Oregon State University

AI534 - MACHINE LEARNING

HW4 - Deep Learning for Sentiment Classification

AUTHOR

Melek Derman - 934382167

December 1, 2024

Contents

1 Word Embeddings

1.1 Load and Query

1.2 Vector Similarity

1.2.1: Can you find the top-10 similar words to wonderful and awful? Do your results make sense?

The top 10 most similar words to "wonderful" are either synonyms or closely related positive terms. On the other hand, while most of the top 10 similar words for "awful" are either synonyms or similarly negative in meaning, a word like "unbelievable," which can carry both positive and negative connotations, also shows a strong correlation with "awful."

Word	Similarity Score
Marvelous	0.8189
Fantastic	0.8048
Great	0.7648
Fabulous	0.7615
Terrific	0.7421
Lovely	0.7320
Amazing	0.7263
Beautiful	0.6854
Magnificent	0.6634
Delightful	0.6575

Table 1: Most similar words to *Wonderful*

Word	Similarity Score
Horrible	0.7598
Terrible	0.7479
Dreadful	0.7218
Horrid	0.6720
Atrocious	0.6627
Ugly	0.6236
Lousy	0.6135
Unbelievable	0.6069
Appalling	0.6062
Hideous	0.5811

Table 2: Most similar words to *Awful*

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

For this question, the results were analyzed for three words: *the*, *claim*, and *movie*. The words *claim* and *movie* produced completely logical results. The word *the* was chosen specifically because it is a neutral word, and the results primarily consisted of words that also have a neutral meaning.

Word	Similarity
this	0.5937
in	0.5429
that	0.5263
another	0.4748
however	0.4748
one	0.4666
entire	0.4620
its	0.4606
which	0.4595
their	0.4575

Table 3: Most similar to *the*

Word	Similarity
claims	0.7393
assert	0.5282
argue	0.5144
contend	0.5010
notion	0.4498
deny	0.4475
arguing	0.4370
argument	0.4299
prove	0.4105
purported	0.4077

Table 4: Most similar to *claim*

Word	Similarity
film	0.8677
movies	0.8013
films	0.7363
sequel	0.6578
flick	0.6322
cinema	0.6317
moviegoers	0.6071
screenplay	0.6014
cinematic	0.5959
actioner	0.5917

Table 5: Most similar to *movie*

1.3 Word Analogy

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

Overall, the results seems to be logical:

In the first example, the top result is *brother*, which is consistent with the expected most highly related result. Except for *daughter*, all other words represent male relatives, which makes them logical. the word *daughter* demonstrates that the embedding can capture other relationships, beyond just gender information.

In the second example, the highest similarity is with the word *faster*, which is an expected result. While words like *bigger*, *cheaper*, and *louder* seem slightly unrelated to the context, they may be logical as they can carry similar meanings in certain phrases.

Word	Similarity Score
brother	0.7967
uncle	0.6754
nephew	0.6596
son	0.6472
father	0.6399
brothers	0.6267
dad	0.5981
siblings	0.5654
daughter	0.5611
sons	0.5581

Table 6: *sister - woman + man*

Word	Similarity Score
faster	0.7065
rapidly	0.5021
easier	0.4884
slow	0.4575
quickly	0.4371
bigger	0.4149
cheaper	0.4101
louder	0.4096
slowly	0.4094
smarter	0.4023

Table 7: *harder - hard + fast*

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

All results are logical and align with the applied query, providing expected outputs.

In the first case (*colorful + bright - dark*), the results emphasize brightness and positivity (Table 8).

In the second case (*amazing + bad - great*), the focus shifts to highlighting negative feelings (Table 9).

Finally, in the third case (*surprising + annoying - predictable*), the results underline a combination of frustration and surprise (Table 10).

Word	Similarity
vibrant	0.4468
cheerful	0.4316
cheery	0.4099
whimsical	0.4054
lively	0.4053
flashy	0.3991
colorfully	0.3954
adorned	0.3939
dazzling	0.3919
energetic	0.3879

Table 8: *colorful + bright - dark*

Word	Similarity
horrible	0.5763
weird	0.5435
awful	0.5378
ugly	0.5327
atrocious	0.5200
scary	0.5148
horrid	0.5114
freaky	0.5026
bizarre	0.4944
shocking	0.4932

Table 9: *amazing + bad - great*

Word	Similarity
irritating	0.5786
irksome	0.5241
puzzling	0.4946
bothersome	0.4933
bothers	0.4849
baffling	0.4761
annoyed	0.4760
infuriating	0.4663
troubling	0.4657
weird	0.4590

Table 10: *surprising + annoying - predictable*

2 Better Perceptron using Embeddings

2.1 Sentence Embedding and k-NN

2.1.1: 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?

The high similarity value is meaningful. Both sentences have a positive tone, and their labels are marked as positive.

```
First positive sentence:
Sentence 0: it 's a tour de force , written and directed so quietly that it 's implosion rather than explosion you fear
Similarity: 1.0000
--------------------------------------------------------
Most similar sentences:
Sentence 400: city reminds us how realistically nuanced a robert de niro performance can be when he is not more lucratively engaged in the shameless self caricature of ` analyze th
is ' ( 1999 ) and ` analyze that , ' promised ( or threatened ) for later this year
Similarity: 0.7564
--------------------------------------------------------
```

Figure 1: Ten closest sentence to the first positive example.

2.1.2: 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?

Both reviews have a critical emphasis and a negative feeling.

```
First negative sentence:
Sentence 0: 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
Similarity: 1.0000
--------------------------------------------------------
Most similar sentences:
Most similar sentences:
Sentence 2091: the plan to make enough into an inspiring tale of survival wrapped in the heart pounding suspense of a stylish psychological thriller ' has flopped as surely as a souffl gone wrong
Similarity: 0.8116
```

Figure 2: Ten closest sentence to the first negative example.

2.1.3: 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`.

Best dev error was 27.8%.

```
k=1  train_err: 0.0% (+:50.0%) dev_err: 37.0% (+:52.2%) running speed: 0.37s
k=3  train_err: 16.15% (+:50.48%) dev_err: 34.8% (+:50.0%) running speed: 0.57s
k=5  train_err: 19.84% (+:50.01%) dev_err: 34.5% (+:48.3%) running speed: 0.44s
k=7  train_err: 21.24% (+:49.86%) dev_err: 33.8% (+:49.0%) running speed: 0.5s
k=9  train_err: 21.99% (+:49.44%) dev_err: 31.9% (+:49.1%) running speed: 0.5s
k=11  train_err: 22.5% (+:49.28%) dev_err: 30.7% (+:49.1%) running speed: 0.5s
k=13  train_err: 22.81% (+:49.14%) dev_err: 31.3% (+:48.9%) running speed: 0.51s
k=15  train_err: 23.61% (+:48.99%) dev_err: 30.2% (+:49.6%) running speed: 0.5s
k=17  train_err: 23.46% (+:48.86%) dev_err: 31.3% (+:47.5%) running speed: 0.52s
k=19  train_err: 23.55% (+:48.1%) dev_err: 30.9% (+:47.7%) running speed: 0.55s
k=21  train_err: 24.04% (+:48.04%) dev_err: 30.9% (+:47.5%) running speed: 0.59s
k=23  train_err: 24.09% (+:48.06%) dev_err: 31.2% (+:46.6%) running speed: 0.47s
k=25  train_err: 24.02% (+:47.85%) dev_err: 30.2% (+:45.8%) running speed: 0.5s
k=27  train_err: 24.44% (+:47.69%) dev_err: 30.4% (+:46.0%) running speed: 0.51s
k=29  train_err: 24.31% (+:47.61%) dev_err: 29.3% (+:46.5%) running speed: 0.52s
k=31  train_err: 24.29% (+:47.34%) dev_err: 29.6% (+:46.6%) running speed: 0.49s
k=33  train_err: 24.28% (+:46.75%) dev_err: 30.5% (+:45.9%) running speed: 0.5s
k=35  train_err: 24.55% (+:46.62%) dev_err: 29.7% (+:46.3%) running speed: 0.51s
k=37  train_err: 24.62% (+:46.35%) dev_err: 30.0% (+:46.6%) running speed: 0.52s
k=39  train_err: 24.84% (+:46.06%) dev_err: 30.0% (+:46.8%) running speed: 0.51s
k=41  train_err: 24.79% (+:46.34%) dev_err: 30.4% (+:46.6%) running speed: 0.5s
k=43  train_err: 24.88% (+:46.28%) dev_err: 29.7% (+:46.9%) running speed: 0.5s
k=45  train_err: 24.64% (+:45.96%) dev_err: 29.3% (+:47.5%) running speed: 0.52s
k=47  train_err: 24.78% (+:46.12%) dev_err: 29.3% (+:46.3%) running speed: 0.53s
k=49  train_err: 25.16% (+:45.99%) dev_err: 29.1% (+:46.1%) running speed: 0.51s
k=51  train_err: 24.92% (+:46.02%) dev_err: 28.9% (+:45.7%) running speed: 0.57s
k=53  train_err: 25.01% (+:46.04%) dev_err: 29.1% (+:45.9%) running speed: 0.48s
k=55  train_err: 25.04% (+:45.89%) dev_err: 29.2% (+:45.2%) running speed: 0.52s
k=57  train_err: 24.75% (+:45.88%) dev_err: 29.3% (+:45.3%) running speed: 0.52s
k=59  train_err: 25.1% (+:45.75%) dev_err: 29.7% (+:45.5%) running speed: 0.51s
k=61  train_err: 24.75% (+:45.58%) dev_err: 28.5% (+:45.5%) running speed: 0.52s
k=63  train_err: 25.04% (+:45.44%) dev_err: 28.3% (+:45.3%) running speed: 0.67s
k=65  train_err: 25.11% (+:45.21%) dev_err: 28.3% (+:45.3%) running speed: 0.52s
k=67  train_err: 25.31% (+:44.91%) dev_err: 28.0% (+:45.4%) running speed: 0.53s
k=69  train_err: 25.11% (+:44.91%) dev_err: 28.6% (+:45.6%) running speed: 0.52s
k=71  train_err: 25.16% (+:44.99%) dev_err: 28.9% (+:44.9%) running speed: 0.53s
k=73  train_err: 25.16% (+:44.66%) dev_err: 27.8% (+:45.6%) running speed: 0.52s
k=75  train_err: 25.3% (+:44.65%) dev_err: 28.9% (+:45.1%) running speed: 0.52s
k=77  train_err: 25.48% (+:44.92%) dev_err: 28.7% (+:44.7%) running speed: 0.54s
k=79  train_err: 25.35% (+:44.75%) dev_err: 28.7% (+:45.5%) running speed: 0.53s
k=81  train_err: 25.12% (+:44.47%) dev_err: 29.0% (+:45.2%) running speed: 0.59s
k=83  train_err: 25.08% (+:44.38%) dev_err: 28.6% (+:44.8%) running speed: 0.51s
k=85  train_err: 25.24% (+:44.24%) dev_err: 28.9% (+:44.7%) running speed: 0.53s
k=87  train_err: 25.21% (+:44.24%) dev_err: 28.1% (+:44.5%) running speed: 0.53s
k=89  train_err: 25.31% (+:44.36%) dev_err: 28.4% (+:44.8%) running speed: 0.59s
k=91  train_err: 25.26% (+:44.36%) dev_err: 28.8% (+:44.4%) running speed: 0.51s
k=93  train_err: 25.28% (+:44.32%) dev_err: 29.2% (+:43.8%) running speed: 0.53s
k=95  train_err: 25.33% (+:44.3%) dev_err: 29.1% (+:43.3%) running speed: 0.56s
k=97  train_err: 25.35% (+:44.52%) dev_err: 29.4% (+:43.4%) running speed: 0.53s
k=99  train_err: 25.44% (+:44.14%) dev_err: 28.7% (+:43.5%) running speed: 0.55s
Best k value: 73.0
Predictions have been added to the test set and a new CSV file has been created.
```

Figure 3: Error rates of k-NN classifier on dev for k = 1,3,..99 using sentence embedding.

2.1.4: 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`.

Best dev error was 40.3%.

```
Training k-NN classifier...
k=1, dev error: 42.8%, time: 1.90s
k=3, dev error: 41.6%, time: 2.10s
k=5, dev error: 42.2%, time: 1.94s
k=7, dev error: 41.7%, time: 1.77s
k=9, dev error: 40.3%, time: 1.77s
k=11, dev error: 41.2%, time: 1.75s
k=13, dev error: 41.3%, time: 1.70s
k=15, dev error: 43.2%, time: 1.73s
k=17, dev error: 42.8%, time: 1.71s
k=19, dev error: 42.7%, time: 1.72s
k=21, dev error: 44.0%, time: 1.80s
k=23, dev error: 44.3%, time: 1.81s
k=25, dev error: 43.0%, time: 1.75s
k=27, dev error: 43.3%, time: 1.81s
k=29, dev error: 42.4%, time: 1.77s
k=31, dev error: 43.1%, time: 1.72s
k=33, dev error: 42.3%, time: 1.75s
k=35, dev error: 42.9%, time: 1.69s
k=37, dev error: 44.8%, time: 1.79s
k=39, dev error: 42.5%, time: 1.84s
k=41, dev error: 43.9%, time: 1.79s
k=43, dev error: 43.8%, time: 1.73s
k=45, dev error: 44.7%, time: 1.76s
k=47, dev error: 44.4%, time: 1.70s
k=49, dev error: 43.4%, time: 1.93s
k=51, dev error: 43.5%, time: 1.80s
k=53, dev error: 43.1%, time: 1.76s
k=55, dev error: 43.0%, time: 1.74s
k=57, dev error: 43.8%, time: 1.72s
k=59, dev error: 43.8%, time: 1.77s
k=61, dev error: 43.0%, time: 1.80s
k=63, dev error: 44.2%, time: 1.71s
k=65, dev error: 43.1%, time: 1.71s
k=67, dev error: 43.3%, time: 1.71s
k=69, dev error: 43.5%, time: 1.72s
k=71, dev error: 43.3%, time: 1.71s
k=73, dev error: 42.3%, time: 1.72s
k=75, dev error: 44.6%, time: 1.72s
k=77, dev error: 43.3%, time: 1.69s
k=79, dev error: 43.5%, time: 1.80s
k=81, dev error: 45.1%, time: 1.69s
k=83, dev error: 44.4%, time: 1.73s
k=85, dev error: 43.7%, time: 1.73s
k=87, dev error: 44.5%, time: 1.70s
k=89, dev error: 44.2%, time: 1.69s
k=91, dev error: 45.0%, time: 1.75s
k=93, dev error: 44.2%, time: 1.71s
k=95, dev error: 43.8%, time: 1.71s
k=97, dev error: 43.9%, time: 1.80s
k=99, dev error: 43.7%, time: 1.72s
Best k: 9, Best dev error: 40.3%
```

Figure 4: Error rates of k-NN classifier on dev for k = 1,3,..99 using one-hot vectors.

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

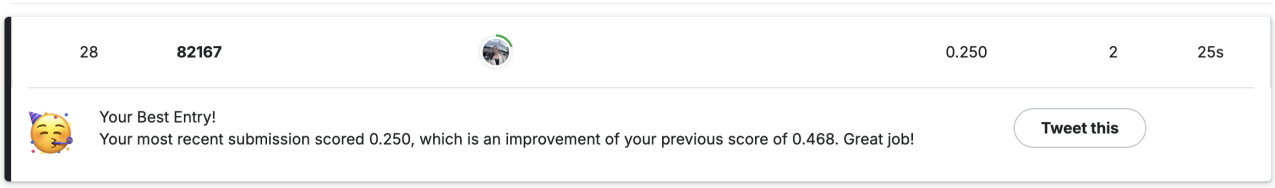

Figure 5: Public leaderboard ranking of k-nn predictions.

2.2 Reimplement Perceptron

2.2.1: For basic perceptron, show the training logs for 10 epochs. Compare your best dev error rate with the one from HW2.

In HW2, the best dev error rate was 30.1%. On the other hand, in the basic perceptron algorithm applied with sentence embedding, the best dev error rate was also 33.2%. We observe that the error rate increased because sentence embeddings are a complex method for simple algorithms like the basic perceptron.

```
epoch 1, update 2485, dev error 37.4%
epoch 2, update 2363, dev error 35.4%
epoch 3, update 2381, dev error 33.2%
epoch 4, update 2331, dev error 40.0%
epoch 5, update 2376, dev error 35.2%
epoch 6, update 2349, dev error 40.4%
epoch 7, update 2352, dev error 38.4%
epoch 8, update 2355, dev error 42.5%
epoch 9, update 2329, dev error 39.0%
epoch 10, update 2332, dev error 39.2%
best dev err 33.2%, time: 2.6 secs
```

Figure 6: The training logs for 10 epochs for basic perceptron

2.2.2: 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.

In HW2, the best dev error rate for the averaged perceptron was 26.3%, whereas in this assignment, the best dev error rate decreased to 23.6% for averaged perceptron with sentence embeddings%.

```
epoch 1, update 31.1%, dev error 24.9%
epoch 2, update 29.5%, dev error 23.9%
epoch 3, update 29.8%, dev error 24.3%
epoch 4, update 29.1%, dev error 24.1%
epoch 5, update 29.7%, dev error 24.2%
epoch 6, update 29.4%, dev error 23.9%
epoch 7, update 29.4%, dev error 23.6%
epoch 8, update 29.4%, dev error 23.8%
epoch 9, update 29.1%, dev error 24.1%
epoch 10, update 29.1%, dev error 24.4%
best dev err 23.6%, time: 2.9 secs
```

Figure 7: The training logs for 10 epochs for average perceptron.

2.2.3: Do you need to use smart averaging here?

Smart averaging is not necessary here because, unlike the sparse vector implementation in HW2, the vector used here is a dense vector.

2.2.4: 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.

In HW2, the best error rate for the averaged perceptron algorithm was 25.9%. In this assignment, using sentence embeddings reduced it to 24.2%.

```
epoch 1, update 32.0%, dev error 25.0%
epoch 2, update 30.4%, dev error 25.1%
epoch 3, update 30.3%, dev error 24.4%
epoch 4, update 29.9%, dev error 24.2%
epoch 5, update 30.4%, dev error 24.2%
epoch 6, update 30.7%, dev error 24.5%
epoch 7, update 30.4%, dev error 24.3%
epoch 8, update 30.1%, dev error 24.5%
epoch 9, update 29.9%, dev error 24.7%
epoch 10, update 30.2%, dev error 24.7%
best dev err 24.2%, time: 2.7 secs
Updated test file saved as test_predictions_hw4p224.csv
```

Figure 8: The training logs for 10 epochs for average perceptron after one-count word pruning.

2.2.5: 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.

The One-Hot method failed to capture the meanings between words effectively, whereas the sentence embedding successfully captured semantic meanings and understood contexts more accurately.

Sentence	Target	Prediction	One-Hot Prediction
- An atonal estrogen opera that demonizes feminism while gifting the most sympathetic male of the piece with a nice vomit bath at his wedding	-	-	+
- Two tedious acts light on great scares and a good surprise ending	-	-	+
- A worthy addition to the cinematic canon, which, at last count, numbered 52 different versions	+	+	-
- Jaunty fun, with its celeb strewn backdrop well used	+	+	-

Table 11: Comparison of sentence embedding and One-Hot Predictions

2.2.6: Submit your averaged perceptron models (both with and without pruning) to Kaggle, and record best your public error rate and ranking (take a screenshot). (should be $\sim$22.4%)

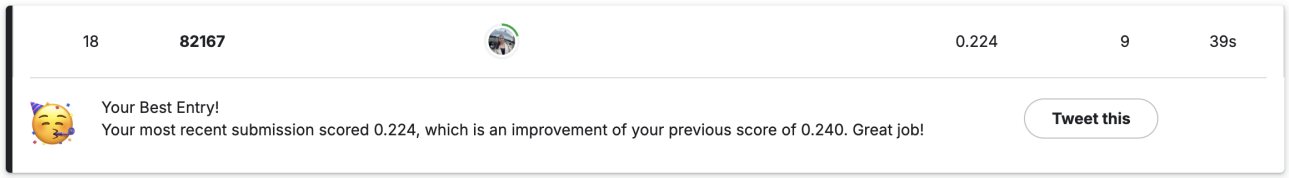

Figure 9: Kaggle score for averaged perceptron without pruning.

2.3 Summarize the error rates in this table

	one-hot, best dev	embedding, best dev	best kaggle public
k-NN	40.3%	23.6%	25.0%
perceptron	26.3%	24.2%	22.4%

3 Try some other learning algorithms with `sklearn`

3.1: Which algorithm did you try? What adaptations did you make to your code to make it work with that algorithm?

For this part, I tried the SVM algorithm with and without one-count word pruning.

3.2: What's the dev error rate(s) and running time?

My dev error rate was 23.4%, and the running time on my personal computer was 4.6 seconds.

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

The difference in dev error rates between k-NN with one-hot and sentence embedding features is notable (40.3% and 27.8%, respectively). On the other hand, perceptron and SVM algorithms provide much better results, with error rates of 23.6% and 22.4%, respectively. However, as the complexity of the data increases, it is likely that the perceptron algorithm's capacity to learn complex relationships will remain limited. The better dev error rate of SVM reflects its ability to handle high-dimensional data more effectively.

Based on my evaluation of other algorithms applied in HW2, HW3, and this assignment, I concluded that while the averaged perceptron performs well in simpler tasks, as the data becomes more complex -such as complex relationships, noisy, or high-dimensional data- more advanced algorithms tend to yield better results.

4 Deployment

4.1: What's your best error rate on dev, and which algorithm and setting achieved it?

The best dev error rate I achieved was with the averaged perceptron without pruning (22.4%). I was unable to achieve a lower error rate with any other algorithm.

4.2: What's your best public error rate on Kaggle, and which algorithm and setting achieved it?

The best public error rate was the same for both SVM without pruning and averaged perceptron without pruning, which was 22.4%.

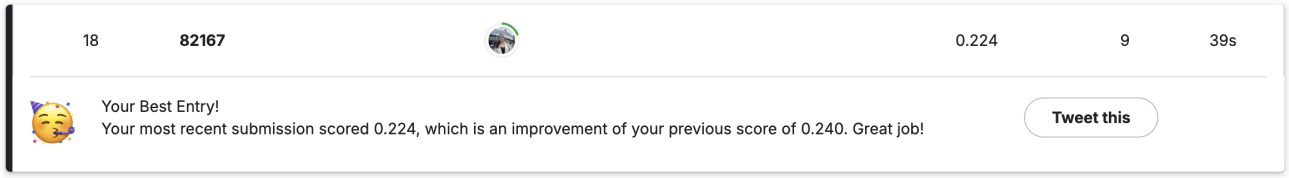

Figure 10: Kaggle score for averaged perceptron without pruning.

5 Debriefing

5.1: Approximately how many hours did you spend on this assignment?

I spent approximately 14 hours on this assignment.

5.2: Would you rate it as easy, moderate, or difficult?

I would rate it as moderate in difficulty.

5.3: Did you work on it mostly alone, or mostly with other people?

I worked completely on my own.

5.4: How deeply do you feel you understand the material it covers (%-100%)?

I feel I have learned about 85% of the material.

5.5: Any other comments?

Thank you!