Ben Tankus

HW 4

Question: why is each of these steps necessary or helpful for machine learning?

1. All words are lowercased;

This is helpful for machine learning because the potential for multiple factors for the same word, potentially spreading your data thin. For example, if your word was ‘fight’ and ‘Fight’, python will treat these as two separate features, splitting your data to two different items with the same meaning.

2. Punctuations are split from adjoining words and become separate “words” (e.g., the commas above);

This allows for easier parsing and splitting of the data, and ensures ‘children,’ and ‘children’ are not considered two features.

3. Verb contractions and the genitive of nouns are split into their component morphemes, and each morpheme is tagged separately, e.g.: children's -> children 's parents' -> parents ' won't -> wo n't that's -> that 's gonna -> gon na i'm -> i 'm you'll -> you 'll we'd -> we 'd you're -> you 're

This allows for clean feature creation as in the question above. Here, ‘children’s’ would become ‘children’, allowing for the one meaning to be held for multiple calls. This also might cause issues with reading code as potentially strings, typically notated with ‘ quotes.

4. Single quotes are changed to forward- and backward- quotes (e.g., 'solaris' --> ` solaris ') and double quotes are changed to double single forward- and backward- quotes, e.g.: "catch me if you can" --> `` catch me if you can '' 1

Again, this prevents double featuring, as in my previous answers.

5. There were a small amount of reviews in Spanish; I’ve removed them

The Spanish words would likely be unique features that were only very minimally used. Removing them will speed up the analysis and reduce wasted computing expense. Cognates with different meanings could also potentially cause issues.

1 Naive Perceptron Baseline

1. Take a look at svector.py, and briefly explain why it can support addition, subtraction, scalar product, dot product, and negation.

Svector.py supports these by including the __add__, __sub__, __mu1__, __dot__, and __neg__ methods in the svector class.

2. Take a look at train.py, and briefly explain train() and test() functions.

The train function takes the input of a file path (both train and dev), and epochs and creates a feature map for it. It then uses the test function to calculate dev error, and prints the response. It's pretty much a way better perceptron feature mapping implementation of HW2.

The test function takes the devfile and weight vector (model) and calculates the test error from it.

3. There is one thing missing in my train.py: the bias dimension! Try add it. How did you do it? (Hint: by adding bias or ?) Did it improve error on dev?

See the code, I added a bias dimension of 1, and it slightly improved the error rate.

4. Wait a second, I thought the data set is already balanced (50% positive, 50% negative). I remember the bias being important in highly unbalanced data sets. Why do I still need to add the bias dimension here??

The bias dimension is still important to handle fringe cases. With a positive bias and minimal input from the weights, the code will default to positive.

2 Average Perceptron and Vocabulary Pruning

1. Train for 10 epochs and report the results. Did averaging improve on dev? (Hint: should be around 26%). Did it make dev error rates more smooth?

```
epoch 1, update 38.8%, dev 30.7%
epoch 2, update 25.1%, dev 29.0%
epoch 3, update 20.7%, dev 28.4%
epoch 4, update 16.7%, dev 27.8%
epoch 5, update 13.8%, dev 28.0%
epoch 6, update 12.1%, dev 28.1%
epoch 7, update 10.3%, dev 27.8%
epoch 8, update 9.2%, dev 27.2%
epoch 9, update 8.4%, dev 27.2%
epoch 10, update 7.0%, dev 27.8%
best dev err 27.2%, |w|=15805, time: 3.9 secs
```

The averaging improved the results significantly, with an improvement in best model of 3 percentage points (30.1% to 27.2%). Another change was in the error rate conversion. The naïve method still jumped around quite a lot whereas the average method remained consistent.

2. Did smart averaging slow down training?

Smart averaging did not slow down by a significant margin, both naïve and average methods took around 4 seconds.

3. What are the top 20 most positive and top 20 most negative features? Do they make sense?

Top 20 Negative:

Placing	Negatives	Positives
1	‘intentions': -522182.0	'happy': 467754.0
2	'backdrops': -460321.0	'journey': 409196.0
3	'base': -459251.0	'provides': 405540.0
4	'overstays': -436170.0	'unfolds': 404350.0
5	'mindless': -434776.0	'smarter': 403524.0
6	'flimsy': -431224.0	'helluva': 396146.0
7	'exhausting': -428729.0	'strength': 392337.0

8	'possesses': -422190.0	'marvel': 387324.0
9	'diary': -413108.0	'suck': 386624.0
10	'niro': -411772.0	'liberating': 381995.0
11	'bogus': -395236.0	'conduct': 378817.0
12	'foot': -388187.0	'tribute': 376540.0
13	'incoherent': -385770.0	'involving': 371253.0
14	'foreign': -380453.0	'assured': 362499.0
15	'gay': -379216.0	'effectively': 361394.0
16	'bore': -377441.0	'among': 360389.0
17	'coherent': -376942.0	'emergency': 360110.0
18	'meanders': -372662.0	'neo': 358888.0
19	'shooting': -371771.0	'bizarre': 358416.0
20	'intelligence': -371155.0	'dense': 357896.0

In large part the top and bottom 20 factors make a lot of sense. Most of the positives have good words like happy, smarter, helluva, strength, etc whereas negative has words that typically go with negative things like intentions, overstays, flimsy, mindless, etc.

4. Show 5 negative examples in dev where your model most strongly believes to be positive. Show 5 positive examples in dev where your model most strongly believes to be negative. What observations do you get?

This was mostly done manually by looking through the code predictions and text file, so

(Predicted as negative in error)

+ we learn a lot about dying coral and see a lot of life on the reef

+ daughter from danang sticks with its subjects a little longer and tells a deeper story

+ could use a little more humanity , but it never lacks in eye popping visuals

+ this is so de palma if you love him , you 'll like it if you do n't well , skip to another review

+ there are moments of hilarity to be had

(Predicted as positive in error)

- the film 's maudlin focus on the young woman 's infirmity and her naive dreams play like the worst kind of hollywood heart string plucking

- kapur weighs down the tale with bogus profundities

- i 'm guessing the director is a magician after all , he took three minutes of dialogue , 30 seconds of plot and turned them into a 90 minute movie that feels five hours long
- sex with strangers will shock many with its unblinking frankness but what is missing from it all is a moral what is the filmmakers ' point ? why did they deem it necessary to document all this emotional misery ?
- the script is high on squaddie banter , low on shocks

Many of these incorrect items either use colorful language that likely includes words with sparse data, or has positive and negative words. For example, “We Learn … dying … on the reef” includes the term *dying* which has a strong negative value that offsets the positive values. On the alternative, “Kapur weighs down the tale with bogus profundities” uses unique, or neutral words which are difficult for the algorithm to classify correctly.

5. Try improve the speed by caching sentence vectors from training and dev sets, and show timing results.

I tried, but I haven’t really been able to figure out how this one works. Close enough.

3 Pruning the Vocabulary

1. Try neglecting one-count words in the training set during training. Did it improve on dev? (Hint: should help a little bit, error rate lower than 26%).

```
epoch 1, update 39.0%, dev 31.6%

epoch 2, update 26.4%, dev 27.5%

epoch 3, update 22.8%, dev 26.8%

epoch 4, update 18.8%, dev 26.6%

epoch 5, update 17.2%, dev 25.9%

epoch 6, update 14.8%, dev 26.5%

epoch 7, update 13.2%, dev 27.0%

epoch 8, update 12.7%, dev 26.7%

epoch 9, update 11.4%, dev 26.6%

epoch 10, update 10.6%, dev 26.2%

best dev err 25.9%, |w|=8425, time: 1.3 secs
```

The error rate is slightly improved with using a paired training (25.9% dev error). The main benefit is the error rate did not drop significantly and the time to completion is cut almost in half, likely due to the word count drop from 13,428 to 8,153

2. Did your model size shrink? (Hint: should almost halve).

The size did shrink from 13,428 to 8,153 words.

3. Did update % change? Does the change make sense?

The update % changed, but only slightly. This makes sense because these 1-ct factors should not have been significant to the model, as they had minimal data to train on.

4. Did the training speed change?

The training speed improved by around 90%

5. What about further pruning two-count words (words that appear twice in the training set), etc?

```
epoch 1, update 38.9%, dev 31.1%
epoch 2, update 28.2%, dev 29.2%
epoch 3, update 23.7%, dev 28.7%
epoch 4, update 21.7%, dev 28.0%
epoch 5, update 18.5%, dev 27.9%
epoch 6, update 17.7%, dev 26.6%
epoch 7, update 16.2%, dev 26.8%
epoch 8, update 15.2%, dev 26.6%
epoch 9, update 14.1%, dev 26.6%
epoch 10, update 12.6%, dev 26.6%
best dev err 26.6%, |w|=5934, time: 1.5 secs
```

Dropping 2 count words looks like it is a step too far. We lose our sub-26% dev error rate, and are likely under-fitting the data. I recommend dropping 1-count words, but including 2-count words in the model.

4 Try some other learning algorithms with sklearn

Q: What did you learn in terms of the comparison between averaged perceptron and these other (presumably more popular and well-known) learning algorithms?

Using logistic regression from the sklearn package I achieved my best dev error of 25.6%. This improved error rate is likely due to the advanced methods used in the sklearn package that have improved on my fairly naïve average perceptron model.

best dev err 25.6%, |w|=15805, time: 18.1 secs

5 Deployment

You can also try other tricks such as bigrams (two consecutive words) and removing STOP words (such as “the”, “an”, “in”, “and”, just Google it). Collect your best model and predict on test.txt to test.txt.predicted. Q: what’s your best error rate on dev, and which algorithm and setting achieved it?

My “stop-drop” method really didn’t improve the model too much, and was *very* computationally expensive. The best dev error rate was 26.9%, and it took 25.28 hours to run!!! The full length was 12,304, so it only saved about 1000 words. The lack of improvement may be attributed to the bank of words considered “stop-words”, or maybe some of those words being relevant to the model.

epoch 1, update 39.3%, dev 31.0%

epoch 2, update 23.0%, dev 27.7%

epoch 3, update 16.8%, dev 27.2%

epoch 4, update 12.9%, dev 26.9%

epoch 5, update 10.2%, dev 27.1%

```
best dev err 26.9%, |w|=12304, time: 91017.2 secs
```

```
In [30]:
```

drop stop length 12,304

6 Debriefing (required):

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

This assignment was easier compared to the previous three. I think I spent around 15 hours on this one.

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

Not too bad, it didn't help that I'm super burned out. Probably moderate.

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

I worked mostly with myself and help from the TA's. Probably 90% - 10% split.

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

It's kind of hard to gauge. I can tell this was the "put it all together" homework, but it was kind of just a different way of asking the same thing, improving model accuracy. I think about 90% understanding.

5. Any other comments?

There are lots of ways to put on a "Machine Learning" class. It's difficult to design a course for this because of how wide the interpretation is, either a very low-level, almost binary mathematical code, all the way to very high-level import-package code. This class obviously leaned towards the former, with lots of math and low-level code.

I consider myself pretty good at code. I am a Data Analyst at Linkedin on a team that codes frequently, and I am one of the better coders on that team. To be honest I have been very lost for most of this class. I am aware that there are many who code better than me, but this course was provided to Data Analytics students.

It was clear that there was very minimal prep for designing this class to students from the Data Analytics degree, and while the TA's were great support, I expect more from OSU. I was really looking forward to this class, and it really let me down.