NaturalLanguageProcessing-Lecture06

Instructor (Christopher Manning):Hi again. Welcome back to 224n. Okay. So today's lecture is hopefully a little bit of a respite where we can finish off some stuff and do high-level introduction to information extraction and you can, hopefully, feel relaxed. You handed in the first program assignment and are getting a good start on your second assignment. And then on Wednesday, we kinda get into the next big area of technical content, which is then doing discriminative sequence classifiers. So what I'm gonna do today is, first of all, finish off with a bit that I didn't say about machine translation, about using some tech models. And then I want to give a high-level introduction to information extraction and some of the motivations and talk about a couple of historical – and still used – but straightforward method to go about information extraction. And then next time, we'll go into the main technique that we'll cover, which is discriminative sequence classifiers. Okay. For about the first quarter of the class, I just want to finish off by saying a little bit more about what I didn't get to in machine translation. So at the end time, if you remember, we were talking about statistic phrase-based system. And statistical phrase-based systems are kinda just the de facto state of the art as to what people build and use for machine translation systems. And that the deal was there that we actually still used IBM style alignment models to build good alignment. So we don't actually use them to translate any more, but we use them to learn alignments using EM, kind of the way we've discussed and you're doing for the assignment. We're then using an algorithm to merge their outputs. And there are various algorithms that are used, and I'm not gonna go into the details. But conceptually, this is just fixing the brokenness of the fact that the IBM models allow one too many in one direction, but not in the other direction. So to some extent, this looks like a kind of an ad hoc procedure and I think it's fair to say it is.

But it's basically state of the art what everyone does. And so then once you've got that final phrase table over here, you do the phrase extraction process. And the phrase extraction process, as I said last time, is that you have legal phrases and legal phrases are things that have complete, in that it has to contain all the alignment points for all the words in both phrases. So you can't have this as a phrase, because there's an alignment point that's outside it. And you can't have this as a phrase, because there are alignment points outside it. But nevertheless, there are lots of possible phrases you can extract. So this is a possible phrase. That's not a possible phrase. But this is a possible phrase. And then the whole thing's a possible phrase. And then this gets cut off, but maybe kind of all of this is a possible phrase, so you can recursively extract bigger and bigger possible phrases. And people do, normally, up to some fixed limit. And then the decoder runs, using these kinda multiword phrase chunks. And that's proven to be a very successful technique, essentially because these multiword phrase chunks can capture a lot of the word selection and fluency of translation in common idioms. But still, these methods don't solve all the world's problems, and so I wanted to go on and say a little bit about syntax-based machine translation systems. So the general idea of what we're doing here is this is the Vauquois triangle that I showed earlier on. And the idea of this was you could be operating just at the word translation level. You could be heading to deeper levels of linguistic analysis. And, although people had previously done a lot of work with syntax-based machine translation before statistical machine translation came to the fore, essentially, with the IBM models, things went right back to the beginning.

Although there are kind of a couple of little hacks like the Model 4, remembering the displacement, and so you didn't get penalized twice, which a little bit captured the idea of phrase structure, basically you're down at the level of doing word-for-word translation at the bottom of this triangle. And so recent work in machine translation has been starting to push back up this triangle. And so now these statistical phrase-based systems are usual, and most people, but not quite everybody, believes that pretty soon the wave of the future is building syntax-based machine translation systems. And so I'm just going to show a little bit of one early system that used syntax. Some of the ideas of why you want a syntax-based system is you'd like to have output sentences, which are grammatical. If you get far enough in your assignment, have your Model 2 working decently, you'll hopefully see that some things are kind of okay. It's finding a lot of the content words that it should be, and so on. But you're not really gonna get out sentences, which are grammatical sentences that make it past your English teacher. So somehow it would be nice if we could actually kind of at least translate into sentences. They may not be perfect translations, but shouldn't they at least be grammatically well formed. And another place where statistical phrase-based system still badly fall down is reordering and word order between languages, so that's where when you have something like Arabic commonly has verb initial sentences, German commonly has verb final sentences. All of these different word order possibilities that, to some extent, statistical phrase-based systems can reorder phrases, but they're very weak models of doing that. And you'd hope that syntax-based models could do a lot better. Phrase-based systems are also normally bad at putting in the right function words. That's knowing that when you translate into English you have to put thus and us in various places, where in other languages you have to put various kinds of case particles or verb endings, because they don't really understand any of the syntax. Okay. So here's the kind of motivating example for trying to do something better. This is a Japanese sentence and we're translating it to, "He adores listening to music." First thing to notice is that there's a ton of reordering here. So we have the "here" at the beginning, the "kare," but then the next thing, "adores," is right at the end of the sentence, "daisuki." And then more than that, there's kind of a lot of sort of syntax stuff going on.

So here in English we have, "listening to music," whereas in the Japanese we have "ongaku ow." So this is kind of an object marker. So you're not getting "to," you're kind of getting, "listening music" in Japanese. So it's one of these syntax facts that you have to know about how things get translated. And then in both of them, effectively, you've got a nominalization. So "listening to music" is the English nominalization, and in Japanese, you're also getting this nominalization of "ongaku." But it's got a lot of its own syntactic structure, these little function words that are going together to put it together, and somehow it seems like to actually be able to translate well, we want to understand some of that syntactic structure. And then to know general facts about word order. So that in Japanese, you're generally getting the predicates at the end of phrase. So we're getting the "adores," at the end of this phrase, and then inside the nominalization, we're getting the "listening" at the end, and the "music" before it. So that part's the opposite to English. So there's lots of syntax going on, which we might want to model. And so the suggestion in this Yamada and Knight paper was we can just take the – so ultimately, we're going to be translating Japanese to English. So doing the usual Bayesian version still, we have an English to Japanese channel model, and so maybe what we could do is, since we have parsing technology for English that's pretty good, that we could build the channel model around an English parser. And it's probably better to just immediately move to the picture. So this is the idea of what the channel model now looks like. The channel model starts with an English sentence and then it does the sequence of operations that changes things around. So the first part of it is to treat this tree like a mobile that you can kind of change around the parts of. So at each level, you're allowed to do reordering operations. So I can swap around "to music" to "music to." And then I can swap around the phrase and the verb that used to be "listening to music," and swap them to "music to listening." And then I can recursively do the same one more time up the top, and get "adores" at the end and the object before it. So I can sort of just by sweeping things around, turn things into the right word order for Japanese. If you want to, in your spare time, you could think about whether that's always possible or whether there are counter examples to that. Then the second phase of this is putting in these spurious words, the extra words that are turning up for no reason. So now we've got a syntax-based model. And so Japanese has these case marking particles that denote grammatical functions, like what's the subject or what's the object. And that's just the kind of thing that you can very straightforwardly do if you have a syntax tree. Because you could look at the syntax tree and say, oh, here's the subject, so it should have the subject – I guess the topic – case particle "ha" after it. And then down – actually, I should be careful. This is actually a topic comment structure, so this is the topic and then this is actually the subject with another marker. But you can put in the appropriate case markers. You can put in extra bits of verbal morphology that you have to, with reference to the syntax. Yes?

Student:[Inaudible].

Instructor (Christopher Manning):So there certainly could be, yes. To be honest, in the model of Yamada and Knight, the chances of putting these words in didn't depend on the context. They depended solely on the parent category. Now, obviously, you could imagine better than that, but essentially, its model was I'm going to be able to put in the appropriate grammatical markers in the appropriate syntactic positions. Yes?

Student:[Inaudible] a lot of language, not just Japanese, these types of words where having then in there [inaudible].

Instructor (Christopher Manning):Well, I guess in a lot of languages, the corresponding things are bits of morphology that are endings on words, i.e., cases that normally occurred. So that's something like in German, where you have nominative case, accusative case. Or if you got tortured with Latin in your early youth, you saw nominative, accusative case. So I think more commonly the answer is that they're kind of endings on words that have done this morphology, which you'd have to handle slightly differently. But in principle, you could do the same sort of thing. It's not that there aren't other languages that do things in this kind of way, more these sort of stand-alone particles. So you see that in things like the Austronesian languages. Okay. So we've done reordering, inserting extra bits of grammar, and then we just translate the words into Japanese words. And if we're really, really lucky, we can just read off the tree and have the right sentence. Great. Of course, it doesn't always work, but that's the principle. So when they first published this, they published this as a channel model and showed some alignment results and that seemed kind of cool. But I kind of wondered how they were ever going to reverse this to turn it into a real translation model in a decoder. But it turns out it's actually not that hard to do. Essentially, what you can do is write a grammar that incorporates all of this stuff into it so that your grammar rules have extra annotations to say whether in the Japanese version things gets swapped. And so this is still just being written as context-free grammar rules. You can add grammar rules that then taking a word in one language and turning it to a word in another language, that kind of funny terminal rewrite rules. And you can turn it all into a big phrase structure grammar context-free grammar, and that's what they did and built a decoder out of it. And so they did this in sort of small-scale experiments for Japanese-English and the general result was that they could actually get a very significant increase, especially in human judgments of acceptability of their translations by doing this. Because I think this is particularly emphasizing the fact that – they can mess up as well, but providing they kind of get it right, that they've got much better control of the grammar and structure of their sentences and so they sound a lot better to human beings. So they had three human beings judging 50 sentences, and this was jest a three-way judgment that the human beings could say that it was good, so-so, or bad. And that as you can see in these results, they were getting significantly better results than IBM Model 5 is here. In particular, ten sentences were judged perfect, as opposed to zero. And so here's just a couple of pictures that are showing the alignments, where the idea here is that the very dark lines are high probability alignments, and the thinner lines are low probability alignments by the alignment model, or the channel model. And so here's IBM Model 3. And some of the time, this works perfectly well. So you are learning to translate it with the verb here at the end. That's the right thing to do. "Me," "watashi," that's doing well. "Kare," "he," that's doing well. Most of it is okay. But somehow it's completely lost the correct alignment of revolvers. So revolvers should be going to this guy here. But instead, it's being aligned with function words. So this was the object particle, and this is dative particle. And I think you'll probably learn, to your frustration, when you build your own Model 2, that these IBM models just really commonly do that kind of crazy stuff. I think it's partly in the sort of biases that are there. And unless you're careful, these kind of function words – and you don't get precisely the same function words for the cases that you're doing, but both French and English have lots of function words. They have kind of articles and auxiliary verbs and little introducing words, like "that." All those kind of words are really common and can easily be that, in the model's mind, the easiest way to do these alignments is to sort of align things to function words, because they're just around a lot and you can kind of have them lined up.

Student:[Inaudible].

Instructor (Christopher Manning):The word order, yeah.

Student:[Inaudible].

Instructor (Christopher Manning):It did affect the way – I maybe didn't say this clearly enough. So each of these steps here is a probability distribution. So that you've got each one of them is a little probabilistic model. So this one here, the translation one, is kinda like all the ones that we've seen of the probability of fj given ei, that they're just probabilities of word substitutions. But this reordering one is also a probability distribution. So what it's actually got is okay, here's a rule, VB2 goes to VB2. What's the probability distribution over different orderings of those components? And so they could stay the same or they could swap. Or extra things can be inserted into them. And you have a probability distribution of those possibilities. So free word order can be modeled by effectively just having a flat probability distribution over reorderings.

Student:[Inaudible].

Instructor (Christopher Manning):Yes. Note that this is like a context-free grammar that the reordering is only being done relative to a single level of tree. So looking at something like this, so if there are subtle facts about particular words preferring different reorderings, it's not capturing it.

Student:[Inaudible].

Instructor (Christopher Manning):So short answer to that, no. I mean, I think to some extent, this is part of why – so at the moment, there are lots of people actively researching syntax-based models and there are some promising results, especially for some languages, like Chinese or Japanese, with a lot of different things going on. But in other circumstances, statistical phrase-based models still do better. And I think part of the reason for that is, at the moment, there isn't kinda the perfect technology and the different models are good for different things. So statistical phrase-based models do excellently for that case, because they'll see that "listening to music" is just being translated as this one word lots of times, if you've got enough data. They'll learn that little phrasal alignment rule, and translate it beautifully. Whereas, this kind of model, although it's doing some clever stuff with syntax, in another sense, it's right back where the IBM models were, because at the end of the day, every leap here is being translated independently, just like in Model 1, 2, or 3. So it isn't capturing the advantages of a statistical phrase-based system, and so, I guess that's where, obviously, to do better is the goal. And you can kind of think of ideas to do better. You could say, can I take a whole phrase and translate it at once? And that's an obvious thing to think about. But I mean, one of the things that makes it difficult and part of why statistical phrase-based systems have been very successful by ignoring syntax is a lot of their useful translation units don't actually correspond to syntactic units. So like in this example, the useful translation unit is actually "listening to," because "listening to" gets translated in a certain way. It's not that "to music" is a good translation unit, because if you were talking about something different, where you're actually going to music tonight, the "to" would be translated as a Japanese particle, which is really used for motion towards. But here, because somehow just this English idiom of "listen to," they're not using that particle. They're just regarding it as a direct object. So that's kind of part of why this is tricky. Yeah?

Student:[Inaudible].

Instructor (Christopher Manning):So there certainly have been – I don't think I'm very expert on this field, I will say first off. I mean, there certainly is quite a lot of work and studying of human translator. I guess, especially, there has been a lot of psychological studies of people doing simultaneous translation, because that's seen as a fairly special skill. But I mean, absolutely I agree that human translators are recognizing chunks that they can handle. But I think it's fair to say that the chunks that they're recognizing, some of them are idiomatic. I think it's clear that humans store a ton of idiomatic chunks in their head, like we all know all of those regular things we see in the newspaper. Ethnic cleansing and improvised explosive devises and all your other favorite newspaper terms. We all remember those little chunks and then people start using them. So humans carry a huge number of chunks in their heads and will translate chunks. But I think it's also fair to say that the human being is also doing grammatical analysis in their heads. So that if you're trying to translate something like a Chinese sentence, you really, really need to work out which parts are different clauses and where the verb with the different clause is before you can start to make any sense of how to translate it. Okay. That's one more example that maybe kind of exhibits the same thing. Okay. So here's my final machine translation side. So where's machine translation at the moment? I mean, it's just still not the case that you can get machines to do high quality automatch machine translation. Just doesn't exist. So for most of the translation industry, where the action and technology actually is, is producing aids for human translators. And that means that the machine doesn't translate the text at all. Because it turns out that if the machine does a kind of a bad job at translating, it just take the human being more time to fix up all the junk that they've produced. And it would take the human being to do it right in the first place.

So all the technology, which commonly goes under this name of translation memories, is effectively building big databases of translation chunks so that the system will know well how to translate chunks. So if it's some phrase like, "move your mouse," and you're wondering how to translate your software manual into Icelandic, or some language like that, well, the translation memory will be able to tell you that "move your mouse" should be translated as this. And it actually then even sort of fits into standardizations, so large companies wanting their translations to be consistent. So just as the same terminology of icon window dialogue box is used in English that you don't want in the translation, five different translators have chosen different creative ways to translate "dialogue box" into Slovenian, or something. So precisely by having this kind of translation memories, you can standardize on a single way to translate things. The exception to that, which has effectively been the enormous growth area that's revived a lot of MT technologies, is effectively there's now this new concept of user-initiated translations. So I think the traditional model is that really the only thing that existed was authorial producer generated translation, and if you're the authorial producer, not withstanding some of those appliance manuals you remember from the 1980s, by and large, you don't want to be producing manuals that are clearly horrible and make you look bad. And so you want a certain quality of translation. But if you're just a user looking at something that you can't understand at all, you're pretty happy to have something that kind of gives you the gist of it, even if the translations aren't perfect. And so that's led to this resurgence of medium- to low-quality machine translation that appears a lot on the Web.

And I think I sort of said a bit about some of the future things. So I'll stop there. Okay. So moving ahead now for sort of the remainder of the course, I guess, what I'm wanting to do is sort of start talking about different natural language processing topics, which start to sort of move up from fairly surface level processing, to starting to do deeper processing, which can be a syntactic kind, like parsing. But a lot of it can be thought of as starting to understand more of a text so you can do more of a text. And so the lowest level of starting to understand some bits of a text that have been explored quite a lot are these tasks of name density recognition and information extraction. So what I'm gonna do today is motivate those tasks and start to stay a little about a couple of ways that they're approached early on. But let me motivate them in a slightly discursive way by talking about some negative examples. So we have this technology of natural language processing, and we want to do something with it. And there's the Web out there, so an obvious thing to say is, well, can't we use MLP to improve Web search? And if you think about that for two minutes, the answer is, of course you can. There are all kinds of places in which you could use at least low-level MLP technologies to do a better job with word sense disambiguation. The most overused example of word sense ambiguity on the World Wide Web is the word, "jaguar," because it has all of these different meaning that are common in the computer world. So there's the big cat's meaning. There's the version of Apple Mac OS. There are jaguar cars. And there are all sorts of miscellaneous other littler ones, as well. I'd say there's molecule geometry and solvation energy package that's called Java. And where you get those same kind of word sense disambiguation problems with people. And this happens more and more as greater and greater numbers of people have blogs, home pages, etc.

So one name, like Michael Jordan, will refer to lots of people. And then you also have the reverse problem, so that you have more synonyms of laptop you don't find references to notebook. So there are lots of places in which you think a little bit of MLP would help. And there are then slightly higher-level places with which MLP might help more, so people point to various semantic search kinds of applications. So that if you're wanting to do something like "clean espresso machine porter filter," that if you kind of know that "espresso machine porter filter" is the object of "clean" and you want to make sure you search for that relationship, maybe you'll get more precise searches. And it sort of seems like there's some technologies that you can apply to some of these problems. So word sense disambiguation is certainly a technology that people have worked on a ton. And there are fairly good word sense disambiguation systems. Learning synonyms or just listening to synonyms, that's a technology that's worked on quite a bit. And so at various times, there have been tons of companies that have, essentially, their mission statement being to bring MLP to Web search. There are a whole bunch of them that were back around the kind of the bubble days around 2000. There are some newer ones now. Traditional key word search technology is hopelessly outdated. But it seems like, in practice, somehow this is kind of an idea that hasn't quite worked for people. And it's not that I think people should give up all hope of this, because in some sense there are places in which it should be useful, and I think in kind of little edges and corners of the search world, it is starting to be shown useful. But it turns out that some of these ideas kind of aren't as useful as an LP person thinks, or aren't as easy to apply successful as an MLP person thinks. So we can do things like correctly stemming words to their form of getting singulars and plurals and things like that in the kind of good linguistic way.

But it doesn't actually work much better in practice than doing it in crude ways. We can do word sense disambiguation fairly well, but it turns out it's not clear that we can do it well enough, so there's actually an early study over a decade ago, now, by Mark Sanderson, who's an IR person, in England. Essentially, it was a user study, which was as accurate did word sense disambiguation have to be before, on balance, users were happy with the system. And so it was attempting to do word sense disambiguation and give improved results based on it. And the crucial observation is that if you disambiguate wrongly, you lose big time. Whereas, if you don't do disambiguation, and you show a smattering of results, a lot of the times people can just choose the one that they want. And so his results were that you had to have about 90 percent accurate word sense disambiguation before for user utility you could measure an increase on it. And the problem is, a lot of the time, somewhere around 80 and 90 percent is the kind of accuracy we can do word sense disambiguation with. Okay. So syntax. I'm very fond of syntax and parsing, and I'll talk about it for lots of lectures any week now, so surely have the structure of sentences, just as I was motivating for machine translation, that should help for doing Web search. Maybe someday, I actually hold much less hope for that for machine translation. I think syntax will be proved useful in machine translation much sooner than for Web search. It seems like for most of the things that people want to do with Web search, that statistical phrases, that same kind of technology that's been so successful for MT, where you're just taking any substring of words that you're finding distinctively often, has just been a very successful technology. So Web search engines have gotten a lot of value out of using various notions of statistical phrases, so as well as when people overtly put double quotes around things. Because it turns out that, although you kinds of people put double quotes around things, most regular people don't use the double quotes operator. So you can get all the benefits, or nearly all the benefits, for recognizing quotes, recognizing covert phrases for them, and treating things as phrase, even though they didn't type the double quotes. Or more generally, various kinds of proximity weighting, where you're looking for search terms to appear near to each other. Okay. So in practice, if you kind of look at what's been driving a lot of the progress in Web search, I couldn't honestly say that a lot of it has been high-level natural processing, that lots of other things, like anchor text, have been giving much more value.

And there are kind of interesting ways to think about that. I mean, effectively, anchor text, one of the big things it's giving you is it's giving you synonyms. Because people who are writing anchor text are writing alternative ways to express the topic of some page. So often they will do synonyms. So effectively, you're getting this very domain specific, hand-written synonym or disambiguation technology, which is tending to be more useful. Okay. So what should you do about that if you're a natural language processing person? So I kind of feel like if you want to be useful as a natural language person, it is somewhat unwise thing to aim at to say I'm going to build a better search engine. And probably what you should be trying to do is actually to come up with technologies that actually require more understanding. It turns out that doing document retrieval doesn't actually require very much understanding of what's on the page. That if you chose other tasks, which actually require much more understanding of what's on the page, then it seems that there are much more opportunities to show that deeper MLP techniques actually deliver value. So this is a kind of an old slide now from Oren Etzioni, who's CS prof from the University of Washington, who's done a lot of much more recent work on the Know It All project, which is a very interesting project for information acquisition from the Web. But this was a slide, I think it actually dates back to the very late '90s, that he made, where he wanted to suggest that things like search engines, they're the information herbivores, and what we should be doing is, on top of the information herbivores, building the information carnivores, which work in a much higher level of operation. And I think when you get to wanting to build information carnivores, that maybe that's where MLP gets much more useful. Okay. And so I want to talk about a couple of examples of that. And one of the obvious places on the Web is trying to do much more in the way of trying to do product finding, comparison shopping in more automated fashions.

So it seems like a good higher level task is, rather than me spending half an hour variously browsing around websites and doing Web searches and finding different pages and clicking through various ecommerce sites, why can't I just tell my computer to go and collect information for me on the prices and cost of shipping and handling, all of that, of different products? And actually Oren Etzioni was one of the first people that tried to do this. He built the Jango shopbot, which was an automatic price comparison system. I mean, when these were first introduced, I think vendors hated them. But I think that's kind of changed now, because of the way that the Web has changed and these days it's more important than for vendors to actually be found, rather than not found than for them to worry about the comparison shopping angle. Here is a very old screen shot of Froogle. I will use this to illustrate that information extraction isn't easy. But also to pose a question as to what the future is. So you can still go to Froogle. It doesn't look quite like this anymore. And you can type in your product name. I put in here "Lego fire engine." And it will give you some results. These results aren't very good. If you go and look at it today, the results are slightly better. But in the old days, if you kind of look down at your results, you got to this green bar, where it says, "The results below were automatically extracted from web pages. Price and category information are uncertain." And below the green bar, they did information extraction. They went around Web pages and automatically tried to extract details about products. Now, with no internal information, from what I can see from the Web, they just dropped this and got rid of it. And they just show the stuff up here. And what's the stuff up here?

The stuff up here is vendors give data to Google, instructed XML formats, and they just stick it up as the results, so you're doing the Web search over structured data, where you have product names and attributes and things like that. Well, that's a bit sad for an MLP person, because we'd be doing the part down here. But I mean, I presume why that was done was because the accuracy down here was just too low and it wasn't perceived as as good. So there's still plenty of work for MLP people to do to get good accuracy. But I still think that there's actually very good prospects to doing automatic information extraction. Because the fact of the matter is that there’s just tons of stuff where you have scriptive content, which isn’t very structured, but you’d like to process it automatically. So you see that in a lot of the most popular sites. Something like Craigslist, they have a little bit of typing of the data, but nearly all of it is just written as unstructured text. Something like ebay has a little bit more typing of the data, but still a lot of the details that you want to know are unstructured text. So the thing that’s actually unclear is one possibility in the world is everything will go the structured way, and the other possibility in the world is that a lot of stuff will stay in unstructured natural language text. And different people have different bets on that. So the much hyped semantic web of Tim Berners-lee is essentially a bet that people will find it worth their while to start turning things into highly structure data that makes it easy for computers to process. Doesn’t seem to have really succeeded. I’m a little bit skeptical. My bet, and my livelihood, is lots of stuff will stay down here and someone will want to get information out of it at some point. So this is the question. Will human beings keep on writing their emails to their friends or their IMs or whatever other medium saying, “Hey, Sue, can we have lunch tomorrow at noon?”

Or will they start filling out structured lunch request forms and send them as structured data to their friends? So why is doing this information extraction automatically hard? It’s kinda not hard for humans. We’re very good at interpreting this kind of stuff, but it turns out that when you’re a machine, there’s a lot of stuff to notice here. So here we’ve got a product. What kind of a product is this? A car? Or a book? Yes, this is a book. Could you imagine a computer being fooled and not thinking it was a book? Yeah, because it has yo-yos and it says here “toys.” But actually, it’s a book. Of course, there are other good clues that it is a book that’s paperback. So working out what the person is selling, that’s a categorization task. It’s not necessarily tricky. Okay. Well here’s the title of the book. Nothing explicit says it’s the title. It does kind of explicitly here say “author” in front of the authors, but for the title, you’re just kinda meant to know, based on how it’s placed and the fact that it’s bold and so on. In particular, yeah, this could be a book title, but it’s kinda hard to know if you’re a computer. It’s not so obvious what’s the book title. And then we’d want to know how much this sells for. And among other things, note even if you’re clever enough to have your regular expression for U.S. dollars, there are two prices here that are in U.S. dollars. Say you need to know enough about how to interpret the context and where the buyer is to be able to give them the right price in U.S. dollars. Okay. So this is the space of information extraction systems. So the goal of information extraction systems is to say we’re not going to try and do full natural language understanding. What, instead, we have is a simple information need, and the simple information need you can think of as a database record. Or you can think of it as kind of an attribute value. You’ve got several attributes for which you want to find values. So in the previous example, I want to know, perhaps, the category of the item, the name of the item, and the price of the item.

And that’s all the information I need. All the rest is superfluous. I’m just getting out a simple record and so I want to be able to run information extraction system over a lot of pages of different kinds and fill in that database table as accurately as possible. If you think about it, there are kind of tons of things that people would like to do that you can think of as basically information extraction problems. So maybe I want to collect company information from all of those forms they have to file for the SCC. So I want to find out for a company what's its earning, what's its profit, what's all those other numbers that people report. Or the book value or things like that. And I might want to get out different kinds of relations. So I might have kind of entering relations. I want to find out all the members of the board of that company and things like that. There's been a lot of work in information extraction in the biomedical domain. So the sexiest part of the news has been doing genome sequencing. But if you just have a sequence of base pairs, that doesn't tell you anything about what they do. So if you actually want to know what they do, basically all of the information about what things do is then contained in papers where people write natural language about experiments. So there's been lots of work in recent years. And then trying to mine from research papers information about what proteins enable some other protein to do something, or disable certain things going on, things like that. You're also increasingly starting to see this just in consumer operating systems. So I guess here I've got Microsoft "Smart Tags," where you see exactly the same thing in the recent release of Mac OSX, where you see it in Apple mail, where you get dates and things like that tagged. You see the same thing in gmail. So that people are starting to do various kinds of automatic recognition of that sort. Here's the very first information extraction project I ever worked on, which was back in 1999, a decade ago in Australia.

And this was for News Corporation, that well known and much loved media company. So like their name, News Corporation Australia actually owns most of the newspapers, as opposed to most of the television stations. And so what they were wanting to do was take all of their real estate listings and be able to provide Web search over them. And at that time, at least, all of those real estate listing were just in plain text. But you know, still, if you look at something like Craigslist, most of what you see is in plain text. And I think this is just a nice example, actually, because with just plain text and plain text searching, you kind of can give very little value with something like real estate or other forms of classified ads. Where if you can actually do information extraction over these, and semantically interpret the information, there's all sorts of interesting things that you could do. So in this application, it extracted out addresses. It can use those to geocode to locations. It extracted out open house times. I think around here, every open house is from 1:00 to 4:00 p.m. on Saturday. So there's not very much information about open houses. But in Australia, all the places just had different random open times on the weekend, so it could be useful to fill in a little kind of – if you're looking at a bunch of places, fill in calendar entries of which order to do them in. You could get out the price, you could get out various other requirements, like how many bathrooms that you wanted. And it could give a really nice application, in which you could do semantic level searching over these listings, and then interpret the results to show them in various nice ways that gave value to the user. Whereas, in a domain like this, just trying to do plain text search works absolutely horribly. And just to say a few sentences on that. So by and large, one of the three biggest things that people want to search for in houses, well, they'd want to search for where they are, how much they cost, and how many bedrooms they have. That's the three big starting points. Now the one that most well works for text search is searching for a location. But it turns out that even location doesn't search, doesn’t work very well for all sorts of reasons. First of all, there are normally lots of stuff in the listings that’s misleading. So normally, real estate agents want to promote themselves in the listing, so they put information in about themselves, including where their office is.

So it’ll say something like, “Caldwell Banker,” which isn’t where the house it, it’s where their office is. People use creative phrases like, “only 45 minutes from Parramatta,” which doesn’t mean it’s in Parramatta. They might mention multiple properties in a single ad. And then the reverse direction. You kind of really want geographical information, because commonly you want to search in an area that’s bigger than a particular town or suburb. But this is really the best end of it. If you go onto other things, it just kind of gets worse. Because for money, you don’t want to textual match, you want a range. But you also want more interpretation than that. People get quite creative in these ads. Was $155K, now $145K. So you want to use this price. Offers in the high $700s, that means $700 and something thousand. Whereas, if you have something like rents for $270.00, that means it Australia, $270.00 a week. That doesn’t mean $270,000.00. Bedroooms, there are all of these creative ways that you can abbreviate bedroom. So you want to be, again, recognizing a number in the context of something like abbreviated bedrooms. So finding the things to extract is a big problem. A second big problem is canonicalization of product information. So these pages are from CNET, where I’ve searched – or at least, I searched a couple of years ago, I guess, for an IBM X31 laptop in the days that they still made them. And here’s sort of some of my results list. And what you see is, boy, they have a lot of X31 laptops listed. And if you look very carefully at the fine points, there is one of them that’s different. This one’s a 1.3 gigahertz model. But this one, this one, and this one, all identical. And this one kind of you can’t tell, because this has too little information.

But effectively, what they’re doing is failing to merge together things that should be treated as identical, so this is the problem of canonicalization, which is a big problem in all of these domains is recognizing things that are actually the same. And because they fail to do canonicalization sufficiently well, you kind of get crazy effects, because the price range here is different from the price range there, where they should have merged that information together. That’s hard to do well, because people in natural language just describe things differently because humans are so good at interpreting it. So here’s a digital camera website. And this is looking at their technical specs pages for different camera. And what you can see is, for the resolution of the camera, they just use all kinds of different terms, image-capture devise, image sensor, imaging sensor, total pixels. You’re just kind of meant to know that somehow those are referring to the same thing. And I mean, what’s kind of interesting is I took all of these off one manufacturer’s website. It’s not that different camera manufactures use different terminology. Essentially, it sort of seems like it’s just not worth the manufacturer having consistent terminology just for themselves, because it’s sort of, effectively, when human are reading it, there isn’t the cost. Because unless you’re me making an example for class, people just kind of look at the tech specs and they’re happy because they can see the resolution of the camera and that they don’t actually have difficulties with these canonicalization issues. But our computers do. I’ll skip that. Okay. So first problem that we’re gonna start off with is the problem of named entity recognition. And we'll look at this a bit before we get back to high-level issues, like the canonicalization. So the idea of named entity recognition – this was kind of invented in the late '90s as a subtask of information extraction – was what we should do is have a separate component that starts with text and just picks out these things that we called named entities.

So named entities are kind of a term of art, but what's being meant by named entity is something that is a name. It could be a person name, but it could also be a product name or a place. But there are these kinds of things with definite reference that have proper names, in some sense, that are called named entities. And then having identified them, the second task we then want to do is to classify them. So in this particular data here, there's a four-way classification at work, so there's organizations, locations, persons, and miscellaneous. And if you have nothing else, kind of having person, organization, location are the first big three for information extraction. And so the idea here is that, well, kind of a lot of the key information in text comes with these named entities. And a lot of the higher-level things that you want to do with text often involve named entities. So a lot of the time when people are asking questions, they're asking questions of named entities. So if they're saying, "Who starred in the 1993 cult film –" whatever cult films there were in 1993, I don't know. Then the answer to that will be a person name, so you'll be looking for a person name as the answer. To give one more example, sometimes you can do quite well, even with crude techniques that are nothing more than named entity recognition. So in the biomedical domain, where a lot of the time people are wanting to study the task of protein interaction, which means that the presence of one protein affects another protein in some way – it might be up regulating it, down regulating it – but there's some effect between the two proteins. The kind of baseline method of doing protein-protein interaction is you simply look for sentences that mention two protein names, and the implicit assumption is if two proteins have been mentioned in the same sentence, they must have something to do with each other. And so you simply just return pairs of proteins that occur in the same sentence, having done named entity recognition.

And it actually turns out that that works not so badly. And so then all the kind of work on top of that you essentially write your papers saying, "If I do this clever information extraction, syntactic parsing and other stuff, I can get results that are 5 or 10 percent better than just looking for ones in the same sentence." Okay. And so, essentially, these are the kind of tasks that you'll get as a named entity recognition task. You're going through words for text. You might actually have extra information, like the parts of speech of the word that we'll talk about later. And your goal is to identify entities. So here's an entity that's a person name. Here's an entity that's an organization name. Here's an entity, which is another organization name. And a very common way to do this, which is what we'll do in the next class, is doing what are called sequence models, where, effectively, you're going through each token in turn and you're assigning a class to it. So I'm assigning a class to these tokens, and then you have one extra class that's kind of the background class that I'm calling "O" for other, which are all the words that aren't named entities. So I'll just spend a couple minutes talking about evaluation of named entities. And one way you can evaluate named entities is sort of saying whether these individual tokens are right or wrong. But that’s not the standard way for a couple of reasons. Firstly, most of the text is background text that is other, in general. And it seems like you shouldn’t really gain a lot of points for getting that right. And secondly, it sort of seems like you want to capture whole entities. So the standard way that people do things is ask about whole entities, as to whether you’re getting them out correctly. And so then evaluation is done in a similar method to information retrieval of using notions of precision and recall, which I imagine a bunch of you have seen, but let me quickly say. So you draw this kind of two by two contingency matrix. And so you have four cells here: the true positives, which are the things that were entities and the system returned them; the false positive, which are the things that the system said were a person name, but actually weren’t; the false negatives, which are the things that are a person’s name, but the system didn’t return; and then this other box here is the true negatives. We don’t worry about that for the evaluation. And we evaluate just based on these three cells. And so we worked out precision, which is the proportion of the things that you returned, which were correct things to return, and recall, which is the percentage of the things that you should have returned that you did return.

Do those notions make sense? Can I rush ahead? Okay. And so that gives us two measures. And they’re interestingly different measures. Quite a lot of the time we just want to combine them into one measure, and that’s standardly done with something that everyone calls the F measure. And what the F measure is, is the harmonic mean. If you remember back to some high school class, if you did harmonic means. So this is the way to see it as the harmonic mean. So it’s a weighted – it’s a potentially weighted harmonic mean. So you can kind of weigh the two components of precision and recall, and then that’s the harmonic mean formula. But normally people don’t use that formula. Normally, people paramaterize it differently, like this, and so you’ll see references to beta equals three, or something, in the weighting of the harmonic mean. But in particular, the simplest case of the harmonic mean is when beta equal one, which corresponds to alpha equal a half, where you just weight precision, recall equally. And if you do that, and it’s easiest to do the math there, what you end up with is that the F measure is two times the precision times the recall divided by the precision plus the recall. And that’s the measure we’ll use later on in the third assignment. And the idea of the harmonic mean is it’s effectively a very conservative average. It means that to do well, you have to do well on both precision and recall. So that the nature of the harmonic mean is if the two values are very different, like if you have 90 percent precision and 10 percent recall, if you just did an arithmetic mean, you get 50 percent. Whereas, if you do a harmonic mean, I can’t quite do it in my head, but you’ll find out that you get something like about 21 percent, or something. That you’re way down in the direction of the minimum. Okay. Precision and recall are widely used, very standard, and straightforward for tasks like information retrieval, where you just return documents. There’s actually a subtlety in the way they work for information extraction, which is good to be aware of. That in classic information retrieval there’s only one grain size, and that’s the document grain size. And so it’s straightforward to draw that two by two contingency matrix. But when you have information extraction systems, you’re picking out entities, which can be of different sizes. And it turns out that the most common mistakes that systems make is that they get the boundaries wrong. So here the system has returned Bank of Chicago as an organization name, where what it should have returned if First Bank of Chicago as an organization name.

And you can kinda see how that's a very easy mistake to make, because the first word of a sentence is capitalized in English, so that's not much of a signal that it's a company name. And starting a sentence with "first," that's not unusual. First we did this, then we did that. So it's just regarding this is a common word, and then this is the name. So if you do an F measure evaluation of your named entity recognition system here, if you do something like this, you actually get dinged twice. You get dinged once for not returning First Bank of Chicago, that's a recall error. And then you get dinged the second time for returning Bank of Chicago, which is a precision error. And so, actually, you do better if you return nothing than returning something that's almost right, because then you'd only be dinged for the recall error. You could conclude that that's kind of broken, and I think many people think, in a sense, it is kind of broken. And there are various more complex scoring systems that have been used that have partial credit systems for things that are almost right. I mean, the problem with those partial credit systems is you then kind of have to have extra parameters and models as to how much credit to give for being how much of a percent right. If it's a six-word company name and you return one of those words, is that worth anything or nothing at all? And so a lot of the time, people don't worry about it and just use the F measure. Okay. So if you're computer scientists and have been around a bit and you're thinking about information extraction, the first thing you should think about is regular expressions. You can write regular expressions for some of these things. So if you want to recognize prices for things, that's the kind of thing you can write a regular expression for. If you're wanting to recognize phone numbers, you can start writing regular expressions for that. And if you start thinking more creatively, there are a lot more things that you can write, which are effectively regular expressions. I mean, in particular, for some of the higher-level things that you want to do, you might want to use parts of speech, that you want to recognize proper nouns or you want to recognize prepositions before them, or something like that. But anything that's a dictionary or lexicon is really just equivalent to a big regular expression. So if you have a word list of here's a long list of known digital camera manufacturers, that's just 100 percent equivalent to a regular expression, which is name one vertical bar, or name two vertical bar, name three, name four. All name lists are equivalent to big disjunction regular expressions.

So the only work in named entity recognition information extraction was, essentially, people handwriting rules, which started at the low level with regular expression, but then made use of various kinds of syntactic categories, like parts of speech and even syntactic parsing, to work out patterns of words and then to build patterns for extraction. And so people would write patterns. I think I have a couple of better examples later to recognize various kinds of patterns. And so this was actually where information extraction technology began. That information extraction, like the Internet, was invested by DARPA, not quite as famous. And so DARPA in the late – well, early to late 1990s – sponsored a whole series of workshops called the MUC Workshops, Message Understanding Conference Workshops. And the idea of them was to try and get people to build systems that did information extraction tasks. And essentially, that was a task that no one had considered before then. And then it was really as that task progressed that people invented the even lower-level task of maybe before doing full information extraction, we should build good systems that do named entity recognition. And it was during those years that at the start of the MUC competitions, essentially, all of the systems were hand-coded and grammar-based. And as the MUC competitions went on, statistical methods started to then infiltrate and start to show that they could do these things better. So here’s a classic example of MUC system. This was the system FASTUS that was done at SRI, which is over in Menlo Park. And FASTUS was, perhaps, a classic example of pattern matching system for doing regular expressions, so for doing information extraction. So the whole of it was cascaded regular expressions. So this is the sort of task that you wanted to do. You had the piece of text at the top and you wanted to do information extraction, where you were filling out these kind of templates.

So here you were having things like a relationship between two companies, what the companies were, and what the amount was. And then it had some structure here, so this table points over to this table. And this is saying what the activity was, it’s production activity. Here’s the company. There’s the product. And there’s the start date. So you have this somewhat complex database design that you want to be extracting facts into. And there are various bits of text that you want to match up. So effectively, what FASTUS did was very complex cascaded finite automata, i.e., fancy regular expressions, where the fanciness is that you’re allowed to have recursion in your regular expression so that you can write a base regular expression for a phone number, and then you can write more complex regular expressions that can use phone number as a subroutine. And so if would have regular expressions and lots of lexicons to be, actually, trying to parse up quite a bit of syntax. But this is the kind of thing that was actually being built. So that you were building noun groups up by using simple regular expressions, where a noun group could be a proper noun, where you could stop or then go with a possessive. Or it could be an article followed by a noun, where you could stop. Or you could have a preposition and another noun. You’re building up various bits of syntactic structure by doing regular expressions, in terms of lower level things of parts of speech, which could be recognized as a big list and regular expression. Okay. And so these are the kinds of patterns that you can write. And there’s some other work that I won’t talk about that then goes on to learn these kinds of patterns.

So if the relation that you want to extract is “where is an organization located,” if you can match organizations and locations, you can then write patterns that are kind of higher-level regular expressions. So [org] in [location] or [org] followed by [location], and then a word like “division,” “branch,” or “headquarters.” That that will work fairly well. If you want to determine who holds various positions, you can have patterns like: [person], [office] of [organization], which would match a pattern like this. And so people, essentially, would, in the early days, handwrite these patterns to match common things, and then some other work for automatically learning these patterns. Okay. So then for the last few slides today, I just wanted to sort of start to introduce the sort of very beginnings of how one might do that in a more probabilistic way, and then come back to that next time. First a question. Has everyone seen naïve Bayes classifiers, or are there people who have not seen naïve Bayes classifiers in some context? Nobody? Has everyone seen them? I hope. If not, you should definitely have a look at material on naïve Bayes classifiers. But I will say it very quickly. So a naïve Bayes classifier is going to be using Bayes rule, like everything else we do, and what we're wanting to do is find the class based on some attributes. We do Bayes rule and invert things around and say we have a prior probability of classes, and then we're using the probability of the attributes given the class. And the naïve bit comes in of saying, well, we can't possibly estimate this as a join probability, so we make this conditional independence assumption, where we pretend each thing is independent conditioned on the class, and so we just use estimates of this form. Okay. Now let me say the MLP particular part. For MLP, what we usually do is say that the xi are the words in a document. So we treat them as independent words in a document, but they are the actual word tokens that occur in a document. And so if you remember back to our language model lectures, that naïve Bayes model is neither more nor less than a class conditional unigram language model. And so, in particular, I won't dwell on this, but there's another way you can make a naïve Bayes model for text, where you take the x variables to be word types. So you have a variable for house and a variable for garden and a variable for travel and things like that. And you can do that and if you paid attention to 228, it might seem the more natural way to do it. But it actually doesn't work as well for just about any MLP application. So think of it as class conditional unigram language model, and you won't go astray. Yeah?

Student:

Where c is class?

Instructor (Christopher Manning):Yeah, c is the class that we're gonna assign things to. And if you think of them as language models, you'll hopefully remember all of the stuff that we did before that you'll want to smooth them and you'll probably want to work in log arithmetics so that things don't underflow. So if you have one of these as a little classifier, can you already do some kind of useful information extraction? And it turns out that you can. And this was a paper by Nick Kushmerick in the '90s, that what he wanted to do was to build a system that learned change of address emails, so he could automatically update his address book. So here's a change of address email. And he built it as a two-stage text classifier. So the first stage was binary text classification, kind of like spam or not spam. But his text classification was, is this a change of address email or is it not a change of address email? And then the second text classifier, it used a regular expressions to detect candidate email addresses, and then it used a window around that regular expression, where these windows are denoted in red. And each of those windows around the email address was then being classified by text classifier to say, is this a new email address in a change of address or is this not the new email address? So what you want to get out is that this one isn't the new email address, and this one is the new email address. And the clues for that are in the context. Things like "new email address is." Pretty good clues. And he did that and got quite good results. You can see the results there. They're not perfect, but they're quite good. So for some simple kinds of information extraction, you can just treat it as a text classification problem and use text classification technology and you're done. But note crucially, and this motivates what I'll do next time, this only works in situations where the entities are obvious. So he could just use a regular expression to identify email addresses, and then you can do a text classification problem, looking at the context to say, is this the person’s new email address? Whereas, typically, when we’re doing named entity recognition information extraction, we can’t use a regular expression to identify the content. That’s part of the problem that we want to solve probabilistically. And so that leads us to use more complex methods, which leads into the sequence models that I’ll talk about next time. Okay. Done.

[End of Audio]

Duration: 76 minutes