rule is a way of determining-- so for a document d and a class c, the probability of a class given the document is equivalent to the probability of a document given the class times the probability of a class divided by the probability of a document. It's a very simple thing.

So using a Naive Bayes classifier, we want to try to maximise the probability of a class given a document. That was find the class which has the largest probability for a particular document, the most likely class. And given the Bayes equation, we see that we want to maximise-- or find-- the probability of a document given a class times the probability of a class divided by the probability of a document.

Now, we know what document we've got, so we can drop the denominator. You don't have to divide. The probability of a document is 1 because we know what document it is. So we want to maximise-- find, for all the possible classes, the probability of a document given the class times the probability of the class, so we do that for each of the classes and figure out which one is the highest score.

In other words, the document is actually a set of features-- the words in the document. So you want to find what's the maximised probability of the set of words in the document given the class times the probability of the class. You want to do that for each of the different classes. Your document is represented as a set of features-- essentially, the words in it. And this thing want to do is basically a likelihood of a document-- or sorry, the likelihood of a class given the known classes.

Now, what we need to know is, what's the probability of the class? And we can look at that straightforwardly. We just count the relative frequencies in a corpus if we have the labeled training set, which has, for each document-- for example for each tweet, is it positive or negative? And then we can count how many positives there are and how many negatives there are in that example.

The other thing is the set of words given the class, and that's much more difficult. You can estimate, for each word, how likely it is for a particular class. That means we have to have a very, very large number of training examples. You have to have a reasonably large corpus because we want to work out, for every word, how good-- what's the probability of that word given a class, which means you've got to have lots of examples of each word for each of the classes.

Most of the words are quite rare, in even in a large corpus, so that could be quite difficult to collect. That's why we do this bag of words assumption. We don't have to look at a particular word sequence, but we just assume that the word is counted up regardless of where it occurs. And we also have to assume, independent of the conditions, we have to assume that each of these features, the probability of each word given a class is independent given the class c.

So we don't assume that the probability of one word is dependent on the probability of another word, but they are just a bag of words. Some words, the probability of a particular word sequence, like "the cat sat on the mat," given a particular class, is it doesn't matter on the order of the words. And the probability of the and cat and sat are all independent of each other.

So the Naive Bayes classifier essentially assumes we want to maximise the probability of each of the words given a particular class, and you work out the probability of a document by simply multiplying together the probabilities of each of these words given that particular class. So all the word positions in a test document are multiplied together, so you apply the Naive Bayes classifier to text classification by simply saying, for each of the possible positions in a document, multiply all them all of them together and figure out which one is the highest.

There's a practical problem with this multiplication-- we've seen this before in the N-gram modeling-- that if you multiply together lots of probabilities, each probability is in the range 0 to 1, so each fraction will be quite small. And multiplying together a lot of them-- for a document, there'd be many words-- ends up with a very small number multiplying lots of probabilities can result in floating point under flow. So what we do instead is we use logarithms of these numbers, and then we add together the logarithms.

So that has got two benefits. One is that you avoid underflow because the logarithms will stay in the same sort of range. Adding them together doesn't produce ever smaller numbers.

And also, adding is faster than multiplying in computer terms, so logarithms are just basically more efficient. So we do everything in logarithmic space. Instead of lots of multiplications, you do a sum of the logarithms, and sum of a logarithm is faster and doesn't involve underflow.

Notice, also, what we actually want is to find the best class, and in this case, the ranking of the classes is all important. So the highest probability in terms of multiplication is also the highest log probability. So all that matters is we want these scores for each class, and rank the classes according to the scores. And if the scores are logarithms rather than probabilities in a range 0 to 1, it doesn't matter as long as the rank order is the same.

So that means it's basically a linear model. I mean, if you're a mathematician, this is significant. You want to maximise the sum of weights. Therefore, it's a linear function of the inputs, and that's why Naive Bayes is called a linear classifier.

This is a sort of important thing for mathematicians. I'm not so sure, in terms of data mining, that it is important. Nevertheless, that's the point.

So that's the Naive Bayes classifier in a nutshell. For more details, if you like the mathematical modeling, have a look at the textbook. Let's just see how it's applied to learning.

So the first attempt, you want to have maximum like estimates, so you can simply use the frequencies in the data to try to work an estimate of probabilities. You count up how many words appear under each class and use the word frequencies as an estimate of probabilities, and you want to find out the estimate of the probability of a word belonging to a class. You count the fraction of times the word appears amongst all words in documents of topic c or sentiment c.

So basically, you create a mega document for a particular topic j or a particular sentiment, positive or negative, by concatenating all of the documents in this topic, and then use the frequency of the word in this particular mega document. Basically, you count up how many times does a word appear in all of the positive documents or all of the documents to do with blood clotting, if you have a medical document class.

Of course, there is a slight problem with this counting thing. We've seen this before in N-gram modeling. What if we have no training documents with the word fantastic, which are classified as positive?

So the probability of fantastic being positive can be the count of the fantastic in all the positive documents divided by the count of all of the words in the positive documents while the frequency count of all the words is basically the number of words, the size of the sub-corpus. But if fantastic never occurs, that'll be 0 divided by some number, which will be-- the probability will be 0.

So we have a problem. If a word just happens not to occur in the positive class, then the probability will be 0. And maybe fantastic does occur in the positive class, but it's not so likely to occur in a negative class. So therefore, we have a probability of negative being 0. 0 probability is our problem, despite all the other evidence, because remember, you're multiplying together all the probabilities. And that means if one of those probabilities is 0, then the multiple of all of them will be 0, so this is a problem.

If you're looking at all of the words which occur, if some of the words, or any one of the words, has a 0 probability, then that means the probability overall of a whole text becomes 0. So this is why we have to do smoothing of some sort.

So this is, remember, add 1 or laplace smoothing for N-gram modeling. You have to do the same thing for probability modeling in Naive Bayes. You don't just take the count. You add 1 to the count, and you add 1 to all of the counts.

So in other words, the probability for a particular word, like fantastic, is the count of fantastic in all of the documents, add 1, divided by the count of all of the words-- the corpus size. Add on 1 for every word. In other words, add on the size of the vocabulary.

So from the training corpus, you extract the vocabulary, the dictionary of all the words that appear there. And then you calculate the probability of classes just by looking at-- for each of a document, count up how many documents have that class, and that gives you the probability of a class. So for a particular class, how many documents divided by the total number of documents.

So for example, in a corpus, you might say, collect some tweets, half of which are positive and half which are negative. So the probability of positive-- let's say you've got 100 documents. If 50 of them are positive, then the probability of positive is 50 divided by 100, or a half. The probability of negative is also 50 divided by 100, or a half.

We also have to calculate the probability of each of the words given the class, so for a particular text, a text is a single document, and you want to take all the documents containing positive things. And for each word in the vocabulary, count up how many times it occurs in this set of documents. So the 50 documents which are positive, count up how many times the word appears in there.

And then for a probability, you want to have the count plus some constant divided by the total number of words plus some constant times the vocabulary size. And this content is usually 1, so when you add 1, you say the probability of a word like fabulous-- was it fabulous? I've forgotten the word now. Fabulous plus 1 divided by the total vocabulary size plus 1.

So what about unknown words? This was another problem in N-gram modeling, that if you have an out of vocabulary word in the test-- in the test data, you come across a tweet which has got a word in which wasn't in the training set. This was a problem in N-gram modeling because you'd get a 0 probability, and we don't like 0 probabilities.

But actually, if you have a tweet which has an unknown word in it, you can just ignore the words. If you remove them, pretend they're not there, then rather than trying to get any probability-- well, how does this work? Why don't we build an unknown word model?

Remember, we had-- before, you had a word unknown, and you counted those frequencies. Well, actually, it's not worth doing. It doesn't really help. Knowing which class has more unknown words doesn't really help you very much.

Knowing that, for example, there are more unknown words in positive tweets than in negative tweets isn't really going to help. What really matters is for the known words of a positive or negative. So actually, in classification, this is one difference between classifiers and N-gram modeling. Naive Bayes, you can just ignore the unknown words.

And the same thing for stop words. Stop words are very frequent words like the and uh and and. And in some NLP applications, what you have to do is take out the list of stop words and remove them.

But actually, again, that's probably not worth doing because again, in Naive Bayes algorithms, words like the and uh will appear in all of the classes. The positive tweets and the negative tweets will both have the and uh in it, so it's not really worth the effort of removing them because Naive Bayes algorithms basically just use all of the words. And you might say it might save a little bit of time to not process them, but removing them is time consuming. And it's not just the and uh, but typically, you may have 50 or 100 stop words, and preprocessing all of those words is just extra hassle which isn't worth it.

So that's Naive Bayes. You just apply the algorithm very naively, and it seems to work reasonably well. So let's have a look at a worked example of binary Naive Bayes applied to sentiment analysis.

Let's take a very small training corpus. We've got just five tweets. In fact, not even tweets-- they're just phrases.

And the training set has got three negative examples-- just plain boring, entirely predictable and lacks energy, no surprises and very few laughs. We've also got two positive examples-- very powerful, the most fun film of the summer. And in comes a new comment-- predictable with no fun. Now, is that positive or negative?

Well, the CRISP-DM methodology says, first thing you should do is, are there any obvious results? What do you think the result should be? Well, to me, that looks like a negative, so I, as a human expert, think this is negative.

Now, we're going to see what does the Naive Bayes classifier predict it to be? Remember, we have to do add-1 smoothing, so we have to add 1 to all of the frequency counts. Otherwise, we'll end up with 0 probabilities, which mess things up.

So here's our data set, so the first thing we have to do is work out the probability of each of the classes from this training set. Well, there are three negative examples, so the probability of negative is 3 out of 5. And there are two positive examples, so the probability of positive class is 2 out of 5.

Next, we have to look at our test instance predictor with no fun. Remember, we said with is not anywhere in the training sets. It's out of vocabulary. Rather than try to do some unknown token processing, we'll just forget about it. We'll drop with.

Next, we have to calculate, for each of the words, for each of the classes, what is the probability of that word given the class? And this is, in terms of counting, how many times that word appears in that class,

adding 1, and then dividing by the count of all of the words in all of the data. Adding on the vocabulary size that-- a 1 for every word in the vocabulary.

The model should actually do that for all of the words, but we're only interested in doing it for predictable no and fun in the test set. We don't need to do it for all the other words, as well. So for example, the word predictable, being negative-- well, it does appear once in the first three tweets, so that's 1. But we add on 1, so it's 1 plus 1, or 2.

And there are how many words altogether in the negative set? 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14-- there are 14 words in the negative set. So therefore, divide by 14, and then you add on the size of vocabulary. Well, that's the entire training set. There's also powerful, fun, film, summer, most. That's six extra words, so that's 20 is the entire vocabulary.

So the probability of predictable, given that it's a negative tweet, probability of predictable in only the negative tweets is 2 divided by 34. And you could do the same calculation for predictable being positive, but predictable doesn't appear in any of the positive cases. So it's 0, but we don't want 0.

So you add on 1 for smoothing, so it's 1 divided by-- well, there's 1, 2, 3, 4, 5, 6, 7, 8, 9 words in the positive samples, but we have to add on the vocabulary size. So the probability of predictable given it's positive is 1 divided by 29.

I'm not going to do this for all of the words, but we can see, in principle, we want to work out what's the probability of negative times the probability of a sentence given that it's negative. And that's a probability of a negative is 3 out of 5. We've already done that, probability of a class being negative.

And the probability of a sentence given that it's negative, that's the probability of predictable, no fun, given it's negative. Well, that's 2 out of 34 times 2 out of 34 times 1 out of 34. 2 out of 34 for predictable, 2 out of 34 for no, and one out of 34 for fun. That's 2 times 2 times 1 over 34, and this whole thing is 6.1 times 10 to power minus 5, so 6.1.

Whereas the probability of a class being positive times the probability of a sentence given that it's positive, predictable is 1 out of 29. Sorry, the class, to start off with, being positive, is w out of 5. And then predictable is 1 out of 29, and no is 1 out of 29, and fun is 2 out of 29. So that's 3.2 times 10 to the power of minus 5.

So in other words, we can see that, essentially, this score for negative is twice, more or less, the score for positive. So it must be negative. That's what we're predicting, which is good because that's what we thought in the first place.

So that works as a nice worked example. If you're not sure, have a look at the textbook and go through it again. It's explained in better detail in the textbook.

So for tasks like sentiment analysis, it looks like the occurrence of the word is more important than the word frequency. So the fact that fantastic is there is important. If it occurs five times rather than once, it doesn't really help any. We still know it's fantastic is important.

So in other words, this gives us an idea that maybe you don't need to count up everything. You just have to say, is it there or not? So that gives us the binary multinominal Naive Bayes or binary NB. This is different from Bernoulli Naive Bayes, which is another mathematical model. And it's in the textbook, but I'm not going to really talk about that here.

So for binary, it's a bit easier. Rather than calculating all the frequencies of the words, we just have to basically count up if it's there or not. So basically, you remove all the duplicates in the document, and for each word, we basically only retain a single instance of a w. And this affects the count somewhat. Let's have a look at an example of this.

So we remove all the duplicates of a word from all the document, and then we do all the calculations we did before. But we use cut down versions of a document. Using the same equation, using the same procedures, but you basically ignore all the duplicates of the word.

So let's have a quick work example. If we have four documents, negative ones are "it was pathetic" and "the worst part was the boxing scenes" and another one, "no plot twists or great scenes." And we have two positive examples-- "and satire" and "great plot twists." Another one is "great scenes great film."

Well, we don't want to count-- so in the original count, we have, for example, great appears three times because the final one is great scenes great film, so it's duplicated. But actually, the fact that it's got great scenes great film, it's still positive if you just have the word great in it. It doesn't matter it's repeated.

So after predocument binarization is getting rid of all the duplicates, we have, "it was pathetic," "the worst part boxing scenes." You don't say was twice. And this is a nice way of cutting out all these function words like was and the and so on. "Was the" are cutout. That the fact that "was the" appear several times in many documents-- you throw that away because it's not actually going to help any.

Similarly, "great scenes great film--" we throw away the second occurrence of great. This is the simplify documents, and we do the counts all over again. And this time, most of the counts are 1s and 0s. But notice that "and," which appeared twice in the original count, is down to just one, and this is because and-- where is and?

And satire and great plot twists-- well, it's just down to and satire great plot twists. Have a notice that some of the counts can be 2. So you get rid of repeats within a document. Binarization is within a document, but if a word appears in two separate documents, then it can be pulled.

So for example, the word great appears in satire great plot twists, and also great scene films. So great is in both of the positive documents, and this is worth keeping. It shows that the word great is a good indicator of positive documents.

And this is actually, if you like, it's sensible. If a word is repeated within a document, then we don't-- it's not that important. But if a word is repeated across several documents, then it is a good indicator of positive or negative.

And that's enough on sentiment analysis and Naive Bayes. Let's have a look a bit more on what are the other problems in sentiment classification. Well, one issue we haven't dealt with is negation.

I really like this movie. What happens if I say, I really don't like this movie? That's actually negative. But words like really and like are positive, so what do you do?

Well, negation essentially changes the meaning of like to negative. So negation can also do the other thing. If I say don't dismiss this film, then dismiss is negative, and don't dismiss is positive. Or doesn't let us get bored. Well, bored is negative. Doesn't bored is positive.

So what can you do about this? Well, basically, a simple method that's been used is to add-- to negate every word between negation and the end of the clause or phrase, and that's basically the following

punctuation. So if you say I didn't like this movie, but blah, blah, blah, blah, blah, well, didn't doesn't apply to everything, but it does apply to not like. Or not this, and not movie.

And actually, there isn't a not movie elsewhere, but not like is important. So we have didn't not like, not this, not movie, but not all the rest of the words as well because the comma tells you that's the end of that phrase. So the negation only applies within that phrase.

That's one thing. Another thing is we need lots and lots of labeled training data. We need to have lots of examples of tweets where someone's read them and said they're positive or they're negative.

What happens if you haven't got that? What we have got, we can get hold of prebuilt lexicons. That is words someone's already gone through and decided particular words are definitely positive, like "like" or "fabulous" or "nice." And here's a couple of examples.

Here's one example. You can get hold of the MPQA Subjectivity Cues Lexicon, and this has lots of positive words like admirable, beautiful, confident, dazzling, and lots of negative words like awful, bad, bias, and so on. And you can build a classifier which simply has these as the features, and if a new text coming in has one of the positives-- you count up how many positive words there are, and then you count up how many negative words there are using this lexicon.

Here's another one. The General Inquirer-- this has a lot of positive words and negative words, and it will have other categories as well, which you may ignore or not as you see fit. So essentially, what you do is, to use it in sentiment classification, we have a feature which is incremented. You get a count every time a word from the lexicon is found.

So the feature is essentially, this word occurs in the positive lexicon, or this word occurs in a negative lexicon. So that works. You basically just have a positive feature and a negative feature, and it counts up and increment it if a word is there or not there.

The trouble is, this is not really as good as using all of the words because there are-- you're limited to the words which are in the lexicon. And there are other words, which also affect the sentiment in some way-- not directly because the positive words, you'll only have really positive words like good and great, and negative words like bad and awful. But there are other words, which are just ignored by this method. When the training data is sparse, you haven't got much training data, or it's not particularly representative of the test set, then this can help.

That's enough on sentiment analysis. What about other tasks? Well, as I said before, spam filtering is also a sort of classification task, and you can have a large training set of good documents, good emails, and spam emails and extract a Naive Bayes model from that. But that requires a lot of analysis.

It also means there's lots of words, so you have quite a large vocabulary if you've got lots of emails. Maybe only some of them are important, so you can have, instead of having a dictionary of positive and negative words, you can have a small number of features which an expert has decided are good indicators of spam, like mentions of millions of dollars or if the From address starts with a lot of numbers, then it's probably not a person.

Or if the subject is all capitals-- that doesn't happen very often anymore, but that used to be a good indicator. Or various of these other things that might be there-- those are good indicators that it's spam.

Another application in Naive Bayes is in language identification-- determining what language a piece of text is written in. Essentially, if you look at the character N-grams, that can work very well. In particular languages, particular character sequences occur or don't occur.

It is important, if you're going to do this, to have not just-- have a good variety of each language. So if you're building a classifier for English as opposed to French as opposed to Spanish, for example, then don't just include American English in your input because you should also include British English, African English, Indian English, and so on.

So in summary, Naive Bayes, while it's called that, isn't so naive. It does actually work. It's very fast. It has low storage requirements, and it works reasonably well with even small amounts of training data. So this is a great advantage over deep learning and other machine learning methods because they tend to be much-- require a lot more training data and require a lot more processor.

So what I'm saying is deep learning is all the rage at the moment, but Naive Bayes and other very simple methods can work reasonably effectively. So what you should do is try these first, and only go on to deep learning if it turns out that Naive Bayes isn't good enough. It's also very robust. Irrelevant features, words like "the," "and," and "of," they don't really help, but they cancel each other out anyway. So it doesn't matter.

It's also good in domains where there are lots of important features. For example, in sentiment analysis, the vocabulary of good words or happy words is quite large. So there may be thousands of positive words and thousands of negative words. A decision tree like j48 or-- algorithms which give you decision trees are problematic if there's thousands of features because you end up with a very huge tree, which is very difficult-- is not easy to understand.

Naive Bayes also relies on independence. It assumes that each of the words is positive or negative regardless of the other words. And we've just seen, for example, "not" is not independent. If negation is used, then that affects the following words in the following phrase. So independence doesn't always work, but there's a sort of kludge around that.

So Naive Bayes is a good, dependable baseline for text classification. Baseline means it's something you should try first, and then try other things. And if they don't make much improvement, then stop there. There's no point in doing a much more complicated model if it's not much better than the baseline.

There are other classifiers that give better accuracy. For your project, if you do a project in text analytics, then definitely want to try some others. But Naive Bayes is OK for now.

That's enough on sentiment analysis. Finally, I want to point out-- well, nearly finally-- that there are-- we've mentioned N-gram modeling as being similar in a number of ways. Let's have a look at this again.

So for classification, if you want to see if a text is positive or negative, then you want to look at the class of the first word like I, the class of a second word like love, the class of the third word like this, and the course of the fourth like fun, and the class of the fifth word like film. So basically, you want to work out, for I, love, this, fun, and film, how are they like to be positive or negative?

So Naive Bayes essentially looks at the words, but it can look at other features, too. So if, in deciding if it's spam or not, you don't just look at the words in the text, but you can also look at the email address that it came from. You may look at the URL if there's a URL involved. You may look at other features.

For fake news detection, which is a bit like spam detection, you don't just want to look at the news text to see if it's true or false or fake or not fake. But very often, you want to look at where it came from, the trustworthiness of the source. So that's analysis of other features other than the text itself, and this hearkens back to, if you're Muslim, you might know, in Islam, there's the notion of a hadith.

A hadith is something that Muhammad, the original messenger, said or did, and this was written down, and written down by someone who then passed it on to someone else, who then passed it on to someone else, who then eventually, we have it in written form. And this is called the isnad. The isnad the chain of narrators who-- where it came from, and this is separate from the matn, which is the text itself.

So it's a bit like fake news detection. You don't just look at the text, but you also have to look at the narrators or the sources of the text. However, if you're only looking at the words and not looking at any other features, and if we use all of the words in the text, not just one or two of the words, then Naive Bayes classification starts to look very similar to N-gram language modeling.

Basically, you say, for each class, you have a unigram language model, which says assigning a word is-- basically, the probability the word given a class, and assigning each sentence is the probability of sentence given the class, which is basically multiplying together the probability of each of the words given the class. So this looks like, I love this fun film.

The probability of I is, and the probability of love, and the probability of this, and the probability of fun, and then the probability of film. And you take all these probabilities and multiply them all together, and that gives us the probability of a whole sentence. This is basically the same as N-gram or unigram modeling of a sentence probability-- I love this fun film.

And if you do this for positive, and you do this for negative, then you basically have-- I love this fun film being positive or negative using an N-gram modeling for positive sentences and an N-gram modeling for negative sentences. And you multiply together or the N-grams, and you work out the probability of the positive sentence is higher than the probability of a negative sentence. That's exactly the same thing as N-gram modeling.

Let's look at how to evaluate our classifiers. So the obvious evaluation measure might be accuracy. That is, how many it got right out of all of the answers. But there are other metrics called precision recall and F measure, which can be better in certain circumstances. So let's have a look at an example.

Imagine you're the CEO of Delicious Pie Company, and you want to know what people are saying about your pies. So you build a Delicious Pie tweet detector. Now, I'm not saying are they nice about pies or not nice about pies, so it's not sentiment analysis. I'm simply saying, the positive class, the thing I'm trying to find, is tweets about Delicious Pie Company, and the negative class is all the other tweets.

So I need a classifier, and I need a training set, to start off with, of ordinary tweets. Now, chances are if I collect 1,000 tweets, most of them will be about other things, so the negative class will dominate. And this is typical in text analytics.

You often have a binary classification system where one class is very small or skewed, and there's a lot of the other class. So the positive class is very small, but we need a detector for detecting that positive class. It's the same in spam. Hopefully, most of the emails you get are not spam. Or in detecting offensive tweets, most tweets are nice, and there's only a few offensive tweets.

So we want to find the Delicious Pie tweets, so in collecting the data, we have gold negatives and gold positives. Remembering, the positives are the ones which are about the pie company, so there's only a few of those. There's only a few gold positives, and there'll be a large number of gold negatives.

And then the system has to output the labels, and it depends on what the system does. So the system could give positive or negative, but remember that the total counts will be dominated by the gold negatives. The gold negatives will be much more than the gold positives.

So accuracy is essentially all the correct ones, the true positives plus the true negatives, the ones that are correctly predicted, divided by all of the labels. That is, the true positives and the false positives and the true negatives and the false negatives.

Now, the trouble with accuracy is that, if you've got a very large number of gold negatives, then probably true negatives and false positives will be large numbers. So this accuracy is dominated by true negatives, which would be a very large number on top, and true negatives would be a very large number below. So it will tend to be one or 99% or very, very close to 1 because it's more or less a very large number plus some extras divided by a very large number plus a few extras more because true negative will be the biggest thing.

Whereas precision doesn't take into account true negative at all. Precision is the true positives, the ones you've correctly predicted as being positive, that is, the ones that are about the pie company, ignoring all of the false ones-- gold negatives. Divided by the true positives plus the false positives, so if you've got most of-- if true positives, you've got, let's say, there's only 10 of them, but you've got most of those, and you've missed a few, then the precision could be quite high.

Where recall is the true positives divided by the true positives plus the false negatives. Again, you're ignoring the true negatives, so recall could be quite high if you get most of them.

So why don't we use accuracy as our metric? Well, imagine that you saw a million tweets, and only 100 of them talked about Delicious Pie Company. So in our gold sample, the positive case is very small, but you could build a dumb classifier which just labels every tweet as not about pie.

In Wecker terms, this is a 0R classifier. You take the most popular class and always predict the most popular class, and that way, you're guaranteed to get an accuracy quite high. In this case, the accuracy will be 99.99%, a very good accuracy, but actually it's a very useless classifier because, remember, we're trying to find the ones about pie, and none of them are predicted as being about pie.

So that's why we want to use precision and recall, instead. The precision is the true positives divided by the true positives and false positives. So here, the true positives are the ones which are about pie, which correctly predicted about pie. So if you have a system which does get most of those right, then precision will be a high number. Whereas the one we just saw, the dumb one, true positives is 0, so it gets a very bad score in terms of precision.

Recall is the same thing. True positives are things that are counted most, and if you get most of the true positives, if you get most of the ones which are about, then this can end up being quite a high score.

And then for a dumb pie classifier, which doesn't get any of the true positives, then the recall is 0, and precision is also 0, as it turns out. But the accuracy is very high. So precision recall emphasise the true positives, which is usually the thing we're looking for, whereas accuracy counts how many times you got the ones you're not interested in correct as well as the ones you are interested in, which is not very useful.

So precision and recall are somewhat different scores. Typically, you can increase precision, but you reduce recall. Or you can increase recall, but reduce precision.

So you've got two scores. Now, in competitions, like in SemEval, which we'll look at later on, typically, you want to have all the contestants want a single score-- how good are they. So you don't want to have one prize for best precision, another prize for best recall. So there is an F measure which combines a sort of balance between them.

So typically, it's this calculation, and typically the beta is taken to be 1. So the F1 score is 2 times the precision times the recall divided by the precision plus the recall, and this is an arbitrary function, if you like, to some extent. But it gives you a score which balances precision and recall in a neat way, which gives you an overall score, which combines them.

So that's one thing you want to do. Another thing to do in evaluation is how do you get a development test set? There is typically a problem that you train on a training set and test it on a test set. But if you train on a training set, you may overfit on the training set. If a test set isn't, in some sense, very similar to the training set, then you may end up with a high score on the training set and a much lower score on the test set.

One thing you can do is, from the training set, take out a bit of it and calling it a development test set. So you train on the training set, and then you can fine tune it on the dev set, and then report your scores on the test set, and this avoids overfitting. You don't tune on the test set or a training set. You have it tuned separately. So you get a lower score if you just scored on the training set, but a better score than if you scored on a completely separate test set.

On the other hand, we want to use as much data as possible for training, and also as much data as possible for development. So how do we do this? Well, there's this cross validation method. I think you should have come across cross validation before in the data science module.

So cross validation is a way of combining-- using all of a data set for training and testing. But what you can do is you can do cross validation to get a development set and a training set, and then the best model-- the model comes out, overall, basically an average of all of the modules. You can use that as your model and then test it on a test set.

So there's more along that in the textbook. The basic idea is accuracy is not a very safe score, particularly for skewed data sets, where most of the data is one class, and only a small amount is the other class. And it's particularly bad if what you're trying to find is the small class, if you're trying to look for a needle in a haystack.

If you're trying to find the occasional tweet which is offensive or the occasional email which is spam or the occasional tweet which is about pies, then you want to use precision or recall, or even better, use the F measure. And also in those sort of cases, you might want to have a separate development set which you use for training. So you have a training model and a development model and a separate test set.

Next section, I want to look at is, if there's more than two classes, and in this case, you still have the same model works. But there's a slight problem in terms of evaluation because you can have precision and recall for each of in separate classes.

If you have, for example, an email, if you want to separate the emails into urgent, normal, and spam rather than just good and spam, then you can have recall for urgent, recall for normal, and recall for spam, precision for urgent, precision for normal, and precision for spam, and these are different scores. So it may be important to optimise the recall for urgent messages. We don't care about anything else, for example. This depends on the actual user and what they want to get out of this.

An issue in this is that then, if you've got three classes, then we may get two different ways of getting an overall score you could do macro averaging, which is computing the performance for each class and then averaging those. Or you could do micro averaging, which is collecting decisions for all the classes and then computing precision and recall for the overall table.

So remember, that's what we did-- you can take precision and recall overall, or you can compute precision and recall for each individual class and then take the average. And this can end up with different results. Here, we see, going back to the example, the precision for urgent is 0.42.

The precision for normal is 0.52, and the precision for spam is 0.86. The average overall is 0.73. Whereas if you don't take precision individually for each class separately, but just do the precision for the entire data set, then you get a macro average precision of 0.6

Now, it really depends on the particular problem as to which one you take. All I'm pointing out here is that simply saying you should use precision and recall still isn't the overall solution. There are many possible metrics.

You tend to think of accuracy as being the obvious metric. I've just shown that, for skewed data sets, you don't want to do that. You want to take precision and recall, or the F measure, which is the combination of precision and recall. But even that isn't the end of the answer because, if you've got several classes, then there are different sorts of precision and recall, and you have to decide which one is best for your particular application. More on that in the textbook.

Finally, I want to look at-- a problem with Naive Bayes classifiers is that you will reproduce whatever is in the training set. And if a training set has implicit bias in it, then this will come out in the Naive Bayes classifier.

So for example, there's a famous finding that sentiment classifiers can assign lower sentiment and more negative emotion to sentences which have African-American names in them, and this is because, in the training set, it turned out that when there's an African-American name in it when it tends to be a negative emotion. And therefore, this will reproduce in the test set.

But this perpetuates negative stereotypes associate African-Americans with negative emotions. If training data has this in it, then because the Naive Bayes classifier seems to learn combinations of words that come together, then it will just simply reproduce them without meaning anything.

Another common application of classifiers is in detecting hate speech or abuse or harassment. But the trouble is some of these abusive things sometimes will say that mentioning blind people or women or gay people tends to coincide with hate speech. So you may find that hate speech is directed at gay people, and that means negative comes with the hate word but also with the word gay.

So this could lead to censorship of discussion about gay people because nasty things are said about them. That means nastiness coincides with the word gay and other words to do with gay people. Therefore, if you're detecting toxicity, you make sentences about gay people which aren't actually toxic are still being flagged because they have gay words in them. So it may end up with censorship of discussion about these groups.

An actual practical example which is sort of similar is that my wife used to be in charge of sex education at school. And she went on the web to try to find information about sex education and found a lot of the stuff was filtered out by her school's filter because it was about sex. So you can't have documents about sex.

So this is caused by problems in the training data, but also the machine learning systems amplify the biases in the training data. It's also problems to do with the human labels and the resources, like the lexicons, that are used. So what can you do about this? It is an open research area at the moment.

But one thing you can do, at least, is to note down what all the information that you're doing about the training data. This is the idea of model cards. A model card is, when you release an algorithm in a data set,

then you don't just say, here is my toxicity classifier. But you also specify-- you write a document about this where it says what's the training data, what are the sources. Don't just, here is the training data, but how was it developed? What is the motivation? What's the preprocessing? How was evaluated?

When it was developed in the first place, what was this intended use and users? Does it come across as being only for one demographic or environmental group? So document all you can about the data set and the algorithm. Then the users may be aware, or more likely to be aware, of what you might get from it.

I think I'm going to stop now. So we've talked about text classification, looked in more detail about Naive Bayes classifier and how it's using machine learning. We've looked at a worked example of sentiment analysis. It's generally useful for binary Naive Bayes classification, like is it good or is it bad.

It looks very similar to N-gram language modeling in various ways. You might want to evaluate Naive Bayes classifier or text classifiers in terms of accuracy, but this is a problem with skewed data problems. Many classifiers are trying to find a needle in a haystack, trying to find the small class out of lots of data, which is the other class. And for those purposes, precision and recall are better metrics than accuracy, and the F measure, which is a sort of combination of precision and recall into one measure.

If you have three or more classes, then you have to decide, do you want the macro or the micro precision and recall average? And you should still watch out for all these text classifiers, particularly Naive Bayes, which takes into account all of the words, and this can include some inherent bias in the classification. One way of avoiding harms is to document, in great detail, what presumptions were made in defining the classes and in labeling the data sets because then it may be more clear that there are inherent biases in the data.

This is quite a long lecture. I'm going to stop here now, and hopefully, the future-- the other lectures in this module will be a bit shorter and less taxing. Thank you for listening.

[END]