Applied Machine Learning HW4

Due Sunday 5/26 @ 11:59pm on Canvas

1. You should submit a single `.zip` file containing `hw4.pdf`, `test.txt.predicted`, and all your code. Again, LaTeX'ing is recommended but not required. Make sure your `test.txt.predicted` has the same format as `train.txt` and `dev.txt`.

0 Sentiment Classification Task and Dataset

To simplify your job I have done the following preprocessing steps:

1. All words are lowercased;
2. Punctuations are split from adjoining words and become separate "words" (e.g., the commas above);
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
```

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 ''`

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

Question: why is each of these steps necessary or helpful for machine learning?
I will break down each of the above for why they are necessary or helpful for machine learning:

1. If you did not make all words lower case (or all uppercase would work fine as well), then you would get repeat words that are the same to a person but not the computer. For instance, if I have the sentence, "The dog went to the park." the words, "The" and "the" would be counted individually because they appear different character wise but we know they are the same.
2. Similar reason to above since things like "person," and "person" would be counted separately but should be the same.
3. Since a morpheme is a unit of a language that cannot be further divided there is less chance that the classifier will confuse it with something else.
4. Changing quotes in this way is easier to identify when a quote starts and ends.
5. Since everything else is in English, Spanish does little to help unless it is translated first (which could also not be helpful depending on the translation.)

1 Naive Perceptron Baseline

Questions:

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

   - First it uses some of the built in properties of Python classes to all for symbolic representation of operations such as + and -. Using those representations along with two other methods, dot and copy, it then performs simple operations by iterating through the dictionary keys and values and returns the new values. One thing I found was the __isub__ method is different from the __iadd__ method and if you a -= a you make the class object NoneType but a += a just doubles all the values. I'm not sure if this the behavior that was intended.

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

   - The train function implements a basic perceptron with no bias. It starts off with a blank model of the system labeled "model" as an svector object. From there it reads the `train.txt` file line by line converting the "+,-" symbol into a +1 or -1 value to be used as the output value of the line. Each line is converted into an svector object with the key being the individual word and the value the number of times that word appears. From there is takes the dot product of the current model (only the keys that they share) and multiply it by the label value. If that is ≤ 0 then it was misclassified and the model gets updated. The update takes the current model and adds to is the label * svector for the line. If the label is negative it will subtract value from the keys in model meaning those words are more associated with negative sentiment. This repeats for a set number of epochs.

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 `<bias>`?) Did it improve error on dev?

   - I decided to keep it simple and added it as an external dimension that was updated independently when a misclassification had occurred. This is similar to the CIML method. The error rate did improve slightly as you can see below. The best dev error was 28.9%, which is better than 30.1% originally.

```
$ python train.py train.txt dev.txt

epoch 1, update 39.0%, dev 39.6%
epoch 2, update 25.5%, dev 34.1%
epoch 3, update 20.8%, dev 35.3%
epoch 4, update 17.2%, dev 35.5%
epoch 5, update 14.1%, dev 28.9%
epoch 6, update 12.2%, dev 32.0%
epoch 7, update 10.5%, dev 32.0%
epoch 8, update 9.7%, dev 31.5%
epoch 9, update 7.8%, dev 30.2%
epoch 10, update 6.9%, dev 29.8%
best dev err 28.9%, |w|=16743, time: 1.4 secs
```

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??

   - You still need to add the bias dimension here because there is still the question of separability. Just because the data is evenly split does not mean it is separable about the origin.

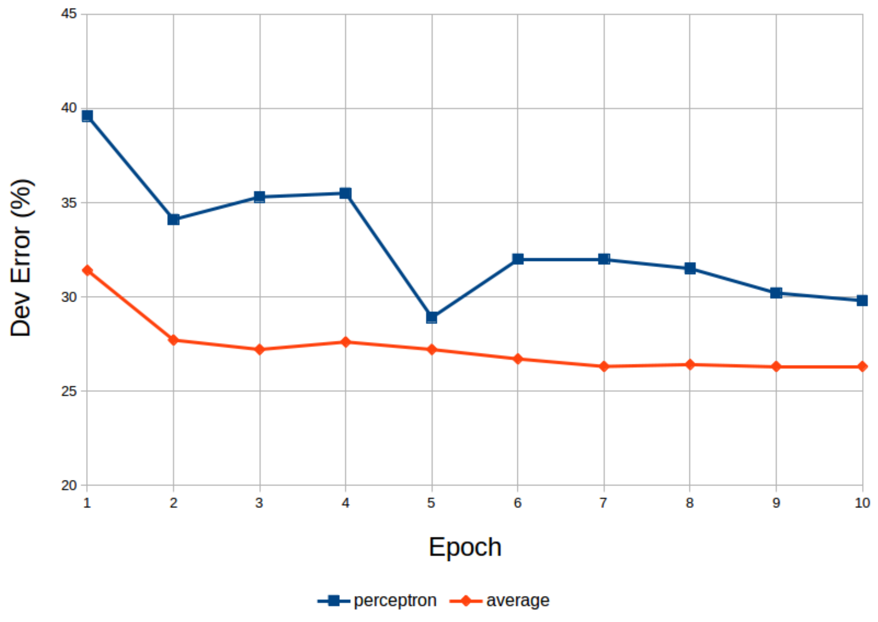

Figure 1: Smoothness comparison between original and average perceptron.

2 Average Perceptron and Vocabulary Pruning

Now implement averaged perceptron based on the version with the bias discussed above. Note that unlike in HW1, this time you have to use the smart implementation of averaging, otherwise it will be way too slow to run, due to the much bigger dimensionality (sparse vectors).

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?

   - Below are the results for the Average Perceptron. The dev error improved to 26.3% down from 28.9% with the bias above. Figure 1 demonstrates a comparison of the dev over 10 epochs from the original and average perceptron. We can see that in fact the average perceptron does look smoother as epochs increase where as the perceptron is prone to larger drops and rises between epochs.

```
epoch 1, update 39.0%, dev 31.4%
epoch 2, update 25.5%, dev 27.7%
epoch 3, update 20.8%, dev 27.2%
epoch 4, update 17.2%, dev 27.6%
epoch 5, update 14.1%, dev 27.2%
epoch 6, update 12.2%, dev 26.7%
epoch 7, update 10.5%, dev 26.3%
epoch 8, update 9.7%, dev 26.4%
epoch 9, update 7.8%, dev 26.3%
epoch 10, update 6.9%, dev 26.3%
best dev err 26.3%, |w|=15805, time: 1.6 secs
```

2. Did smart averaging slow down training?

   - Yes but only be 0.2 seconds which is pretty negligible.

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

   Top 20: ['flaws', 'smarter', 'imax', 'delightful', 'powerful', 'open', 'refreshingly', 'wonderful', 'dots', 'cinema', 'culture', 'pulls', 'treat', 'skin', 'french', 'provides', 'rare', 'unexpected', 'triumph', 'engrossing']

   Bottom 20: ['boring', 'generic', 'dull', 'badly', 'routine', 'fails', 'ill', 'too', 'instead', 'tv', 'attempts', 'unless', 'incoherent', 'neither', 'flat', 'seagal', 'problem', 'scattered', 'worst', 'suffers']

   These make sense as I would expect top performing movies to be things like 'powerful' and 'engrossing' where as movies by Steven 'seagal' tend to be 'generic', 'incoherent', and 'dull'.

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?

   5 negative examples that my model believes are positive:

   (a) the entire movie is in need of a scented bath
   (b) a handsome but unfulfilling suspense drama more suited to a quiet evening on pbs than a night out at an amc
   (c) it never quite makes it to the boiling point , but manages to sustain a good simmer for most of its running time
   (d) dodgy mixture of cutesy romance , dark satire and murder mystery
   (e) the most surprising thing about this film is that they are actually releasing it into theaters

   From these reviews we can see why the model might get confused as to why these are positive. The phrase, "in need of a scented bath" indicates that the movie stinks, but the words used are not negative. Use of words like 'handsome', 'good', 'cutesy' also have positive connotations but are used sarcastically which is difficult to detect.

   5 positive examples that my model believes are negative:

   (a) adam sandler ! in an art film !
   (b) thirteen conversations about one thing lays out a narrative puzzle that interweaves individual stories , and , like a mobius strip , elliptically loops back to where it began
   (c) it has the feel of a summer popcorn movie nothing too deep or substantial explosions , jokes , and sexual innuendoes abound
   (d) overall , interesting as a documentary but not very imaxy
   (e) the ending feels at odds with the rest of the film

   These are a little more difficult to unpack. Adam Sandler has made a number of terrible films so he generally is associated with negativity. The other ones use words that are either ambiguous or have a meaning that is more neutral than negative but use slightly negative words like not or nothing.

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

   By caching sentence vectors from the training and dev set I was able to see a 0.1 second increase to 1.5 seconds.

   No caching:

   ```
   best dev err 26.3%, |w|=15805, time: 1.6 secs
   ```

   With caching:

   ```
   best dev err 26.3%, |w|=15805, time: 1.5 secs
   ```

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%).

   Pruning the single words did help a little bit with a dev error of 25.9%.

   ```
   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|=8424, time: 1.6 secs
   ```

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

   Yes the weights went from 15805 to 8424, a 47% reduction in model size.

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

   The update percentage appears to have gone up slightly. Since the updates value only goes up when the classifier is incorrect, removing the single words must cause the weights to start further away from their true value.

4. Did the training speed change?

   Mine went back up to 1.6 seconds from 1.5 seconds. It was running really slow (13.7 seconds) but then I remembered I could use a python `set()` on my single words list to greatly speeds things up by making it a hash-able object.

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

   For two-count words I had the below results. You can see that I actually got worse results at 26.6% dev error, but a reduction in the weight size. I tried out several other values higher than 2 but all returned worse dev error rates that just removing the one-word.

   ```
   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|=5933, time: 1.5 secs
   ```

Table 1: Dev errors using sklearn classifiers with one and two-count words removed.

Classifier	One-Count Dev Error	Two-Count Dev Error
SVM	26.6	26.7
Neural Network	50.0	26.8
Gradient Boosting	37.5	37.6

4 Try some other learning algorithms with sklearn

For this portion I tried 3 different algorithms on sklearn: SVM, neural network, and Gradient Boosting Classifier. I tested all three on one and two count pruned sets. Table 1 shows the dev error results using these methods.

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

None of the well known algorithms seemed to fair that well compared to the average perceptron since the best dev error for the average I received by pruning one-count words was 25.9% which was better than all of the sklearn algorithms. I believe the training data set is simply too small for these sklearn classifiers to adequately learn thus resulting in worse performance. Another possibility is that my conversion of the data from svectors to pandas data frames was incorrect.

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?

The best error I was able to get 24.7% using the average perceptron, removing single frequency words, removing stop words, and by using bigrams. I tried many different combinations and frequencies with the perceptron and the other sklearn algorithms and I was not able to get any lower than this value. I verified my strings to make sure I was removing and adding things correctly and everything looked correct.

```
epoch 1, update 38.1%, dev 29.0%
epoch 2, update 22.3%, dev 27.1%
epoch 3, update 16.1%, dev 26.8%
epoch 4, update 12.2%, dev 26.3%
epoch 5, update 10.0%, dev 26.1%
epoch 6, update 7.2%, dev 25.8%
epoch 7, update 6.3%, dev 25.2%
epoch 8, update 5.7%, dev 25.0%
epoch 9, update 4.5%, dev 24.7%
epoch 10, update 3.6%, dev 24.9%
best dev err 24.7%, |w|=11504, time: 2.8 secs
```

6 Debriefing (required):

1. Approximately how many hours did you spend on this assignment? 15 hours.
2. Would you rate it as easy, moderate, or difficult? Moderate. It was nice learning how to use some of the existing tools out there for machine learning but also showed that it takes some skill to get them to work.
3. Did you work on it mostly alone, or mostly with other people? Alone.

4. How deeply do you feel you understand the material it covers (0%–100%)? 70%. The perceptron part I felt like I understood but since I mostly just plugged and played the SVM, neural network, gradient boosting classifiers I don't think I had a good grasp on how they worked. For future HW I recommend you just pick one and make us implement an easy version of it as implementing a neural network by hand is only slightly more difficult than a perceptron.

5. Any other comments? To make code run faster you should provide additional tips like using hashable objects. When I found my list of single words to remove from each string my running time went from almost 14 seconds back down to 1.6 seconds by just going from using a list() object to a set() object.