Natural Language Processing (JHU 601.465/665)
Answers to "NLP Applications" practice problems

1. (a)
```
      B     O  O    O    O   B    I     I    I       O
  Yoyodyne is a company in San Narciso , California .
```

   (b) The illegal bigrams are O I and <s> I, since the first word of a chunk is supposed to be tagged as B rather than I. These can be ruled out by setting the transition probabilities $p(I \mid O) = p(I \mid \langle s \rangle) = 0$.

2. (a) ROOT -> S ROOT
   ROOT -> S

   where the first rule has some probability p and the second rule has probability (1-p). If p > 0, then this allows ROOT to rewrite as any sequence of 1 or more sentences. Of course, you'd better have p < 1 to avoid infinite recursion!

   (b) Increase p.

   (c) Instead use the rule ROOT -> S S S S S S (with probability 1).

   (d) You could classify every period -- or every character -- as to whether it ends a sentence or not. This is a binary classification problem. Features might include whether the next word is capitalized, whether the period follows a word that is known to be an abbreviation, whether the period appears to be an initial in the middle of a name, whether the period is only a few words away from another period (so that treating them both as sentence boundaries would give a very short sentence), etc.

3. (a) 90% precision means being nice to readers.

   (To be nice to writers, we'd want 90% recall, i.e., the system finds 90% of the good comments and displays them to users, perhaps with a lot of bad comments as well. It is possible to have high recall but very low precision, or vice versa.)

   (Meanwhile, choice iii. would be called 90% accuracy.)

   (b) It also needs high recall. It is very easy to get high precision by deleting almost everything and keeping only a few comments that are sure to be good.

   (c) 1. Download a selection of comments from the website.

   2. Get the publisher or some readers to annotate the comments as to whether they should be considered good or bad.

      One of you suggested a way around this: collect comments from a SIMILAR website where comments have already been publicly rated by the users.

   3. Divide this corpus into a large training set and a smaller test set.

      It is possible to also set aside some of the training set as a development set, but this isn't necessary if you are going to do jackknifing on the training data.

   (d) Some examples:
   - decision list (benefit: easy to train;
     problem: allows one feature to force the decision

against the objections of others)
- decision tree (problem: features are only considered in conjunction; the tree can only grow to a few levels deep because it quickly fragments the data)
- naive bayes (problem: assumes conditional independence of features. Notice that for this unigram feature set, it's equivalent to measuring the probability of the document against UNIGRAM language models of good and bad comments.)
- nearest neighbor (similarity in vector space to a training document or a cluster of training documents) (problem: curse of dimensionality -- in a high-dimensional space, everything looks similar; may need PCA to reduce dimensionality; also, similarity measure is sensitive to scaling of dimensions, and may not work well when the features are raw counts, as they are here)
- log-linear, also known in this case as logistic regression (allows features to interact without making assumptions; but does not consider interactions among features unless specifically added)
- perceptron or SVM (similar to log-linear, but trained differently)

In all of these cases, you could choose to transform the collection of x vectors by PCA before you start, in order to reduce the number of features and smooth them. (You could alternatively use feature hashing to reduce the number of features.)

Some of you suggested PCA all by itself. But PCA doesn't classify anything; it is a technique for reducing the dimensionality of the feature vectors.

Some of you also suggested EM. But EM doesn't classify anything; it is a technique for coping with missing data (for unsupervised or semi-supervised training). In fact, here we're still discussing a fully supervised problem.

(e) - Decision list: would usually be set up to pick the majority class ("good comment" or "bad comment")

- Decision tree: would be classified by some leaf that handles such examples

- Naive Bayes: would pick the majority class

- nearest neighbor: would agree with the document whose feature vector is close to 0 (this will lead to a tie if the "cosine distance" is used; it will look for short documents if Euclidean distance is used)

- log-linear: would give 1/2 probability to each class, if there are no features other than the unigram features (but probably there will be a "bias" feature that favors one class, which then gets higher probability)

- perceptron or SVM: similar to log-linear

(f) i. Is it grammatical? Log-probability PER WORD under a PCFG. Note that looking at TOTAL log-probability of a comment or sentence is not a good idea, because long comments will score poorly on this! Also, it may be a good idea to divide the PCFG probability by the unigram probability, to correct for rare words.

(Must make sure that the PCFG is robust, i.e., will not assign low probability to comments just because they're comments rather than Wall Street Journal articles.)

Does it look like well-written text? Log-probability per word under an n-gram language model trained on large amounts of such text. Similar to previous, but no PCFG.

Does it look more like good comments than like bad comments? That is, what is its log-probability per word under a smoothed trigram language model trained on good comments, minus its log-probability per word under a similar model trained on bad comments? (Note that in part (c), we were only using unigrams.)

Does it have a high reading level? Average number of syllables per word.

Is it substantive? Length of the comment in characters or words.

Is it correctly spelled? Fraction of words that are corrected by a context-sensitive spelling correction system.

Does it use too much capitalization? If a high percentage of characters are capitalized, the commenter is SHOUTING.

Does it use proper capitalization? For example, look at the fraction of letters following periods that are capitalized. This won't be 100% even in well-written text, because periods are sometimes abbreviations, but if it is very low, that suggests a careless writer.

Does it use proper punctuation? This is a bit hard to detect; you might be able to use a parser. But simply the presence of commas, semicolons, etc. might be helpful.

Does it curse? Number of words appearing on a list of rude words.

Apply any of the above measures to the individual sentences of the comment, and return the worst score of any sentence, on the theory that if any sentence is poorly written then the whole comment is bad.

ii. Train a language model on the news article (backing off to a general language model). Does the comment score more highly under this language model than under the general language model? (Or instead of comparing to the general language model, how good is the comment's score if we compare it to the scores of other comments or a held-out portion of the original article?)

Alternatively, how many n-grams in the comment also appeared in the article? Gives a different feature for each value of n.

Does the comment quote material from another relevant comment?

Is the comment close to the document in LSA space?

Use a topic classifier to identify topics for both the doc and the comment; make sure the topics match.

Could apply any of the above measures to the individual sentences of the comment, and return the best score of any sentence,

on the theory that if any sentence is relevant then the comment is worth keeping.

Some of you talked about using a threshold here to decide whether the comment was relevant. But you are not trying to make a final decision based on relevance! You are just computing some practical measures of relevance that you can use as SOME of the features of a classifier. The classifier gets to consider the actual relevance scores together with other features before making a final classification decision.

(g) Viterbi EM: Use the current imperfect classifier to automatically annotate all comments that were not manually annotated. Add these to the training set as if they had been manually annotated. Iterate.

Bootstrapping (self-training): Usually like Viterbi EM, except only add comments to the training set if the current classifier is quite sure of their class.

EM: Like Viterbi EM, but count the automatic classifications fractionally. If the current classifier is only 70% sure that the document is good, then add it to the training sets as 0.7 of a good comment and 0.3 of a bad comment.

4. (a) p(Author=Palin | Text=B)

$$= \frac{p(\text{Text=B} \mid \text{Author=Palin}) * p(\text{Author=Palin})}{p(\text{Text=B})}$$

$$= \frac{p(\text{Text=B} \mid \text{Auth=P})*p(\text{Auth=P})}{p(\text{Text=B} \mid \text{Auth=P})*p(\text{Auth=P}) + p(\text{Text=B} \mid \text{Auth=V})*p(\text{Auth=V})}$$

Note how p(Text=B) is expanded in the denominator.
Some people incorrectly expanded it as just
p(Text=B | Auth=P) + p(Text=B | Auth=V)
or equivalently as
p_P(B) + p_V(B)
But that sum could be greater than 1, in principle, if both the Palin language model and the Vincent language model tended to generate exactly the book B. The probability of generating B is not a sum of what the two models would do, but a weighted average of what they would do, where the relative weights 0.1 and 0.9 are given by the prior probability of which model it is. That is what is shown above.

(b) The above posterior probability rewrites as

$$\frac{p_P(B) * 0.1}{p_P(B) * 0.1 + p_V(B) * 0.9}$$

Requiring it to be > 0.9, we simplify to

p_P(B) * 0.1 > p_P(B) * 0.09 + p_V(B) * 0.81
p_P(B) * 0.01 > p_V(B) * 0.81
p_P(B)/p_V(B) > 81

So we will call the press if the Palin model was 81 times as likely to generate the text as the Vincent model was.

But there is a quicker and more direct way to see this, in terms

of "odds ratios." Our plan was to call the press if the posterior odds are at least 9 to 1 in favor of Palin:

$$\frac{p(\text{Author=Palin} \mid \text{Text=B})}{p(\text{Author=Vincent} \mid \text{Text=B})} > 9$$

Bayes Theorem implies that these posterior odds are given by the likelihood ratio times the prior odds. So we can rewrite the condition as

$$\frac{p(\text{Text=B} \mid \text{Author=Palin})}{p(\text{Text=B} \mid \text{Author=Vincent})} * \frac{p(\text{Author=Palin})}{p(\text{Author=Vincent})} > 9$$

Since the second factor is 1/9, we need the first factor to be > 81.

In short, the odds started out (a priori) at 9 to 1 in favor of Vincent. To cancel this out and swing the odds to 9 to 1 in favor of Palin instead, we need Palin to have a likelihood ratio of 81.

(c) Z = (exp sum_i f_i(Palin,T)*theta_i) + (exp sum_i f_i(Vincent,T)*theta_i)

(d) i. Discriminative training. Then some other process is assumed to have generated the sentence, and the only job of the discriminative model is to slap a label on it.

What's wrong with generative training here? Suppose that Palin's training corpus contains

... in Washington DC they ...
... in Washington DC they ...
... in Washington DC they ...
... in Washington they ...

whereas an equivalent amount of Vincent's training corpus contains

... in Washington DC they ...
... in Washington they ...
... in Washington they ...
... some other words ...

For simplicity, let's use only UNIGRAM models throughout this example, for both generative and discriminative training.

A generative unigram approach (without smoothing) observes that Palin is 4/3 times as likely to generate "Washington," and 3 times as likely to generate "DC." So it thinks she is (4/3)*3 = 4 times as likely to generate "Washington DC." In other words, adding "Washington DC" to a sentence will raise the generative system's odds in favor of Palin by a factor of 4 altogether.

This behavior is overconfident, just as we saw in class for Naive Bayes. (In fact, our UNIGRAM generative model is a special case of Naive Bayes!) It treats the appearance of "Washington" and "DC" in the same sentence as two independent pieces of evidence for Palin. But in fact they are hardly independent, since "DC" never occurs without "Washington." The unigram model is simply not a true model of the data.

A discriminative approach instead trains the weights of the unigram features to work together to predict the output. In this case, other things equal, it turns out that a discriminative log-linear model (without smoothing) would learn that "DC" multiplies the odds in favor of Palin by a

factor of 6, and "Washington" multiplies those odds by a factor of 1/2.

As a result, seeing "Washington DC" will multiply the odds in favor of Palin by a factor of (1/2)*6 = 3, which makes sense because she does say Washington DC three times as often. But seeing "Washington" without DC will actually halve Palin's odds, which also makes sense because it is Vincent who says "Washington" without DC twice as often.

Notice that these patterns arise simply by learning the interaction among unigram features. We are not able to learn any bigram features, like "Washington DC" to indicate Palin, or "Washington they" to indicate Vincent. In fact, we would take "Washington they in DC" to indicate Palin, not Vincent, since it contains both "Washington" and "DC" -- it has exactly the same unigram features as "in Washington DC they."

ii. This favors generative training.

The discriminative approach, as given, will incorrectly learn to predict Vincent with very high probability. Most of the training sentences are in fact from Vincent and it is rewarded for getting them right. It is simply respecting what it sees in training data.

The generative models do not have any opinion about whether Vincent or Palin is more probable. They only generate text GIVEN the author. They are to be combined with a separate prior over authorship, which we can specify freely, independent of the corpus size. Earlier in the problem, in fact, we decided to specify p(Palin)=0.1, p(Vincent)=0.9.

(In practice, one could fix the discriminative method, for example, by concatenating together many copies of the Palin corpus until it was exactly 1/9 as big as the Vincent corpus.)

iii. EM works with a generative model only. Its goal is to raise the probability (reduce the perplexity) of the observed data. Thus we need a generative model to assign some probability to the observed data. (The discriminative model only assigns some probability to the category, GIVEN the observed data.)

EM would use its current generative models to compute the posterior probability that each raw sentence was written by Palin. It would augment the Palin training corpus with these raw sentences, counting them fractionally in proportion to their probability of being written by Palin. Then it would retrain the Palin generative model on this augmented corpus. It would retrain the Vincent model similarly. Repeat until convergence.

iv. Decision list: discriminative.

Decision tree: discriminative.

Clustering: generative. For example, Gaussian mixture model clustering is an example of EM, so it only works with generative models (see above). More generally, clustering methods are unsupervised, so they cannot be looking to discriminatively predict the labels in training data: we didn't see the labels! Rather, they are looking for parameters that would predict (explain,

generate) the pattern that we actually saw.

(e) A mixture model: Flip a (weighted) coin to decide which author would write the next sentence. Then generate the sentence from that author's n-gram model. Notice that each sentence is generated independently of the others.

A PCFG with author attributes: Instead of S -> NP VP, you'll have

```
S[Author=Palin]   -> NP[Author=Palin]   VP[Author=Palin]
S[Author=Palin]   -> NP[Author=Palin]   VP[Author=Vincent]
S[Author=Palin]   -> NP[Author=Vincent] VP[Author=Palin]
S[Author=Palin]   -> NP[Author=Vincent] VP[Author=Vincent]
S[Author=Vincent] -> NP[Author=Palin]   VP[Author=Palin]
S[Author=Vincent] -> NP[Author=Palin]   VP[Author=Vincent]
S[Author=Vincent] -> NP[Author=Vincent] VP[Author=Palin]
S[Author=Vincent] -> NP[Author=Vincent] VP[Author=Vincent]
```

These rules allow different authors for different phrases even within the same sentence. However, the first and last rules are much more probable than the others, because most sentences are written by a single person. Also, the first rule might be more probable than the last rule, because Palin's writing style might expand S into NP VP more often than Vincent's.

A noisy channel model: Let's say that Palin writes and Vincent edits. So the model has the form

```
p(Draft=A) * p(Text=B | Draft=A)
```

i. To compute the probability of generating the text, you would probably take a product over all sentences in the text, assuming they're independent:

```
p(Text=B) = p(Text_1=B_1) * p(Text_2=B_2) * ...
```

The probability of each sentence depends on the model you chose.

Under the mixture model, it's the sum of the 2 ways to generate the sentence:

```
p(Text_i=B_i)
    =   p(Author_i=Palin)   * p(Text_i=B_i | Author_i=Palin)
      + p(Author_i=Vincent) * p(Text_i=B_i | Author_i=Vincent)
```

Under the PCFG, you'd use the inside algorithm to sum up all of the parses of the sentence.

Under the noisy channel model, let's suppose that the source model p(Draft_i=A_i) is given by a probabilistic FSA called Source, and the channel model p(Text_i=B_i | Draft_i=A_i) is given by a probabilistic PFST called Channel. Then the composition

```
Source .o. Channel
```

is a probabilistic FST that accepts a string pair (A,B) with probability

```
p(Draft_i=A_i) * p(Text_i=B_i | Draft=A_i)
= p(Draft_i=A_i, Text_i=B_i)
```

To get p(Text_i=B_i), you need to find all of the paths that produced B_i as the lower string. Those are exactly the accepting paths in the FST

```
Source .o. Channel .o. B_i
```

and to get their total probability, you'd run the forward or backward algorithm over that FST.

ii. Under the mixture model:

The probability that Palin wrote sentence B_i is

$$p(\text{Author\_i=Palin} \mid \text{Text\_i=B\_i}) = \frac{p(\text{Author\_i=Palin}) * p(\text{Text\_i=B\_i} \mid \text{Author\_i=Palin})}{p(\text{Text\_i=B\_i})}$$

Thus, a good estimate of the fraction of words of book B written by Palin is

$$\frac{\sum_i \text{length}(B_i) * p(\text{Author\_i=Palin} \mid \text{Text\_i=B\_i})}{\sum_i \text{length}(B_i)}$$

Under the PCFG model:

This is a similar idea. Instead of asking which sentences were probably written by Palin, we want to ask which constituents were probably written by Palin. Of course, some of the possible constituents were written by no one!

The estimate in this case is

$$\frac{\sum_i \sum_c \text{length}(c) * p(c \mid B_i)}{\sum_i \text{length}(B_i)}$$

where c ranges over possible Palin-written constituents of sentence B_i. Each c is a triple such as (NP[Author=Palin], 5, 12), where 5 and 12 are the start and end positions, and NP[Author=Palin] is the nonterminal. Note that we sum over only nonterminals with the attribute Author=Palin.

The probability p(c | B_i) is obtained by the inside-outside algorithm.

Under the noisy channel model:

Again, let's assume that Palin wrote the draft and Vincent edited it. In this case it's not clear what it means to say that Palin wrote half of the book, but a reasonable idea is to say that half of the words in the final book were just copied from words written by Palin.

An "easy" approach is that for each sentence B_i, you find
    Source .o. Channel .o. B_i
as before. Then you take the most probable path (Viterbi path) as your best guess about the draft A_i and its alignment to B_i. If B_i has 20 words and this path says that 10 of them were emitted by arcs such as Alaska:Alaska that just copied one of Palin's word, then we can say that Palin wrote half of this sentence.

A slightly better approach is to use all paths instead of just the Viterbi path. You can use the forward-backward algorithm to compute the posterior probability of each arc. Then by adding up the posterior probabilities of the "copy" arcs, we can say that an expected 10.2 words (of 20 total) were copies of Palin's words.