AI 534 Machine Learning – HW2

Name: Nirmit Patel
Email: patenirm@oregonstate.edu

0 Sentiment Classification Task and Dataset (0.5 pts)

Question: why is each of these steps necessary or helpful for machine learning?

> The preprocessing steps, including lowercasing, punctuation splitting, etc. are essential for machine learning because they create a standardized and consistent text dataset. These steps help the model better understand and generalize from the text data, improving its performance.

1 Naive Perceptron Baseline (2 pts)

1. Take a look at svector.py, and briefly explain why it can support addition, subtraction, scalar product, dot product, and negation (0.5 pts).

> **Addition (__add__ and __iadd__):** svector overloads the + and += operators to allow for the addition of two sparse vectors. When you add two svector objects using +, it creates a new svector that combines the keys and values of both vectors. __iadd__ allows in-place addition by modifying the current vector.
>
> **Subtraction (__sub__ and __isub__):** svector overloads the - and -= operators to support subtraction of two sparse vectors. Subtraction is achieved by adding the negation of the second vector to the first vector. __isub__ allows in-place subtraction.
>
> **Scalar Product (__mul__ and __rmul__):** Multiplying a sparse vector by a scalar is supported using the * operator. It creates a new svector where each element of the original vector is multiplied by the scalar. __rmul__ supports scalar multiplication in the reverse order (scalar * vector).
>
> **Dot Product (dot method):** The dot method calculates the dot product between two sparse vectors. It iterates through the keys and values of both vectors, multiplying corresponding elements and summing up the results. The order of the vectors is chosen to minimize computation time.
>
> **Negation (__neg__):** The negation of a sparse vector is obtained using the - operator. It creates a new svector with negated values of the original vector.

2. Take a look at train.py, and briefly explain train() and test() functions (0.5 pts).

> **train():**
>
> - Trains a perceptron model using the training data.

- The training process iterates over the training data for the specified number of epochs. For each training example, it computes the dot product of the model's weights and the feature vector of the example. If this product is less than or equal to 0 and the label is positive, it updates the model's weights by adding the feature vector with a positive label.
- Evaluates the model on the dev data.
- Adjusts the model's weights based on the training data.
- Prints progress during training, including the best dev error.

test():

- Tests a given model on the dev data.
- It iterates over the dev data examples and calculates the error rate. It increments the error counter when the predicted label based on the model's dot product with the feature vector is incorrect (i.e., the product is less than or equal to 0).
- It returns the dev error rate, which is the percentage of misclassified examples in the dev data.

3. There is one thing missing in my train.py: the bias dimension! Try to add it. How did you do it? (Hint: by adding bias or <bias>?) Did it improve error on dev? (0.5 pts)

To add the bias term, I modified the make_vector function

```
def make_vector(words):
    v = svector()
    v['<bias>'] = 1  # Add the bias term with a value of 1
    for word in words:
        v[word] += 1
    return v
```

I used '<bias>' instead of 'bias' to avoid potential conflicts with existing words in the dataset. There is a word 'bias' in the dataset which can create errors in the model if used otherwise.

This modification improved the best dev error rate, reducing it from 30.1% to 28.9%.

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?? (0.5 pts)

Adding the bias dimension is essential even in balanced datasets. It enables the model to learn an intercept, ensuring it can effectively capture the decision boundary and converge to a better solution. While it's particularly important in unbalanced datasets, including the bias term is a good practice in most scenarios to enhance the model's ability to fit the data accurately.

2 Average Perceptron (5.5 pts)

1. Train for 10 epochs and report the results. Did averaging improve the dev error rate? (Hint: should be around 26%). Did it also make dev error rates more stable? (1.5 pts)

Following are the results for 10 epochs:

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: 0.6 secs

Yes, averaging improved the dev error rate, resulting in the best dev error of 26.3%. This modification also enhanced the stability of dev errors when compared to the vanilla perceptron.

2. Did smart averaging slow down training? (1 pt)

Smart averaging did not significantly slow down the training process when compared to the vanilla perceptron. In fact, it was noticeably faster than the naive average perceptron. Here's a comparison of the training times:

- Vanilla Perceptron: 0.5s
- Smart Average Perceptron: 0.6s
- Naive Average Perceptron: 66.8s

3. What are the top 20 most positive and top 20 most negative features? Do they make sense? (1 pt)

Here are the top 20 most positive and top 20 most negative features along with their corresponding weights:

Top 20 positive features		**Top 20 negative features**	
Feature	***Weight***	***Feature***	***Weight***
engrossing	1337627.0	boring	-1597973.0
triumph	1305738.0	generic	-1476776.0
unexpected	1241664.0	badly	-1368663.0
provides	1232912.0	dull	-1367881.0
rare	1231857.0	routine	-1339289.0

skin 1187300.0	ill -1261794.0
treat 1167944.0	fails -1211466.0
french 1155265.0	incoherent -1183230.0
smarter 1151195.0	inane -1111475.0
culture 1139283.0	too -1110680.0
refreshingly 1136154.0	instead -1101381.0
dots 1128579.0	tv -1096413.0
pulls 1124863.0	unless -1094366.0
cinema 1104206.0	flat -1085261.0
flaws 1103116.0	seagal -1084992.0
open 1079666.0	scattered -1079592.0
wonderful 1077079.0	neither -1078160.0
imax 1068125.0	results -1070090.0
delightful 1066127.0	attempts -1065508.0
proves 1061380.0	plodding -1059694.0

Both positive and negative features make sense. The positive features include words like "engrossing," "triumph," and "wonderful," which are typically associated with positive sentiments. Conversely, the negative features include words like "boring," "generic," and "dull," which are often associated with negative sentiments.

There is one notable exception in the list of positive features: "flaws" While the term "flaws" might typically be associated with negative sentiment, it appears among the top 20 positive features in this model. This observation highlights that the model may be interpreting the word "flaws" in a different context or considering it positively in the given dataset it was trained.

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? (2 pts)

5 negative examples in dev where model most strongly believes to be positive

- True Label: -1, Prediction: 3989108.0, Example: ` in this poor remake of such a well loved classic , parker exposes the limitations of his skill and the basic flaws in his vision '
- True Label: -1, Prediction: 3492266.0, Example: how much you are moved by the emotional tumult of fran ois and mich le 's relationship depends a lot on how interesting and likable you find them
- True Label: -1, Prediction: 3434348.0, Example: bravo reveals the true intent of her film by carefully selecting interview subjects who will construct a portrait of castro so predominantly charitable it can only be seen as propaganda
- True Label: -1, Prediction: 3059300.0, Example: mr wollter and ms seldhal give strong and convincing performances , but neither reaches into the deepest recesses of the character to unearth the quaking essence of passion , grief and fear
- True Label: -1, Prediction: 2543569.0, Example: an atonal estrogen opera that demonizes feminism while gifting the most sympathetic male of the piece with a nice vomit bath at his wedding

5 positive examples in dev where model most strongly believes to be negative

- True Label: 1, Prediction: -4551118.0, Example: neither the funniest film that eddie murphy nor robert de niro has ever made , showtime is nevertheless efficiently amusing for a good while before it collapses into exactly the kind of buddy cop comedy it set out to lampoon , anyway
- True Label: 1, Prediction: -4547888.0, Example: the thing about guys like evans is this you 're never quite sure where self promotion ends and the truth begins but as you watch the movie , you 're too interested to care
- True Label: 1, Prediction: -3339214.0, Example: if i have to choose between gorgeous animation and a lame story (like , say , treasure planet) or so so animation and an exciting , clever story with a batch of appealing characters , i 'll take the latter every time
- True Label: 1, Prediction: -3312722.0, Example: even before it builds up to its insanely staged ballroom scene , in which 3000 actors appear in full regalia , it 's waltzed itself into the art film pantheon
- True Label: 1, Prediction: -2231477.0, Example: carrying off a spot on scottish burr , duvall (also a producer) peels layers from this character that may well not have existed on paper

Observation:

English is a very versatile language, where the meaning of a sentence can be highly dependent on the specific words and context used. In many cases, a sentence can use positive words to convey a negative meaning and vice versa. While the model may recognize positive words within a sentence and predict a positive sentiment, humans can grasp the intended meaning by considering the context and the combination of words in the sentence.

Additionally, English often employs sarcasm and irony, which can completely flip the meaning of a statement. Our model is being trained for individual words rather than taking context from the other words and hence cannot properly understand and predict properly for this particular set of data.

3 Pruning the Vocabulary (2.5 pts)

1. Try neglecting one-count words in the training set during training. Did it improve the dev error rate? (Hint: should help a little bit, error rate lower than 26%). (1 pt)

> By removing 1-count words in the training set, the best dev error rate achieved was 26.0% for epoch=8. This is slightly better than the previous version where the best dev error was 26.3%.

2. Did your model size shrink, and by how much? (Hint: should almost halve). Does this shrinking help prevent overfitting? (0.25 pts)

> Yes, my model size did shrink from 15,806 to 8,425, which is approximately 53% of the original size.
>
> This reduction in model size helps prevent overfitting by making the model less complex, denser and more generalizable.

3. Did update % change? Does the change make sense? (0.25 pts)

> Yes, the update percentage changed as well. After removing 1-count words, the model required more updates compared to the previous version, indicating that the model's convergence speed decreased.
>
> This change makes sense because removing 1-count words makes the data more dense and reduces the number of features, which in turn affects the learning dynamics of the model. The increased updates are necessary for the model to adapt to the denser feature space and find an optimal decision boundary.

4. Did the training speed change? (0.25 pts)

> Yes, the training speed changed. The model trained faster with fewer epochs (epoch = 8).

5. What about further pruning two-count words (words that appear twice in the training set)? Did it further improve dev error rate? (0.75 pts)

> By pruning 2-count words, my best dev error increased. However, the fact that it achieved the best dev error for epoch = 4 indicates that the model is being trained more quickly and predicts well on unseen data. This suggests that the model has become more generalized. Nevertheless, it's important to note that this approach filters out some words that might be important in the training process.

4 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 used the SVM algorithm and adapted my code to read and preprocess data with pandas, filter n-count words using 'TfidfVectorizer' from sklearn.feature_extraction.text, and trained the model with the SVM classifier (svm.SVC) from sklearn.

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

Following are the dev error rate and running time after removing n-count words:
n = 0: dev err 26.8%, |w|=15747 time: 4.9 secs
n = 1: dev err 26.5%, |w|=8352 time: 4.5 secs
n = 2: dev err 26.9%, |w|=5856 time: 4.2 secs
n = 3: dev err 26.5%, |w|=4500 time: 3.9 secs
n = 4: dev err 27.5%, |w|=3621 time: 3.9 secs

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)

- Performance: The SVM achieved competitive results with a dev error rate around 26.5% to 26.9%, which is almost similar to the averaged perceptron.
- Training Time: The training times for SVM were significantly longer, taking around 3.9 to 4.9 seconds, whereas the averaged perceptron was much faster (0.6 seconds). This suggests that the averaged perceptron is more computationally efficient.
- Scalability: The averaged perceptron can be more suitable for tasks where efficiency and speed are critical, making it a scalable choice for large datasets. In contrast, SVMs may be more appropriate when computational resources are not a primary concern.

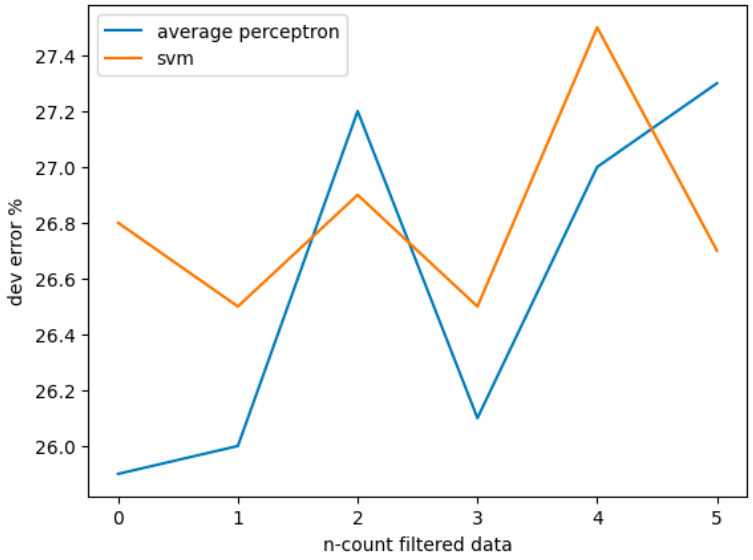
5 Deployment (3 pts)

I used the Average perceptron algorithm with the setting: epoch=8 and filter_n_count = 1, to do the predictions on blind dataset.
Using this setting I got following results on blind dataset:
Positive count: 492 Negative count: 508

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

My best dev error rate on the development set was 25% when I used the averaged perceptron algorithm with the setting of epoch=20 and without filtering any words. While this setting achieved a low error rate, it was computationally inefficient for large datasets.

For a more efficient approach, I achieved a best dev error of 26.0% by setting epoch=8 and filtering out one-count words. Another competitive result was 26.1% for epoch=9 when filtering out three-count words. These settings strike a balance between model performance and computational efficiency.



Debriefing

1. Approximately how many hours did you spend on this assignment?
   20 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%)?
   95%

5. Any other comments?
   It thought it would be difficult in the beginning but as the code was already provided and pseudo code in the exploration, the average perceptron implementation became very easy.