HW4

Part 1: Word Embeddings

1.2 Vector Similarity

Q: Can you find the top 10 similar words to 'wonderful' and 'awful'? Do the results make sense?

```
wv.most_similar('wonderful', topn=10)
```

```
[('marvelous', 0.8188857436180115),
 ('fantastic', 0.8047919869422913),
 ('great', 0.7647868990898132),
 ('fabulous', 0.7614760398864746),
 ('terrific', 0.7420831918716431),
 ('lovely', 0.7320095896720886),
 ('amazing', 0.7263179421424866),
 ('beautiful', 0.6854085922241211),
 ('magnificent', 0.6633867025375366),
 ('delightful', 0.6574996113777161)]
```

```
wv.most_similar('awful', topn=10)
```

```
[('horrible', 0.7597667574882507),
 ('terrible', 0.7478911280632019),
 ('dreadful', 0.7218177318572998),
 ('horrid', 0.6720177531242371),
 ('atrocious', 0.6626645922660828),
 ('ugly', 0.6236302852630615),
 ('lousy', 0.6135216951370239),
 ('unbelievable', 0.6068726181983948),
 ('appalling', 0.6061566472053528),
 ('hideous', 0.5811460614204407)]
```

The results make sense. The similar words are positive for wonderful and negative for awful

Q: Also come up with 3 other queries and show your results. Do they make sense?

```
wv.most_similar('future', topn=10)
```

```
[('upcoming', 0.4648454785346985),
 ('current', 0.462577223777771),
 ('potential', 0.4569127857685089),
 ('viability', 0.4525926411151886),
 ('someday', 0.4397056996822357),
 ('next', 0.4187477529048919 7),
 ('uncertain', 0.4095073938369751),
 ('any', 0.37744396924972534),
 ('regarding', 0.37411409616470337),
 ('fate', 0.3728441894054413)]
```

```
wv.most_similar('machine', topn=10)
```

```
[('machines', 0.7677488327026367),
 ('machinery', 0.5260838866233826),
 ('grinder', 0.4667676091194153),
 ('apparatus', 0.4539251327514648 4),
 ('computerized', 0.44946369528770447),
 ('computer', 0.4277688264846802),
 ('equipment', 0.4147760570049286),
 ('contraption', 0.41036173701286316),
 ('conveyor', 0.4065694808959961),
 ('system', 0.38828083872795105)]
```

```
wv.most_similar('technology', topn=10)
```

```
[('technologies', 0.8332265019416809),
 ('innovations', 0.6230790615081787),
 ('innovation', 0.5921422243118286),
 ('tech', 0.5745568871498108),
 ('technological', 0.5707634091377258),
 ('software', 0.5341452956199646),
 ('solutions', 0.5268415212631226),
 ('innovative', 0.524644672870636),
 ('devices', 0.4894959628582001),
 ('innovators', 0.4637777507305145)]
```

I came up with 3 words future, machine, and technology. They make sense because the results are similar/related to the words

1.3 Word Analogy

Q: Find top 10 words closest to the following two queries. Do your results make sense?

```
wv.most_similar(positive=['sister', 'man'], negative=['woman'], topn=10)
```

```
[('brother', 0.7966989874839783),
 ('uncle', 0.6753759980201721),
 ('nephew', 0.6596081852912903),
 ('son', 0.6472460031509399),
 ('father', 0.6398823857307434),
 ('brothers', 0.6266913414001465),
 ('dad', 0.5981076955795288),
 ('siblings', 0.5654128789901733),
 ('daughter', 0.5610913634300232),
 ('sons', 0.5580724477767944)]
```

```
wv.most_similar(positive=['harder', 'fast'], negative=['hard'], topn=10)
```

```
[('faster', 0.7064899206161499),
 ('rapidly', 0.5021132826805115),
 ('easier', 0.48843100666999817),
 ('slow', 0.4575234651565552),
 ('quickly', 0.4370786249637604),
 ('bigger', 0.4148872196674347),
 ('cheaper', 0.41006121039390564),
 ('louder', 0.409576416015625),
 ('slowly', 0.40936195850372314),
 ('smarter', 0.40232229232788086)]
```

Yes the results make sense. For the first query, the top word ‘brother’ is logical counterpart to ‘sister’ and the rest of the words also reflect male familiar roles. Words like slow and slowly might be because of their semantic relationship. For the second query, the top word ‘faster’ presents a more intense form of fast and the rest of the words align with the idea of speed

Q: Also come up with 3 other queries and show your results. Do they make sense?

```
wv.most_similar(positive=['water', 'equipment'], negative=['boat'], topn=10)
```

```
[('sewage', 0.4311523735523224),
 ('sewer', 0.39392757415771484),
 ('chemicals', 0.367214173078537),
 ('supply', 0.351889044046402),
 ('machinery', 0.3363999128341675),
 ('pumps', 0.3344215154647827),
 ('materials', 0.333029180765152),
 ('softener', 0.32268568873405457),
 ('brine', 0.31937310099601746),
 ('hardware', 0.3171658515930176)]
```

```
wv.most_similar(positive=['engine', 'oil'], negative=['car'], topn=10)
```

```
[('gas', 0.4534226357936859),
 ('crude', 0.4472402334213257),
 ('fuel', 0.3908027410507202),
 ('wells', 0.38713544607162476),
 ('barrels', 0.37867018580436707),
 ('barrel', 0.3694090247154236),
 ('diesel', 0.3444880545139313),
 ('fuels', 0.34446173906326294),
 ('delta', 0.3432126045227051),
 ('combustion', 0.3239583969116211)]
```

```
wv.most_similar(positive=['doctor', 'woman'], negative=['man'], topn=10)
```

```
[('nurse', 0.647728681564331),
 ('dentist', 0.5684018731117249),
 ('surgeon', 0.5513334274291992),
 ('psychiatrist', 0.5257454514503479),
 ('mother', 0.4850187301635742),
 ('clinic', 0.4773845672607422),
 ('medical', 0.47469067573547363),
 ('pregnant', 0.466005802154541),
 ('she', 0.46509498357772827),
 ('hospital', 0.45820727944374084)]
```

They mostly make sense most of the results capture the related context of the given words.

Part 2: Better Perceptron using Embeddings

2.1 Sentence Embedding and k-NN

1. **For the first sentence in the training set (+), find a different sentence in the training set that is closest to it in terms of sentence embedding. Does it make sense in terms of meaning and label?**

```
First Positive Sentence: it 's a tour de force , written and directed so quietly that it 's implosion rather than explosion you fear (Label:
1)
Closest Sentence: a semi autobiographical film that 's so sloppily written and cast that you can not believe anyone more central to the creat
ion of bugsy than the caterer had anything to do with it (Label: -1)
Cosine Distance: 0.2229
```

Yes, it makes sense that the sentences are close because they talk about similar topics like films and writing. But the difference in tone shows a limitation embeddings focus more on the meaning of the words than the emotional tone, so they don't fully capture whether the sentiment is positive or negative.

2. **For the second sentence in the training set (-), find a different sentence in the training set that is closest to it in terms of sentence embedding. Does it make sense in terms of meaning and label?**

```
Second Sentence: places a slightly believable love triangle in a difficult to swallow setting , and then disappointingly moves the story into
the realm of an improbable thriller (Label: -1)
Closest Sentence: the plan to make enough into an inspiring tale of survival wrapped in the heart pounding suspense of a stylish psychologica
l thriller ' has flopped as surely as a souffl gone wrong (Label: -1)
Cosine Distance: 0.1884
```

Yes, the meanings are pretty similar. Both sentences criticize thrillers that fell flat, pointing out issues like unrealistic plots and failed execution. The connection probably comes from shared words like "thriller" and "suspense," as well as the overall negative vibe they both have.

3. **Report the error rates of k-NN classifier on dev for k = 1, 3, …, 99 using sentence embedding. You can reuse your code from HW1/HW2 and/or use sklearn.**

k= 1	train_err 0.0% (+:50.0%)	dev_err 36.3% (+:53.3%)	k=51	train_err 24.6% (+:43.6%)	dev_err 27.8% (+:44.0%)
k= 3	train_err 15.5% (+:49.6%)	dev_err 33.4% (+:48.6%)	k=53	train_err 24.5% (+:43.6%)	dev_err 27.5% (+:43.5%)
k= 5	train_err 19.4% (+:49.1%)	dev_err 31.7% (+:48.1%)	k=55	train_err 24.7% (+:43.4%)	dev_err 28.3% (+:43.3%)
k= 7	train_err 21.0% (+:48.5%)	dev_err 32.1% (+:48.1%)	k=57	train_err 24.8% (+:43.3%)	dev_err 28.0% (+:43.4%)
k= 9	train_err 21.8% (+:47.9%)	dev_err 32.0% (+:47.4%)	k=59	train_err 24.7% (+:43.1%)	dev_err 28.4% (+:43.0%)
k=11	train_err 22.2% (+:47.9%)	dev_err 30.5% (+:47.5%)	k=61	train_err 24.8% (+:43.1%)	dev_err 28.4% (+:43.0%)
k=13	train_err 22.8% (+:47.2%)	dev_err 31.0% (+:45.8%)	k=63	train_err 24.8% (+:42.7%)	dev_err 28.3% (+:42.5%)
k=15	train_err 22.8% (+:47.1%)	dev_err 30.9% (+:45.7%)	k=65	train_err 24.9% (+:42.9%)	dev_err 28.3% (+:42.5%)
k=17	train_err 23.3% (+:46.6%)	dev_err 30.6% (+:47.2%)	k=67	train_err 25.0% (+:42.8%)	dev_err 28.5% (+:42.1%)
k=19	train_err 23.4% (+:46.3%)	dev_err 30.0% (+:46.2%)	k=69	train_err 24.8% (+:42.6%)	dev_err 28.4% (+:42.0%)
k=21	train_err 23.2% (+:46.5%)	dev_err 30.0% (+:44.8%)	k=71	train_err 24.8% (+:42.7%)	dev_err 28.3% (+:42.5%)
k=23	train_err 23.4% (+:46.3%)	dev_err 29.4% (+:44.4%)	k=73	train_err 24.8% (+:42.6%)	dev_err 28.3% (+:42.9%)
k=25	train_err 23.5% (+:46.1%)	dev_err 30.6% (+:44.2%)	k=75	train_err 24.9% (+:42.6%)	dev_err 28.5% (+:43.1%)
k=27	train_err 23.6% (+:45.5%)	dev_err 28.6% (+:44.2%)	k=77	train_err 25.1% (+:42.3%)	dev_err 28.1% (+:42.9%)
k=29	train_err 23.6% (+:45.5%)	dev_err 29.5% (+:43.9%)	k=79	train_err 25.0% (+:42.4%)	dev_err 28.2% (+:42.2%)
k=31	train_err 24.0% (+:45.1%)	dev_err 28.4% (+:43.2%)	k=81	train_err 25.1% (+:42.4%)	dev_err 28.1% (+:42.1%)
k=33	train_err 24.1% (+:44.9%)	dev_err 29.3% (+:43.3%)	k=83	train_err 25.4% (+:42.3%)	dev_err 27.9% (+:41.3%)
k=35	train_err 24.2% (+:44.8%)	dev_err 28.1% (+:44.7%)	k=85	train_err 25.4% (+:42.4%)	dev_err 28.4% (+:41.0%)
k=37	train_err 24.3% (+:44.7%)	dev_err 28.5% (+:44.3%)	k=87	train_err 25.6% (+:42.4%)	dev_err 28.2% (+:40.8%)
k=39	train_err 24.4% (+:44.3%)	dev_err 29.2% (+:43.8%)	k=89	train_err 25.7% (+:42.0%)	dev_err 28.6% (+:41.0%)
k=41	train_err 24.5% (+:44.4%)	dev_err 29.1% (+:43.7%)	k=91	train_err 25.6% (+:42.0%)	dev_err 28.4% (+:40.4%)
k=43	train_err 24.5% (+:44.1%)	dev_err 28.6% (+:43.6%)	k=93	train_err 25.5% (+:41.9%)	dev_err 28.8% (+:40.2%)
k=45	train_err 24.3% (+:43.8%)	dev_err 28.2% (+:43.6%)	k=95	train_err 25.3% (+:41.9%)	dev_err 28.1% (+:40.3%)
k=47	train_err 24.4% (+:44.1%)	dev_err 28.3% (+:44.3%)	k=97	train_err 25.4% (+:41.9%)	dev_err 28.4% (+:40.6%)
k=49	train_err 24.3% (+:44.0%)	dev_err 27.8% (+:44.4%)	k=99	train_err 25.4% (+:41.7%)	dev_err 28.7% (+:40.3%)

The best error rate is 27.45% at k = 53

4. **Report the error rates of k-NN classifier on dev for k = 1, 3, ...99 using one-hot vectors from HW2. You can reuse your code from HWs 1-2 and/or use sklearn.**

```
k= 1, dev_err=40.20%  k=51, dev_err=40.60%
k= 3, dev_err=40.60%  k=53, dev_err=40.00%
k= 5, dev_err=41.40%  k=55, dev_err=41.00%
k= 7, dev_err=41.80%  k=57, dev_err=42.60%
k= 9, dev_err=40.70%  k=59, dev_err=41.00%
k=11, dev_err=39.10%  k=61, dev_err=40.90%
k=13, dev_err=39.10%  k=63, dev_err=40.30%
k=15, dev_err=38.30%  k=65, dev_err=41.00%
k=17, dev_err=40.20%  k=67, dev_err=41.60%
k=19, dev_err=40.10%  k=69, dev_err=41.30%
k=21, dev_err=39.20%  k=71, dev_err=40.90%
k=23, dev_err=39.90%  k=73, dev_err=40.60%
k=25, dev_err=39.60%  k=75, dev_err=40.00%
k=27, dev_err=40.00%  k=77, dev_err=39.70%
k=29, dev_err=39.30%  k=79, dev_err=40.70%
k=31, dev_err=39.90%  k=81, dev_err=40.60%
k=33, dev_err=39.80%  k=83, dev_err=39.40%
k=35, dev_err=41.30%  k=85, dev_err=39.90%
k=37, dev_err=41.50%  k=87, dev_err=41.00%
k=39, dev_err=40.40%  k=89, dev_err=41.80%
k=41, dev_err=40.10%  k=91, dev_err=41.20%
k=43, dev_err=40.00%  k=93, dev_err=39.80%
k=45, dev_err=40.50%  k=95, dev_err=39.50%
k=47, dev_err=41.10%  k=97, dev_err=40.30%
k=49, dev_err=41.20%  k=99, dev_err=40.00%
```

The best error rate is 38.3% at k = 15

5. **Submit your best k-NN classifier to Kaggle and report the public error rate and ranking (take a screenshot).**

   I got 0.272 and my ranking is 50

2.2 Reimplement Perceptron

1. **For basic perceptron, show the training logs for 10 epochs. Compare your best dev error rate (should be ~31%) with the one from HW2**

```
Epoch 1, updates 32.3%, dev error 32.7%
Epoch 2, updates 30.2%, dev error 36.2%
Epoch 3, updates 29.2%, dev error 39.4%
Epoch 4, updates 28.8%, dev error 41.6%
Epoch 5, updates 29.3%, dev error 33.7%
Epoch 6, updates 28.7%, dev error 31.3%
Epoch 7, updates 29.3%, dev error 36.2%
Epoch 8, updates 28.7%, dev error 39.1%
Epoch 9, updates 28.9%, dev error 41.8%
Epoch 10, updates 28.8%, dev error 39.0%
Best dev error 31.3%
```

   The error is a bit higher compared to the one from HW2 which is 28.9%

2. **For averaged perceptron, show the training logs for 10 epochs. Compare your best dev error rate (should be ~23–24%) with the one from HW2**

```
Epoch 1, updates 32.3%, dev error 25.4%
Epoch 2, updates 30.2%, dev error 25.3%
Epoch 3, updates 29.2%, dev error 25.2%
Epoch 4, updates 28.8%, dev error 25.0%
Epoch 5, updates 29.3%, dev error 24.2%
Epoch 6, updates 28.7%, dev error 24.3%
Epoch 7, updates 29.3%, dev error 24.5%
Epoch 8, updates 28.7%, dev error 24.4%
Epoch 9, updates 28.9%, dev error 24.3%
Epoch 10, updates 28.8%, dev error 24.6%
Best dev error 24.2%
```

The best dev error for average perceptron is better in this case compared to the one from HW2 (26.3%)

3. **Do you need to use smart averaging here?**

   No I did not use smart averaging here and the code was still efficient enough

4. **For averaged perceptron after pruning one-count words, show the training logs for 10 epochs. Compare your dev error rate (should be ~23.5%) with the one from HW2**

```
Epoch 1, updates 33.5%, dev error 24.4%
Epoch 2, updates 30.7%, dev error 24.8%
Epoch 3, updates 30.3%, dev error 23.5%
Epoch 4, updates 30.4%, dev error 23.5%
Epoch 5, updates 30.0%, dev error 24.2%
Epoch 6, updates 30.7%, dev error 23.9%
Epoch 7, updates 30.3%, dev error 23.7%
Epoch 8, updates 30.3%, dev error 23.8%
Epoch 9, updates 30.3%, dev error 23.9%
Epoch 10, updates 30.3%, dev error 24.0%
Best dev error 23.5%
```

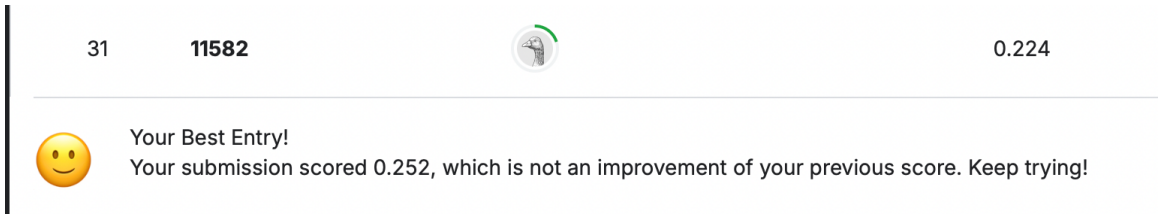
   The dev error after pruning is lower than the one from HW2 (25.9%)

5. **For the above setting, give at least two examples on dev where using features of word2vec is correct but using one-hot representation is wrong, and explain why**

```
Examples where Word2Vec is correct and One-Hot is wrong:
Sentence: you 've already seen city by the sea under a variety of titles ,
 but it 's worth yet another visit
True Label: +
Word2Vec Prediction: +
One-Hot Prediction: -
--------------------------------------------------
Sentence: it may scream low budget , but this charmer has a spirit that ca
n not be denied
True Label: +
Word2Vec Prediction: +
One-Hot Prediction: -
```

   Word2Vec understands the relationships between words, which helps it generalize to new or similar phrases, unlike one-hot encoding, which treats every word as separate and unrelated. It also focuses on the overall meaning of a sentence, so irrelevant words have less impact. In contrast, one-hot struggles to tell which words matter more. Plus, Word2Vec works better with new word combinations, while one-hot can't handle words it hasn't seen before.

6. **Submit your averaged perceptron models (both with and without pruning) to Kaggle, and record best your public error rate and ranking (take a screenshot)**



The average perceptron model without pruning seems to perform better.

2.3 Summarize the error rates in this table

	One-hot, best dev	**Embedding, best dev**	**Best Kaggle public**
k-NN	38.3	27.45%	0.272
Perceptron	25.9	23.5%	0.224

Part 3: Try Some Other Learning Algorithms With sklearn

1. **Which algorithm did you try? What adaptations did you make to your code to make it work with that algorithm?**

   I tried Logistic Regression with word embeddings. I added negation handling, replaced the perceptron with LogisticRegression from scikit-learn, and used accuracy_score for evaluation. Dense vector generation was adapted to work with Logistic Regression.

2. **What's the dev error rate(s) and running time?**

   Dev Error Rate: 24.00%

   Total running time: 0.54 seconds

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

   Logistic Regression is faster, more accurate, and easier to adapt than the averaged perceptron. It is more robust to noise, supports probabilistic outputs, and integrates well with enhancements like negation handling.

Part 4: Deployment

1. **What's your best error rate on dev, and which algorithm and setting achieved it?**

   My best dev error on dev is 23.5% from average perceptron with pruning.

2. **What's your best public error rate on Kaggle, and which algorithm and setting achieved it?**

   My best public score is 0.222 from Logistic Regression with word embeddings.

Part 5:

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

   I spent about 16 hours on this homework

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

   I would rate it as difficult.

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

   I worked on it with a friend

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

   I feel like I understand about 90% of the material it covers

5. **Any other comments?**

   No comments