CS519 HW #2

Matthew Frantz

April 25, 2019

1 Feature Map

1.1

For the perceptron it is better to binarize all the fields because we want weights to update only for the specific input being activated. If we kept it as a numerical field, the weight would never really converge since every value would be different for each update causing constant cycling and the value of the wight would be dependent on the last few training examples.

1.2

From Table 1, we have a total of 230+1 features if we include the bias term. The number of features allocated per field is shown in the second column of Table 1.

Table 1: Field and number of Feature for each field.

Field	Features
Age	1 (67 if Binary)
Sector	7
Education	16
Marital-Status	7
Occupation	14
Race	5
Gender	2
Hours-per-Week	1 (73 if Binary)
Country-of-Origin	39
Bias	1
Total	92 (231 if all Binary including bias)

2 Perceptron and Average Perceptro

2.1

Q: whats your best error rate on dev, and at which epoch did you get it?

Based on Table 2 the best error rate on dev occurred at epoch 3 with 19.0% error.

2.2

Based on Table 3 the best error rate on dev occurred at epoch 2 with 14.7% error.

Table 2: Epoch updates and Dev set error rate and predicted positive with basic Perceptron.

Epoch	Updates	Update %	Dev Error Rate	Positive %
1	1257	25.1	23.9	27.6
2	1221	24.4	22.4	25.6
3	1177	23.5	19.0	21.6
4	1170	23.4	20.8	12.3
5	1172	23.4	20.4	17.8

Table 3: Epoch updates and Dev set error rate and predicted positive with average Perceptron.

Epoch	Updates	Update %	Dev Error Rate	Positive %
1	1257	25.1	14.9	17.1
2	1221	24.4	14.7	16.7
3	1177	23.5	15.4	14.8
4	1170	23.4	15.8	15.8
5	1172	23.4	15.7	16.3

2.3

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

Based on the results of Tables 2 and 3, we can obviously see that the average perceptron outperforms the basic perceptron in terms of error rate on the dev set. Interestingly the number of updates at each epoch was the same but I guess the update criteria for each algorithm is the same so maybe its not that surprising. The best results appear in epochs 2/3 for each algorithm and then get worse after that. This is interesting because given a data set of this size I would have assumed it would have taken longer to converge.

2.4

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

The top 5 most positive feature are:

1. Doctorate
2. Married-Civ-spouse
3. Prof-School
4. Iran
5. 65 years old

The above list is mostly not surprising as we expect older, married, highly educated people to generally make more money. The outlier here is Iran and it was discussed in the lectures that most of the people who from Iran also happened to be highly educated individuals making them associated with being positive values.

The bottom 5 negative features are:

1. 28
2. 7th-8th
3. Farming-Fishing
4. 26

5. 24

This again is not surprising as the list consists of young less educated people or lower skill jobs like farming that would be associated with making less money.

2.5

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?

Since both males and females have more people that make less that $\leq 50K$ in this data set I would expect them to both be negative as the weight update would see more negative than positive examples.

2.6

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

For the average perceptron I got a bias of -6.9972 and for the basic I got -7. This makes sense since the data set is heavily biased towards negative examples ($\leq 50k$) as roughly 75% of the data is negative. As such I would expect the bias to become negative as that is what it is expecting more often.

2.7

Q: Is the update % above equivalent to training error?

No, update % is just the number of wrong predictions during the training process where as training error would be how far off the weights were from the real values at the end of each epoch.

3 Comparing Perceptron with k-NN

1. What are the major advantages of perceptron over k-NN?
   (a) The first one that comes to mind is testing. Once the network is trained the amount of time to execute a test example is very short since you are basically calculating the dot product between the test vector and the weights vectors which very fast. This means you can apply your trained network in near real time to new data coming in. K-NN however is much slower since you still need to compute the new distance and sort them for every new test example which is not negligible.
   (b) I would argue the implementation of the perceptron is also easier as the math is just very simple and does not seem to require any crazy tricks like K-NN to speed it up.
2. Design and execute an experiment to demonstrate your point.
   (a) To prove the time difference I will execute the k-NN algorithm with a single k value of 15 as that was my best value for HW1 and compare that against the 5 epoch average perceptron. For the k-NN algorithm it took 12.082s on my computer to fully test the dev set. For the average perceptron it only took 0.317s to train and test on the dev set for 5 epochs. Not only was it faster but it achieved a better dev error with 16.7% for k-NN and 14.7% for the perceptron. The there is 38x speed boost using the perceptron algorithm over k-NN.

4 Experimentation

4.1

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)?

For the positive sorted training set, Table 4 shows a complete degradation of the dev error (equal to percent of total positive examples) for both the basic and average perceptrons. This is expected as the weights saw over 3000 training examples that were all negative for the last part of the training effectively drowning out the positive examples effect on the weights.

Table 4: Epoch updates and Dev set error rate and predicted positive for both basic and average Perceptron for the sorted data with positive examples first.

Epoch	Updates	Update %	Dev Error Rate	Positive %
1	4	0.08	23.6	0.0
2	12	0.24	23.6	0.0
3	14	0.28	23.6	0.0
4	14	0.28	23.6	0.0
5	17	0.34	23.6	0.0

For the negative first examples we see a pretty big difference between basic and average perceptron. Table 5 shows that the basic perceptron appears to have updated the weights to match all the positive examples that occurred at the end of the training set. As such the dev error matches the inverse actual positive rate for the dev set. For the average perceptron, Table 6 shows that even with the negative sorted data the average performs much better than the basic in this case. This is most likely because of the larger number of negative examples having an effect on the cached weights and bias caused by the averaging.

Table 5: Epoch updates and Dev set error rate and predicted positive the basic Perceptron for the sorted data with negative examples first.

Epoch	Updates	Update %	Dev Error Rate	Positive %
1	3	0.06	76.4	99.9
2	8	0.16	76.2	99.7
3	8	0.16	75.6	98.1
4	11	0.22	76.3	99.9
5	12	0.24	75.4	98.8

Table 6: Epoch updates and Dev set error rate and predicted positive the average Perceptron for the sorted data with negative examples first.

Epoch	Updates	Update %	Dev Error Rate	Positive %
1	3	0.06	45.0	58.7
2	8	0.16	32.0	37.8
3	8	0.16	26.9	30.6
4	11	0.22	37.9	53.7
5	12	0.24	33.0	47.8

4.2

Try the following feature engineering:

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

Table 7 shows the results using the original numerical categories for age and hours worked instead of the binary features. Overall they both perform worse than the all binary version.

Table 7: Epoch updates and Dev set error rate and predicted positive for both the basic and average perceptron using the original numerical features in place of binary ones.

Epoch	Updates	Update %	Basic Error Rate	Basic +%	Avg Error	Avg +%
1	1841	36.82	23.7	0.1	23.6	0.0
2	1746	34.92	23.8	0.2	23.6	0.0
3	1623	32.46	24.0	1.0	23.6	0.0
4	1557	31.14	21.1	31.5	23.6	0.0
5	1540	30.8	23.9	48.3	23.2	0.1

Table 8 shows the results by adding the original numerical categories for age and hours worked to the binary features. Overall they both perform worse than the all binary version. Interestingly the basic perceptron did much better than it did by just replacing the binary features with the numerical ones. The bias was much larger coming around -350 for both sets.

Table 8: Epoch updates and Dev set error rate and predicted positive for both the basic and average perceptron by adding the original numerical features to all the binary ones.

Epoch	Updates	Update %	Basic Error Rate	Basic +%	Avg Error	Avg +%
1	1858	37.16	23.8	0.2	23.6	0.0
2	1676	33.52	23.7	0.1	23.6	0.0
3	1601	32.02	18.6	23.8	23.6	0.0
4	1516	30.32	19.5	26.7	23.6	0.0
5	1510	30.2	23.4	30.2	23.6	0.0

4.2.2 centering of each numerical dimension to be zero mean

Table 9 shows the effect of applying a zero-mean to the numerical fields. The number of updates required was reduced for this example most likely because the bias was reduced from -350 to -117 resulting in less updates needed. The basic perceptron performed much worse with this application and we saw improvement on the average perceptron but still not better than the all binary version.

Table 9: Epoch updates and Dev set error rate and predicted positive for both the basic and average perceptron by adding the original numerical features to all the binary ones and taking the zero-mean for the numerical features.

Epoch	Updates	Update %	Basic Error Rate	Basic +%	Avg Error	Avg +%
1	1489	29.78	33.1	45.1	19.0	14.4
2	1256	25.12	33.2	44.4	16.7	14.9
3	1286	25.72	24.7	36.7	16.2	14.4
4	1203	24.06	30.0	42.0	16.2	14.2
5	1195	23.9	30.9	42.7	16.2	14.6

4.2.3 based on the above, and make each numerical dimension unit variance

Table 11 shows the results of adding the unit variance to the above formulation. We see a great improvement in performance for both the basic and average perceptron and a large decrease in the bias from -117 to -8 which also correlated with less updates needed. Although this still does not perform as well as the all binary example.

Table 10: Epoch updates and Dev set error rate and predicted positive for both the basic and average perceptron by adding the original numerical features to all the binary ones and taking the zero-mean for the numerical features and the unit variance.

Epoch	Updates	Update %	Basic Error Rate	Basic +%	Avg Error	Avg +%
1	1138	23.66	24.4	36.0	15.3	17.1
2	1144	22.88	18.8	19.0	15.8	13.8
3	1139	22.78	19.8	18.2	16.2	12.8
4	1127	22.54	19.5	25.7	15.3	15.7
5	1117	22.34	18.9	20.3	15.6	14.2

4.2.4 adding some binary combination features

After looking through several sets of combinations the best I found was Never-married and Sales with a dev error of 14.3 and 20.1 +% using the average perceptron. This makes sense since younger people tend to be in sales and are more likely to not be married to this feature should improve the accuracy.

Table 11: Epoch updates and Dev set error rate and predicted positive for both the basic and average perceptron by combining the marital status = Never-married and occupation = Sales features.

Epoch	Updates	Update %	Basic Error Rate	Basic +%	Avg Error	Avg +%
1	1268	25.36	21.2	16.8	16.1	13.1
2	1190	23.8	21.0	17.7	15.1	15.9
3	1186	23.72	23.0	25.0	15.0	17.8
4	1182	23.64	24.0	32.7	14.4	19.6
5	1184	23.68	22.6	26.3	14.3	20.1

4.3

Q: whats your best error rate on dev? which setting achieved it?

My best error occurred when I used all binary features and added the combination of Never-married and Sales with a dev error of 14.3 using the average perceptron.

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

I had a very close positive % of 20.1 to the real one of 23.6% leading me to believe that it was a useful feature combination.

Q: whats your predicted positive % on test?

On my test set the best positive % I got was 20.2% which seems to be in agreement with the training and dev set.

5 Debriefing

1. Approximately how many hours did you spend on this assignment?
   (a) Roughly 20 hours between reading, coding, and writing.
2. Would you rate it as easy, moderate, or difficult?
   (a) Moderate. Programming the perceptrons was actually really easy but doing some of the feature manipulation got pretty messy.
3. Did you work on it mostly alone, or mostly with other people?

(a) Alone, but I looked at CANVAS to get some questions answered which was useful.

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

   (a) 85%. Most of this was pretty straight forward but I'm not sure I fully understand the importance of the margin. It seems to me it wouldn't matter to a computer to much whether the data was close or a little further apart.

5. Any other comments?

   (a) A little better explanation of what adding some combination of binary features means. I had to dig into canvas discussion boards to figure that one out.