ANLP Tutorial Exercise Set 4 (for tutorial groups in week 8) WITH SOLUTIONS

v1.2
School of Informatics, University of Edinburgh
Sharon Goldwater

The solutions for this week's tutorial include a lot of additional notes for those who are interested in more details. The information in these notes is not part of the core material required for this course, but may be useful for other courses or to understand these models at a deeper level.

Exercise 1.
In lecture we introduced *pointwise mutual information* (PMI)[1] a measure of statistical independence which can tell use whether two words (or more generally, two statistical events) tend to occur together or not. We'll see it again in the Week 9 lab. The PMI between two events x and y is defined as

$$PMI(x,y) = \log_2 \frac{P(x,y)}{P(x)P(y)} \tag{1}$$

Let's consider two examples:

- x is "*eat* is the first word in a bigram" and y is "*pizza* is the second word in a bigram".
- x is "*happy* occurs in a Tweet" and y is "*pizza* occurs in a Tweet".

a) For each example, what does $P(x,y)$ represent?

b) What do negative, zero, and positive PMI values represent in terms of the statistical independence of x and y? (*Hint:* consider what must be true of the relationship between $P(x,y)$ and $P(x)P(y)$ for the PMI to be negative, zero, or positive.) Give some example pairs of words that you would expect to have negative or positive PMI (in either the bigram or Tweet scenario).

c) Normally in NLP we do not know the true probabilities of x and y so we must estimate them from data (as we will do in next week's lab with a real dataset of Tweets). Assume we use MLE to estimate probabilities. Write down an equation to compute PMI in terms of *counts* rather than *probabilities*. Use N to represent the total number of observations (e.g., the total number of bigrams in the first example, or the total number of Tweets in the second example).

d) Now, using your MLE version of PMI and the following toy dataset, compute $PMI(x,y)$, $PMI(y,z)$, and $PMI(x,z)$. You'll use the answers to error-check your code in next week's lab.

$$\begin{array}{rcrcrcr} N & = & 12 & & & & \\ C(x) & = & 6 & \quad & C(x,y) & = & 2 \\ C(y) & = & 4 & & C(x,z) & = & 1 \\ C(z) & = & 3 & & C(y,z) & = & 2 \end{array}$$

[1] In NLP this is sometimes also just called *mutual information*, although that term is used elsewhere for a related but not identical concept. So I'm using PMI here to avoid confusion.

Solution 1.

a) The probability of the bigram *eat pizza*, or the probability of using the words *happy* and *pizza* in the same Tweet.

b)
- Negative: x and y are *less* likely to occur together than if independent. Ex: *treaty* and *pizza*, two words which are very unlikely to be used together in the same conversation/text, much less in a single Tweet or bigram.
- Zero: x and y are independent.
- Positive: x and y are *more* likely to occur together than if independent. Ex: *pepperoni* and *pizza*.

Note: We are talking about independence between *events* here. As a reminder (see also the definitions in the Basic Probability Theory reading), two events x and y are independent iff $P(x,y) = P(x)P(y)$. Whereas for *random variables* X and Y to be independent, we require that $P(X,Y) = P(X)P(Y)$ for all possible values of X and Y. There is an analogous distinction between PMI and (non-pointwise) Mutual Information (MI, or just I). Mutual Information is defined between two RVs X and Y as the expected value of PMI across all possible values of X and Y (here, all possible choices of word pairs):

$$I(X,Y) = \sum_{x \in X} \sum_{y \in Y} P(x,y) log_2 \frac{P(x,y)}{P(x)P(y)} \tag{2}$$

$I(X,Y)$ is zero if X and Y are independent, and positive otherwise (with larger values indicating less independence: knowing X tells you more about the value of Y and vice versa.) Unlike PMI, MI cannot be negative. MI is used in information theory and other areas of computer science, but not as much in NLP, where PMI is more useful.

c) PMI using counts is:

$$PMI(x,y) = \log_2 \frac{N \cdot C(x,y)}{C(x)C(y)} \tag{3}$$

which can be derived from the fact that the MLE estimates are $P(x) = C(x)/N$, $P(y) = C(y)/N$, $P(x,y) = C(x,y)/N$.

d) $PMI(x,y) = 0$, $PMI(x,z) = \log_2(2/3) = 1 - \log_2 3$, $PMI(y,z) = 1$.

Exercise 2.

Suppose we are using a logistic regression model for disambiguating three senses of the word *plant*, where y represents the latent sense.

y	sense
1	Noun: a member of the plant kingdom
2	Verb: to place in the ground
3	Noun: a factory

a) In lecture (and textbook) We saw the equation for $P(y|\vec{x})$ in a logistic regression model. Write down a simplified expression for the log probability, $\log P(y|\vec{x})$. Can you see why logistic regression models are also called *log-linear* models?

b) Imagine we have already trained the model. The following table lists the features $\vec{x}$ we are using and their weights $\vec{w}$ from training:

feat. #	feature	weight
1	`doc_contains('grow')` & y=1	2.0
2	`doc_contains('grow')` & y=2	1.8
3	`doc_contains('grow')` & y=3	0.3
4	`doc_contains('animal')` & y=1	2.0
5	`doc_contains('animal')` & y=2	0.5
6	`doc_contains('animal')` & y=3	-3.0
7	`doc_contains('industry')` & y=1	-0.1
8	`doc_contains('industry')` & y=2	1.1
9	`doc_contains('industry')` & y=3	2.7

where `doc_contains('grow')` means the document containing the target instance of *plant* also contains the word *grow*.

Now we see a new document that contains the words *industry*, *grow*, and *plant*. Compute $\sum_i w_i f_i(\vec{x}, y)$ and $P(y|\vec{x})$ for each sense y. Which sense is the most probable?

c) Now suppose we add some more features to our model:

feat. #	feature
10	`POS(tgt)=NN` & y=1
11	`POS(tgt)=NN` & y=2
12	`POS(tgt)=NN` & y=3
13	`POS(tgt)=VB` & y=1
14	`POS(tgt)=VB` & y=2
15	`POS(tgt)=VB` & y=3

where `POS(tgt)=NN` means the POS of the target word is `NN`.

We train the new model on a training set where all instances of *plant* have been annotated with sense information *and* the correct POS tag. What will happen to the weights in this model? (*Hint:* You might want to start by considering w_{14} in particular. If you can figure that one out, then start to think about the others.)

d) Suppose we stop training the new model after a large number of training iterations. We then use the model on a test set where POS tags have been added automatically (i.e., there may be errors). What problem will this cause with our WSD system? What are some ways we could change our model or training method to try to solve the problem?

Solution 2.

a) Using the dot product notation $\vec{w} \cdot \vec{f}(\vec{x}, y)$ to indicate $\sum_i w_i f_i(\vec{x}, y)$, we have

$$\begin{aligned} \log P(y|\vec{x}) &= \log \frac{1}{Z} \exp\left(\vec{w} \cdot \vec{f}(\vec{x}, y)\right) & (4) \\ &= \log\ \exp\left(\vec{w} \cdot \vec{f}(\vec{x}, y)\right) + \log \frac{1}{Z} & (5) \\ &= \vec{w} \cdot \vec{f}(\vec{x}, y) - \log Z & (6) \end{aligned}$$

The expression on the right-hand side is a linear combination of the feature values (that is, each value is scaled linearly by its weight, and then we add them together; $\log Z$ is a constant). So these models are called log-linear because the log probability is a linear function.

Note: I used a natural log (base e), so it cancels the exp (because $\log_a a_x = x$). But if we used a different base (say, 2), we could use the fact that $\log_b a = \log_c a / \log_c b$. Then $\log_2 P(y|\vec{x})$ is just Eq (6) divided by $\ln 2$: still a linear function.

b)
- $y = 1$: Only f_1 and f_7 are active (have value 1), and all other features have value 0. So $\sum_i w_i f_i(\vec{x}, y) = 2.0 - 0.1 = 1.9$.
- $y = 2$: Only f_2 and f_8 are active, so $\sum_i w_i f_i(\vec{x}, y) = 1.8 + 1.1 = 2.9$.
- $y = 3$: Only f_3 and f_9 are active, so $\sum_i w_i f_i(\vec{x}, y) = 0.3 + 2.7 = 3.0$.

Without even computing the probabilities, we can see that 3 is the most probable, followed by class 2 and class 1.

To compute the actual probabilities, first take exp() of each value to get 6.68, 18.17, and 20.09. Then normalize by the sum of those values, giving 0.149, 0.404, and 0.447 as the probabilities of class 1, 2, and 3, respectively.

You probably noticed that the set of features that is active for each class is always distinct: that is the role of the '& $y = 1$' part of the feature. This can be a bit confusing at first, but remember these features are really feature *functions*: each feature is a function of the observations $\vec{x}$ and the class y. In training, we see both $\vec{x}$ and y. In testing we see only $\vec{x}$, so to compute the probability of a particular y, we use the feature that matches that $\vec{x}$ and y in the numerator. In the denominator we need to consider features matching all values of y.

Note: In some parts of machine learning, these models are presented slightly differently, with the observations $\vec{x}$ themselves referred to as "features". So, one feature might be `doc_contains('animal')`. In this case, the *same* features are active regardless of the class, but we assume that each class has its own distinct weight vector. So for this example, instead of having a single weight vector with 9 values, we would have three different vectors of length 3, one for each of the three classes. The two formulations are equivalent in the end, but the notation is different. I used the version here for consistency with the textbook, and because (as mentioned in class), it makes it possible to use MaxEnt (logistic regression) for n-best re-ranking when the classes are not pre-defined.

c) In the training scenario I described, the tag `VB` is perfectly predictive: every time we observe `VB` (that is, our observations $\vec{x}$ for a particular training example include `POS(tgt)=VB`), we see the label $y = 2$ for that example. So (unless we apply regularization: see below), our model wants to choose $\vec{w}$ such that $P(y = 2|(\texttt{POS(tgt)} = \texttt{VB}) \in \vec{x}) = 1$. For simplicity, let's assume POS tags are the *only* observations we use in our model, so we only have features $f_{10} - f_{15}$ (a similar argument holds even if other features are included). If we want $P(y = 2|(\texttt{POS(tgt)} = \texttt{VB}) \in \vec{x}) = 1$, then we need to have

$$1 = \frac{\exp(\vec{w} \cdot \vec{f}(\vec{x}, y = 2))}{\exp(\vec{w} \cdot \vec{f}(\vec{x}, y = 1)) + \exp(\vec{w} \cdot \vec{f}(\vec{x}, y = 2)) + \exp(\vec{w} \cdot \vec{f}(\vec{x}, y = 3))} \quad (7)$$

$$= \frac{\exp(w_{14} f_{14})}{\exp(w_{13} f_{13}) + \exp(w_{14} f_{14}) + \exp(w_{15} f_{15})} \quad (8)$$

$$= \frac{e^{w_{14}}}{e^{w_{13}} + e^{w_{14}} + e^{w_{15}}}. \quad (9)$$

where the second and third lines follow because for the observation `VB`, exactly one feature is active and has value 1 for each possible class, and all others have value 0.

But now the problem is clear: there is no way to satisfy this equation with finite weights. We would need the first and third terms in the denominator to be zero, but there is no finite value n for which $e^n = 0$.

Instead, what will happen is (because the training procedure changes the weights iteratively), the value for w_{14} will grow larger with every iteration, to make the probability closer and closer to 1, and/or the values for w_{13} and w_{15} will grow more and more negative. But these values will never converge!

As for w_{10} and w_{12}, these are associated with the `NN` observation. Let's say for the sake of argument that 60% of the training examples tagged `NN` are class 1, and 40% are class 3. Then w_{10} and w_{12} will end up with finite values such that the probability of class 1 given `NN` is 0.6 and the probability of class 3 is 0.4. However, we will run into another problem with w_{11}, because this time `NN` is perfectly predictive of *not* being in class 2. Running through a similar argument to the one above shows that the model will try to make $w_{11} = -\infty$.

d) The problem is that whenever there is an error in the automatic POS tag assigned to a test case, our model will give it the wrong sense, regardless of how much evidence there is from other observations. That's because the model treats the POS tags as perfectly predictive and doesn't care about any other features.

To avoid this problem, and more generally to avoid the problem of infinite weights in the model, we could do one of two things. First, we could train the model on the data set where the POS tags are *also* automatic, so that the training examples would also contain some errors and the tags would not be perfectly predictive. (Plus, hopefully the errors in the training set would be similar to those in the testing set and the model would learn to generalize.)

However, this solution isn't as general as we might like, because there might be other observations that are (accidentally) perfectly predictive of some class. This is especially true if we have very many features, some of which are rare. A feature that only fires for one or two training examples is likely to be perfectly predictive in the training data. This is basically a problem of overfitting. And, as mentioned briefly in the lecture, the way to solve that is through *regularization*. Regularization adds an extra term to the objective function. This term can take different forms, but all of them are designed so that infinite weights will no longer be optimal under the regularized objective. So, adding a regularization term allows the model to generalize better and avoid overfitting. Any standard package implementing logistic regression models should have options to include one or more types of regularization.

Note: In practice, logistic regression models should also include a *bias term* for each class. This is a feature that *only* includes the class label, for example we'd add f_{16}, f_{17}, f_{18} to our model above, which would fire whenever $y = 1, 2$, or 3 (respectively). The purpose of these bias terms is to help model the prior probabilities of each class. For example, we might find that the prior probability of class 1 is much higher than classes 2 and 3. We can model this by setting w_{16} appropriately. Then the other weights simply adjust the probability of class 1 up or down from this baseline.

It's important that regularization should *not* be applied to the bias terms, since by definition they are based on many examples and won't overfit.