Lecture 17: Classification based on Unordered Contexts

All the work we have explored using tagging models has classified data using well-defined sequential contexts. For instance, an n-gram model estimates probabilities in terms of sequences of length n. In this class we look at models in which the sequential information is not considered, and the contexts are consider unordered sets of words. These approaches are used for a number of different tasks including

Word sense disambiguation- finding the appropriate senses for ambiguous words
Word clustering – finding words with common properties (e.g., similar syntactic or semantic properties)
Sentence Classification: we might want to classify sentences along some dimension such as the speech act that is performed (e.g., request, offer, promise, warn, ...);
Document Classification: We might want to classify documents as to their general topic, say to cluster similar web pages together in a search engine;
Message classification – say to route your incoming e-mail into different inboxes;

1. Probabilistic Methods for Text Classification

As a first example, consider the problem trying to classify sentences in a corpus as to whether they concern making an appointment or not. Let's assume we have the corpus shown in figure 1.

We meet at three	APPT
We went to the movie with three friends	OTHER
Shall I go to the movie at three thirty	APPT
I will win three at last	OTHER
My assignment is lost	OTHER
I don't have time to finish my assignment	OTHER
My assignment is late	OTHER
My last assignment is due at three	APPT

Figure 1: Some labeled sentences

We can develop an approach to this problem using the tools we have developed already. If we introduce a random variable T that ranges over the tags, then we want to find the tag with the maximum probability of the observed word sequence W1,N, i.e.,

$$\text{argmax}_v \, P(T = t \mid W_{1,N}).$$

by the standard application of Bayes rule and simplification, we know this reduces to

$$\text{argmax}_v \, P(T = t) * P(W_{1,N} \mid t)$$

P(T=t) is the prior. We can estimate this from the data above to determine

P(T=APPT) = 3/8 P(T/OTHER) = 5/8

To estimate P ($W_{1,N}$ | t), we will have to make some independence assumptions. The most radical is assuming all the w_i are independent, which is the unigram approximation. In contexts like this, this is called the The **Naive Bayes** Approach.
Thus would have

$$P(W_{1,N} \mid t) \approx \prod_i P(w_i \mid t)$$

To see how this works, We can estimate the unigram probabilities from the corpus using the counts in Table 2.

Word	Count for T=APPT	Count for T=OTHER
we, the, movie, last	1	1
meet, shall, go, thirty, due	1	0
at	3	1
three	3	2
I, is, to	1	2
assignment, my	1	3
went, with, friends, will, win, lost, don't, have, time, finish, late	0	1
TOTAL	20	30

Figure 2: The counts

Say we use the ELE estimator since we have such little data. Thus we add .5 to each count and the revised totals are 33.5 and 43.5 (size the vocab size is 27). We get the probabilities in table 3.

Word	P(w | APPT)	P(w | OTHER)
we, the, movie, last	.045	.034
meet, shall, go, thirty, due	.045	.011
at	.1	.034
three	.1	.057
I, is, to	.045	.057
assignment, my	.045 .	08
went, with, friends, will, win, lost, don't, have, time, finish, late	.015	.034

Figure 3: The probability estimates using the ELE estimator

Given the sentence

I will meet at three

we can compute the two probabilities:

P(APPT) * Pi (wi | APPT)

= .375 * .045 * .015 * .045 * .1 * .1 = 1.14 * 10-7

P(OTHER) * Pi (wi | OTHER)

= .625 * .057 * .034 * .011 * .034 * .057 = 2.6 * 10-8

Thus the Naive Bayes technique identifies this utterance as an APPT, which agrees with our intuition.

Other Elaborations

We sometimes could improve of this method by using better estimates of the P(W1,N | v). For instance, assuming we have enough data, we could associate bigrams with the different sentence tags and use a bigram approximation to approximate the probability calculations. Alternately, we might use some other estimation technique altogether. For example, we might simply take the maximum probabilty of elements in the sequence

$$P(W_{1,N} \mid v) = \text{Max}_{wi}\, P(w_i \mid v)$$

In other words, we base the estimate on the word most strongly associated with the tag. We could also use this technique with n-gram models, and use just the n-gram that is most closely associated with the tag.

2. Vector-based Representations

Another approach to this problem tries to account for the overall patterns of behavior in the sentence or document. We represent each sentence as a vector of values, with each word idenifying a particular dimension of the vector. When we classify a new sentence, we compute its vector representation and then see which sentence in the training set is closest in the vector space. To make this approach managable at all, we need to identify words that seem strongly associated with the decision we need to make and then use only those. This can help greatly reduce the amount of data we need to maintain because we only keep statistics on these important words. There are many ways in which this can be done. We could, for instance, build what is called a stop list of words that are so common in English that we expect them to appear in every class and thus they provide no useful information. This would include words like the, a, every, of, and so on. The technique is simple and used in many large-scale applications such as web-search engines, where the task is unstructured and it is impractical to use more sophisticated techniques.

For more focused tasks like the one at hand, where we have training data, we can explore other ways to choose informative words. But first, let's explore the basic technique. Let's just pick 6 words at random to form a vector representation. The sentences are then classified as in Figure 5, with the counts of each key word listed for each sentence. With a new utterance to classify, say

could you meet me at three,

we would compute its vector representation, (0,0,1,1,0,0) and then use a matching technique to find which training vector best matches it. The question is what is a good matching criterion?

Sentence	Class	We	Go	At	Three	To	Assignment
1	APPT	1	0	1	1	0	0
2	OTHER	1	0	0	1	1	0
3	APPT	0	1	1	1	1	0
4	OTHER	0	0	1	1	0	0
5	OTHER	0	0	0	0	0	1
6	OTHER	0	0	0	0	1	1
7	OTHER	0	0	0	0	0	1
8	APPT	0	0	1	1	0	1

Figure 4: Vector representations of 8 sentences

One obvious approach to do matching would be to compute the dot product of the vectors, which is written as V • W and defined as

$$V \bullet W = \sum_i v_i * w_i$$

If we use this measure to compare the third and fourth vectors, we get the following:

Sentence 3: (0,1,1,1,0,0) • (0,0,1,1,0,0) = 2
Sentence 4: (0,0,1,1,0,0) • (0,0,1,1,0,0) = 2

Note that this fails to capture the fact the the new sentence has the identical vector as sentence 4, but is not identical to sentence 3. Yet, they both receive the same score. A measure that solves this problem is called the **Dice Coefficient**. To define this, we first need a notion of the length of a vector V, which is defined as usual

$$|V| = \sqrt{\sum_i v_i^2}$$

Then the dice coefficent is defined as $(2*|X \cap Y|)/(|X| + |Y|)$.

Using the dice coefficient as a measure gives a match score of 4/5 to sentence 3 and 1.0 to sentence 4. Thus sentence 4 is the closest match and the new sentence would be classified as not an appointment (unfortunately, since this disagrees with my intuition - we'll fix it in a minute).

A very common measure for comparing vector representations is the **cosine measure**, which intuitively measures the angle between the vectors in n-space. The cosine measure is most easily computed as the dot product of normalized vectors, where a normalized vector is a vector of length 1. We can easily normalize a vector by dividing each element in the vector by the length of the vector: i.e.,

$$new\text{-}v_i = v_i / SQRT(|V|)$$

The normalized vector representation for our training corpus is shown in Figure 6.

The cosine match scores for sentence 3 and sentence 4 are .71 and 1.008 respectively. So as with the dice measure, utterance 4 is the closest and we would classify the new sentence as an OTHER. What if there are several with the same best score, and they differ on the tag assigned? One way to decide in this case would be to allow the tying scores to "vote" for the best tag. The **K-nearest neighbour technique** is a generalization of this technique. Rather than picking the sentences with just the best match, we take the K closest matches and then compute a weighted average vote. For example, say we choose k=2. The cosine match scores for the eight training vectors for *could you meet me at three*, (0,0,.71,.71,0,0) is given in Figure 7. The two best scores are 1.0 and .81, which include three vectors corresponding to sentences 1, 4, and 8. Sentences 1 and 4 are tagged with APPT. The votes we get for each tag are

Sentence	Class	We	Go	At	Three	To	Assignment
1	APPT	.57	0	.57	.57	0	0
2	OTHER	.57	0	0	.57	.57	0
3	APPT	0	.5	.5	.5	.5	0
4	OTHER	0	0	.71	.71	0	0
5	OTHER	0	0	0	0	0	1
6	OTHER	0	0	0	0	.71	.71
7	OTHER	0	0	0	0	0	1
8	APPT	0	0	.57	.57	0	.57

Figure 5: The normalized vector representation of the corpus

Vector	1	2	3	4	5	6	7	8
score	.81	.40	.71	1.0	0	0	0	.81

Figure 6: The closeness scores to each template

APPT: .81 + .81 = 1.62
OTHER: 1.0

With this measure, we would choose the tag APPT rather than OTHER which we obtained from the original 1-best technique. Note that because of the presence of ties, the K-nearest neighbor technique is actually selecting K sets of matching vectors rather than individual vectors (i.e., we take all the vectors with the K highest scores and compute the weighted average using them).

3. Word-sense Disambiguation

Another important application of these techniques is word sense disambiguation. For example, say we have the data

We eat corn chips/FOOD1 at the party
The manufacture of chips/COMP1 is expensive to do
Silicon valley chips/COMP1 are considered some of the best
I loved fish and chips/FOOD1 for dinner every night when I was living in Silicon Valley
We worked all night to diagnose the faulty chips/COMP1, having nothing but potato chips/FOOD1 to eat

For word sense disambiguation, it seems that the words closest to the word in question are most relevant. So rather than considering the entire text that a word appears in, we start with a **window** of words as the context for the algorithm. Say we take four words in each direction as the window. We now have a training set shown in Figure 7. Note that by setting the window size larger we get more context, but then run the risk of including irrelevant material. For example, the window size of 3 each way in sentence 4 would eliminate the misleading words "silicon valley" from being associated with the food sense.

Word (W)	Context ($C_{1,n}$)	Sense Tag (T)
chip	we eat corn _ at the party	FOOD1
chip	the manufacture of _ is expensive to	COMP1
chip	silicon valley _ are considered some	COMP1
chip	loved fish and _ for dinner every	FOOD1
chip	diagnose the faulty _ having nothing but	COMP1
chip	nothing but potato _ to eat	FOOD1

Figure 7: A series of contexts for the word *chip*

Given a sizable corpus, we could then disambiguate words using the techniques discussed earlier. We have random variables T (for the tag), W (for the word in question), and $C_{1,n}$ (for the n words in the context). In particular, given a word w in a context of n words $C_{1,n}$ we'd like to pick the sense tag t that is

$$\text{Argmax}_t \, P(T=t \mid W=w, \; C_{1,n})$$

Rather than rewriting this using Bayes rule as usual, we do a slightly different derivation that produces more intuitive derivation that leaves W as part of the condition, Using the definition of conditional probability we get

$$= \text{Argmax}_t \, P(T, W, C_{1,n}) \, / \, (W, C_{1,n})$$

= Argmax_t P(T, W, $C_{1,n}$) [dropping the denominator since it doesn't involve T]

= Argmax_t P(W) P(T | W) P($C_{1,n}$ | T, W) [rewriting using the chain rule]

= Argmax_t P(T | W) P($C_{1,n}$ | T, W) [dropping P(W) since it doesn't involve T]

The strongest assumption we can make is that all the conditions in the context are independent, which gives us a unigram approximation, which in this context is usually called the **Naïve Bayes** approximation. So we are computing

Argmax_t P(T | W) * $\prod_I$ P(C_i | T, W)

Consider an example. Say we have a hundred sentences involving the word *chip* and obtained the counts of words in each class (ignoring common words like *the, a, of*, etc) shown in Figure 8. Let's also assume we there were 30 instances of chips in the FOOD1 sense and 70 in the COMP1 sense in this corpus. Thus the prior probabilities, P(sense | w), are P(FOOD1 | chip) = 30/100 = .3 and P(COMP1 | chip) = 70/100 = .7

eat	party	manufacture	expensive	silicon	valley	considered	loved	fish
FOOD1	5	2	1	1	1	1	2	3
COMP1	1	4	2	6	4	5	3	4
	diagnose	faulty	potato	corn	dinner	hurt	industry	
FOOD1	1	1	5	6	4	1	1	
COMP1	6	3	1	1	3	1	3	

Figure 8: the counts from one hundred cases involving the word *chip*

Say we have the context,

... the manufacture of potato chips hurt the fish industry ...

After ignoring the common function words, we have a context including the words *manufacture, potato, hurt, fish,* and *industry*. Using the Naive Bayes approach, we compare

P(FOOD1 | chip) * P(manufacture|FOOD1) *P(potato|FOOD1) *P(hurt | FOOD1) * P(fish|FOOD1) * P(industry | FOOD1)
= .3 * 2/30 * 5/30 * 1/30 * 3/30 * 1/30 = 3.7e-7
P(COMP1 | chip) * P(manufacture|COMP1) *P(potato|COMP1) *P(hurt |COMP1) * P(fish|COMP1)
= .7 * 4/70 * 1/70 * 1/70 * 4/70 * 4/70 = 2.7e-8.

Thus we'd choose the food sense (FOOD1) in this context.

More elaborate models

Sometimes additional information, such as the parts of speech of the words in the context is also included, providing some smoothing of the probabilities.

Also, we could develop schemes that weighted words closer in the context to the ambiguous word more strongly than words further away. For instance, we might divide the context into the immediate context (say plus or minus 2 words) and the general context (words from 3 to 10 away). We could develop independent probability models for each of these contexts just as we did above, and then combine them using a linear interpolation scheme, where the weights are set based on performance on some development data. Or as a further generalization of this, if we use a probability function *weight*(j) that assigns a probability to each position j in the context, then we could use this as a linear interpolation of a series of probability functions, one for each position in the context, i.e.,

$$P(t \mid C_{1,n}) = \sum_i \text{weight}(i) * P(t \mid c_i)$$

4. Identifying Word Senses

One of the major problems in performing word sense disambiguation is that we don't have a clear notion of what the senses should be in the first place. Generally, these senses have to be constructed by hand. You might think that dictionaries would provide us with this information, but in general, dictionaries do not define word meanings in a formal enough manner to be useful for computational purposes. As a result, progress in this area is limited by how much training data can be constructed.

An alternate approach has been to use learning techniques to try to identify different word senses for words in an unsupervised fashion (i.e., directly from the data). The key intuition underlying these algorithms in that word senses (actually word properties in general) are reflected by the company they keep, i.e., what words tend to co-occur with them in context. For instance, consider this idea on something simpler that word senses for the moment, namely parts of speech. It makes sense that the different pasts of speech will tend to occur in different word contexts. Adjectives, for instance, will tend to follow words like *the* and *a* much more than other classes of words besides nouns. Adjectives, however, would tend to precede nouns more than any other class (except articles). Furthermore, adjectives often follow the verb *be* (e.g., it is red), and are probably the most common class to follow words like *very*. The idea behind clustering algorithms is that the combination of these tendencies will produce a unique probabilistic "signature" that could be used to group all adjectives together. A similar argument can be made for the senses of a word. If we look at all the contexts that a word occurs in, we may be able to find natural clusters among these contexts that would then tend to reflect the different senses that the word has.

Another useful resource for identifying plausible word senses exploits the knowledge we have in translating languages. If a word has multiple senses, then these senses will tend to be translated into different words in another language. While not always the case, as ambiguities in one language sometimes translate to another, in general this is a good method for identifying different senses. And note, when we have corpora of translations, once we have identified the set of senses, we also have a means of identifying the sense for each word. Thus we cold create a tagged training corpus for developing word sense disambiguation algorithms for each word.