Project1_Report

February 2, 2024

1 Report for CS-165A Coding Project 1: Classifier Agent

1.0.1 Name: David Jr Sim

1.0.2 PERM #: 5416763

1.1 Declaration of Sources and Collaboration:

1.1.1 Collaboration:

- Special thanks to Jacob Eisner for going thorugh the gradient portion with me and helping me understand the process.

1.1.2 Sources:

- Wikipedia, "Cross-entropy," Wikipedia, https://en.wikipedia.org/wiki/Cross-entropy (accessed Jan. 22, 2024).
- D. Shah, "Cross entropy loss: Intro, applications, code," V7, https://www.v7labs.com/blog/cross-entropy-loss-guide (accessed Jan. 22, 2024).
- Python3, "Collections - container datatypes," Python documentation, https://docs.python.org/3/library/collections.html#collections.Counter (accessed Jan. 22, 2024).
- Python3, "5. Data Structures," Python documentation, https://docs.python.org/3/tutorial/datastructures.html (accessed Jan. 22, 2024).
- Gsitk, "Features," Gsitk documentation, https://gsi-upm.github.io/gsitk/features/ (accessed Jan. 30, 2024).
- Maas, Andrew L. and Daly, Raymond E. and Pham, Peter T. and Huang, Dan and Ng, Andrew Y. and Potts, Christopher, "Learning Word Vectors for Sentiment Analysis," Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, June 2011, Portland, Oregon, USA, Association for Computational Linguistics, pp. 142–150, http://www.aclweb.org/anthology/P11-1015 (accessed Jan. 30, 2024).
- Gensim, "Transformations," Gensim documentation, https://radimrehurek.com/gensim/auto_examples/cor (accessed Jan. 30, 2024).
- Scikit, "TfidfVectorizer," Scikit documentation, https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html (accessed Jan. 30, 2024).

1.2 Part 1: Gradient Calculations

The loss function to use is the cross-entropy loss, averaged over data points. We calculate the gradient with respect to the weights as that is what we want to change. The input (x) and label(y) are constants, so we can ignore them when calculating the gradient. The gradient is calculated as follows:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(w,(x_i,y_i))$$

$$\ell(w,(x_i,y_i)) = -(\log \hat{p}_w(x)y + \log(1-\hat{p}_w(x))(1-y))$$

$$\hat{p}_w(x) = \frac{e^{w^T x}}{1+e^{w^T x}}$$

$$\begin{aligned}\nabla L(w) &= \nabla \frac{1}{n}\sum_{i=1}^{n} \ell(w,(x_i,y_i)) \\ &= \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(w,(x_i,y_i))\end{aligned}$$

$$\nabla \ell(w,(x_i,y_i)) = \frac{\partial}{\partial w}[-(\ln(\frac{e^{w^T x}}{1+e^{w^T x}})y + \ln(1-\frac{e^{w^T x}}{1+e^{w^T x}})(1-y))]$$

$$\nabla \ell(w,(x_i,y_i)) = \frac{\partial}{\partial w}[-(1-y)\ln(1-\frac{e^{w^T x}}{1+e^{w^T x}}) - y\ln(\frac{e^{w^T x}}{1+e^{w^T x}})]$$

$$\nabla \ell(w,(x_i,y_i)) = (y-1)\frac{\partial}{\partial w}[\ln(1-\frac{e^{w^T x}}{1+e^{w^T x}})] - y\frac{\partial}{\partial w}[\ln(\frac{e^{w^T x}}{1+e^{w^T x}})]$$

$$\nabla \ell(w,(x_i,y_i)) = (y-1)\frac{1}{1-e^{w^T x}}\frac{\partial}{\partial w}[\frac{1}{1-e^{w^T x}}] - y\frac{e^{w^T x}+1}{e^{w^T x}}\frac{\partial}{\partial w}[\frac{e^{w^T x}+1}{e^{w^T x}}]$$

...

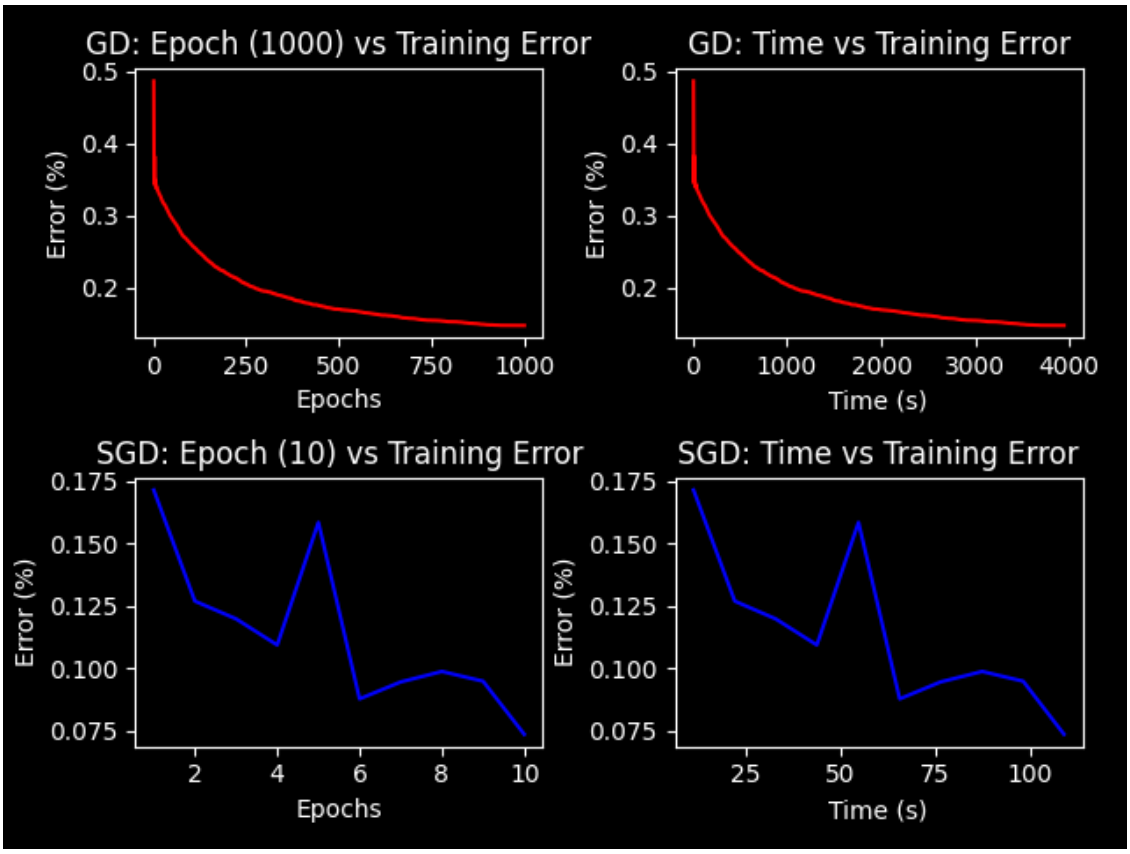
$$\nabla \ell(w,(x_i,y_i)) = -\frac{(y-1)\left(x\mathrm{e}^{w^T x}\cdot\left(\mathrm{e}^{w^T x}+1\right) - x\mathrm{e}^{2w^T x}\right)}{\left(\mathrm{e}^{w^T x}+1\right)^2\left(1-\frac{\mathrm{e}^{w^T x}}{\mathrm{e}^{w^T x}+1}\right)} - \frac{y\mathrm{e}^{-w^T x}\cdot\left(x\mathrm{e}^{w^T x}\cdot\left(\mathrm{e}^{w^T x}+1\right) - x\mathrm{e}^{2w^T x}\right)}{\mathrm{e}^{w^T x}+1}$$

$$\nabla \ell(w,(x_i,y_i)) = -\frac{x\cdot\left((y-1)\,\mathrm{e}^{w^T x}+y\right)}{\mathrm{e}^{w^T x}+1}$$

$$\nabla L(w) = \frac{1}{n}\sum_{i=1}^{n}[-\frac{x\cdot\left((y-1)\,\mathrm{e}^{w^T x}+y\right)}{\mathrm{e}^{w^T x}+1}]$$

1

1.3 Part 2: Gradient Descent vs Stochastic Gradient Descent



In the end, the graphs show that both methods have reached about the same error rate. However, the biggest difference is time. While SGD was a lot less inconsistent with its progress through each iteration, it took a lot less time to reach the same error rate as vanilla GD. On the other hand, vanilla was a lot more consistent with its progress towards the low error rate, however it too much longer to reach it. Based on the graph, SGD took about 2 minutes to reach its final error rate while vanilla GD took over an hour to achieve the same. It is evident, then, that the smoothness of progress gained by using the full gradient is not worth the time cost it takes compared to SGD.

1.4 Part 3: Apply the model to your own text

[18]:
```
from classifier import tokenize
from classifier import feature_extractor, classifier_agent
import numpy as np

# First load the classifier

with open('data/vocab.txt') as file:
    reading = file.readlines()
    vocab_list = [item.strip() for item in reading]
```

```
# By default this is doing the bag of words, change this into your custom␣
 ↪feature extractor
# so it works with your "best_model.npy"
feat_map = feature_extractor(vocab_list, tokenize)

d = len(vocab_list)
params = np.array([0.0 for i in range(d)])
custom_classifier = classifier_agent(feat_map, params)
custom_classifier.load_params_from_file('trained_params_sgd.npy')
```

[19]:
```
# Try it out!

my_sentence = "This movie is amazing! Truly a masterpiece."

my_sentence2 = "The book is really, really good. The movie is just dreadful."

ypred = custom_classifier.predict(my_sentence,RAW_TEXT=True)

ypred2 = custom_classifier.predict(my_sentence2,RAW_TEXT=True)

print(ypred,ypred2)
```

```
[1.] [0.]
```

1.4.1 We can also try predicting for each word in the input so as to get a sense of how the classifier arrived at the prediction

[21]:
```
import pandas as pd

# function for set text color of positive
# values in Dataframes
def color_predictions(val):
    eps = 0.02
    if isinstance(val,float):
        if val > eps:
            color = 'blue'
        elif val < -eps:
            color = 'red'
        else:
            color = 'black'
    else:
        color='black'
    return 'color: %s' % color

my_sentence_list = tokenize(my_sentence2)
ypred_per_word = custom_classifier.
 ↪predict(my_sentence_list,RAW_TEXT=True,RETURN_SCORE=True)
```

```
df = pd.DataFrame([my_sentence_list,ypred_per_word])

df.style.applymap(color_predictions)
```

```
/var/folders/1j/r963mnqs4wsdzdf0mj1qyvjm0000gn/T/ipykernel_34064/1404405345.py:2
3: FutureWarning: Styler.applymap has been deprecated. Use Styler.map instead.
  df.style.applymap(color_predictions)
```

[21]: <pandas.io.formats.style.Styler at 0x1212aef30>

1.4.2 Answer the questions:

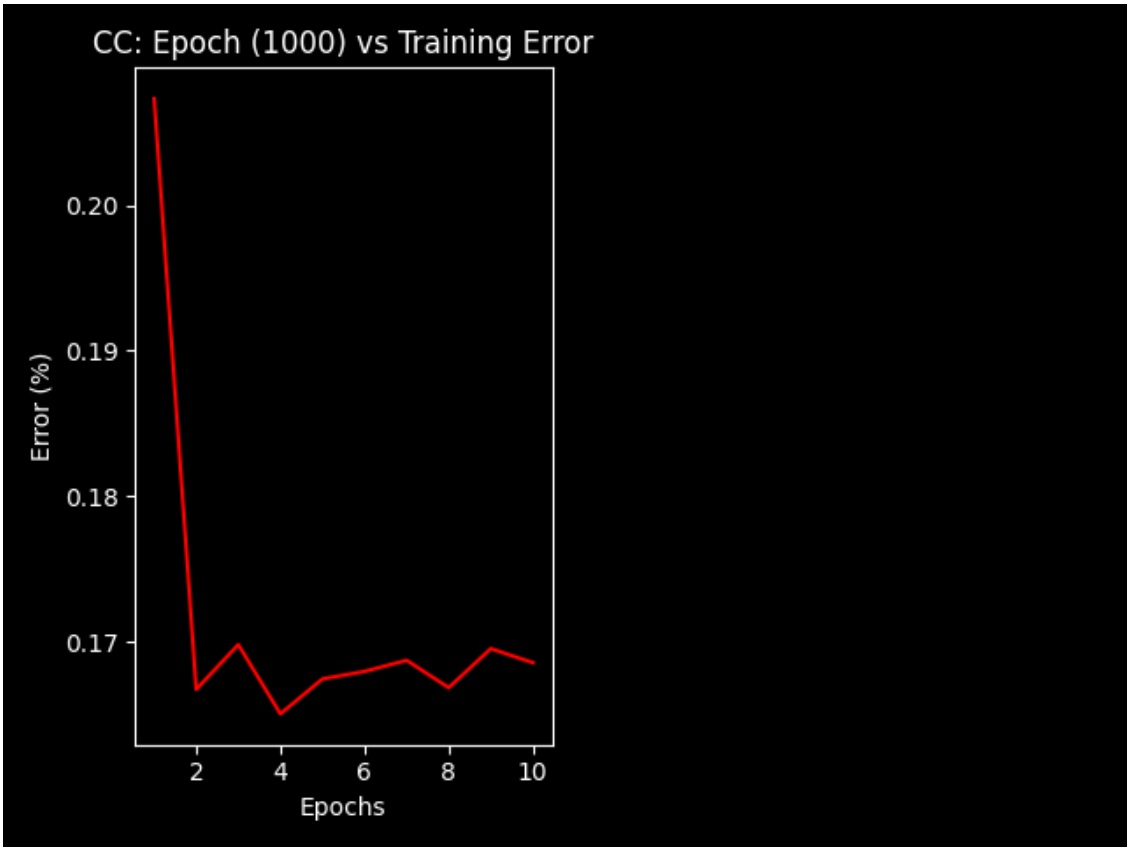
1. Are the above results making intuitive sense and why?
   - Yes, the results make intuitive sens. For the words that are just preopsitions are non-adjectives, they are given close to zero weight as both types of reviews will have these and they do not inform the agent on what kind of review this is. The words "good" and "movie" are both positively weighted and that makes sense. Being a positive adjective, "good" will, in most cases, imply that this review is saying something positive. On the other hand, the word "movie" is closer to a neutal word, which is why it is weighted less than the word "good". However, it is still slightly positive as it mentions the movie itself. Mirroring this, the words "book", "just", and "dreadful" all have negative weights. Ngative adjectives aside, the word "book" has a negative weight because it is talking about the foil of the movie and in many cases, people mention the book as a way to compare the movie to.
2. What are some limitation of a linear classifier with BoW features?
   - Each word is given a weight. However, this also means that each word is treated as an independent feature. This means that the classifier will not be able to take into account the context of the words. For example, the word "good" is given a positive weight. However, if the word "not" is placed before it, the classifier will not be able to take into account that the word "not" negates the word "good".
3. what are some ideas you can come up with to overcome these limitations (i.e., what are your ideas of constructing informative features)?
   - One way to overcome this limitation is to use n-grams. This way, the classifier will be able to take into account the context of the words. For example, the word "not" will be able to negate the word "good" if they are both in the same n-gram.

1.5 Part 4: Document what you did for custom feature extractors

Origionally, I looked into implementing a n-gram feature extractor but that did not improve by much and interger n values were not as comprehensive and flexible as I would have liked it to be. The improvement was less than 10% of the base bag-of-words model. Thus, I did not include its analysis in this report.

After, I decided to simply implement a Ti-idf vectorizer using `sklearn` on a new dataset. I am using a dataset provided by Maas et al. (2011) to train and test the accuracy of the model. I have pre-processed the data into a single file compared to seperate files in the origional dataset. The dataset also comes with a vocabulary file that we will use to train the model. The origional SGD `niter` increased exponentially after each training set, so it took a while to train.

The results below is a similar format as presented in *part 2*. The results are as follows:



Notice that this is somehow slightly worse than the normal hag of words from the origional assignment. I'm not sure if it's because of the dataset or because of the transformation, but it unfortunately was not as successful as I had hoped. I have tried (notice in the training section of `train_custom_model.py`) to adjust the training prameters with different step patterns and iterations to no avail. I tried running the origional classifier model on the same dataset and recieved worse results. This tells me that this new dataset is less correlated and more complex than the origional dataset.

1.6 Part 5: Anything else you'd like to write about. Your instructor / TA will read them.

I relearned that Python3 type hints are not as useful as I want them to. The toughest part was to keep track of the data types that I am working with and to try and not convert things into dense matricies when I don't need to. Otherwise, the implementation of the lab was intuitive enough from the material learned in lecture. I wouldn't say it was easy as there was a moment when everything had to click and be translated from theory in lecture into practice in the lab. The part that I worked on the longest was translating the abstract loss equation into concrete code.