NaturalLanguageProcessing-Lecture08

Instructor (Christopher Manning):Hi. Welcome back to CS224n. Okay. So today, what I'm gonna do is keep on going with the stuff that I started on last time, going through all the technical details of maximum entropy models. And then on Wednesday, I'm gonna get more solidly back into NLP topics as to how they're used for various kinds of information extraction and text mining. Are there any questions up until here, or do I just plow straight on? I'll plow on. So what I wanted to start off doing today is actually talking about this idea of maximum entropy models and why in NLP people call them maximum entropy models. What I introduced were, essentially, these exponential or log-linear classification models. And I mean, I actually think, overall, that's the most sensible way to think about them and connects more closely with what you're likely to see in other classes, like statistics. But in NLP, they're always called MaxEnt models. And the reason for that is largely historical. And it reflects this tradition of a lot of probabilistic ideas coming into NLP from speech. So I guess the idea of maximum entropy goes back to a physicist, Jaynes, who was actually, at one point, a physicist at Stanford in the '60s. And then he moved to St. Louis. And so Jaynes thought a fundamental principle for how to be finding probabilistic probability models was this principle of maximum entropy. And so the idea there was that you could think of the space of all possible distributions, and for any distribution, you can look at whether it's a very spiked, uneven one, or a sort of smooth one. And Jaynes' idea was, essentially, well, if you have a lot of evidence for a very spiked distribution, then you can choose a very spiked distribution, and that's correct. But if you only have a limited amount of evidence, what you should be looking for is as smooth as possible a distribution.

I guess I just looked up his book this morning and saw a quote at the bottom is the quote that starts the chapter where he talks about the maximum entropy principles and it says, "Ignorance is preferable to error. And he who is less remote from the truth believes nothing than he who believes what is wrong," which Thomas Jefferson apparently said sometime, I'm not quite sure when. So this idea of trying to keep the probability distribution smooth and uniform, except when you have a lot of evidence otherwise is this idea of maximum entropy. And so Jaynes propounded this version, which kind of fits into a Bayesian statistics framework. And somehow, that idea then got into the information theory community, which is largely an electrical engineering community, which got to the speech community, which then got to the NLP community. And that's sort of why it's still called maximum entropy models in NLP. But apart from just that little bit of history, I think kind of thinking about them as maximum entropy models in terms of what they're doing actually is a useful alternative way to think about things. So I'm going to present a few slides on that. So you should remember from when we looked at language models that we had this idea of the entropy of a distribution. So it's how uncertain that we are about what's going to happen so that we can look at our surprise, which is this quantity of the log of the inverse of the probability. And then, when we've got a whole distribution, we're taking our expected surprise and that's our entropy. And so the thing that's maybe confusing for the moment here is in all of our work on speech recognition language models, etc., the whole idea was to make our entropy as small as

possible because the assumption was we had a good model if it did a good job at fitting whole-out data, i.e., it had lower surprise when it saw further data.

So our goal was to find minimum entropy models or strictly minimum cross-entropy models. So it seems kind of a bit perverse that Jaynes' principle is the opposite of saying, no, we want maximum entropy models. But the sense is more specific than that. The sense is to assume that what we're going to want to do is have some facts about a reference distribution. So there's some things that we want to model, so we might want to model a probability of a certain event being high. But it's in the places that we're not really sure what we want to model, that's where we're going to aim at maximum entropy. So in some sense, the idea of the maximum entropy principle is kind of the same as what we put under the title of smoothing. That to the extent that you are uncertain about regions of the probability space, that you should be making them kind of smooth and uniform and not making strong predictions about things that you're not very confident about. So we have these two principles. In some sense, fit our data, and in some sense, keep things kind of smooth. And so what a maximum entropy model does is say we've got some constraints on possible distribution, and we will place these constraints, in terms of expectations, on the features that we define, just like last time. So if you remember last time, there were these features and there were empirical expectations of the features and then our model had expectations of the features. So the simple kind of maximum entropy model, which is the only kind that we do, is we have a quality constraints, which last time we derived in terms of a little bit of calculus from our model class. But here we just start with them at the beginning. We want to constrain our model to say that the empirical expectation of a feature is the same as the model expectation of the feature. And then, given those constraints, we want otherwise for you to find the probability model that has the maximum entropy.

And so the idea here is this is a simple example, the classic simple example of entropy, that if you're tossing a coin, how much entropy is there? So if the coin always comes up heads or tails, there's no entropy. If it's an unweighted coin so it comes up half tails and half heads, you're here in the middle, and you have maximum entropy. And so what we're going to do is place constraints on the distribution. So here I placed a constraint that the probability of heads is 0.3. And once I place that constraint, I'm in the restricted subspace. Now in this case, the restricted subspace isn't very interesting, because since I only had one parameter, my restricted subspace is a point. And so I'm right there. So there's nothing for the maximum entropy principle to do. But in general, what we're going to want to do is we're going to have a big probability space and we're going to put some constraints on it, but not too many. And then the maximum entropy principle will smooth the rest of the distribution. But this principle still applies. If you actually pin down in your constraints every fact that you saw in your training data, you'd get exactly the same effect, that there's nothing for the maximum entropy principle to do anymore. You're at a specific point in the probability space and nothing can be smoothed. So the interesting part depends on having somewhat general constraints, and then smoothing the distribution in between those constraints. So in general, adding constraints with the features that we defined, it lowers the possible maximum entropy of the model, but it raises the maximum likelihood of the observed data. And so it brings the distribution

away from uniform and closer to the data. What we want to do is find a balance point between those two things. Here's a fractionally more interesting example, since we've now got one more dimension.

So if we just have – I think when I explain this example, it often confuses people. In the start of my model, I have two factors. One factor for the coin coming down heads, and one factor for coin coming down tails. And at the start, I'm placing no constraints on those factors at all. They're not yet probabilities. If you just start with a factor for heads and a factor for tails, and they're not constrained to be probabilities, the maximum entropy solution is actually right there, where that point is one on e, because if you draw your little minus x log x graph, that's the point that's the maximum entropy point. But normally, that's not what we do. Normally, we place a constraint that says that we want to have a probability distribution coming out. And so we say, look, the solution you find has to be on this line, where heads and tails are related to each other as being complimentary probabilistic events. And so if you restrict yourself to this line, then you get the picture of an entropy curve that you normally see in the textbooks, where the maximum is when you're at 0.5 probability of heads and tails. And so then we get our second constraint, like before, of constraining the probability of heads to 0.3, and then again, we've got down to just a single point and there's nothing for us to do. So maybe I should then go on to a more linguistic example, which sort of starts to illustrate how we might do something useful. So what we're wanting to do is predict simply the probability of different parts of speech. So this is just look at words, 80 percent of them nouns, 30 percent of them verbs. It's not actually specific to words yet. So my simple example here, we have six parts of speech, noun, plural noun, proper noun, plural proper noun, the s form of verbs, like "runs," and the past tense form, "ran." And I've done a very small data count, and this is how many counts of the different ones I have. So again, if I just say, okay, I want to assign a value to each of these cells so as to maximize entropy, I'd again be putting the value one and e on each cell. But if I say, look, I want a probability distribution to come out, that the expectation that I place over it being some member of that set is one per data point, so I'm always going to get something of that set, then the maximum entropy distribution is the uniform distribution. So here's my uniform distribution, which is the maximum entropy distribution, which is a probability distribution. But it looks nothing at all like my observed data. Yes?

Student:

[Inaudible]

Instructor (Christopher Manning):This is if I want to put numbers corresponding to each of these events, and those numbers aren't even constrained to be probabilities, what values for those numbers would maximize entropy?

Student:

[Inaudible]

Instructor (Christopher Manning):It's somewhat analogous to probability, but not constrained to sum to one. Maybe I should leave this out. I think it always causes confusion. Okay. So anyway, we get down to here and now we have a uniform probability distribution. But if we want it to look a bit more like our observed data, we can add some extra constraints. So something obvious that we can notice is that nouns were much more common than verbs in my original data. So in my original data, seven-eighths of the data was nouns. And it seems like a good constraint to put in would be the expectation of seeing some kind of noun on each trial is seven-eighths. That seems like a good one to put in. So I put in this feature. So I've now got a feature that on a single trial, the expectation that you should see one of these is one, and that turned into a probability distribution. Now I add in the second constraint, the expectation that you see one of these four is seven-eighths. And so once I do that, I can then again ask, what is the maximum entropy solution? And now the maximum entropy solution is this one. So my constraint of this being seven-eighths is observed, but otherwise, the maximum entropy principle stills says spread probability mass around uniformly. And so I get two thirty-sixths each year. So I said seven-eighths, but it should have been eight-ninths. Sorry. And then I get eight thirty-sixths in each of these cells. But if I go back to the data distribution, I can say, well, in this data distribution, what's really common is proper nouns. So maybe what I should do is add extra constraints saying for each trial, the expectation that it's a proper noun is two-thirds, 24 out of 36. I'm again matching one of my empirical expectations from the training data. And so then I get out this distribution as the maximum entropy distribution. Yeah?

Student:

[Inaudible]

Instructor (Christopher Manning):It doesn't end up the same as a generative model. And maybe that'll become clearer in a few more slides. Maybe I shouldn't try and ad lib too far than that. It's different from – I kinda see what you're saying, which is, well, you're picking out these bigger events, and those bigger events are just having their relative frequencies matched. And that's true. But the interesting part will come when you have interesting constraints that interact in various ways. And then finding the maximum entropy solution doesn't neatly correspond to anything you get out of a generative model. Maybe I should leave that for a moment and come back to it in the following examples. So now I have three constraints and you should be able to note that all of them are observed. So if you sum up all of these, they sum up to one. If you sum up these four cells, they sum up to 32/36. And if you sum up these cells, they sum to two-thirds. So all the constraints that I placed on the model are observed. But apart from that it's still making the model as uniform as possible. So these are estimated uniformly. These are. And those two cells are. And we could keep on adding more constraints to the model, but again, it would be kind of like the previous example. If we started adding anything much more into the model, the particular cells would be pinned down to their empirical expectation and there wouldn't be anything left for the maximum entropy principle to do. This is a little side note, but one good bit – I just sort of asserted last time that maximum entropy models are convex, and therefore, you could use any convex

optimization procedure to solve the models. If you actually think about it in this entropy space, it's very easy to see that you're in a space of convex models. So convex is this property that the function value at any linear combination is greater than or equal to the linear combination of the function values.

And so if we're thinking in the entropy space that this function of the entropy is a convex function, like the picture I showed before, so then when we take the sum of the convex functions, we've still got a convex function. And then when we find a feasible region subject to constraints, we're then finding a subspace of a convex function, which has to be convex. And so, therefore, our constrained entropy space here remains this convex function that we can do maximization on quite straightforwardly. Now these are the examples that might help to answer the question of why this isn't the same as a generative model. And so these next few slides go through a having overlapping and interacting constraints in various ways and what happens then. And the basic thing that I want to show is in the first instance, this is coming back to my traffic lights and senses examples as to why you get MaxEnt models giving a better handling of overlapping features than does a generative model. These are joint models over a space of four outcomes. So the outcomes are kind of either big A big B, or big A little b. So we have four possible outcomes. And this is our empirical data. We've got two each of these outcomes and one each of those outcomes. And so for this data, what we could do is build maximum entropy models. And so the simplest maximum entropy probabilistic model is just to say, well, we're putting a probability distribution over this space. So if we do that, it'll give probability of one-quarter to each event outcome. And so then the obvious thing to do is say, well wait, outcomes with "capital A" are more common. And so what we can do is put in a feature that covers these two outcomes, and say that per trial, the expectation that you'll see a "capital A" event is two-thirds. And then if you do that, instantly you find the maximum entropy solution subject to that constraint, which precisely matches the observed data. Cool.

The bit that's more interesting is, then, what happens if I throw in a second features, the blue feature? And the blue feature is actually exactly the same as the red feature. The blue feature picks out this event of the "capital A" events. And what I showed last time in the senses examples is that if you had two variables that were picking out the same thing in a naïve Bayes model, that the Bayes model went wrong. It made false independence assumptions and that caused the probability estimates and the predictions to go wrong. So the crucial first observation is that if you do this in a maximum entropy model, nothing goes wrong. You get perfectly correct answers. So what happens is that for each feature, the red and the blue feature, you introduce a weight. Here there was just a weight for the lambda A feature. And here we have two weights, one corresponding to the red feature, and one corresponding to the blue feature. And our maximization procedure is going to adjust those weights up and down. But how's it going to adjust the weights? It's going to adjust the weights so that the model expectation of these cells equals the empirical expectation of these cells. So it'll adjust these weights so that the outcome is still one-third, one-third, one-sixth, one-sixth. You'll get exactly the same predictions as if you had only the red feature. And you can keep on doing that. You could have ten features that picked out the "capital A" class, and the predictions of the model would be the same.

It doesn't have the problems of naïve Bayes model. So formerly what happens is this model will find a solution where the lambda value for the red parameter plus the lambda value for the blue parameter will sum to what the lambda value you would have been if you had only a single parameter. So if I don't say anything else, for this problem, it will find a correct answer, which gives these probability estimates, but this problem is actually under-constrained, because there's actually a whole subspace of values you can set these parameters to that would give that solution. Okay. And why that is useful in the kind of models that we build is that the kind of models that we build, we just sort of throw in all kinds of features that seem to be useful. And commonly, when we're throwing in those features, we throw in a lot of features. And a lot of those features overlap and are extremely correlated with each other.

And so the kind of effects that you get all the time is here's my named entity recognition example again. So I tagged the previous word as "other," and I'm looking at "Grace" and I want to say that it's part of a location. And it's a proper noun and these signature features, I don't think I mentioned last time, the signature features are kind of giving the shape of the word. So this "capital X" "little x" is meaning this is a capitalized word consisting of letters, as opposed to numbers or something like that. So we have lots of features, but among the features that we have is we have what does the word begin with feature. So this feature says this word begins with a "capital G." And we have a current word feature. This word begins with "Grace." Now, by itself, "Grace" is kind of an indicator of being a person name, because if you just see the word "Grace" out of context, it sounds like a person name. So you'd want something here to be voting for person. But these features are clearly strongly overlapping, because the cases in which the "Grace" feature is turned on is actually a subset of the cases when the word begins with "capital G" features is turned on. And similarly, whenever the "Grace" feature is turned on, the current signature is capitalized letters feature, it's also always turned on. So you have lots of these features that are highly correlated or even kind of completely overlapping with each other. And so what's happened in the model is that it's adjusting these weights to give the best matching of the empirical observed data in terms of expectations. And it gives allowance for the overlaps in adjusting those model weights. So what you're finding here is that most of the productive power for something being a person name is actually coming from the fact that it begins with "capital G," which I guess reflects that the fact that there are plenty of common names, like Greg and George and so on, that start with a "capital G."

So that's getting a significant vote for person, whereas actually, knowing that the word is "Grace" is giving you very little additional predictive value, beyond that it starts with "capital G." So the first message to take away – sorry, yeah?

Student:[Inaudible]

Instructor (Christopher Manning):These are totals, yes. Right. So for each feature, it has a weight, so each feature gives a vote for "person" versus "location." And there are other columns for it voting for "other" and "organization" that I just haven't shown. So each feature is giving a vote as a feature weight for what it thinks of the different classes.

And in my linear classifier, I just sum those votes to work out the outcome. And so the overall outcomes is strongly favoring "Grace" being a location here. But so in the classifiers, these have weights and they're just summed to give the prediction. So the adjusting happens in the optimization procedure in how these weights are chosen. And how the weights have been chosen is adjusting to not give double prediction. So in other words, every time that you see the word "Grace," you also saw this. And so, therefore, if you were just kind of independently putting weights on features, like a naïve Bayes model, that this feature should have a high vote for person just like starting with "G" does. But because of the optimization procedure, doing the best job to match expectations, this feature has actually been given a low weight. So the first message was if you have completely overlapping features, they do no harm whatsoever. You can just throw them in freely and the model that you estimate will be exactly the same. And in practice, when you throw in lots of features, you regularly find that you do have some that are just completely overlapping that pick out exactly the same subset of events. So for instance, if you have longer substring features, like you have sort of first three characters of the word and first four characters of the word, they'll just be some cases in which you only observe one word with a certain initial three letters, like "zyg" for zygote or something, as the only thing you observe starting with "zyg." So that the "zyg" and "zygo" features will have exactly the same places where they're active.

But the more common case is that you have features that overlap in when they're active, but are not actually identical. And so the message that you want to take home there is that using MaxEnt models kind of helps in that instance, but it doesn't do magic. So suppose now our empirical data just looks like this. We've seen one data point in each of these three cells. And we've seen no data point in the fourth cell. So if we just start with the constraint that we need to find a probability model, our expectation for each cell for one trial is one-quarter probability for each. If we put in the red feature just as before, well, the red feature's expectation is, again, two-thirds per trial. And so the model we'll build is this model. And they'll be a single weight that votes for these outcomes. And we'll get that picture. Well, it seems like what we want to do is predict that you also tend to get "capital B" outcomes. So it seems like the obvious thing to do is to throw in a blue feature like this that says, you find "capital B" in the data, let's prefer "capital B" outputs. Well, if you do that, you then have the lambda A weight voting for getting a "capital A" outcome, and you have the lambda B weight voting for getting a "capital B" outcome. And the predictions of the model, so this is kind of true of any logistic regression style model, is you get an additive effect in log space, where if both of those are true, the two weights add together and make the outcome more likely. So the predictions of the model are actually now this, that you've got a one-ninth chance of seeing little a little b, you've got a two-ninths chance each chance of seeing these cells, and a four-ninths prediction of seeing that cell. And you can work it out if you want. That's the maximum entropy solution, which doesn't exactly look like this. So if features overlap, something useful happens. But it's not that the model does everything useful for us. What effectively has happened is some expectation has been stolen from this cell and shifted up here, so that the expectations of the model are correctly modeled, namely, two-thirds of the stuff is here and two-thirds of the stuff is here. But it's no longer spreading it around uniformly, because you've got two features active at this point. That make sense? Okay. Well,

suppose we want to fix that and actually get our model expectations correct, what could we do about that?

Student:

[Inaudible]

Instructor (Christopher Manning):No, you get it for all features.

Student:

[Inaudible]

Instructor (Christopher Manning):Right. So there are various ways to fix it and one way you could conceivably fix it would be to make the values of the features non-binary. But I mean, this isn't really effective. The fact that you get this behavior is a general fact about these kinds of MaxEnt models or logistic regression style models in statistics. It isn't a fact about whether they're binary or not, that if you have multiple features are the active, you get this summative effect inside log space. And I mean, if your features are of this kind, you're gonna get this kind of doubling up effect, and it wouldn't matter if the feature value is just binary 1, 0, or whether it's got some other values of 275 or something, you'd still get this doubling up effect.

Student:

[Inaudible]

Instructor (Christopher Manning):Okay. So these are meant to be illustrative examples to think about what happens when you place constraints on data. But precisely what we will want to do when we build models is to say, and I guess I tried to say something like this last time, that what we do when we design features for a model is, okay, I see in the data that a lot of the times you get location after a preposition like, "in." So why don't I put in a features into my model saying the previous word is a preposition and the current word's class is "location." And then I think a bit harder and say, well, usually that's true for proper nouns, so maybe I could make another features, which was the previous word's a preposition, the current word is capitalized, and the class you should assign to it is "location." So that's how people typically go about thinking of features and designing them into the model. And I guess these are extremely simple, abstracted, illustrative examples, but what they're meant to be illustrating is, well, if you go through that kind of thought process, you end up with these overlapping features of various sorts, and what's actually going to happen in terms of the parameter weights of you model and the predictions of your model when you do that. From this slide, this picture, it seems, wasn't perfect because our model is now predicting that seeing "capital A" "capital B" is twice as likely as seeing these cells individually, whereas in the observed data, each of those three cells was equally likely. And so the question is, how might we fix that? And the standard way of how you might fix that is to add what are called interaction terms. So

interaction terms are where you're insisting that a pair of features be true at the same time. So we have the same red and blue features as before, and we add a third feature, which is true only if it's "capital A" and "capital B" at the same time, i.e., the third feature just picks out this quadrant here.

And so this feature's empirical expectation is one-third. And therefore, the model will be built so that the model expectation is one-third. And then, the lambda weights for the red and blue features will be adjusting so that each of these cells now has a value of one-third. And how will it do that? The way it will do that is that these lambda weights will actually be increased to get this value up to one-third, rather than two-ninths. And the green feature will actually have a negative weight, so it's pushing down the intersection cell to give the right answers. So you kind of can get – because you get these feature interactions, you kind of makes the weight of features kind of hard to interpret, because it depends how they go together with each other. So it's not that being "capital A" and "capital B" is bad. Actually, one-third of the observed data is like that. But nevertheless, the interaction feature is effectively being a correction features, because the red and the blue features are overestimating the probability of this cell. So it gets a negative weight, so it's pushed down the cell's expectation.

Student:

[Inaudible]

Instructor (Christopher Manning):These little models that I'm drawing, my pictures, these are just point models, and I will come back in a moment to the kind of predictive classification models that we use for prediction and their relationship to these. But when we build models, essentially, we've got some training data, and we are choosing features that allow us to pick out well what occurs in the training data. So that if we see certain patterns in the training data, we're wanting to choose features that will pick out those patterns. So if we notice that person names being with "capital J" kind of often, we want a feature that the word begins with "capital J" and it's a person name. So that will be a feature. And it will allow us to model our training data, which is observed data, better, i.e., to fit that data better. And then we want to kind of do the usual job of fitting the training data reasonably well, but not overfitting the training data by precisely matching it so well that our data doesn't generalize at all. So throwing in the green features, let's get the model expectations being precisely equal to the empirical expectations. So that's kind of good. But you might also notice that if we wanted to, if we just defined the feature, the yellow feature, which is it's either "capital A" or it's "capital B" and "little a," i.e., disjunctive feature. That we could have had one feature that would have gotten our training data exactly right, because the maximum entropy principle would have evenly split the probability mass over this one feature, and so it would have had a one feature model that could also predict things just right. And so what the take away from that for the linguistic case is that you can – and people regularly do – build models with thousands and thousands and thousands and thousands of features and then they discover that they could make them even better by putting in interaction terms. And so they can

then add in millions of interactions terms, because the number of interactions terms is the square of the number of features.

And it will make the model better. But it turns out in a lot of cases in linguistic modeling, you can get enormous value by effectively cleverly defining disjunction features that pick out natural classes in the data. So typical disjunction features are things like the word starts with “capital J.” That’s a disjunction over all words that you’ve seen that start with “capital J.” Or the word contains a dash. That’s a huge disjunction over a big class of words. And if you have those kinds of features that pick out natural linguistic classes that they can estimate with a small number of features very well what’s going on in the data, providing they’re the right classes, i.e., the right natural classes for modeling the data. So one more note on scale of this. So for doing logistic regression style models and other places like statistics, it’s kind of normal that you have a model where you’ve measured five things and you have five features and you build a model over those five features. And you then say, do we need any interaction terms in our model? And you do some kind of search, which is normally a greedy search. But nevertheless, you search the space of interaction terms and consider what interaction terms to put into your model. There’s a problem with that, which is the space of possible interaction terms is exponential in the number of features. So in NLP-like context, where there are already thousands or hundreds of thousands of basic features, the idea of doing any kind of search for interaction terms that you might need isn’t very appealing. So most commonly, in practice, what interaction terms of put into the model is actually just done by human intuition. There’s then some empirical testing of let’s try putting in these kinds of interaction terms and see if they help the model, and if they don’t, take them back out again. But in terms of which candidates to try, most commonly people just have intuition of, well, you know, knowing whether the word’s capitalized and what the proceeding word, that’s probably useful interaction terms.

Let’s try putting it in and seeing if that helps. There has been a bit of work, which is trying to do automatic adding of what would be good interaction terms based on the strengths of individual features in a simple model. But in practice, it’s still kind of computationally so expensive that you need to do a ton of work in engineering things really well so that you can work out the deltas and model quality for adding features, so that you actually have some framework that’s efficient enough that you could run. But I think it is fair to say that 95 percent of actual model building, that human beings are still deciding what seems like likely interaction terms, and trying them out, and then seeing what happens to the model. Here going back to my MaxEnt example is an example where there are interaction terms. So all of these one’s down here are interaction terms of various kinds. So we have various kinds of features. One of the features is what class is being assigned to the previous word. So here for the word, “at,” the class assigned to the previous word is “other.” And if you just look at that feature alone, “other’s” prediction is it’s unlikely to be a person, it’s unlikely to be a location, which kinda makes sense. Most of the time, you’re just reading along this sentence of, “We were traveling along the road and then we saw.” After one “other” word, you’re most likely to see another “other” word. So “other’s” prediction is negative for all named entities. And so then we have this one for the word is capitalized. So that predicts positively for named entity classes,

because if you see capitalized words, some of them are at the beginning of the sentence, but a lot of the time when you see a capitalized words, it's a named entity, like a person or location. So it has a positive vote for both of these two classes.

And then you can say, okay, here's my interaction in terms of saying the previous thing is "other" and the current signature is capitalized. And then this one gives a further positive vote that is summative on top of these votes. This just about cancels out and says this is almost no vote for anything. But then the interaction term says if you see this combination of features, it's a fairly strong vote for person. And in a perhaps more interesting features, then, is this one down here. This one is saying, well, what about if the previous state is "other," the previous signature is just x, i.e., it's an all lowercase word, and the current word is capitalized, then this interaction term says that's a negative weight for it being a person, but it's a strong positive weight for it being a location. And that's because it's much more common to have location names after a word of that sort, like in the example, "at Grace Road," than it is to have person names. Partly because, commonly, when a person name is used, it's used as a subject at the beginning of the sentence, like "Bill came over last night." And normally, after that, not always, but normally, if you're talking about Bill more, you'd use a pronoun like, "him," and you'd say, "And I showed him my new Lego set," or something like that. So it's sort of less common to have the proper noun following a lowercase word. So that's actually getting a negative weight there. And so all of these terms down here and interaction terms, they're modeling these combination effects in various useful ways. And as you can see, the interaction terms are pretty strong. These are having a big effect in modifying what's happening from the effect of the individual features. Okay. Those last models that I've just been showing for those last bunch of slides, they weren't actually conditional models of the kind I showed last time as my MaxEnt models. They're actually joint models, so they're putting a probability distribution over a space as a joint model. But they're joint models being estimated by this principle of maximum entropy. So let me just quickly connect that back to the models that we had last time, which were these conditional discriminative models, the probability of the class given the data. And so the way that you could think about it is this, that you can think of the cross-product space of the class and the data as just this big, complex x space.

And you could say, okay, I want to make a joint model over this produce space of classes and data. And go off and build a join model. If you tried to do that for the kind of applications we have in natural language processing, you'd be in really, really big trouble. And the reason that you'd be in really, really big trouble is that normally, the number of classes we're interested in is pretty small. So that if we're doing named entity tagging of other person, organization, location, we have a class space of four. Very small. But the space of possible pieces of data is huge, because that's every way that a document could be. I guess, in some sense, really, that's an infinite space. But even if you fixed your vocabulary as every possible sequence of words, it's this huge, huge space. So, although in principle we could build models over this joint space, in practice, we absolutely can't do that. Because if we were going to try, what we'd need to be able to do is expectations over this joint space, which means that we'd need to be enumerating all of the points in this joint space, which is clearly completely impractical. So what we do is

we effectively restrict the models we build so that they just model the observed data. So if we have the joint probability of C and D, we can break that up with the chain law, the probability of D times the probability of C given D. And so what we do is we effectively – you can conceptually think of it that when we build out MaxEnt models, we add constraints to our MaxEnt models, which says the probability in the model of any piece of data should be its empirical probability. This means that the pieces of data we didn't see in the training data, i.e., the documents we didn't see in the training data, have probability of zero. And the ones that we did see in the training data will have probability of one divided by the number of documents. So we tie the probability of documents to their empirical probabilities in our closed set of training data set.

And that might seem a crazy things to do, because you could think, well, wait a minute, the whole idea is I want to take this classifier and then run it on different pieces of data. But actually, it turns out it's not a crazy thing to do. And the reason it's not a crazy thing to do is at run time, all we're gonna use is this part of the model. What's the probability of a class given a data item? We're gonna use our conditional model. And so therefore, it doesn't really matter if we trained the model having tied the model expectations of the training data to the empirical expectations, because we're just not going to use this piece. We're gonna be using this piece. Okay. And so once you've done things like that, maximizing joint likelihood and maximizing conditional likelihood are actually equivalent. So we're building these conditional models that you could think of them as also maximizing joint likelihood under this additional assumption that the marginals of probability of D are tied to what's actually observed. Okay. So there are kind of two more topics I want to get through today. One is smoothing and the other is just sort of saying a little bit about how this connects into sequence inference. So when you build these models for NLP, these models typically have tons and tons and tons of features. So the kind of models that we use for problems like named entity recognition typically have something like 1 million to 5 million features. And for every features, there's then a parameter weight. So we have this humongous number of parameter weights. So from any statistical perspective, the models are enormously overparameterized, because quite typically, we'll be training those models on rather less than that amount of data. So we might be training a model with 5 million parameter weights, but we're only training it over, say, 300,000 words of running text data with classification decisions.

So what does that mean? Well, it means that we have to write our code kind of efficiently, because there are a lot of parameters. There's the practical side of things. And on the more theoretical side of things, it's really, really easy for these models to overfit. And so we need to do something on top of this to stop overfitting, which we can refer to again as smoothing the models or regularizing the models. And the idea there is we won't want to learn facts that are so specific about our training data that they're not good facts for predicting our test data. And in particular, we don't want to set the weights or parameters in ways that are so specific to our training data that those weights aren't useful weights at test time. So what can go wrong? Well, just consider, again, a baby example where we're flipping a coin and we're counting up the number of heads and tails. So we'll get some empirical facts, like if we flip it five times, we might get three heads and two tails. And so what we're gonna be doing in our MaxEnt models is we're

building this little MaxEnt model where we have two outcomes, heads and tails, and so we'll have a lambda weight for heads and a lambda weight for tails. And so this is what our MaxEnt model will look like. Formerly, there's actually only one degree of freedom, because the only thing that's really important to the probability predictions of this model is what's the difference between these two parameters. And indeed, if you look at what's done for multiclass logistic regression, when you see it in statistics, the parameters that are put in the model are these kind of lambda parameters, where you're taking the difference between values of classes. That's standardly not done here. And the reason it's not done is that if you parameterize it this way, you get less non-zero parameters.

And when we try and implement these models efficiently, you get a great deal of efficiency from the fact that, even though in theory there are a ton of features and parameters in your model, the number of them that have a non-zero value for any particular data point is typically very small. Because you have sort of features like the word begins with "grac," that feature won't be active for almost any data point. It'll only be active for words that start with "grac." And so by parameterizing it like this, you minimize the number of features that are active for a particular data point. But anyway, if you think of it as being a single degree of freedom, lambda, what we're going to do is that lambda is going to have a value, and if lambda has the value of zero, you're predicting you're equally likely to have a head or a tail. And if lambda has a value of two, then the probability of heads is about 90 percent. And if lambda has a value of four, the probability of heads is about 99 percent, and vise versa for the probability of tails. You get these logistic curves that look like that. So what happens if we just look at some data and estimate the values of lambda? If our training data, we got two heads and two tails, this is our likelihood distribution over different values of lambda, and the most likely value for lambda, which we take as our estimate of lambda, is a point estimate, is zero. That one's easy. If we saw in our training data three heads and one tail, then the maximum likelihood estimate for lambda is about 1-point-something, and we take that. The problem is, what if we saw in our training data four heads and zero tails? Well then, our distribution over values for lambda looks like this. The bigger we make the estimate of lambda, the more likely our data becomes. Because as the value of lambda climbs, it's following this logistic curve here. As the value of lambda climbs and climbs, it's getting closer and closer to saying the probability of heads should be one. And so since here, heads was always one in our observed data, the higher and higher we make the value of lambda, the higher we might make the likelihood our observed data. And so that's really bad, because that means that the model at the moment, if it sees a categorical result, will maximally over-predict it. What it wants to do is make the lambda weight of that feature as high as possible. So that gives us two problems.

The first problem it gives us is having the optimal weight being infinite makes it kind of tricky when you throw it into a numerical optimization procedure, because it takes a long time for your computer to decide something is infinite. But it has the theoretical problem that your predicted distribution becomes just as spiked as the empirical distribution, that your predicted distribution will be wanting to predict you always see heads. And so that's something that we'll want to fix. And this actually, again, happens all the time with the big models that we produce. So if we start putting in these very specific features, like I

showed for NER, where we had features like the word starts with "G," and the previous word is a preposition, and the previous class is "other." You start taking conjunctions of events, and you very quickly come to the point where, in the observed training data, some of those conjunctions are categorical. The commonest case being that that feature only matches one point in the training data, and therefore, necessarily, it's outcome is categorical. Your feature matched once and when it matched the right class to assign was "person name," and so you've got a categorical feature that says whenever you see this configuration, you should always assign the value, "person name." And if we did nothing else, because that feature was categorical in the training data, our optimization software would want to say that the lambda weight corresponding to that feature should be infinite, so that whenever you see that configuration, it will trump every other feature and it will assign "person name" to that configuration. And so that's not what we want. And so we want to do something other than that. And so what can you do about that?

One way to do that, to solve this problem, which is actually used quite a lot in early NLP work, though people sort of frown on a little now, is to do what's called early stopping. And doing early stopping means you run your optimization procedure for a while and then you stop it. And so, effectively, what happens is you kind of stop the optimization procedure and the value of the lambda parameters start climbing for the ones that are heading infinite. And then you kind of turn it off and so they haven't gone to the value to be infinite, they've only gone to be values like five and ten. You can do that, but it's not very principled. A better way of doing it, and this is kind of the standard way of doing things, is to then put regularization, which in Bayesian returns is then referred to as a prior onto your model, so that you avoid that. So you put a prior distribution on the lambda parameters, which is essentially going to say we don't want the lambda parameters to be very big. And then you look at your evidence. And so what you're going to do is find out the most likely posterior for your model which combines the prior and the evidence. And so this will have the effect that, to the extent that there's a lot of evidence for parameter having a high value, it'll get a high value. But to the extent that there's very little evidence for it, like you only saw a particular feature active once, it'll maintain a low value, because the value that's set for it will be dominated by the prior. So the standard way of doing that is to put on Gaussian priors, which is also referred to as L2 regularization, and so what you're doing is putting a little Gaussian over how likely you are to get a certain value for lambda, which is centered or a particular point, mu. Where conventionally, mu is just taken to be zero, meaning this feature doesn't have any predictive value. And then so the prior is saying that you'll likely have a very small lambda value, and values of lambda away from zero in each directly get progressively less likely. And so if you have – so this is the picture of when you had a categorical feature, the four zero outcome of heads and tails, again. And so if you have no prior, which is equivalent to sigma squared being infinite, you get this curve where an ever bigger lambda value gives higher conditional likelihood to the data.

But if you have very strong prior centered at zero, and you see four heads and no tails, what you get out of the posterior is this red curve, where now the highest likelihood value for lambda is about one. So the strong prior has moved the most likely setting for the lambda all the way from infinity from being about one, based on having seen a

categorical outcome four times. If you make the prior somewhat weaker, which you do by making this nominator bigger, for example, you make two sigma squared equal to ten, you then get this curve where now the maximum likelihood value for the lambda parameter is two and a bit, but it's still enormously smaller than infinity. When I was showing these pictures before, I was sort of leaving out one detail, which is the weights that are shown in these pictures actually incorporate having regularization on the parameters. So no one asked me about it. But when we were looking at these two, someone could have asked, well, wait, explain to me why this one's big and that one's small, rather than that one being big and that one being small. And the answer to that in this picture is actually that's partly a consequence of the regularization of the model. So the regularization of the model means that it's easy for parameters to have high weight if there's a lot of evidence for them in the training data. And where if there's very little evidence in the training data, if a feature is only active two or three times in the training data, then necessarily, it has to have a small weight, because the prior dominates. And so part of the effect that you're seeing here is all words that begin with "G" is a much more frequently evidenced feature than the word is "Grace." Because the word "Grace" might have appeared only once or twice in the training data. And so the model is putting very little weight onto this feature, partly because of the prior. And it is, instead, putting most of the weight into that feature, because it's a common feature that's generally predictive. And so putting these kind of priors on really works. So this is an example from part of speech tagging. And the red one is with a Gaussian prior and the blue one is without a Gaussian prior. And so this is running the optimization procedure and counting the number of iterations. And then this is the accuracy of the tagger going from 96.3 percent to 97.1. And so for the blue line, if you do optimization, for a while, your parameter weights are being moved and your model is getting better at predicting the part of speech tagged. And it reached a maximum. Then after that, there are all these various categorical evidence features, and their weights keep on being increased.

And the model starts to overfit to the training data, which means that when you run it on the test data, your performance actually gets worse again. And so it's precisely the blue curve that motivated early stopping and early work, because if you stop early training at about 100 iterations of your optimization procedure, you've got the best results. But what you find is that if you do regularization, like with a Gaussian prior, that your model not only genuinely converges to an optimum, rather than having to do early stopping, but actually, it converges to a better optimum, because you have an overfit parameter value's the same. So what you do to do this is we have the same model as before, and now this is log probability so that you're putting in this extra term for the priors. And if we just accept the fact that we have the default value for our features being to have weight zero, what you actually have here when you're building the model is a very simple term, where you're penalizing the model for making the lambda value non-zero by squaring its lambda value. And you're then regulating how much it's penalized by this two sigma squared term down here. So if this is a big number, the penalization is weak. And if it's a small number, the penalization is strong. And the central intuition, then, is this penalty that's assessed for lambda is assessed precisely once, whereas if giving a lambda a high weight, positive or negative, is useful for predicting the observed data, you'll gain here for every piece of data for which it's useful. And so, therefore, if there are lots of places

where you can make use of a feature, you'll get an enormous gain in data conditional likelihood. And it doesn't matter that you have to pay the penalty once. Whereas, if the feature is only active at one or two places in the data set, then you get some gain here, but the penalty here is bigger than the small amount of gain that you get over here. Okay.

One other way you could think of doing smoothing is just say, well, can't I just add in virtual data the same as we did with add one for smoothing models? In practice, that's extremely hard to do once you start having all of these different features and interactions, because it's very hard to know how to add in virtual data to make sure that everything is non-zero in a consistent fashion. I've only got a few minutes left, so let me just quickly say a minute about doing high-level inference, and then I'll be done. So I guess I've spent a lot of time on how do we build these classifiers here for logistic regression star models, these MaxEnt models. And so for building one of these classifiers, we're building an individual classifier where we've got training data, which is labeled, we've build features, we've trained one of these up using our optimization software, and once we've found lambda weights, we can then run a linear classifier at test time to classify a new piece of data. But what we want to do for any of these tasks is get beyond that and actually do this sequence labeling that I mentioned earlier, motivating going beyond just doing individual classification decisions and doing sequence labeling. And normally, the answer to sequence labeling is you do a form of search. And one of the commonest ways to be doing the form of search is to be doing dynamic programming. And so, in explaining in lecture, I'm gonna largely punt on this for the moment and you should all have read Chapter 6 of Jurafsky and Martin, which talks about the Viterbi inference and how to extend that to these kind of models. The reason I do that is just kind of a technical decision, because I do go through all of the search and Viterbi inference for parsing models and somehow it seems kind of a bit overkill and boring to do it twice in close succession. So I'll leave it to you to read it for this time. But the general picture is the way we want to do sequence classification is we build our classifiers assuming that we have a local context. So we already made some decisions on preceding things. We have our observed word, and our classifier's going to say, okay, predict the next part of speech tag here. Where for features, you can use preceding decisions and anything about the observed data, either from before straight onto you, or after you. So we're gonna build the classifier like that. So it'll be able to see features like this. But when we're actually running at run time, of course, we don't know these. We want to be working out those part of speech tags, as well.

And so the way we explore the space and come up with a good classification of the sequence is that we have our classifier built just like this, but then we use search process over the top to find good sequences. So the simplest way of doing this, and in practice one that just works fine, is to do beam search. So at each position, you keep just a list of, say, five best ways of labeling the sequence that you've labeled up until then, and you then consider extinctions of the sequence, i.e., you consider each way of labeling the next word as person, location, organization, other. And then once you've tried out all of those 20 combinations, you just keep the best five and throw away the worst and repeat over. That's incredibly simple, but for a lot of these applications, actually, beam search just works fine. And there's not really much reason to do something more complex than that.

But nevertheless, you can do something more complex than that, which is doing this kind of Viterbi search, which means that you can do an optimal search for finding the best sequence through the best sequence of labels, providing that all of your features are local. So if your features are only use some window, say that you only look at the previous two assigned labels, then you can do a dynamic program Viterbi search and find the optimal label sequence. So that's where I just stuck in here. In the Jurafsky and Martin book, you see these kind of pictures that explain how to do that, and I'll leave it to you guys to look at that, or you can wait and hear it when I explain it for parsing. Okay. That's it for today.

[End of Audio]

Duration: 76 minutes