AI 534 Machine Learning – HW4

Name: Nirmit Patel
Email: patenirm@oregonstate.edu

1 Word Embeddings (5 pts)

1.1 Load and Query (0 pts)

```
from gensim.models import KeyedVectors
wv = KeyedVectors.load('embs_train.kv')
✓ 0.4s
```

```
wv
✓ 0.1s
<gensim.models.keyedvectors.KeyedVectors at 0x105cfd6f0>
```

```
wv['big']
✓ 0.3s
Output exceeds the size limit. Open the full output data in a text editor
array([ 0.11132812,  0.10595703, -0.07373047,  0.18847656,  0.07666016,
       -0.3828125 , -0.0625    , -0.07470703,  0.05957031,  0.22167969,
        0.20507812, -0.09228516,  0.05395508,  0.01379395, -0.16992188,
        0.05493164,  0.09619141,  0.06103516, -0.14160156,  0.03173828,
       -0.08642578,  0.12011719,  0.06445312,  0.22070312,  0.06835938,
        0.04956055, -0.22460938, -0.06298828,  0.09179688, -0.00531006,
       -0.11425781,  0.20605469,  0.31054688, -0.0625    , -0.02026367,
       -0.13476562, -0.02697754,  0.2734375 ,  0.27929688,  0.21386719,
```

1.2 Vector Similarity (2.5 pts)

Q: Can you find the top-10 similar words to wonderful and awful? Do your results make sense? (1 pt)

```
wv.most_similar('wonderful', topn=10)
✓ 0.1s
[('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)
✓ 0.1s
[('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 as the similar words for "wonderful" are positive and convey a sense of delight, while the similar words for "awful" are negative and indicate something unpleasant or of low quality.

Q: Also come up with 3 other queries and show your results. Do they make sense? (1.5 pts)

I used ‘love’, ‘complete’ and ‘organized’ for additional queries.

```
wv.most_similar('love', topn=10)
✓ 0.1s
[('loved', 0.6907791495323181),
 ('adore', 0.6816874146461487),
 ('loves', 0.6618633270263672),
 ('passion', 0.6100709438323975),
 ('hate', 0.6003956198692322),
 ('loving', 0.5886634588241577),
 ('affection', 0.5664337873458862),
 ('cherish', 0.5405279994010925),
 ('adored', 0.5365651845932007),
 ('wonderful', 0.5256202816963196)]
```

```
wv.most_similar('complete', topn=10)
✓ 0.1s
[('completed', 0.6287841200828552),
 ('completes', 0.5529298782348633),
 ('full', 0.499875545501709),
 ('completion', 0.45394039154052734),
 ('detailed', 0.4336627721786499),
 ('comprehensive', 0.4189838767051697),
 ('thorough', 0.41830331087112427),
 ('begin', 0.41768521070480347),
 ('entire', 0.4137260317802429),
 ('proceed', 0.3922833502292633)]
```

```
wv.most_similar('organized', topn=10)
✓ 0.9s
[('organizing', 0.6603507995605469),
 ('organize', 0.6334936618804932),
 ('staged', 0.5322155356407166),
 ('orchestrated', 0.4860710799694061 3),
 ('formed', 0.45218321681022644),
 ('conducted', 0.4488440752029419),
 ('undertaken', 0.41358616948127747),
 ('held', 0.41076338291168213),
 ('attending', 0.3867975175380707),
 ('motivated', 0.38134443759918213)]
```

The results make sense as the similar words for each query are semantically related, capturing the context and meaning of the input words.

1.3 Word Analogy (2.5 pts)

Q: Find top 10 words closest to the following two queries. Do your results make sense? (1 pt)

sister - woman + man and harder - hard + fast.

```
wv.most_similar(positive=['sister', 'man'], negative=['woman'], topn=10)
✓ 0.1s
[('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)
✓ 0.9s
[('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)]
```

The results make sense, and they demonstrate the linguistic regularities captured by word vectors.

Q: Also come up with 3 other queries and show your results. Do they make sense? (1.5 pts)

```
wv.most_similar(positive=['him', 'they'], negative=['he'], topn=5)
✓ 0.1s
[('them', 0.7502678632736206),
 ('themselves', 0.5942553281784058),
 ('their', 0.5658615827560425),
 ('us', 0.5205104351043701),
 ('yourselves', 0.5106819272041321)]
```

```
wv.most_similar(positive=['white', 'coffee'], negative=['water'], topn=5)
```

✓ 0.1s

```
[('black', 0.5280864238739014),
 ('brown', 0.42677968740463257),
 ('colored', 0.42052048444747925),
 ('pink', 0.3950096070766449),
 ('bean', 0.3780435919761657)]
```

```
wv.most_similar(positive=['winter', 'hot'], negative=['cold'], topn=5)
```

✓ 0.1s

```
[('summer', 0.5669187903404236),
 ('spring', 0.5186282992362976),
 ('hottest', 0.5085405707359314),
 ('summertime', 0.4723301827907562),
 ('season', 0.4058018922805786)]
```

The results make sense.

2 Better Perceptron using Embeddings (6.5 pts)

2.1 Sentence Embedding and k-NN (3 pts)

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? (0.5 pts)

> Idx: 0 Label: 1 Review: it tour de force written directed so quietly that it implosion rather than explosion you fear
>
> Idx: 2061 Label: -1 Review: semi autobiographical film that so sloppily written cast that you can not believe anyone more central the creation than the caterer had anything do with it
>
> This pairing does not make sense, which highlights potential challenges or errors in the k-NN predictions, as the meanings and labels of these sentences are discordant.

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? (0.5 pts)

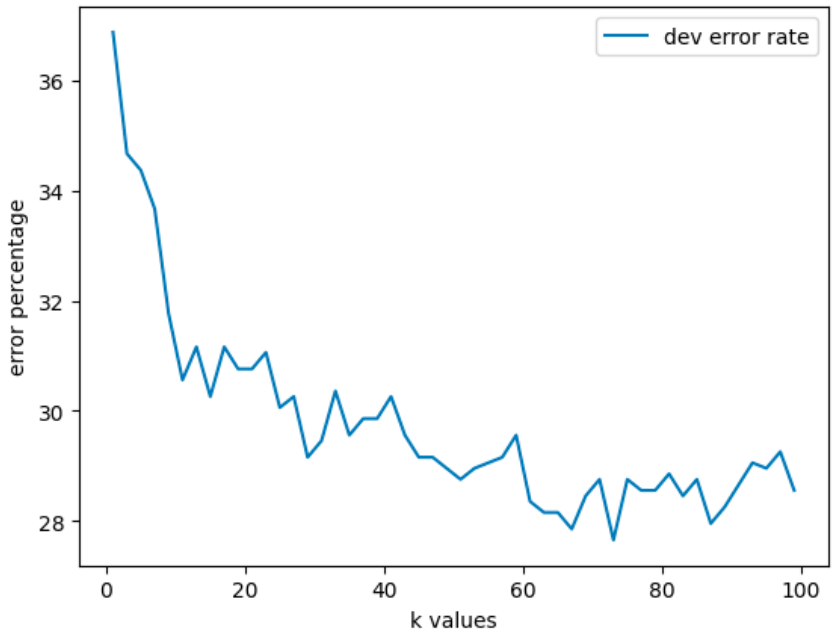
> Idx: 1 Label: -1 Review: places slightly believable love triangle in difficult swallow setting then disappointingly moves the story into the realm an improbable thriller
>
> Idx: 4205 Label: -1 Review: the plan make enough into an inspiring tale survival wrapped in the heart pounding suspense stylish psychological thriller has flopped as surely as souffl gone wrong
>
> In this case, the k-NN prediction appears to be correct, as the meanings and labels of these sentences align well. The sentences share the theme of a narrative turning into an improbable or disappointing thriller, making the pairing sensible.

3. 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. (1 pts)

> k = 1 train_err: 0.00% dev_err: 36.87%
> k = 3 train_err: 16.15% dev_err: 34.67%
> k = 5 train_err: 19.84% dev_err: 34.37%
> k = 7 train_err: 21.24% dev_err: 33.67%
> k = 9 train_err: 21.99% dev_err: 31.76%
> k = 11 train_err: 22.50% dev_err: 30.56%
> k = 13 train_err: 22.81% dev_err: 31.16%
> k = 15 train_err: 23.61% dev_err: 30.26%
> k = 17 train_err: 23.46% dev_err: 31.16%
> k = 19 train_err: 23.55% dev_err: 30.76%
> k = 21 train_err: 24.04% dev_err: 30.76%
> k = 23 train_err: 24.09% dev_err: 31.06%
> k = 25 train_err: 24.02% dev_err: 30.06%
> k = 27 train_err: 24.44% dev_err: 30.26%
> k = 29 train_err: 24.31% dev_err: 29.16%

k = 31 train_err: 24.29% dev_err: 29.46%
k = 33 train_err: 24.28% dev_err: 30.36%
k = 35 train_err: 24.55% dev_err: 29.56%
k = 37 train_err: 24.62% dev_err: 29.86%
k = 39 train_err: 24.84% dev_err: 29.86%
k = 41 train_err: 24.79% dev_err: 30.26%
k = 43 train_err: 24.88% dev_err: 29.56%
k = 45 train_err: 24.64% dev_err: 29.16%
k = 47 train_err: 24.78% dev_err: 29.16%
k = 49 train_err: 25.16% dev_err: 28.96%
k = 51 train_err: 24.92% dev_err: 28.76%
k = 53 train_err: 25.01% dev_err: 28.96%
k = 55 train_err: 25.04% dev_err: 29.06%
k = 57 train_err: 24.75% dev_err: 29.16%
k = 59 train_err: 25.10% dev_err: 29.56%
k = 61 train_err: 24.75% dev_err: 28.36%
k = 63 train_err: 25.04% dev_err: 28.16%
k = 65 train_err: 25.11% dev_err: 28.16%
k = 67 train_err: 25.31% dev_err: 27.86%
k = 69 train_err: 25.11% dev_err: 28.46%
k = 71 train_err: 25.16% dev_err: 28.76%
k = 73 train_err: 25.16% dev_err: 27.66%
k = 75 train_err: 25.30% dev_err: 28.76%
k = 77 train_err: 25.48% dev_err: 28.56%
k = 79 train_err: 25.35% dev_err: 28.56%
k = 81 train_err: 25.12% dev_err: 28.86%
k = 83 train_err: 25.08% dev_err: 28.46%
k = 85 train_err: 25.24% dev_err: 28.76%
k = 87 train_err: 25.21% dev_err: 27.96%
k = 89 train_err: 25.31% dev_err: 28.26%
k = 91 train_err: 25.26% dev_err: 28.66%
k = 93 train_err: 25.28% dev_err: 29.06%
k = 95 train_err: 25.32% dev_err: 28.96%
k = 97 train_err: 25.35% dev_err: 29.26%
k = 99 train_err: 25.44% dev_err: 28.56%

best k: 73 best dev error: 27.66

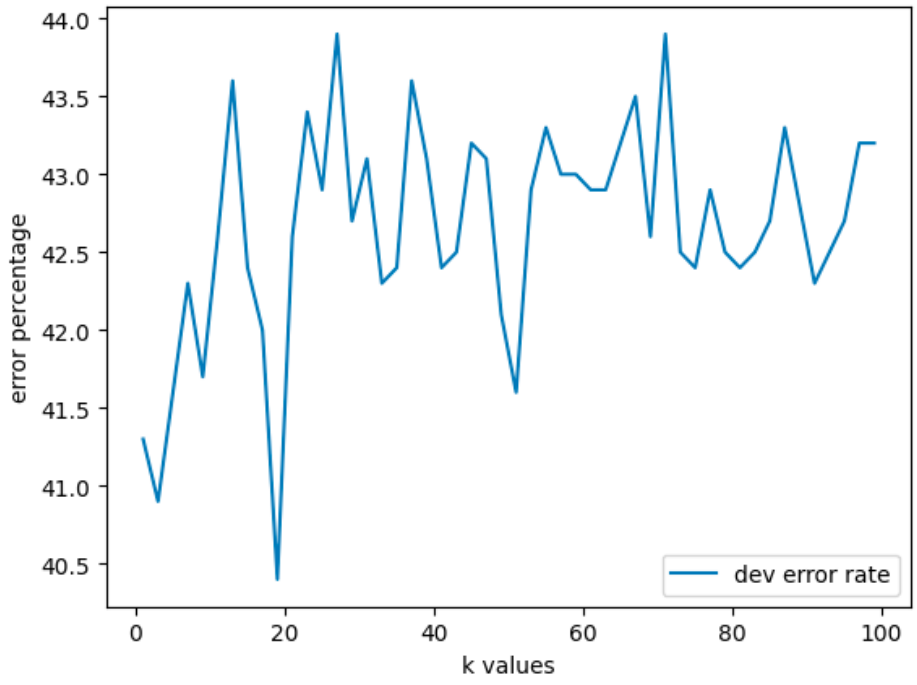


4. Report the error rate of k-NN classifier on dev for k = 1, 3, ...99 using one-hot vectors from HW2. You can reuse your code from HWs 1-2 and/or use sklearn. (1 pt). (should be around 40%).

k = 1 dev_err: 41.30%
k = 3 dev_err: 40.90%
k = 5 dev_err: 41.60%
k = 7 dev_err: 42.30%
k = 9 dev_err: 41.70%
k = 11 dev_err: 42.60%
k = 13 dev_err: 43.60%
k = 15 dev_err: 42.40%
k = 17 dev_err: 42.00%
k = 19 dev_err: 40.40%
k = 21 dev_err: 42.60%
k = 23 dev_err: 43.40%
k = 25 dev_err: 42.90%
k = 27 dev_err: 43.90%
k = 29 dev_err: 42.70%
k = 31 dev_err: 43.10%
k = 33 dev_err: 42.30%
k = 35 dev_err: 42.40%
k = 37 dev_err: 43.60%
k = 39 dev_err: 43.10%
k = 41 dev_err: 42.40%
k = 43 dev_err: 42.50%
k = 45 dev_err: 43.20%
k = 47 dev_err: 43.10%
k = 49 dev_err: 42.10%
k = 51 dev_err: 41.60%
k = 53 dev_err: 42.90%
k = 55 dev_err: 43.30%

k = 57 dev_err: 43.00%
k = 59 dev_err: 43.00%
k = 61 dev_err: 42.90%
k = 63 dev_err: 42.90%
k = 65 dev_err: 43.20%
k = 67 dev_err: 43.50%
k = 69 dev_err: 42.60%
k = 71 dev_err: 43.90%
k = 73 dev_err: 42.50%
k = 75 dev_err: 42.40%
k = 77 dev_err: 42.90%
k = 79 dev_err: 42.50%
k = 81 dev_err: 42.40%
k = 83 dev_err: 42.50%
k = 85 dev_err: 42.70%
k = 87 dev_err: 43.30%
k = 89 dev_err: 42.80%
k = 91 dev_err: 42.30%
k = 93 dev_err: 42.50%
k = 95 dev_err: 42.70%
k = 97 dev_err: 43.20%
k = 99 dev_err: 43.20%

best k: 19 best dev error: 40.40



2.2 Reimplement Perceptron with Sentence Embedding (3 pts)

1. For basic perceptron, show the training logs for 10 epochs (should be around 33%). Compare your error rate with the one from HW2. (0.5 pts)

epoch 1, update 31.2%, dev 41.3%
epoch 2, update 29.2%, dev 37.3%
epoch 3, update 29.7%, dev 36.1%
epoch 4, update 29.2%, dev 34.7%
epoch 5, update 29.7%, dev 37.0%
epoch 6, update 29.1%, dev 33.8%
epoch 7, update 29.5%, dev 40.9%
epoch 8, update 29.5%, dev 35.7%
epoch 9, update 28.9%, dev 42.6%
epoch 10, update 28.9%, dev 36.2%
best dev err 33.8%, |w|=300, time: 0.1 secs

HW2 result:
best dev err 29.5, best dev_ave err 25.9, |w|=9974, time: 0.3 secs

The basic perceptron seems to have a higher dev error rate (33.8%) compared to the HW2 perceptron (29.5% basic, 25.9% average). However, it's important to note that the dimensionality of the weight vector (|w|) for the basic perceptron is significantly lower (300) compared to the HW2 perceptron (9974). The execution time for the basic perceptron is also notably faster (0.1 secs) than the HW2 perceptron (0.3 secs). This trade-off between error rate, dimensionality, and execution time should be considered when choosing the perceptron model for a given task.

2. For averaged perceptron, show the training logs for 10 epochs (should be around 23%). Compare your error rate with the one from HW2. (0.5 pts)

epoch 1, update 31.2%, dev 26.7%
epoch 2, update 29.2%, dev 25.4%
epoch 3, update 29.7%, dev 24.5%
epoch 4, update 29.2%, dev 25.2%
epoch 5, update 29.7%, dev 24.8%
epoch 6, update 29.1%, dev 24.9%
epoch 7, update 29.5%, dev 24.8%
epoch 8, update 29.5%, dev 24.4%
epoch 9, update 28.9%, dev 24.4%
epoch 10, update 28.9%, dev 24.3%
best dev err 24.3%, |w|=300, time: 0.1 secs

HW2 result:
best dev err 29.5, best dev_ave err 25.9, |w|=9974, time: 0.3 secs

The averaged perceptron shows a significant improvement in dev error rate, achieving 24.3% compared to 29.5% (basic) and 25.9% (average) in HW2. Additionally, the dimensionality of the weight vector (|w|) for the averaged perceptron is much lower (300) than the HW2 perceptron (9974). The execution time for the averaged perceptron is also notably faster (0.1 secs) than the

HW2 perceptron (0.3 secs). This indicates that the averaged perceptron is more efficient and effective in this context.

3. Do you need to use smart averaging here? (0.5 pts)

No, the use of smart averaging is not necessary for word embeddings. Since the model's size is constant (300, the size of word embeddings), there is no risk of overloading the model. The fixed size of the model makes it unnecessary to implement smart averaging, which is more relevant in situations where the model's size may vary dynamically.

4. For averaged perceptron after pruning one-count words, show the training logs for 10 epochs. Compare your error rate with the one from HW2. (0.5 pts)

epoch 1, update 31.9%, dev 26.9%
epoch 2, update 31.1%, dev 26.5%
epoch 3, update 30.8%, dev 25.9%
epoch 4, update 30.3%, dev 25.1%
epoch 5, update 29.8%, dev 24.4%
epoch 6, update 29.7%, dev 24.2%
epoch 7, update 30.7%, dev 24.3%
epoch 8, update 30.0%, dev 24.8%
epoch 9, update 29.9%, dev 24.6%
epoch 10, update 30.2%, dev 24.6%
best dev err 24.2%, |w|=300, time: 0.1 secs

HW2 result:
best dev err 29.5, best dev_ave err 25.9, |w|=9974, time: 0.3 secs

It looks like there's an improvement in the error rate after pruning one-count words in the averaged perceptron. The best dev error rate is now 24.2%, compared to the HW2 result. Pruning one-count words likely helped improve the model's generalization by removing less informative or noisy features, resulting in a more effective representation. The reduction in error rate suggests that this preprocessing step contributed positively to the model's performance.

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. (1 pt)

- the film oozes craft (+)
    - Word2Vec captures the semantic relationships between "film" and "craft," understanding the positive connotation associated with the phrase "oozes craft." In contrast, one-hot encoding lacks the ability to capture semantic meaning and may treat individual words independently, missing the overall positive sentiment.
- get out your pooper scoopers (-)
    - Word2Vec recognizes the informal nature of the expression and associates it with a negative context, indicating a need for cleaning. One-hot encoding, lacking context awareness, may treat individual words independently and may not discern the informal tone, potentially misinterpreting the sentiment.

2.3 Summarize the error rates in a 2x2 table: {k-NN, perceptron} × {one-hot, embedding} (0.5 pts).

	ONE-HOT	EMBEDDING
K-NN	40.40%	27.66%
PERCEPTRON	25.9%	24.2%

3 Try some other learning algorithms with sklearn (1.5 pts)

1. Which algorithm did you try? What adaptations did you make to your code to make it work with that algorithm? (0.5 pts)

> I utilized the SVM algorithm for this task. The integration of word embeddings into the SVM framework was smooth, requiring minimal code adjustments. The primary modification involved specifying the radial basis function (rbf) kernel for the SVM to effectively handle the word embeddings.

2. What's the dev error rate(s) and running time? (0.5 pts)

> I got 23.6% dev error rate and running time is 6.5s

3. What did you learn in terms of the comparison between averaged perceptron and these other (presumably more popular and well-known) learning algorithms? (0.5 pts)

> The perceptron algorithm demonstrated notably faster performance compared to popular sklearn algorithms. Despite the perceptron's faster execution, it's noteworthy that the SVM achieved a better dev error rate compared to the perceptron.

4 Deployment (2.5 pts)

Q: What's your best error rate on dev, and which algorithm and setting achieved it? (0.25 pts)

> The best dev error rate achieved was 23.6%, and it was obtained using the SVM algorithm.

Debriefing

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

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

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

4. How deeply do you feel you understand the material it covers (0%–100%)?
   85%

5. Any other comments?
   N/A