AI 534 Machine Learning HW4

Submitted by: Apoorv Malik (malikap@oregonstate.edu)

1.2. Vector Similarity

Q1: Can you find the top-10 similar words to wonderful and awful? Do your results make sense?

For "wonderful":
1. Marvelous
2. Fantastic
3. Great
4. Fabulous
5. Terrific
6. Lovely
7. Amazing
8. Beautiful
9. Magnificent
10. Delightful

For "awful":
1. Horrible
2. Terrible
3. Dreadful
4. Horrid
5. Atrocious
6. Ugly
7. Lousy
8. Unbelievable
9. Appalling
10. Hideous

These results make sense as they reflect words with similar meanings to "wonderful" and "awful". The similar words for "wonderful" are all positive and uplifting, while those for "awful" are negative and express strong disapproval.

Q2: Also come up with 3 other queries and show your results. Do they make sense?

For "technology":
1. Technologies
2. Innovations
3. Innovation
4. Tech
5. Technological
6. Software
7. Solutions
8. Innovative
9. Devices
10. Innovators

For "music":
1. Jazz
2. Songs
3. Musicians
4. Tunes
5. Musical
6. Song
7. Piano
8. Folk
9. Guitar
10. Entertainment

For "happy":
1. Glad
2. Satisfied
3. Proud
4. Delighted
5. Disappointed
6. Excited
7. Unhappy
8. Confident
9. Nice
10. Hopeful

Yes these results make sense. These results align well with the respective contexts of “technology”, “music”, and “happy”. Interestingly for “happy” many of the words that appear in the list are antonyms and contrasting emotions.

1.3. Word Analogy

Q1: Find top 10 words closest to the following two queries. Do your results make sense?

a) Top 10 words closest to **sister - woman + man** are:

1. Brother
2. Uncle
3. Nephew
4. Son
5. Father
6. Brothers
7. Dad
8. Siblings
9. Daughter
10. Sons

These results make sense as they include male relatives and family terms, which align with the analogy aiming to find a male equivalent to "sister". The word "brother" being the top result makes so much sense.

b) Top 10 words closest to **harder - hard + fast** are:

1. Faster
2. Rapidly
3. Easier
4. Slow
5. Quickly
6. Bigger
7. Cheaper
8. Louder
9. Slowly
10. Smarter

These results also make sense. The top result, "faster," is directly relevant to the analogy, suggesting an intensification, much like "harder" does for "hard".

Q2: Also come up with 3 other queries and show your results. Do they make sense?

For **cold - winter + summer**:

1. Chilly
2. Chill
3. Hot
4. Cool
5. Chilled
6. Warm
7. Cooler
8. Summertime
9. Sweating
10. Steamy

For **write - kitchen + office**:

1. Written
2. Writing
3. Read
4. Rewrite
5. Wrote
6. Post
7. Copy
8. Speak
9. Copies
10. Send

For **read - book + computer:**

1. Screenful
2. Laptops
3. Reading
4. Hack
5. Delete
6. Reads
7. Software
8. Listen
9. Modem
10. Programmer

Yes the results make sense.

The results for "cold - winter + summer" are interesting, as they include words associated with both cooler and warmer temperatures, reflecting the complex relationship between the words "cold," "winter," and “summer.” The results for "write - kitchen + office" focus on activities and verbs related to writing, particularly in a professional or office context, which aligns with the intended analogy. The results for "read - book + computer" show a mix of terms related to reading on computers, including "laptops," "software," and "programmer," suggesting a shift from traditional reading to digital forms.

2.1. Reimplement k-NN with Sentence Embedding

Q1. 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?

A different sentence closest to the first (positive) sentence (“It's a tour de force, written and directed so quietly that it's implosion rather than explosion you fear.”) is:

“**- A semi-autobiographical film that's so sloppily written and cast that you cannot believe anyone more central to the creation of Bugsy than the caterer had anything to do with it.**”

Yes, it makes sense. For the first sentence (+), the closest sentence found has a negative sentiment, indicating a mismatch in terms of the sentiment label. This suggests that while the sentence embeddings may capture some aspects of the sentence structure or content, they might not always align perfectly with the sentiment or meaning.

Q2. 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?

A different sentence closest to the second (negative) 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.”) is:

"- 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 soufflé gone wrong."

Yes, it makes sense. For the second sentence (-), the closest sentence also has a negative sentiment, which aligns well with the sentiment of the original sentence. This indicates a better match in terms of both content and sentiment label.

Q3: Report the error rate of k-NN classifier on dev for k = 1, 3, ...99 using sentence embedding. You can reuse your code from HW1 or use sklearn.

```
[LOG] k = 1      dev error rate = 0.37
[LOG] k = 3      dev error rate = 0.348
[LOG] k = 5      dev error rate = 0.345
[LOG] k = 7      dev error rate = 0.338
[LOG] k = 9      dev error rate = 0.319
[LOG] k = 11     dev error rate = 0.307
[LOG] k = 13     dev error rate = 0.313
[LOG] k = 15     dev error rate = 0.302
[LOG] k = 17     dev error rate = 0.313
[LOG] k = 19     dev error rate = 0.309
[LOG] k = 21     dev error rate = 0.309
[LOG] k = 23     dev error rate = 0.312
[LOG] k = 25     dev error rate = 0.302
[LOG] k = 27     dev error rate = 0.304
[LOG] k = 29     dev error rate = 0.293
[LOG] k = 31     dev error rate = 0.296
[LOG] k = 33     dev error rate = 0.305
[LOG] k = 35     dev error rate = 0.297
[LOG] k = 37     dev error rate = 0.3
[LOG] k = 39     dev error rate = 0.3
[LOG] k = 41     dev error rate = 0.304
[LOG] k = 43     dev error rate = 0.297
[LOG] k = 45     dev error rate = 0.293
[LOG] k = 47     dev error rate = 0.293
[LOG] k = 49     dev error rate = 0.291
[LOG] k = 51     dev error rate = 0.289
[LOG] k = 53     dev error rate = 0.291
[LOG] k = 55     dev error rate = 0.292
[LOG] k = 57     dev error rate = 0.293
[LOG] k = 59     dev error rate = 0.297
[LOG] k = 61     dev error rate = 0.285
[LOG] k = 63     dev error rate = 0.283
[LOG] k = 65     dev error rate = 0.283
[LOG] k = 67     dev error rate = 0.28
[LOG] k = 69     dev error rate = 0.286
[LOG] k = 71     dev error rate = 0.289
[LOG] k = 73     dev error rate = 0.278
[LOG] k = 75     dev error rate = 0.289
[LOG] k = 77     dev error rate = 0.287
[LOG] k = 79     dev error rate = 0.287
[LOG] k = 81     dev error rate = 0.29
[LOG] k = 83     dev error rate = 0.286
[LOG] k = 85     dev error rate = 0.289
[LOG] k = 87     dev error rate = 0.281
[LOG] k = 89     dev error rate = 0.284
[LOG] k = 91     dev error rate = 0.288
[LOG] k = 93     dev error rate = 0.292
[LOG] k = 95     dev error rate = 0.291
[LOG] k = 97     dev error rate = 0.294
[LOG] k = 99     dev error rate = 0.287
Best k: 73
Best error rate: 0.278
```

Best error rate = 27.8% (k = 73)

Q4: Report the error rate 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.

```
[LOG] k = 1      dev error rate = 0.402
[LOG] k = 3      dev error rate = 0.406
[LOG] k = 5      dev error rate = 0.414
[LOG] k = 7      dev error rate = 0.418
[LOG] k = 9      dev error rate = 0.407
[LOG] k = 11     dev error rate = 0.391
[LOG] k = 13     dev error rate = 0.391
[LOG] k = 15     dev error rate = 0.383
[LOG] k = 17     dev error rate = 0.402
[LOG] k = 19     dev error rate = 0.401
[LOG] k = 21     dev error rate = 0.392
[LOG] k = 23     dev error rate = 0.399
[LOG] k = 25     dev error rate = 0.396
[LOG] k = 27     dev error rate = 0.4
[LOG] k = 29     dev error rate = 0.393
[LOG] k = 31     dev error rate = 0.399
[LOG] k = 33     dev error rate = 0.398
[LOG] k = 35     dev error rate = 0.413
[LOG] k = 37     dev error rate = 0.415
[LOG] k = 39     dev error rate = 0.404
[LOG] k = 41     dev error rate = 0.401
[LOG] k = 43     dev error rate = 0.4
[LOG] k = 45     dev error rate = 0.405
[LOG] k = 47     dev error rate = 0.411
[LOG] k = 49     dev error rate = 0.412
[LOG] k = 51     dev error rate = 0.406
[LOG] k = 53     dev error rate = 0.4
[LOG] k = 55     dev error rate = 0.41
[LOG] k = 57     dev error rate = 0.426
[LOG] k = 59     dev error rate = 0.41
[LOG] k = 61     dev error rate = 0.409
[LOG] k = 63     dev error rate = 0.403
[LOG] k = 65     dev error rate = 0.41
[LOG] k = 67     dev error rate = 0.416
[LOG] k = 69     dev error rate = 0.413
[LOG] k = 71     dev error rate = 0.409
[LOG] k = 73     dev error rate = 0.406
[LOG] k = 75     dev error rate = 0.4
[LOG] k = 77     dev error rate = 0.397
[LOG] k = 79     dev error rate = 0.407
[LOG] k = 81     dev error rate = 0.406
[LOG] k = 83     dev error rate = 0.394
[LOG] k = 85     dev error rate = 0.399
[LOG] k = 87     dev error rate = 0.41
[LOG] k = 89     dev error rate = 0.418
[LOG] k = 91     dev error rate = 0.412
[LOG] k = 93     dev error rate = 0.398
[LOG] k = 95     dev error rate = 0.395
[LOG] k = 97     dev error rate = 0.403
[LOG] k = 99     dev error rate = 0.4
Best k: 15
Best error rate: 0.383
```

Best error rate = 38.3% (k = 15)

2.2. Reimplement Perceptron with Sentence Embedding

Q1: For basic perceptron, show the training logs for 10 epochs (should be around 33%). Compare your error rate with the one from HW2.

```
[INFO] Using Sentence Embeddings
[INFO] Training basic perceptron...
epoch 1, update 31.1%, dev 37.4%
epoch 2, update 29.5%, dev 35.4%
epoch 3, update 29.8%, dev 33.2%
epoch 4, update 29.1%, dev 40.0%
epoch 5, update 29.7%, dev 35.2%
epoch 6, update 29.4%, dev 40.4%
epoch 7, update 29.4%, dev 38.4%
epoch 8, update 29.4%, dev 42.5%
epoch 9, update 29.1%, dev 39.0%
epoch 10, update 29.1%, dev 39.2%
best dev err 33.2%, |w|=300, time: 2.1 secs
```

```
[INFO] Using One-Hot Vectors
[INFO] Training basic perceptron...
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|=16744, time: 0.6 secs
```

For basic perceptron:
- Best dev error using sentence embedding is: **33.2%**
- HW2 (one hot vector) best dev error is: **28.9%**

Q2: For averaged perceptron, show the training logs for 10 epochs (should be around 23%). Compare your error rate with the one from HW2.

```
[INFO] Using Sentence Embeddings
[INFO] Training average perceptron...
epoch 1, update 31.1%, dev 24.9%
epoch 2, update 29.5%, dev 23.9%
epoch 3, update 29.8%, dev 24.3%
epoch 4, update 29.1%, dev 24.1%
epoch 5, update 29.7%, dev 24.2%
epoch 6, update 29.4%, dev 23.9%
epoch 7, update 29.4%, dev 23.6%
epoch 8, update 29.4%, dev 23.8%
epoch 9, update 29.1%, dev 24.1%
epoch 10, update 29.1%, dev 24.4%
best dev err 23.6%, |w|=300, time: 2.2 secs
```

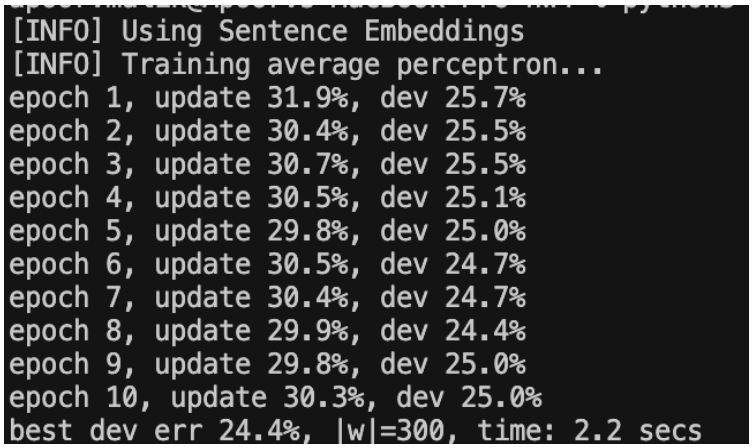
```
[INFO] Using One-Hot Vectors
[INFO] Training average perceptron...
epoch 1, update 39.0%, dev 31.4%
epoch 2, update 25.5%, dev 27.7%
epoch 3, update 20.8%, dev 27.2%
epoch 4, update 17.2%, dev 27.6%
epoch 5, update 14.1%, dev 27.2%
epoch 6, update 12.2%, dev 26.7%
epoch 7, update 10.5%, dev 26.3%
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|=15806, time: 0.7 secs
```

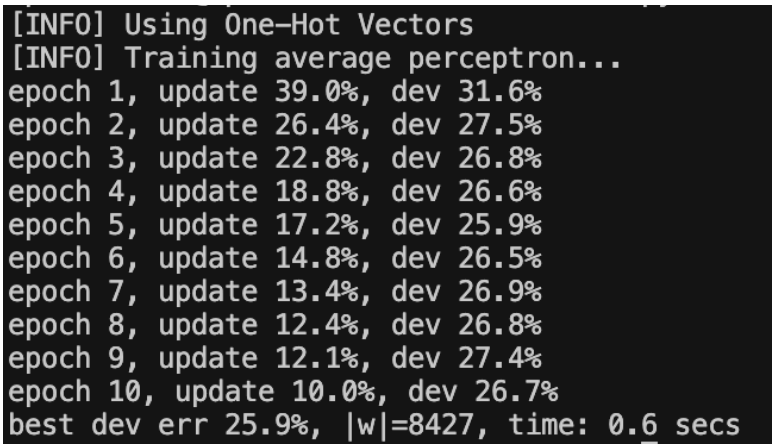
For Average perceptron:
- Best dev error using sentence embedding is: **23.6%**
- HW2 (one hot vector) best dev error is: **26.3%**

Q3: Do you need to use smart averaging here?

Its not necessary, but I have used smart averaging in my average perceptron implementation.

Q4: For averaged perceptron after pruning one-count words, show the training logs for 10 epochs. Compare your error rate with the one from HW2.





For Average perceptron, after pruning one-count words:
- Best dev error using sentence embedding is: **24.4%**
- HW2 (one hot vector) best dev error is: **25.9%**

Q5: 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.

Sentence 1: “It may scream low budget, but this charmer has a spirit that cannot be denied.”
- Using word2vec features, the average perceptron classifies it correctly as positive.
- Using one hot representation, the average perceptron classifies it incorrectly as negative.

The one-hot model focuses on "low budget" and predicts a negative sentiment. In contrast, the sentence embedding model (word2vec features) correctly interprets the overall positive sentiment in the sentence (“but” indicating a shift from a negative beginning to a positive ending).

Sentence 2: “The action sequences are fun and reminiscent of combat scenes from the Star Wars series.”
- Using word2vec features, the average perceptron classifies it correctly as positive.
- Using one hot representation, the average perceptron classifies it incorrectly as negative.

This sentence is overall positive, praising the action sequences by comparing them to "Star Wars. The one-hot model misses the context of "fun" and "reminiscent of combat scenes from the Star Wars series," leading to an incorrect prediction. The sentence embedding model, captures the positive sentiment.

2.3. Summarize the error rates in a 2x2 table

Table 01: Error Rates Summary

	k-NN	Avg. Perceptron
one-hot	38.3%	26.3%
embedding	27.8%	23.6%

3. Try some other learning algorithms with sklearn

Q1: Which algorithm did you try? What adaptations did you make to your code to make it work with that algorithm?

I choose SVM because of its effectiveness in high-dimensional spaces and its capability to handle non-linear relationships through the use of different kernel functions.

Adaptations Made:
- The sentence embeddings generated from the word embeddings were converted into NumPy arrays as SVM in scikit-learn expects data in this format.
- The SVM algorithm has several critical hyperparameters like C (regularization), kernel, and gamma. These were tuned to find the best combination for the given dataset. I got best results with C = 1, gamma = 1.3, kernel = ‘rbf’.

Q2: What's the dev error rate(s) and running time?

- I got the **best dev error rate** with **SVM** so far, which is **23.00%**.
- The training took **6.15 seconds** to complete.
- The evaluation on dev-set took **1.07 seconds** to complete.
- SVM hyperparameter tuning step almost took about **30 minutes**.

Q3: What did you learn in terms of the comparison between averaged perceptron and these other (presumably more popular and well-known) learning algorithms?

- SVMs, especially with non-linear kernels, can capture more complex patterns in the data compared to the linear decision boundary of an averaged perceptron.
- This complexity often translates to better performance, particularly in high-dimensional spaces or when the data is not linearly separable.
- Training time for SVMs can be longer than for perceptrons, especially when dealing with large datasets or when conducting extensive hyperparameter tuning.
- SVM's ability to generalize (perform well on unseen data) is often better due to its margin maximization principle and regularization, which are not explicit features of the perceptron algorithm.

4. Deployment

- I got the **best dev error rate** with **SVM** so far, which is **23.00%**.
- I got best results with these hyperparameters value: C = 1, gamma = 1.3, kernel = 'rbf'.

5. Debriefing

1. I spent about 8 to 10 hours on this assignment.
2. I rate it as moderate.
3. I worked on it alone.
4. I understand the material 95%.
5. This was fun!