Natural Language Processing (JHU 601.465/665)
Answers to "Log-Linear Modeling" practice problems

1. (a) 0.2.

Remember that with the maximum-likelihood parameters, the expected features of a log-linear model match the observed features. (Because that's what makes the gradient vector 0 -- recall http://cs.jhu.edu/~jason/tutorials/loglin/#6 and the log-linear handout.)

The given formulas can be interpreted as saying that there are 3 binary features. One fires only on "bwa," one fires only on "bwee," and one fires only on "kiki."

The "bwee" feature is observed to fire 0.2 of the time, so in the maximum likelihood solution, the model matches this and predicts that it will fire 0.2 of the time. In this case, that simply means that p(bwee) = 0.2.

(b) log 0.3, log 0.2, log 0.5 gives probabilities 0.3, 0.2, 0.5 as required. In this case Z=1.

(c) For any constant b, b+log 0.3, b+log 0.2, b+log 0.5 still gives probabilities 0.3, 0.2, 0.5. In this case, Z=exp b. So just choose any b != 0 to get a different solution.

(d) The L1 regularizer prefers smaller values of
|theta_bwa| + |theta_bwee| + |theta_kiki|

Which value is best depends on what value of b you chose. In this case, the L1 regularizer is minimized if you choose b = -log 0.3, so that the median weight is 0 and the other weights are close to 0.

(Of course, the regularizer would be even happier if you also reduced the distance between the weights, bringing them all closer to 0! That gives you the regularized solution, which is NOT a maximum-likelihood solution.)

(e) 0.25. We observe the "bw" feature in 0.5 of the training data, so under the maximum-likelihood parameters, the model expects this feature to fire 0.5 of the time. That means p(bwa)+p(bwee) = 0.5. But according to the formulas, p(bwa)=p(bwee): the features don't distinguish between them. Therefore p(bwee) = 0.25.

(f) Yes. The new model will correctly predict the observed counts of bwa, bwee, and kiki in an average sentence. Thus, it will correctly predict the SUM of these counts, which is 4.0.

To be more concrete, if there are N training sentences, then the information in the problem implies that there are 4N training words, comprising 4N*0.3 bwa tokens, 4N*0.2 bwee tokens, and 4N*0.5 kiki tokens. Thus the average observed count per sentence is 4*0.3 bwa, 4*0.2 bwee, 4*0.5 kiki. The trained model's predictions will match these counts, and therefore it predicts 4*0.3 + 4*0.2 + 4*0.5 = 4 tokens per sentence.

2. (a) f_1(s,l,y) = (if l==Esperanto then y else 0).

Note that this is a real-valued feature (think of the "number of sides" feature in the log-linear model

over triangles, squares, and pentagons).

If this has a negative weight, then p(y | s,l)
will decrease exponentially with y (when l==Esperanto).
That is, it is unlikely that an Esperanto student
will take a lot of years.

(When l != Esperanto, p(y | s,l) is constant at 1/11.
No features fire, so all 11 outcomes are equally likely.)

(b) In training, we are trying to maximize log p(y | s,l), summed over all training triples (s,l,y).

The main point: this value is optimized when the model predicts an expected total value of f_1 over the training set that equals its observed value. In other words, the model should expect that the average Esperanto student will take 1.6 years to become fluent, just as we observed.

We would expect theta_1 to be negative: as explained above, that's when the model prefers as few years as possible.

To be formally precise about it, the observed number of firings of f_1 (summing over examples i in the training set) is

```
   sum_i f_1(s_i,l_i,y_i)
```

and the expected number is

```
   sum_i sum_y p(y | s_i,l_i) f_1(s_i,l_i,y)
```

By the definition of f_1, the observed number is

```
   1.6
```

and the expected number is

```
   sum_y p(y | s_i,l_i) * y
   = sum_y (1/Z) exp(theta_1*y) * y
```

where Z = sum_y exp(theta_1*y) and y ranges over 0,1,...10.

If we let t = exp(theta_1), then we can rewrite the expected number as

```
    sum_y (t^y / Z) * y
    = sum_y t^y * y / (sum_y t^y)
```

The exam asked for a strategy to find the optimal theta_1. We could use the same gradient ascent strategy that we've been using all along: increase theta_1 when expected < observed, decrease theta_1 when expected > observed.

Or we can just solve for the point where expected = observed. That is, theta_1=log(t) is optimized when

```
    sum_y t^y * y / sum_y t^y = 1.6
```

and this could be solved quite simply by the bisection method ("binary chop") since it is merely a function of one variable. Actually doing this gives theta_1 = log(t) = -0.471.

Just for fun, if you like playing with formulas:

We can get a good approximation to the optimal t by letting y in the summation range over 0,1,...infinity instead of 0,1,... 10. This is a more natural problem anyway than artificially capping the number of years at 10. And it won't make much difference since t^y decays exponentially fast and will be tiny for large y.

In that case, sum_y t^y = 1/(1-t) by the usual formula for the sum of a geometric series. And we can write a recurrence for sum_y t^y * y as well:

```
   if S = sum_y t^y * y
```

```
            = 1 * 0 + t * 1 + t^2 * 2 + ...
      then S - (t + t^2 + t^3 + ...) = tS
      so (1-t)S = t + t^2 + t^3 + ... = t/(1-t)
      so S = t/(1-t)^2.
   Thus, the equation we're trying to solve becomes
      (t/(1-t)^2) / (1/(1-t)) = 1.6
      t/(1-t) = 1.6
      t = 1.6 - 1.6t
      2.6t = 1.6
      t = 1.6/2.6 = 0.6153
    and we estimate theta_1 = log(t) = -0.486.
```

Checking this, this value of t does give a pretty accurate answer even to the original problem where we only sum over y=0,1,...10. It turns out that sum_y t^y * y / sum_y t^y = 1.55 which is close to the desired 1.6.

(c) Because theta_1 is negative, the most common amount of time is 0 years. The probability of y falls off exponentially as y increases.

(d) f_2(s,l,y) = (if y==0 then 1 else 0).

(e) theta_2 = -infinity. This means that p(y=0 | s,l) = 0: it is impossible to take 0 years. After all, we've never observed 0 years! This may be overfitting a bit, but it leaves as much probability as possible for the things that we did observe.

This will change the optimal value for theta_1.

(f) To capture whether Helen is good at languages, we can have f_Helen(s,l,y) = (if s==Helen then y else 0).

This is completely analogous to the Esperanto case. A negative weight means that Helen tends to take few years. A positive weight means that she tends to take many years.

(g) The weight of the feature f_you had no effect on the objective function. You didn't appear in training data; the model had no knowledge of whether you'd take many years or few years, and had no incentive to predict that you'd take many or few years.

So without regularization, the answer is underdetermined -- the optimizer could equally well pick any value of theta_you. All values are tied.

Regularization breaks the tie: then the optimizer will pick theta_you = 0 in order to make the regularizer as happy as possible. This says that you are neither good nor bad at languages. The model will treat you as a typical student without any personal adjustments for you. And that seems like the appropriate result since the model never saw you during training and has no special information about you.

3. (a) China black cars: {f1} -- total weight 1
China white cars: {} -- the empty set, with total weight 0
India black cars: {}
India white cars: {}

Observing that exp(1)=e and exp(0)=1,

p(c = black | r = china) = e / (e+1+1+1+1+1+1+1) = e/(e+7) = 0.27

p(c = black | r = india) = 1 / (1+1+1+1+1+1+1+1) = 1/8 = 0.125

(b) China black cars: {f1,f2,f3} -- total weight 6
China white cars: {f3} -- total weight 3
India black cars: {f2} -- total weight 2
India white cars: {} -- total weight 0

p(c = black | r = china)
= e^6 / (e^6 + 7 * e^3) = e^3 / (e^3 + 7) = 0.74

Notice that f3 had no effect here because it was always true whenever r=china. It fired on every term in the numerator and denominator, so the extra factors of e^3 in the numerator and denominator canceled out.

p(c = black | r = india) = e^2 / (e^2 + 7) = 0.51

(c) 500 car tokens (100 from each region).

(It would be only 481 tokens if you left out the OOVs, but as we'll see below, you shouldn't do that. They are part of the data.)

DOESN'T MATTER whether the first objective includes the OOV row. If the model perfectly predicts all the answers in the first 7 rows, then it must also perfectly predict all the answers in the 8th row (OOV), as being the values that make each column sum to 1.

YES, the second objective should include the OOV row. Otherwise, it would assign 0 probability to OOV in order to maximize the probability of the in-vocabulary events that *are* in the objective colors. But this would not match the table, which shows that OOV colors have positive probability.

Note: As HW3 said, the vocabulary is whatever you define it to be. It doesn't have to be the set of words or colors observed in training data, and in fact it's not in this case (unlike in HW3).

(d) Formula for regularizer:
-C * (sum[c,r] theta[c,r]^2 + sum[c] theta[c]^2 + sum[r] theta[r]^2)

No, the optimizer will no longer fit the data exactly, because there is a tension in the regularized objective between fitting the data and keeping the weights small. The weights will be a little closer to 0 than before.

With a regularizer, the learned weights may be more useful for predicting car colors in other countries or other years that are not in this training dataset. In effect, it will do better on test data -- data that don't appear in the training table. (Without a regularizer, we will fit the training data better, but possibly by picking crazy weights that won't generalize well.) Some of you said that the regularizer would help with predicting unseen colors, but there are only 8 categories that will ever be seen (including OOV), and all of them have already been seen in training data.

theta[brown]: NEGATIVE because brown cars are less likely than the average color, across almost all regions.

(This could also be modeled by giving

f[brown] a positive weight that is still less positive than the other f[c] features. However, this solution can't be optimal, since if all the f[c] weights are positive, we can improve the regularizer without changing the likelihood, by subtracting a small constant from all weights theta[c].)

theta[india]: ZERO because this feature doesn't discriminate between different car colors and thus will have no effect at all on likelihood, as we saw in part (b). So the only place where theta[india] actually matters is in the regularizer, which prefers to keep it at 0.

theta[brown,india]: POSITIVE because brown cars are more popular in India than elsewhere; this positive weight partly cancels out the negative weight for theta[brown]

Between 0 and 1/8: Although none of the f[c,r] or f[r] features will fire on Australia, the f[c] features will fire. And these features should recognize that brown is generally less common than the other colors.

Like f[india], the Australia-specific features will never fire in either the training events or competing events in the same training contexts. Therefore, their weights will not affect the probabilities that are being measured by the objective. Only the regularizer cares about their weights, and prefers those weights to be 0.

(e) 1 if c=black or c=white

This will get a positive weight. (This following variant would also work, but would get a negative weight: "1 if not (c=black or c=white)".)

This feature will fire on every event that has a black or white car color, resulting in a positive weight being added for those events. As a result, the other weights of those events can be reduced to compensate.

Therefore, the weights theta[black,india] and theta[white,india] will go down.

(f) [extra credit]

This does not change the ratio of colors in any region. However, it increases the number of European cars in the training corpus from 100 to 1000. Thus, the objective more strongly emphasizes matching the probabilities for European cars. The resulting $p(c \mid r)$ will fit the European column of the DuPont table more accurately, and will tend to fit the other columns less accurately.