NaturalLanguageProcessing-Lecture04

Instructor (Christopher Manning):Okay. Hi, everyone. And back to CS224n. So this week is gonna be the remainder of the content on machine translation. And today I'm gonna talk more about the kind of alignment models that people have used to learn about how words translate from one language to another language. And then I'm going to imbed inside of that a discussion of the EM algorithm that we use for a lot of NLP tasks, including this alignment algorithm.

So the stuff today is really the core stuff that's the second programming assignment. And then on Wednesday, I do more of a survey of more modern and recent work in machine translation.

So you'll hopefully remember from last time that the general picture is that we've made this sentence-aligned date, whereby in large, the sentences are aligned so that their translations of each other. And we we're essentially then wanting to run this algorithm, which is looking at words that could translate each other, and settle down and learn possible translations for a word, as in this baby corpus.

And so, formally, starting off for what we did for IBM Model 1, that the overall picture we have is we have a lot of French and English parallel sentences. So we have a complete conditional probability of our corpus, that the probability of all of our French stuff times the probability of all of our English stuff. And I tried to add this into this slide that wasn't there last time. And I made a boo boo in my copy and paste already since this should be a product not a sum. So we take that probability, and it's just simply the product of the probability of each French sentence given each English sentence. (Product, right there on the top line.)

So then for an individual French and English sentence, what's its probability? Well, there's, first of all, a probability of how long the translation is going to be. So the length of the French sentence J, given the length of the English sentence I. But then the core of it is that what we're doing is considering over here how different French words align to English words.

And we do that in a slightly funny way. Once we've decided the length of the French sentence, we can just take these positions and say, okay, that the probability distribution over this position being aligned to any English word, including the possibility of it being aligned to no English word, the NULL, and so we have a probability of aj = i. And then given that we make a certain alignment, we then have the probability of having generated [inaudible] given that the English is being. So the probability is fj given ei.

And in particular, in Model 1, the alignment probabilities are just taken to be uniform. So there's kind of no real content in them at all, and so what we really have is these probabilities of fj given ei.

So what we want to do to learn these alignment models is run the EM algorithm to maximize the likelihood of big F given big E, that's ultimately what we want to do. But given that some of the assumptions this model makes about chances of different alignments being uniform, etc., if you actually boil it down, what you end up having to do is what's in this slide. And, as I said last time, essentially, this slide can just be a recipe for what you have to implement for the first half of the second assignment for re-implementing Model 1.

So what you do is that you just start off with fj given ei just being uniform. You just give every English word a uniform chance of being translated by the different French words that you find in your training corpus. And that includes, in particular, that you can have the special English word null generate various French words. (The French words that appear in French that aren't really translations of any English various kinds of function words.)

So what you then do is you iterate through your data and you consider each position in turn in the French sentence, and you calculate a posterior over alignments. So you want to work out what's the probability that this French position is aligned with a particular English word. And the way that you do that is that you use your current estimates of what's the probability of fj given ei. Because, remember, there's no independent information about which alignments are likely. That's assumed to just be uniform probability distribution that we're not changing. And then you just normalize it based on all the different possibilities for English words that you could have.

So this gives you a probability of a particular alignment for position j. So you work these out for all positions j. And then based on doing that, you then increment these counts. And these counts are kind of fractional counts, of according to your current model, how much of an evidence do you have for there being alignments between particular French words and particular English words.

So if you work out here that there's a .3 chance that j=5 aligns to i=2, you then look at what the French and English words there are, and you increment the count for that pair based on that probability that you add .3 to it. So you're adding these fractional counts of different alignments.

So you do this over the entire corpus. And so this is the E-Step of the EM algorithm. And then at the end, you have all of these fractional counts from going through the corpus. And what you do is you just renormalize them to turn these into probabilities. So you have fractional counts for every fj given ei. You total up how many of those there are for a particular ei, and then you divide through by that total again. And so these become the probability of fj given ei. And so then you have new estimates for this quantity. And you go back up to the top, and you go through and do it again. And you keep repeating this for a while and lo and behold you get a translation model.

So the first crucial question for understanding this is why does this work at all?

Here's the argument why it shouldn't work at all.

We've got this simple model. And I've told you that there's no information about the length of the French sentence that translates the English sentence; we just assume a uniform distribution up to some convenient large upper bound like 10,000 words. There's no information, when we went back a slide, there's no information about the probability of aj = i; we just assume that that's uniform and we never change those estimates. And when we start off, we start off by assuming that the probability that fj = ei is uniform. And so it sort of looks like every one of these is the same.

And when you're considering different ei prime down here, this quantity is the same for every possible ei. So it sort of seems like you've got the same thing dividing over some of the same things. And it kind of looks like it'll just come out the same for everything. So you'll do this for your normalization, and all your estimates will still be the same and it won't go anywhere. That's not actually true. So let me pause for a moment. I mean why does this algorithm learn anything?

Student:[Inaudible.]

Instructor (Christopher Manning):Yeah, we are gonna get something out of this. I guess I want something a little bit more precise. Why does that manage to give us something out of it?

Student:[Inaudible.]

Instructor (Christopher Manning):Yeah. So in each iteration – like, yeah, I guess I kind of just say it here, I don't actually have an equation. At each iteration, once I've made these counts, count of fj given ei, I then renormalize them. So I sum these counts for all j to get the total of the count mass of things involved ei, and I divide through it so I've got a new estimate for next time around for fj given ei. So yes, I'm changing this at each iteration through.

But the question is, you know, given that it starts off uniform, and I'm having uniform divided by some of uniform, why do the estimates change? Why do I actually learn something? Any good ideas for that?

Student:[Inaudible.]

Instructor (Christopher Manning):Well, so what is the evidence that I'm using?

Student:[Inaudible.]

Instructor (Christopher Manning):What's that?

Student:[Inaudible.]

Instructor (Christopher Manning):I guess the way of focusing this question is, I mean, what is the crucial source of constraint that this algorithm is exploiting that means it actually learned something?

Yeah?

Student:[Inaudible.]

Instructor (Christopher Manning):I think that's kind of close to it, yeah. I'll call that near enough. And I'll say the answer. So that's kind of it, yeah.

So if here you were considering all French words and all English words in your vocabulary, and you were doing this recalculation, I mean, you would just learn nothing because it would be uniform over some of uniform, which would be the same every time, and you'd get the same thing out every time, and you'd learn absolutely nothing.

The reason that this algorithm learned something is exploits one crucial source of information, which is that you've started off with these aligned sentences. So you have some information if for all the sentences that have the word bongo in it that 80 percent of them have some particular word in their translation then you learn something.

So the crucial reason this algorithm learned something is kind of hidden effectively over what we're summing over here, where we're not summing over every possible word in the vocabulary, we're summing over the candidate words in the English sentence. And so it's precisely because we've actually got real sentences, and we're considering the alternatives in a real sentence, and only those words, that that gives us a source of constraint.

In particular, if we run it on a one-word sentence, where there's a French word and an English word, what we're going to learn from this (if we ignore nulls for a minute there are also the chance of nulls, but if we ignore nulls, and null wasn't a possibility), if we had a one-word sentence, we would learn that the probability of this is whatever it is a. There's only one thing positioned to sum over, so it would be the same quantity a, so the probability of the alignment would come out one. So, therefore, we'd get one count mass, which would be given for saying that that English word is translated by that French word. So we are kind of learning from what we saw in the data.

And you can extend up from there, and say, well, what about if I get a two-word sentence? Well, the first time around, again, if you ignore the possibility of null generating words, well, this is just whatever it is. And then there are two possibilities down here. And, again, both of them have the same estimate, because we started off uniform, and so for each French word you're gonna get half of a chance that it aligns with each English words, and so these counts here are gonna be a half.

And so, effectively, the first iteration through you run this, all you're doing is counting sentence co-occurrence counts scaled by the number of words in a sentence (i.e., if it's a

12-word sentence, and the words occurs once each in a sentence, you're counting a twelfth). So you're just counting fractional counts like that for the number of times that things co-occur.

And that already gives you a lot of information. Because, you know, that's like back to the example of the [inaudible]. And if you just see things co-occurring a lot in sentences that translate each other, that's pretty good information that they have something to do with each other.

So I'll go on then later to some to some of the later models of the IBM models. But let me just sort of get around this point. I'll step back and say a few remarks and then go off and do the EM part.

So in terms of typologies of learning, how many people know about supervised versus unsupervised learning?

Almost everyone. What is this? Unsupervised.

Now, in general, in machine learning does unsupervised learning work well?

I mean, people use it sometimes. But I mean I think it's true to say that throughout most of machine learning unsupervised learning doesn't work that well. I mean, that's being a bit harsh. I mean, so we have the classic contrasts between clustering algorithms, which are unsupervised. You throw in a bit of data and you cluster it somehow and see what you get out versus classification algorithms, where what you do is you hand label data. So for something like email spam, you say, this one's spam, that one's not spam, this one's spam, that one's not spam, and you learn a spam classify using kind of standard classification algorithms.

And so I think it is fair to say, that in most of machine learning that although doing unsupervised learning is kind of fun and sexy and interesting, that for a very large space of problems, you kind of sort of get a clustering that looks kind of tantalizing, like it's capturing some structure. But the results just aren't that good and you don't know what to do with it. And so, by and large, most of modern machine learning has been driven by supervised training, supervised classification methods, where precisely somebody sits around and hand labels pieces of data and you go and learn a classifier off of it.

Student:[Inaudible.]

Instructor (Christopher Manning):Well, I'll say that in just a moment.

So it's kind of interesting that what you're doing here for learning these alignment models is unsupervised learning. And this is actually one of the best cases of large scaled unsupervised learning that actually works really well. So all the state of the art big machine translation systems, that are statistical machines translation systems, there's a lot of other stuff going on. But apart from the alignment algorithms, everyone uses supped

up versions of this, where you're doing unsupervised alignment algorithms. And they just work extremely well once you have tons of data.

But there has also been some work doing supervised alignment algorithms with small hand-done alignments of small numbers of sentences. But, effectively, you can get the unsupervised data at such scale, and the unsupervised algorithms work quite well enough that it's what everyone uses for big MT systems.

And then the question was, well, wait a minute, this isn't quite unsupervised because, well, you started off with translations of sentences, and that's kind of like supervised data, because someone provided those translations and that gave you information. And, I mean, the short answer is, yeah, that's totally right.

I guess I should have narrowed my claim that the part that's purely unsupervised is learning this alignment model. So that learning of the alignment model is completely unsupervised, that, yeah, there's sort of supervision over the overall problem of how to translate a particular source. And that may be a moment when I can say just a general comment that I think turns up a lot in natural language processing. That in machine learning, there's this classic distinction between unsupervised algorithms versus supervised algorithms. And it's a sensible thing to think of.

But a lot of the time, in natural language processing, that the binarity of that distinction isn't very useful. Because a lot of the times in NLP the kinds of things that we'd like to do is work out how we can get a lot of value from a small amount of supervision.

And so a lot of things people do in all sorts of domains, they're not purely unsupervised algorithms, but on the other hand, to do all of the learning supervised would require an enormous investment of resources that may not be likely or practical. And so a lot of the time people would like to do things where you have a little bit of supervision.

And so the idea there is, well, you know, suppose I could easily get my hands on a dictionary, and there are lots of dictionaries around, suppose I let myself use a dictionary but I had no other information about how words that translate in context, and I'll learn all the rest based on the context. That seems a useful thing to try and do because we know we can exploit dictionaries that are pre-existing. So a lot of the time people are wanting to exploit pre-existing resources and easily obtainable stuff, and then build the rest of it out unsupervised.

So at this point I'll deviate off and say a bit about the EM algorithm. But before I do, just sort of to mention a couple of the announcements. A couple of the SCPD students said, gee, it would be much nicer if the quizzes were due on Sunday rather than Friday. And so we're going to have the quizzes due on Sunday. Don't forget that Assignment Number 1 is due on Wednesday. And, now, there's always one or two people that blow all their late days on Assignment 1, but you're really much better off saving them until later. So do get working on Assignment 1, and then we'll be handing out Assignment 2 just to keep everyone busy.

And the final thing I thought I'd just mention is for contacting me and the TA's for questions. We have both the mailing list and we've got a newsgroup. If you've got variety questions which aren't something about your personal problems but are just not understanding how the assignment works, we really encourage you to use the newsgroup if you can because that kind of makes it easy for us to give answers where other people can go looking for answers in case they have similar problems. Of if you get really lucky, maybe one of your fellow students will tell you the answer to the problem before we even get back to it. Well, that's lucky for you and for us, actually. So do think about using the newsgroup.

Student:[Inaudible.] It's almost like a [inaudible].

Instructor (Christopher Manning):I could do midnight if you want, sure. Sure, we can have them due at midnight if you want.

Okay. The EM algorithm. All right. So this is this expectation maximization algorithm, which is often used for unsupervised learning.

So I know it's always the case in CS224n that there's kind of a spectrum between people who've never seen the EM algorithm before and people who have already seen it in three different classes. I mean, certainly, I know for myself, the EM algorithm confused me about the first five times I saw it. So, therefore, I hope that nobody has seen it enough times that they can't learn something from my presentation here.

But, I mean, in particular I hope to kind of work through a little practical example. Because, I mean, I think a lot of the time, certainly for the kind of discreet data that we work with mainly in NLP, that, you know, the EM algorithm theory is kind of confusing. But in practice, when you're actually sort of working through a concrete pace of what you have to do, it's actually kind of straightforward what you work out and how you work with it. Let's hope so.

So what I'm gonna do here is just for discreet data, and for an easy case in these slides of estimating the parameters of an N-gram mixture model. So the general idea is EM's a method of doing maximum likelihood estimation.

So what we want to do is we have some data, and we have some parameters, which are meant to be a model of that data. And somehow we want to fiddle around with those parameters, just like usual, to make the observed data as likely as possible. And, you know, in theory, anything you've ever learned about in optimization you could attempt to use for that problem. But in practice, the EM algorithm is used a lot as an iterative algorithm in places where you can't find the kind of gradients that you need to use other optimization methods.

So in particular, the classic kind of place where you see the EM algorithm is when you're doing things like mixture modeling. So here we have some data, the xi data, which is mixture of some probability distributions. So this is exactly what we saw in the language

modeling part of the class, where we said, well, we can do linear interpolation of several models. We could estimate things with a trigram model or a bigram model or a unigram model, and we're going to combine those estimates together with some waste data.

So in particular, for the problem I'm looking at here, we're now saying the trigram, bigram and unigram model are completely fixed. Their parameters aren't being changed. All we're wanting to do is assign these theta weights that combine them together. So we really only have two parameters in this little baby example, for a trigram, as to how to weight these models.

So for when we have the entire data set, we've got the entire data set x, and we want to say, okay, what's the likelihood of the data, given our parameters? Well, what it's going to be is the product over predicting each word, which is then going to be worked out as a sum over our different component models weighted by the likelihood that's given to each component model by our theta parameters.

So you have this picture where what you want to calculate is a product over a sum. And it's when you see that configuration, that's the classic place where you always end up using the EM algorithm. So the EM algorithm is used when you have incomplete data.

And so when the EM algorithm was originally developed, the idea of incomplete data was you were somehow missing a couple of data points. So that the idea was you were doing some experiment in the lab, and some klutz knocked over a couple of test tubes, and you're missing those two test tubes, and that you'd like to do analysis of your experiment anyway. And so what you wanted to do was reconstruct what would have been in the two test tubes that you didn't actually get to measure the content of. And so that's the case of really incomplete data. That's never the case that we're dealing with, and the kind of models that we're making.

What we're assuming is that there's certain data that we can observe, which is here, the x's, but we're assuming because we have some kind of generative story that underlying that data is other data that we can't observe, which is here being called the y. So the y is referred to as the completions of the data. And so we are pretending our data is artificially incomplete, that there's some underlying extra stuff going on and that you can't actually see but that we're going to be trying to reason with.

So what we're doing here is that our complete data is saying, well, then for the N-gram model there are the actual words that we can observe. But, secondly, we're assuming that there are these y choices. And the y choices, according to our generative story was, was this data point generated by the trigram model, the bigram model or the unigram model so that our generative model is corresponding to our mixture model.

So when you want to generate the next word, you first of all roll your dice to see whether to generate from the trigram, bigram or unigram component models. And then having chosen one, you roll the dice again, and then you choose a particular word to have generated. And so it's the choice of which model to generate from is our y variables. And

so the y variables effectively have three values, 1, 2, 3, as to whether you're generating from the unigram, bigram or trigram model.

Student: Does the y value [inaudible]? Like [inaudible] is the y gonna be the actual testing [inaudible], like the ideal actual claim, or is it what our modeling will be filling in the blank with?

Instructor (Christopher Manning):I think it's kind of what our model will be filling in the blank with. I mean, in particular, when we actually do this, I mean for the standard case, what we're going to actually be interested in with the y, is the probability distribution over its value. So we're using the model to say, well, the thing that we couldn't see, it's 80 percent likely it had this value, 15 percent likely it had this value, and five percent likely it had the remaining value.

Yeah, so our y's here are just one, two, three for unigram, bigram or trigram, so that he likelihood of the completed data, where we assume that the y's were observed, is just that we're generating the probability of the word according to a different component model. So if we imagine completed data, that we knew where the y's were, the sum of the components is gone since a yi variable tells you which component it's coming from.

So an important thing to get straight, in general, is that there end up being two data likelihoods around here. There's the actual observed data likelihood, the probability of x given theta. And then if we assume completions, i.e., we give values to the y, we then have a complete data likelihood, which is the probability of x,y given theta. And the idea of the EM algorithm is that we want to maximize the observed theta likelihood, but we're going to use completions to make that easier to do.

So why is that? That's because if we have products of sums that's hard to work out the observed data likelihood, on the other hand, it's easy to work out the complete data likelihood if we know the y's. So the general idea of the EM algorithm is you alternate these two steps. That you assume some completions field data, and then work out what good values for the theta parameters would be. And then you use those theta parameters to try and work out what the completions are most likely to be, and you repeat over and again those two alternating phases. Which is what is expressed essentially in that slide. But I'll go on fairly quickly to the more concrete part of it.

So the E-Step is we work out expectations which tell us how likely different completions are. And then the M-Step we maximize the likelihood of the complete data, which isn't the same as maximizing the logged likelihood of the observed data that's the least close to it. And that we hope by doing that in an iterative algorithm we'll get better results.

An important thing to notice here is when you're running the EM algorithm, the completed data isn't a constant, it's something that varies with each iteration as you come up with better estimates.

So here's my final trying to do it a bit more formally. So for a certain set of parameters we have the likelihood of the data, which is the probability of all of the data given the parameters, which will then be, since we haven't observed the y's, the sum of the y of the probability of x,y. And in particular, since we assume that each word is generated independently, we have a product over the x's of the sum over of the y's of a probability of x,y given theta.

And so that's precisely why you have this product of sums, which is hard to optimize. And the reason it's hard to optimize, is in general, products of sums give you a non-convex surface, so you get all of these local maxima. And so even running the EM algorithm you have problems with these local maxima.

And so what we're going to want to do is sort of the answer to the question. Since we don't know what the y's are, we're not actually going to want to have the sign the y values, what we're actually gonna want to have is the distribution over how likely the different values for y are. And we're going to effectively treat them like little mini partial data items, where say we have .7 of a data item where y is the trigram model; .15 of a data item where y is the bigram model; and .15 of the data item where y is the trigram model.

Now, around this point is when conventional presentations of the EM algorithm people drag out Jensen's inequality. People heard of Jensen's inequality? Some people? Well, anyway, I was gonna try and do it without doing Jensen's inequality in an easier way. And so this is an easier way.

This way of doing it uses a very simple fact. What is the arithmetic meaning of the numbers one and nine? Five. Okay. What is the geometric mean of the numbers one and nine? Three. And so just like in that example, the geometric mean is always less than the arithmetic mean of numbers. And so if you stay out of log space, you can represent what you can do with Jensen's inequality just with arithmetic means and geometric means.

So if you just remember that the square root of 1 times 9, 3, is less than 1 plus 9 divided by 2, 5. And that's the arithmetic mean, geometric mean inequality, which is the same idea that you use for doing EM algorithm. And so here it is in its general form. So you can put arbitrary weights on the different components. And where these weights add up to one, and that your geometric mean is less than your arithmetic mean.

And we're gonna use this to do the EM algorithm. Because what we have is a product of sums. And if we can turn that inside sum into a product, we then have something smaller, a lower bound, which would be a product of a product, and so that would lower bound what we're interested in and would give us a basis. Because if we can kind of push up that lower bound, then we'd have made progress and we'll be able to have an attempt at optimizing things.

And so where we have this probability of a completion, the probability of x,y given theta, so that can be z,i here. But somehow we want to get in a notion of what can that w,i be to

run this algorithm. And so how can we do that? The way we do that is by introducing an iterative algorithm, where we go through a succession of updates of updating our weights.

So that means we start off with some guess for our parameters. It doesn't have to be a very good guess, it has to be some guess for our parameters, which is called theta prime. And so according to our original guess for the parameters that gave the data some likelihood value. It doesn't matter what it is, it gave the data some likelihood value.

Now what we're gonna want to do is come up with an improved set of estimates theta that is better than our old theta prime. Well, our old estimate was just whatever it was, you know, like it was some estimate of the parameters that gave some likelihood to the data. So this previous likelihood we can regard as a constant because it was just whatever it was. And what we want to do is come up with a new theta that makes the observed theta more likely.

And so what we can do is instead of directly trying to optimize theta, we can instead try and make this relative change in likelihood as big as possible. So the likelihood, according to the old thetas, is some constant. And so if we can make the likelihood given out new thetas, such that that ratio is big, we've improved our estimates of theta.

Student:[Inaudible?]

Instructor (Christopher Manning):Yes. Well, you're training data – strictly speaking for the case of doing the mixture model for a language model, your training data for learning the mixing weights is the validation data. So the validation data serves as the training data for the purpose of learning the mixing weights.

So it's sufficient if we can make this likelihood ratio large. Well, how can we do that? Well, here's a little derivation that's so slowly spelled out that even I can understand it. So we just write that down.

So here's the likelihood of the data given our new model theta. And there's the likelihood of the data given our old parameters theta prime. We can pull the product out the front. We can pull the sum out the front, where the sum is just concerned with the numerator not the denominator. In this step here, we multiply by one. So the terms on the right-hand side is just this number over number, so that's multiplying by one. And then in the final step, we take this term over to the left, and we multiply those two denominator terms to get the product of x,y given theta prime. That's not too hard math. That hopefully makes sense.

But if we look at this, what we've now got on the inside is that we've got a sum of relative likelihoods that are weighted by the probability of y given x, theta prime. So this is something in the shape of the arithmetic mean, geometric mean, inequality, because here we have a sum over some quantities with some weights. So we can apply the

arithmetic mean, geometric mean, inequality right there. And so that's what we do on the next slide.

So this is just we had at the end of the last slide. And if you quickly look. Remember back to the arithmetic mean, geometric mean, inequality, we take the weights by which we're combining things and turn them into exponents and the we have a product. And so we can say that this quantity is greater than this quantity. So now we have a lower bound, which is expressed just in terms of products. So that's kind of pretty.

But in terms of maximizing this lower bound, we kind of don't need the denominator here, because that denominator is just a constant. So for the purpose of maximizing the lower bound, we can throw it away and instead just maximize something that's proportional to lower bound. And so that gives this q equation, which gets called the auxiliary equation in the EM algorithm. Where we've got the likelihood according to the new theta estimates raised to the power of the likelihood given the old theta estimates.

Student:[Inaudible.]

Instructor (Christopher Manning):In our mixture model, what we're guessing generated the individual data points. So the values of y are just 1, 2, or 3. As to did our unigram model generate this data point; did our bigram model generate this data point; did our trigram model generate this data point. I bet it will make more sense when I get to the concrete example in just a minute. This is to just confuse you before I do the concrete example to show you all how easy it is.

So this is sort of a summary of that. And the crucial thing is this here is something we can easily maximize. Because products of products, they're nice, easy things to maximize. We can use our calculus on that, and we can work out how to maximize them. In particular, as I've already pointed out earlier, for these discreet probability distributions, actually the way you can maximize them is just using relative frequency. You don't even have to remember calculus. But if you remember some calculus, you can prove that the way to maximize them is just using relative frequencies.

So we run this algorithm. In each step we guess the completions. We then change the parameters to make the guess completions as likely as possible. We then re-guess the completions, change the parameters again, and do that. And so provably from doing that, and I've sort of almost proved, is that each iteration your theta guesses must make the observed data more likely.

What I've presented here isn't then a complete proof that the EM algorithm works. Because the other half of it is actually showing that my lower bound gets tight as I approach a solution. Which I haven't actually tried to do here. But I'll leave that part out. And let me move instead for a bit to my [inaudible].

So I think all of this makes a ton more sense, and is really quite easy if you see it being done in a spreadsheet. So what I'm wanting to estimate a probability distribution over

words that come after comes across. And so in my original training corpus, I saw comes across ten times. And eight times after comes across was the word as (comes across as a common idiom). Once there was comes across the, and once there was comes across three. And that was what I saw in my training corpus. I saw ten instances of comes across.

So based on my training corpus, I trained, by maximum likelihood estimates let's say, a trigram model. So that's easy. This is .8; this is .1; this is .1; and every other words is estimated at 0. And the I've also estimated a bigram model, and a unigram model. And while I haven't shown you the rest of my training corpus, let's just assume this is my bigram model and my unigram model.

Now, what I want to learn is mixing weights between these three models to make the likelihood of some held out data as likely as possible. So in my held out corpus, I get to see comes across five times. So I see comes across as three times; I see comes across the zero times; comes across three zero times; and I see comes across a once; and comes across one once. So this is now my hold out corpus.

And so what I'm going to do in the EM algorithm is I'm going to, I call them Lambda here, but here are my Theta, or Lambda, my mixing weights between the unigram, bigram and trigram model. And so I'm going to want to set these weights so as to make the likelihood of the held out corpus as good as possible. And I'm gonna do that with the EM algorithm.

So to start off, I have to start off with some initial guesses for what these mixing parameters are. And I guess .7 for the trigram. I mean, it's not gonna really matter what I pick, as I'll show you in a minute. So here's how concretely I run the EM algorithm.

So what I want to do is say, okay, the things I observed in my observed corpus is as, a, and like. Let me work out how likely each model is to have generated them, or how likely it is to have generated them in different ways. So if I see the word as, well, what's the kind of chance of it being generated by the trigram model? The chance of as being generated by the trigram model is there's a .7 chance of the trigram model being selected, and then there's a .8 chance of it generating the word as. So if I run my mixture model, the chance of it generating the word as, by having used the trigram model, is just the product of those two quantities. Does that make sense?

Student:[Inaudible.]

Instructor (Christopher Manning):Yeah. All through this, I'm assuming that the context is this comes across before it. I've got my mixture model. It's gonna generate the next word. What's the probability that it'll generate as when using the trigram model to do it it's .56?

It can also generate the word as using the bigram model. And what's its chance of doing that? Well, it's chance of .2 of selecting the bigram model, then given its selected the

bigram model, this is a bit of a high estimate, but the model isn't sitting here, there's a 20 percent chance of it generating the word as after across. So this is effectively the probability of as given across. So, again, we just multiply those together, and we get .04. And so I fill in all of this table.

Well, if I then want to ask for mixture model what's the total chance of it generating the word as after comes across, well, then I'm summing this column. Because the mixture model is the sum over the probability of generating it with the component models. So these are the overall estimates that my model gives to generating these different words after comes across.

So my initial data likelihood is then to take, since I saw as three times, and the other words once, is to take this probability and cube it, multiply it by this probability, and multiply it by that probability, which is what I do with this equation. So I take this probability cubes times that probability times that probability. And so I get my small number as my data likelihood. And so then at this point, I run the EM algorithm.

So for the EM algorithm what I want is the probability distribution over completions, which is over the choice of whether a unigram, bigram or trigram model generated different things.

And this is actually easy, right. So there's a total .6 chance of generating as next. And of that .6, 0.56 out of .6 chance it came from the trigram model. So I literally take this by that, and that's the probability distribution the trigram model generated it. And, similarly, I just do the same, divide through and say there's a six percent chance the bigram model generated it, a tiny chance the unigram model generated it. So I get here. My chances of different models generating things.

In particular, since the trigram model gave zero probability estimate to generating a after comes across, the probability of the trigram model, being the thing that generated it, is also zero. So this is my probability distribution over my completions. And so from there I work out my expectations.

And so the expectations is then just a count of how often I expect to have seen different completions. So since I saw as three times in my held out corpus, my expectations for this column are simply three times my probability as distribution here. So I'm taking that number and multiplying it by three. And so this column I'm multiplying by three there. And since these words I saw one time each, their expectations are simply the probability estimates.

So this gives me expectations of how many times I used the trigram model to generate as after comes across. If I then want to just work out my total expectation for my held out corpus, as to how often I was in the trigram model, what I then do is sum across the row. But since these are zeros, the expected number of times I was using the trigram model to generate something was 2.7 times. But then for the bigram model and the unigram model, I sum these rows, and I say, okay, my best guesses were I used the trigram model 2.7

times, the bigram model 1.4 times, and the unigram model 0.7 times. And so if I sum those expectations they add up to five. The number of items in my validation corpus.

If for some reason they don't add up to the size of your data set, you know you've done something wrong during the course of it. So up until there that's my E-Step. I've worked out expectations over the completions.

And so now I do the M-Step, which is to change the parameters to make my data more likely. And so the way I do that is I just re-estimate the mixing weights based on my current guesses of how much I use different models.

So it looks like I used the trigram 2.7 out of five times, i.e., slightly more than 50 percent of the time. So I do that division, and I get a new estimate for the mixing weight of the trigram model, which has shrank. Remember I started at .7, but it shrunk to .55, and then these two weights have grown.

And intuitively that makes sense, right? Because for my held out corpus, since the trigram model can only generate three of the observed five words, because it's giving a zero estimate to the other two words, it just can't be the best thing that gives highest likelihood to the data, to give a mixing weight of .7 to the trigram model, because at most it can have generated 3/5 of the data. So it sort of seems like its weight must be less than or equal to .6.

Student: So do we do this for every [inaudible]?

Instructor (Christopher Manning):So you're kind of getting to a point where I'm making this example sort of easy. So if you're doing this for real, you're certainly, yes, summing over every history to work out good mixing weights. And then there's a question of how are the mixing weights done? And I sort of mentioned in class that there are a couple of possibilities for that. One possibility was if you're just doing the simplest form of linear interpolation. Your mixing weights are you just have one for each order model. And so then you sum this over all possible contexts, and then in the same way you just re-estimate and you get weights on the trigram, bigram, and unigram model.

There are more subtle ways to do it, in which you kind of put different estimates based on things like the count of the context so you have more parameters in your mixture. And you can explore that if you want for the homework. That makes things a little bit more complicated but it's in principle the same. What you don't want to do is actually have different mixing weights for each possible history.

In real life, if you have different mixing weights for each possible history, it's hopeless because then your mixing weights are being estimated with just as, or even more sparse data than your original distributions, and so the mixing weights you learn are completely over trained. Because if in your held out data you saw only things that you saw in the trigram model before, you'll put all the weight on the trigram model and zero on the other

models and then you're not achieving what you'd like to achieve of having this doing smoothing for you between the models.

So at that point, I've changed my estimates for my mixing weights. And then I can go back to the start and I can work out, okay, for my new model with these weights, which I've just again put down here, I can work out the likelihood of the observed data, i.e., the held out data according to my new model.

So it's exactly the same as before. I work out the chance of the trigram model, the trigram component generating as, which is simply the weight here times the probability in the trigram model. And then I sum these. And now the probability of as is 0.5, which is lower than before. The probability of a is this and this. And if I work out my data likelihood in the same way as before, the crucial thing to notice is the hold. My data likelihood has gone much bigger than it was before. It used to be 4.9 times 10 to the minus 5, and now it's 6.4 times 10 to the minus 5. So my data likelihood has gone up by 30 percent. So fiddling these Lambda weights is doing something useful for me.

Student:[Inaudible] the actual data [inaudible].

Instructor (Christopher Manning):Right. This is the actual data likelihood.

Student:[Inaudible.]

Instructor (Christopher Manning):No, you have to see an increase. But because the completion lower bounds the actual data likelihood, it – I think that's right. Yeah, no, you really have to see it if – well, strictly, it could stay the same. That it has to stay the same or increase. You can't possibly have a reversal in the actual data likelihood.

Yeah, but this is just working out the actual data likelihood of my hold out data. I'm just working it out by taking a product of the sum, which is the sum is going down these columns.

So at that point I just repeat. So I work out probability over completions again. I then work out expectations to be in different states again. I work out probabilities for my mixing weights by, again, just dividing through these expectations via the total. And, again, I get these different weights. Where, again, the probability estimate of using the trigram model shrinks a little bit. It's not nearly as much as the first time.

But I do it again, and lo and behold my data likelihood has increased a little. It's now increased from 6.4 times 10 the minus 5 to 6.6 times 10 to the minus 5. And I keep on doing this, and now it's 6, 6, 4, 4 and 6, 6, 6, 6 and 6, 6, 8, 0. I was able to do this by cut and paste. I didn't have to fight this all in. And I keep on going, 6, 6, 8, 9; 6, 6, 9, 4; 6, 6, 9, 8; 6, 6, 7 – one of the things that you should learn from this about the EM algorithm is convergence of the EM algorithm is pretty slow. You kind of have to run it for a while and go through iterations. But, nevertheless, the likelihood of the data keeps on going up, and I keep on jugging by. And essentially what it's converging to is that the weight on the

trigram model is a half. The weight on the bigram model is .4. And the weight on the unigram model is a 1/10.

So let me just ask kind of a couple of questions on this. I chose these mixing weights at the beginning. What would happen if I had sort of started with the bigram rate as 0, and the unigram weight as 0.3? What would I expect to be the end result after I'd run this convergence?

Student:[Inaudible.]

Instructor (Christopher Manning):No, that is not the right answer.

Student:[Inaudible.]

Instructor (Christopher Manning):Yes, that is the right answer.

So your answer is right most of the time, but it's not right if something is zero.

So if something is zero, it's got no chance of generating particular data points. And so when I work out what's the probability of having used the bigram model to have generated different things, it's still zero because it would have no chance of generating them. So the expectations is zero, and it stays zero. And, well, as you can see, I had these strings hard coded rather than them dynamically generating the truth. So this one's growing, but that one isn't changing at all. And if we run it to convergence, the weight is just being split between the trigram and the bigram model.

Student:[Inaudible.]

Your answer is right most of the time, so I didn't really have to start with these numbers. If I started off with 0.1, 0.6 and 0.3, it actually makes no difference. And if you go down to the bottom you're getting the same, kind of a half, four-tenths, a tenth distribution.

So a lot of the time what you find with the EM algorithm is that if you choose extremely extreme values of these parameters, that you can get to bad local maxima and you'll get different answers in the time. But fairly often there'll be a fairly broad range of middle values of starting parameters and that'll converge the same local maxima.

Student:[Inaudible.]

Instructor (Christopher Manning):That certainly should be the case.

So if I choose any kind of middling values, my data likelihood ends up as about 6.705 times 10 to the minus fifth. If I go to the case where this was zero, and – well, here's another trick. It doesn't actually matter on the first iteration of what you give it as a probability distribution because when it re-estimates things it'll turn it into a probability

distribution. So if I start off with the middle one zero, I guess my data likelihood never gets nearly as good, yeah. Because the bigram model is useful.

Student:[Inaudible.]

Instructor (Christopher Manning):I can try it. I think that will be bad news. Minus 0.3. Student:

[Inaudible.]

Instructor (Christopher Manning):Yeah, you don't actually want to have zeros. Providing they're all positive you'll be in good shape.

So this spreadsheet is on the web page. You can play around with it more if you want to. Another thing that you can play around with is you can also just change these underlying models. So if you, for example, suppose the bigram model had given a slightly higher probability to like coming after comes across. So suppose it was 0.02, that that estimate is still less than the unigram's estimate. But, nevertheless, if you run it to convergence, what's happening as it runs to convergence is the unigram's weight is converging to zero, and weight is just being split between the trigram and bigram model.

But it turns out that even though the unigram model assigns a higher probability to like, and even though the word like appears in the held out corpus, that the optimization essentially finds it can just get a lot more value in terms of increasing likelihood by giving weight to the bigram model. Because the bigram model gives much higher likelihood to as, and much higher likelihood to a, so that data likelihood is actually maximized by giving just more and more weight to the bigram model, despite the fact that the unigram model is more likely to generate 1/5 of the training data.

So you guys can play with that. Let me just for the last couple of minutes then say a moment about what comes after Model 1, and is then the heart of the rest of what you have to do for the assignment. So the first part of the assignment is implement that and get Model 1 to work. And then the second part of the assignment is we go onto IBM Model 2. So there's the sequence of IBM models the rest of which I will talk about next time.

But the idea of these models is to make further refinements to the probability models that make them more complex, but also make them do a slightly better job at modeling how human languages work.

So if you look at human languages as translations, commonly what you get is these alignment grids kind of look like this example I showed earlier. That there's various stuff going on reflecting the fact that different languages put words in different orders from each other. They don't all use the same word order. But generally it tends to be the case that you sort of get this sort of overall kind of, you know, goes along the diagonal.

And if you know something about languages, you might say, well, wait minutes, that's not true of some languages. There are some languages that have very different words orders. In English you get subject, verb, object. There are some languages that have completely opposite, they have object, verb, subject, they should get completely the opposite word order.

And that's a little bit true. I mean, so for language pairs with similar typology, something like French/English, you get a very strong diagonal effect like this. Whereas if you kind of think of languages that have very different word orders, the diagonal effect becomes less strong, it's true, but it normally is still there.

And the reason it's still there is in written pros, or even when people are speaking, most sentences, except in kindergartener readers, most sentences aren't, The boy sees the bal. The boy grabs the ball. Right. They're not those s, v, o sentences. What you get is these long multi clause sentences of, While Jim was listening to the morning news, his wife called out it was time to make the coffee, and he went over and started the machine. And so that even if each of those individual clauses sort of has its order flipped in the language with very different word order, normally the translation will still have those three clauses in the same order. Because that's basically just the iconical order in which events happen. And so you'll still get a kind of blocked diagonal structure.

Student:[Inaudible.]

Instructor (Christopher Manning):I mean there is a bit of bias, you're right. Because clearly in some sense it's more mental effort for translators to reorganize the clauses than just to do them one at a time. Although, you know, translators really will do that if it's appropriate to do sometimes. But I agree that's a source of bias.

But it's something more than that. There's also – I mean this is the kind of thing people talk about in functional linguistics, there is just an iconic ordering of events. And although you don't have to present events to a reader in the order in which they happened, and sometimes people don't, and then you use things like, before clauses, where you kind of invert the order. But by and large 90 percent of the time people do present events in the iconic order, and that exists independent of it being a translation. At any rate, we're learning from translations.

Okay. So what we want to do in Model 2 is then to say, well, we somehow want to capture that translations normally lie along the diagonal. And so in the programming Assignment 2, the general idea is when you're translating a word here, that it could be translated by any other word. But a word somewhere around here should be more likely, and a word somewhere up here should be less likely to be the translation of it. And so I'll put that into our models, which gives it a bias on absolute the kind of learned roughly diagonal translations. And you'll see that that makes the MT system work a lot better.

Okay. I'll stop for now.

[End of Audio]

Duration: 74 minutes