Ben Tankus

CS 519 HW 2

4-24-21

1 Feature Map

1. In HW1, we binarized all categorical fields and kept the numerical fields, which makes sense for distance metrics. But for perceptron (and most other classifiers), it's actually better to binarize the numerical fields as well (so that "age" becomes many binary features such as "age=40"). Why?

We binarized select features only in kNN because otherwise it would be too computationally expensive. Time complexity increases with additional features in kNN, much more so than in the perceptron method. It's better to binarize all features in the perceptron method because it increases the accuracy of the model without increasing time complexity as much as in KNN.

2. How many binary features do you get in total after binarizing all fields? (Hint: around 230; Recall that HW1 had about 90 features). Again:

for i in `seq 1 9`; do cat income.train.txt.5k | cut -f $i -d ',' | sort | uniq | wc -l; done

Do not forget the bias dimension.

I get exactly 231 fields with all data binarized + the one bias dimension.

2 Perceptron and Averaged Perceptron

1. Implement the basic perceptron algorithm. Run it on the training data for 5 epochs (each epoch is a cycle over the whole training data, also known as an "iteration"). After each epoch, print the number of updates (i.e., mistakes) in this epoch (and update %, i.e., # of updates /|D|), and evaluate on the dev set and report the error rate and predicted positive rate on dev set, e.g., something like:

epoch 1 updates 1257 (25.1%) dev_err 23.8% (+:27.5%)

...

epoch 5 updates 1172 (23.4%) dev_err 20.5% (+:17.7%)

epoch 1 updates 1257 (26.48%) dev_err 22.4% (+:27.5%)
epoch 2 updates 1221 (26.64%) dev_err 20.8% (+:25.4%)
epoch 3 updates 1177 (26.22%) dev_err 18.0% (+:21.5%)
epoch 4 updates 1170 (26.2%) dev_err 19.7% (+:12.3%)
epoch 5 updates 1172 (26.22%) dev_err 18.7% (+:17.7%)

Q: what's your best error rate on dev, and at which epoch did you get it? (Hint: should be around 19%). Actually, my dev err rates here are the upper bound. This is due to tie-breaking scenarios (when dotproduct is exactly 0). Depending on how you deal with such cases (always predict +1 or always predict -1 or random), you might get 21.1% for epoch 1 and 18.7% for epoch 5 or even lower rates, but the the update rates and positive rates should be same. My dev err rates are achieved when I always consider myself wrong in tie-breaking cases (extreme unlucky case), but in reality, at testing time you don't have the correct answer in hand, and you have to make a prediction no matter what (you can't abstain even if you're uncertain), so what I did was incorrect.

On the other hand, during training, it's important to consider yourself always wrong in all tie-breaking cases; in other words, we want to be super strict to ourselves during training (e.g., study time), but allow ourselves to guess during testing (e.g., exam time).

Also note that tie-breaking is only possible with the non-averaged perceptron, as weights are all integers. With averaged perceptron, it's no longer a problem (weights are real numbers).

Best error rate

- **Using 5 epochs: 18.0% at epoch 3**
- **Using 1000 epochs: 16.8% at epoch 511, minimal convergence pattern.**

2. Implement the averaged perceptron. You can use either the naive version or the smart version (see slides). Note: the difference in speed between the two versions is not that big, as we only have around 230 features, but in HW4 the difference will be huge, where we'll have sparse features. Run the averaged perceptron on the training data for 5 epochs, and again report the error rate and predicted positive rate on dev set (same as above).

Q: This time, what's your best error rate on dev, and at which epoch did you get it? (Hint: around 15%).

```
epoch 1 updates 1257 (26.48%) dev_err 15.0% (+:18.6%)
epoch 2 updates 1221 (26.64%) dev_err 15.1% (+:19.3%)
epoch 3 updates 1177 (26.22%) dev_err 14.8% (+:20.0%)
epoch 4 updates 1170 (26.2%) dev_err 14.7% (+:19.3%)
epoch 5 updates 1172 (26.22%) dev_err 14.8% (+:20.0%)
```

3. Q: What observations can you draw by comparing the per-epoch results of standard and averaged perceptrons? What about their best results?

Per-epoch results seem to vary much less using the average perceptron method, only 1 or 2 percentage points in comparison to the 6 percentage point variance in the vanilla method (which concurs with lecture). The average method is also much more accurate, typically seeing scores in the 14.7% compared to 18.7% errors for their respective best results.

When running out to 1000 epochs, the average perceptron method converges at 15.7 – 15.8% around 200 epochs, whereas the vanilla method never seems to converge.

4. Q: For the averaged perceptron, what are the five most positive/negative features? Do they make sense?

These are the most negative and positive features. Most of the most negative features are ages, either retirement, or minor ages. This is because those people typically do not make an income, which makes sense.

The most positive features are doctorate, Iran, Guatemala, etc. This makes sense because high education correlates with higher salary, and all of the people from Iran are positive in this data set.

```
Positive Index: 65 Positive Weight: -983544.0 Positive Feature: (0, '88')
Neg Index: 85 Neg Weight: 919197.0 Neg Feature: (2, 'Doctorate')

Positive Index: 54 Positive Weight: -964693.0 Positive Feature: (0, '77')
Neg Index: 201 Neg Weight: 902858.0 Neg Feature: (8, 'Iran')

Positive Index: 118 Positive Weight: -895000.0 Positive Feature: (7, '13')
Neg Index: 215 Neg Weight: 664571.0 Neg Feature: (8, 'Guatemala')

Positive Index: 52 Positive Weight: -764391.0 Positive Feature: (0, '75')
Neg Index: 83 Neg Weight: 639343.0 Neg Feature: (2, 'Prof-school')

Positive Index: 44 Positive Weight: -740119.0 Positive Feature: (0, '66')
Neg Index: 90 Neg Weight: 632116.0 Neg Feature: (3, 'Married-civ-spouse')
```

5. Q: We know that males are more likely to earn >50K on this dataset. But the weights for Sex=Male and Sex=Female are both negative. Why?

The weights for female is less negative than the weights for male. The more negative the value, the more it will pull the prediction towards a negative salary prediction. They can both be negative, but because female is *more* negative, this still makes sense.

6. Q: What is the feature weight of the bias dimension? Does it make sense?

The feature weight of the bias dimension is -687816.0. This is because it is more likely for the data to be negative than positive, so the bias dimension pulls it this way. This makes sense.

7. Q: Is the update % above equivalent to “training error”?

The update percent is equivalent to training error. If this model is overfit, the update percent will go towards zero.

3 Comparing Perceptron with k-NN

1. What are the major advantages of perceptron over k-NN? (e.g., efficiency, accuracy, etc.)

The main benefits of perceptron over KNN are speed and additional accuracy.

Using the KNN method the computer must store the entire comparison vector in memory during the prediction process. This is very computationally expensive. Alternatively, average weight perceptron method stores a very small amount of data in memory, allowing for much faster runtime.

Perceptron also has the potential to be more accurate on blind data as the algorithm can be more effectively trained.

2. Design and execute a small experiment to demonstrate your point(s).

Accuracy – My dev error for my best KNN model was 15.7% whereas using perceptron I am closer to 14-15% using typical average perceptron.

Time – The duration for the best model using KNN is 15.8 seconds (with error rate of 15.7%). The perceptron method achieves a 14.8% error rate using a simple average method with a time of 0.156 seconds, a full 100x faster.

4 Experimentations

1. Try this experiment: reorder the training data so that all positive examples come first, followed by all negative ones. Did your (vanilla and averaged) perceptron(s) degrade in terms of dev error rate(s)?

```
Vanilla
epoch 1 updates 4 (25.02%) dev_err 76.4% (+:100.0%)
epoch 2 updates 8 (25.02%) dev_err 76.4% (+:100.0%)
epoch 3 updates 9 (25.04%) dev_err 76.3% (+:98.9%)
epoch 4 updates 9 (25.06%) dev_err 76.4% (+:100.0%)
epoch 5 updates 10 (25.08%) dev_err 76.3% (+:99.7%)

Best Dev Error: 76.3%

Average
epoch 1 updates 4 (25.02%) dev_err 27.8% (+:4.4%)
epoch 2 updates 8 (25.02%) dev_err 23.7% (+:0.3%)
epoch 3 updates 9 (25.04%) dev_err 23.6% (+:0.0%)
epoch 4 updates 9 (25.06%) dev_err 23.6% (+:0.0%)
epoch 5 updates 10 (25.08%) dev_err 23.6% (+:0.0%)

Best Dev Error: 23.6%
```

Both average and vanilla degrade in dev error rate. Vanilla predicts a massive positive rate, while average predicts a very small number of positives. This shows how important it is to shuffle data before analysis, to prevent like-items being together

2. Try the following feature engineering:

(a) adding the original numerical features (age, hours) as two extra, real-valued, features.

Vanilla:

```
epoch 1 updates 1858 (27.42%) dev_err 23.8% (+:0.2%)
epoch 2 updates 1676 (26.54%) dev_err 23.7% (+:0.1%)
epoch 3 updates 1601 (26.24%) dev_err 18.6% (+:23.8%)
epoch 4 updates 1516 (26.02%) dev_err 19.6% (+:26.8%)
epoch 5 updates 1510 (25.9%) dev_err 23.4% (+:0.2%)
```

Average:

```
epoch 1 updates 1858 (27.42%) dev_err 23.6% (+:0.0%)
epoch 2 updates 1676 (26.54%) dev_err 23.6% (+:0.0%)
epoch 3 updates 1601 (26.24%) dev_err 23.6% (+:0.0%)
epoch 4 updates 1516 (26.02%) dev_err 23.5% (+:0.1%)
epoch 5 updates 1510 (25.9%) dev_err 22.7% (+:1.5%)
```

The error rates are worse when the ages and hours worked are included, and the positive results numbers vary wildly. I recommend using the normal binarization over this method.

(b) centering of each numerical dimension (or all dimensions) to be zero mean;

```
Vanilla
epoch 1 updates 1177 (25.08%) dev_err 22.9% (+:30.1%)
epoch 2 updates 1121 (25.04%) dev_err 20.9% (+:25.1%)
epoch 3 updates 1150 (25.02%) dev_err 22.0% (+:26.0%)
epoch 4 updates 1107 (25.0%) dev_err 25.3% (+:33.7%)
epoch 5 updates 1129 (25.04%) dev_err 23.6% (+:25.4%)

Average
epoch 1 updates 1177 (25.08%) dev_err 14.1% (+:20.1%)
epoch 2 updates 1121 (25.04%) dev_err 14.5% (+:20.5%)
epoch 3 updates 1150 (25.02%) dev_err 14.5% (+:21.1%)
epoch 4 updates 1107 (25.0%) dev_err 14.8% (+:21.0%)
epoch 5 updates 1129 (25.04%) dev_err 15.4% (+:21.4%)
```

These provide the best model at epoch 1 of the average model (14.1%). The vanilla model is pretty bad though.

(c) based on the above, and make each numerical dimension (or all dimensions) unit variance.

```
Vanilla
epoch 1 updates 1162 (25.24%) dev_err 22.8% (+:26.6%)
epoch 2 updates 1091 (25.0%) dev_err 23.7% (+:28.9%)
epoch 3 updates 1134 (25.06%) dev_err 23.3% (+:29.1%)
epoch 4 updates 1128 (24.98%) dev_err 25.3% (+:35.9%)
epoch 5 updates 1119 (25.08%) dev_err 20.2% (+:19.2%)

Average
epoch 1 updates 1162 (25.24%) dev_err 16.5% (+:21.3%)
epoch 2 updates 1091 (25.0%) dev_err 16.2% (+:20.6%)
epoch 3 updates 1134 (25.06%) dev_err 15.8% (+:21.2%)
epoch 4 updates 1128 (24.98%) dev_err 16.1% (+:21.1%)
epoch 5 updates 1119 (25.08%) dev_err 15.7% (+:21.5%)
```

This transformation did not improve the data by much.

(d) adding some binary combination features (e..g, edu=X-and-sector=Y)

```
Vanilla
epoch 1 updates 1546 (26.14%) dev_err 23.1% (+:0.7%)
epoch 2 updates 1295 (25.32%) dev_err 17.6% (+:28.0%)
epoch 3 updates 1210 (25.18%) dev_err 16.0% (+:22.8%)
epoch 4 updates 1186 (25.22%) dev_err 19.0% (+:6.6%)
epoch 5 updates 1158 (25.14%) dev_err 19.6% (+:34.4%)

Average
epoch 1 updates 1546 (26.14%) dev_err 18.1% (+:9.3%)
epoch 2 updates 1295 (25.32%) dev_err 16.6% (+:13.6%)
epoch 3 updates 1210 (25.18%) dev_err 16.1% (+:15.5%)
epoch 4 updates 1186 (25.22%) dev_err 16.2% (+:17.0%)
epoch 5 updates 1158 (25.14%) dev_err 15.8% (+:18.2%)
```

Adding the age and hours metrics to the unit variance model didn't really help the Averaged model, however it was helpful in the vanilla case. In vanilla it helped convergence around 19%, but it did not have much impact on average.

Q: which ones (or combinations) helped? did you get a new best error rate?

My new best error rate is 14.1% epoch 1 part b. Most of the transformations did not help outside of this one.

3. Collect your best model and predict on income.test.blind to income.test.predicted. The latter should be similar to the former except that the target (>=50K, <50K) field is added. Do not change the order of examples in these files.

Complete

Q: what's your best error rate on dev? which setting achieved it? (should be able to get 14% or less)

My best error rate is 14.1% epoch 1 part b using the centering transformation. Unfortunately I couldn't get that 14.1% error weighting vector to successfully calculate labels, so I used the conventional perceptron average calculated earlier in the assignment (14.7% error).

Q: what's your predicted positive % on dev? how does it compare with the ground truth positive %?

My predicted positive rate on dev is 19.3% for my best model, which is 23.6%, 4.3% off. Not terrible.

Q: what's your predicted positive % on test?

My predicted positive on test is 20.4%

which I am certain is 100% correct, 60% of the time.

Debriefing (required)

1. Approximately how many hours did you spend on this assignment?

I donno like a billion.

Not as bad as HW1 and not as long as many in this class, I spent roughly 21 hours on this assignment with lecture, and the assignment itself.

2. Would you rate it as easy, moderate, or difficult?

This assignment was much easier than HW1. I had a very difficult time figuring out how to succeed with HW1, and now that I have attended every office hour it is much easier, and I'm learning more.

I would still rate it as moderate difficulty. Part 4 was difficult to understand what the question was asking.

3. Did you work on it mostly alone, or mostly with other people?

I worked on it 10%/90% in office hours / alone. A majority of my time was working alone, but I broke through serious barriers at office hours.

4. How deeply do you feel you understand the material it covers (0%–100%)?

Much better than last Homework. Probably around 80-90%.

5. Any other comments?

The TA's are both very helpful. Are we done learning yet? This was the last homework right?