DEVYAN BISWAS HOMEWORK 2 REPORT

- python version 3.7.5
- addtl reference: https://machinelearningmastery.com/pytorch-tutorial-develop-deep-learning-models/

```
import warnings
warnings.filterwarnings('ignore')
```

```
import pandas as pd
import numpy as np
import re
from bs4 import BeautifulSoup
import contractions
```

1. Dataset generation

```
df = pd.read_csv('./data.tsv', sep='\t', usecols =
['star_rating','review_body'], header=0)
```

```
df = df.loc[df['star_rating'].isin([5, 4, 3, 2, 1])]
```

Some basic data cleaning

```
df = df.dropna()
```

```
df
```

```
        star_rating                                        review_body
0                 5  so beautiful even tho clearly not high end ......
1                 5  Great product.. I got this set for my mother, ...
2                 5  Exactly as pictured and my daughter's friend l...
3                 5  Love it. Fits great. Super comfortable and nea...
4                 5  Got this as a Mother's Day gift for my Mom and...
...             ...                                                ...
1767046           4  It is nice looking and everything (it is sterl...
1767047           4  my boyfriend bought me this last christmas, an...
1767048           4  This is a great way to quickly start learning ...
1767049           5  the 14kt gold earrings look remarkable...would...
1767050           5  It will be a gift to my special friend. We kno...

[1701275 rows x 2 columns]
```

```
df['star_rating'] = df['star_rating'].astype(int)
# df['review_body'] = df['review_body'].astype(str)
```

```
df['review_body'] = df['review_body'].str.lower()
```

```
df['review_body'] = df['review_body'].str.replace(r'<[^<>]*>', '',
regex=True)
df['review_body']  = df['review_body'].str.replace(r's*https?://S+(s+|
$)', ' ').str.strip()
```

```
df['review_body'] = df['review_body'].astype(str)
df['review_body'] = df['review_body'].apply(lambda x:
contractions.fix(x))

df['review_body'] = df.review_body.str.replace('[^a-zA-Z\s]', ' ')

df['review_body'] = df['review_body'].replace(r'\s+', ' ', regex=True)

df['review_body'] = df['review_body'].replace('', np.nan)
df['review_body'] = df['review_body'].replace(' ', np.nan)
df['review_body'] = df['review_body'].replace('nan', np.nan)
df = df.dropna()

df
```

```
         star_rating
review_body
0                  5  so beautiful even though clearly not high
end ...
1                  5  great product i got this set for my mother
as ...
2                  5  exactly as pictured and my daughter s friend
l...
3                  5  love it fits great super comfortable and
neat ...
4                  5  got this as a mother s day gift for my mom
and...
...              ...                                             ..
.
1767046            4  it is nice looking and everything it is
sterli...
1767047            4  my boyfriend bought me this last christmas
and...
1767048            4  this is a great way to quickly start
learning ...
1767049            5  the kt gold earrings look remarkable would
def...
1767050            5  it will be a gift to my special friend we
know...

[1700118 rows x 2 columns]
```

Selecting the 20k of each rating type

```
star_5_df = df[df['star_rating'] == 5]
star_4_df = df[df['star_rating'] == 4]
star_3_df = df[df['star_rating'] == 3]
star_2_df = df[df['star_rating'] == 2]
star_1_df = df[df['star_rating'] == 1]

from sklearn.model_selection import train_test_split
```

```
# CHOOSING 20k random entries from each
# Seeding them so that data is more consistent
df_20_5 = star_5_df.sample(n=20000, random_state=100)
df_20_4 = star_4_df.sample(n=20000, random_state=100)
df_20_3 = star_3_df.sample(n=20000, random_state=100)
df_20_2 = star_2_df.sample(n=20000, random_state=100)
df_20_1 = star_1_df.sample(n=20000, random_state=100)

# Splitting them 16k and 4k to make new datasets for training and
testing
training_5 = df_20_5.iloc[:16000,:]
testing_5 = df_20_5.iloc[16000:,:]
training_4 = df_20_4.iloc[:16000,:]
testing_4 = df_20_4.iloc[16000:,:]
training_3 = df_20_3.iloc[:16000,:]
testing_3 = df_20_3.iloc[16000:,:]
training_2 = df_20_2.iloc[:16000,:]
testing_2 = df_20_2.iloc[16000:,:]
training_1 = df_20_1.iloc[:16000,:]
testing_1 = df_20_1.iloc[16000:,:]

# Merge all the ones above into one dataframe for training
# training_data = [training_5, training_4, training_3, training_2,
training_1]
training_data = pd.concat([training_5, training_4])
training_data = pd.concat([training_data, training_3])
training_data = pd.concat([training_data, training_2])
training_data = pd.concat([training_data, training_1])
training_data=training_data.reset_index(drop=True)

# Merge all the remaining ones above into one dataframe for testing
testing_data = pd.concat([testing_5, testing_4])
testing_data = pd.concat([testing_data, testing_3])
testing_data = pd.concat([testing_data, testing_2])
testing_data = pd.concat([testing_data, testing_1])
testing_data=testing_data.reset_index(drop=True)

whole_dataset = pd.concat([training_data, testing_data])
whole_dataset=whole_dataset.reset_index(drop=True)

whole_dataset
```

```
       star_rating                                        review_body
0                5  this is my third set purchased from them and i...
1                5             great little reminder just have faith
2                5  works well with black ceramic watch and other ...
3                5  great necklace the amount of bling is perfect ...
4                5  the glass beads are very pretty it did come a ...
...            ...                                                ...
99995            1      very cheap looking looks like costume jewelry
99996            1  the bracelet had a very musty smell that i cou...
```

```
99997            1                  i cannot get them in my piercing
99998            1                                      did not like
99999            1   this bracelet is crap the lock sucks complete a

[100000 rows x 2 columns]
```

```
only_reviews = whole_dataset['review_body']
only_reviews
```

```
0        this is my third set purchased from them and i...
1                   great little reminder just have faith
2        works well with black ceramic watch and other ...
3        great necklace the amount of bling is perfect ...
4        the glass beads are very pretty it did come a ...
                                ...
99995        very cheap looking looks like costume jewelry
99996    the bracelet had a very musty smell that i cou...
99997                   i cannot get them in my piercing
99998                                       did not like
99999     this bracelet is crap the lock sucks complete a
Name: review_body, Length: 100000, dtype: object
```

```
testing_data.to_csv('test.csv', index=False)
training_data.to_csv('train.csv', index=False)
whole_dataset.to_csv('whole_dataset.csv', index=False)
only_reviews.to_csv('only_reviews.csv', index=False, header=False)
```

FROM HERE ON OUT we do not need to run anything before; just gotta read from our friends test and train

2. Word Embedding

First, getting the google news word2vec model and testing it out

```
import gensim.downloader as gensim_api

wordvec = gensim_api.load('word2vec-google-news-300')

for index, word in enumerate(wordvec.index_to_key):
    if index == 10:
        break
    print(f"word #{index}/{len(wordvec.index_to_key)} is {word}")
```

```
word #0/3000000 is </s>
word #1/3000000 is in
word #2/3000000 is for
word #3/3000000 is that
word #4/3000000 is is
word #5/3000000 is on
word #6/3000000 is ##
word #7/3000000 is The
```

```
word #8/3000000 is with
word #9/3000000 is said
```

3 tests for semantic similarity and the like

```python
def np_cosine_sim(a, b):
    return np.dot(a, b)/(np.linalg.norm(a)*np.linalg.norm(b))

def e_dist(a,b):
    return np.linalg.norm(a - b)
```

```python
wordvec.most_similar(positive=['king', 'woman'], negative=['man'],
topn=5)
```

```
[('queen', 0.7118193507194519),
 ('monarch', 0.6189674139022827),
 ('princess', 0.5902431011199951),
 ('crown_prince', 0.5499460697174072),
 ('prince', 0.5377321839332581)]
```

```python
# EXAMPLE 1
necklace = wordvec['necklace']
neck = wordvec['neck']
wrist = wordvec['wrist']
bracelet = wordvec['bracelet']
anklet = wordvec['anklet']
vec_test = (necklace - neck) + wrist

#Euclidean distance comparison
edist = e_dist(vec_test, bracelet)
edist_t = e_dist(vec_test, anklet)
cosine_sim = np_cosine_sim(vec_test, bracelet)
cosine_sim_t = np_cosine_sim(vec_test, anklet)
print("Euclidean distance of test_vec and bracelet: ", edist, " vs
Euclidean dist of test_vec and anklet: ", edist_t)
print("Cosine Similarity of test_vec and bracelet: ", cosine_sim ," vs
cosine sim of test_vec and anklet: " , cosine_sim_t)
```

```
Euclidean distance of test_vec and bracelet:  3.9348783  vs Euclidean
dist of test_vec and anklet:  4.7187786
Cosine Similarity of test_vec and bracelet:  0.60587525  vs cosine sim
of test_vec and anklet:  0.4625539
```

```python
# EXAMPLE 2
daughter = wordvec['daughter']
son = wordvec['son']
brother = wordvec['brother']
boy = wordvec['boy']
girl = wordvec['girl']
vec_test = (daughter - girl) + boy
```

```
#Euclidean distance comparison
edist = e_dist(vec_test, son)
edist_t = e_dist(vec_test, brother)
cosine_sim = np_cosine_sim(vec_test, son)
cosine_sim_t = np_cosine_sim(vec_test, brother)
print("Euclidean distance of test_vec and son: ", edist, " vs 
Euclidean dist of test_vec and brother: ", edist_t)
print("Cosine Similarity of test_vec and son: ", cosine_sim ," vs 
cosine sim of test_vec and brother: " , cosine_sim_t)
```

```
Euclidean distance of test_vec and son:  1.1582639  vs Euclidean dist 
of test_vec and brother:  2.0276282
Cosine Similarity of test_vec and son:  0.91892457  vs cosine sim of 
test_vec and brother:  0.74743086
```

```
# EXAMPLE 3
ring = wordvec['ring']
finger = wordvec['finger']
ear = wordvec['ear']
earring = wordvec['earring']
necklace = wordvec['necklace']
piercing = wordvec['piercing']
vec_test = (ring - finger) + piercing + ear


edist = e_dist(vec_test, earring)
edist_t = e_dist(vec_test, necklace)
cosine_sim = np_cosine_sim(vec_test, earring)
cosine_sim_t = np_cosine_sim(vec_test, necklace)
print("Euclidean distance of test_vec and earring: ", edist, " vs 
Euclidean dist of test_vec and necklace: ", edist_t)
print("Cosine Similarity of test_vec and earring: ", cosine_sim ," vs 
cosine sim of test_vec and necklace: " , cosine_sim_t)
```

```
Euclidean distance of test_vec and earring:  5.6522703  vs Euclidean 
dist of test_vec and necklace:  5.812169
Cosine Similarity of test_vec and earring:  0.3556472  vs cosine sim 
of test_vec and necklace:  0.31888828
```

For the two examples above, each vector is closer/more similar to the vector it's intended to be for over another control vector. This shows the basic functionality of word2vec and its usefulness

Now, we want to train a model with our own dataset!

```
from gensim.test.utils import datapath
from gensim import utils
```

```
# FIX THIS: ONLY HAVE SENTENCES NOT THE OTHER NONSENSE BROH
class MyCorpus:
    """An iterator that yields sentences (lists of str)."""
```

```python
    def __iter__(self):
        corpus_path = 
datapath('/Users/devyanbiswas/Desktop/CSCI544/Homeworks/HW2/only_revie
ws.csv')
        for line in open(corpus_path):
            # print(line)
            # assume there's one document per line, tokens separated 
by whitespace
            yield utils.simple_preprocess(line)
```

```python
import gensim.models

sentences = MyCorpus()
model = gensim.models.Word2Vec(sentences=sentences, vector_size=300, 
window=11, min_count=10)
```

Let's test these out with our examples from above (yes it's copy and pasted but this is what i can do rn lol)

```python
# EXAMPLE 1
necklace = model.wv['necklace']
neck = model.wv['neck']
wrist = model.wv['wrist']
bracelet = model.wv['bracelet']
anklet = model.wv['anklet']
vec_test = (necklace - neck) + wrist

#Euclidean distance comparison
edist = e_dist(vec_test, bracelet)
edist_t = e_dist(vec_test, anklet)
cosine_sim = np_cosine_sim(vec_test, bracelet)
cosine_sim_t = np_cosine_sim(vec_test, anklet)
print("Euclidean distance of test_vec and bracelet: ", edist, " vs 
Euclidean dist of test_vec and anklet: ", edist_t)
print("Cosine Similarity of test_vec and bracelet: ", cosine_sim ," vs 
cosine sim of test_vec and anklet: " , cosine_sim_t)
```

```
Euclidean distance of test_vec and bracelet:  12.631125  vs Euclidean 
dist of test_vec and anklet:  19.925135
Cosine Similarity of test_vec and bracelet:  0.8182945  vs cosine sim 
of test_vec and anklet:  0.46293825
```

```python
# EXAMPLE 2
daughter = model.wv['daughter']
son = model.wv['son']
brother = model.wv['brother']
boy = model.wv['boy']
girl = model.wv['girl']
vec_test = (daughter - girl) + boy

#Euclidean distance comparison
```

```python
edist = e_dist(vec_test, son)
edist_t = e_dist(vec_test, brother)
cosine_sim = np_cosine_sim(vec_test, son)
cosine_sim_t = np_cosine_sim(vec_test, brother)
print("Euclidean distance of test_vec and son: ", edist, " vs Euclidean dist of test_vec and brother: ", edist_t)
print("Cosine Similarity of test_vec and son: ", cosine_sim ," vs cosine sim of test_vec and brother: " , cosine_sim_t)
```

```
Euclidean distance of test_vec and son:  17.356606  vs Euclidean dist of test_vec and brother:  16.919842
Cosine Similarity of test_vec and son:  0.45576268  vs cosine sim of test_vec and brother:  0.34291872
```

```python
# EXAMPLE 3
ring = model.wv['ring']
finger = model.wv['finger']
ear = model.wv['ear']
earring = model.wv['earring']
necklace = model.wv['necklace']
piercing = model.wv['piercing']
vec_test = (ring - finger) + piercing + ear


edist = e_dist(vec_test, earring)
edist_t = e_dist(vec_test, necklace)
cosine_sim = np_cosine_sim(vec_test, earring)
cosine_sim_t = np_cosine_sim(vec_test, necklace)
print("Euclidean distance of test_vec and earring: ", edist, " vs Euclidean dist of test_vec and necklace: ", edist_t)
print("Cosine Similarity of test_vec and earring: ", cosine_sim ," vs cosine sim of test_vec and necklace: " , cosine_sim_t)
```

```
Euclidean distance of test_vec and earring:  27.762281  vs Euclidean dist of test_vec and necklace:  38.68862
Cosine Similarity of test_vec and earring:  0.6088801  vs cosine sim of test_vec and necklace:  0.003107671
```

Cosime similarity scores are consistently better for the specific model, and euclidean distances are also more or less better, but there is an error in example 2 (the dist of the test_vec and brother is slightly closer than that of it and son). Other than that, it actually seems that in terms of semantic similarity, our specific model does better at modeling relationships given the difference in the two measures.

With that being said, let's move onto the simple models with training from Word2Vec embeddings.

3. Simple Models

So now, we're gonna train a perceptron and an SVM model on the pre-trained google W2V data First, let's develop the "average vector" for each review, which is described as: $\frac{1}{N}\sum_{i=1}^{N} W_i$ with N words in a review

```
# grabbing testing and training data, splitting them into appropriate
X and y pairs as well
train = pd.read_csv('./train.csv', header=0)
test = pd.read_csv('./test.csv', header=0)

train

       star_rating                                        review_body
0                5  this is my third set purchased from them and i...
1                5             great little reminder just have faith
2                5  works well with black ceramic watch and other ...
3                5  great necklace the amount of bling is perfect ...
4                5  the glass beads are very pretty it did come a ...
...            ...                                                ...
79995            1  wore it once still covered in an itchy red ras...
79996            1  this ring arrived in timely fashion which i li...
79997            1  poorly made and combined good thing is i am cr...
79998            1                        this came late and broken
79999            1  i was happy to find these and was going to wea...

[80000 rows x 2 columns]

def meanEmbeddings(model, sentence):
        words = sentence.split()
        # remove out-of-vocabulary words
        words = [word for word in words if word in model]
        if len(words) >= 1:
            return np.mean(model[words], axis=0)
        else:
            return []

train['review_body'] = train.apply(lambda x: meanEmbeddings(wordvec,
x['review_body']), axis=1)
test['review_body'] = test.apply(lambda x: meanEmbeddings(wordvec,
x['review_body']), axis=1)
```

There's an issue: some embeddings are just empty. So for now, instead of doing the training/testing data split here so that we have the full 80k/20k split, im gonna just drop empty vectors. TODO: Do the splitting *AFTER* you drop empty list embeddings

```
train = train[train['review_body'].map(lambda d: len(d)) > 0]
test = test[test['review_body'].map(lambda d: len(d)) > 0]
```

```
train=train.reset_index(drop=True)
test=test.reset_index(drop=True)
```

```
train
```

```
       star_rating                                        review_body
0                5  [0.021781074, 0.010730558, 0.03842163, 0.09902...
1                5  [0.0777181, 0.0241038, -0.003133138, 0.1139322...
2                5  [0.03963216, 0.0152542675, 0.009416086, 0.0662...
3                5  [0.0351429, 0.040046692, 0.016231537, 0.068573...
4                5  [0.022198932, 0.0065338784, -0.010754097, 0.10...
...            ...                                                ...
79950            1  [0.040590923, 0.019532362, 0.06068675, 0.08143...
79951            1  [0.008141665, 0.013809791, 0.024548898, 0.0918...
79952            1  [0.01965768, -0.007548264, 0.051713128, 0.0990...
79953            1  [0.049072266, 0.12234497, 0.024169922, 0.03753...
79954            1  [0.041787915, 0.051063508, 0.029582731, 0.1004...

[79955 rows x 2 columns]
```

```
test
```

```
       star_rating                                        review_body
0                5  [0.018040033, 0.043087665, 0.03163665, 0.12524...
1                5  [0.015490723, 0.08129883, 0.031079102, 0.10883...
2                5  [0.01468811, -0.040407658, -0.012677002, 0.150...
3                5  [-0.032828193, 0.000926154, 0.06468855, 0.0201...
4                5  [0.02191816, 0.023200626, 0.04187157, 0.096947...
...            ...                                                ...
19983            1  [0.011849539, 0.0987636, -0.065767564, 0.11478...
19984            1  [0.016875131, -0.010713373, 0.034745354, 0.113...
19985            1  [0.026611328, 0.039376397, 0.057128906, 0.1531...
19986            1  [0.1295573, 0.065592445, 0.07306417, 0.1007080...
19987            1  [0.018993378, -0.04724121, 0.009490967, 0.1369...

[19988 rows x 2 columns]
```

```
train_X = pd.DataFrame(train['review_body'].to_list())
train_y = train['star_rating']

test_X = pd.DataFrame(test['review_body'].to_list())
test_y = test['star_rating']
```

Perceptron Training and Metrics

- Gonna compare these to HW1 metrics, which will be imported in

```
from sklearn.linear_model import Perceptron
perc = Perceptron()
perc.fit(train_X, train_y)
```

```
Perceptron()
```

```python
# Testing and score calcs
from sklearn.metrics import recall_score, precision_score, f1_score, 
accuracy_score

perc_y_pred = perc.predict(test_X)

print("METRICS FOR PERCEPTRON")
print("======================")

# Averages
recall_avg = recall_score(test_y, perc_y_pred, average='macro')
accuracy_avg = accuracy_score(test_y, perc_y_pred)
precision_avg = precision_score(test_y, perc_y_pred, average='macro')
f1_avg = f1_score(test_y, perc_y_pred, average='macro')

print("Recall Avg: ", recall_avg)
print("Accuracy Avg: ", accuracy_avg)
print("HW1 Accuracy: ", 0.41665)
print("Precision Avg: ", precision_avg)
print("F1 Avg: ", f1_avg)
```

```
METRICS FOR PERCEPTRON
======================
Recall Avg:  0.40106992135312886
Accuracy Avg:  0.400990594356614
HW1 Accuracy:  0.41665
Precision Avg:  0.4499730442232609
F1 Avg:  0.3678551740823453
```

```python
from sklearn.svm import LinearSVC
lin_svc = LinearSVC(max_iter=2000)
lin_svc.fit(train_X, train_y)
```

```
LinearSVC(max_iter=2000)
```

```python
svc_pred = lin_svc.predict(test_X)

print("METRICS FOR SVC")
print("======================")

# Averages
recall_avg = recall_score(test_y, svc_pred, average='macro')
accuracy_avg = accuracy_score(test_y, svc_pred)
precision_avg = precision_score(test_y, svc_pred, average='macro')
f1_avg = f1_score(test_y, svc_pred, average='macro')

print("Recall Avg: ", recall_avg)
print("Accuracy Avg: ", accuracy_avg)
print("HW1 Accuracy: ", 0.4849)
```

```
print("Precision Avg: ", precision_avg)
print("F1 Avg: ", f1_avg)
```

```
METRICS FOR SVC
======================
Recall Avg:  0.48418447370576717
Accuracy Avg:  0.48409045427256353
HW1 Accuracy:  0.4849
Precision Avg:  0.46487209033571564
F1 Avg:  0.4646735269946672
```

Seemingly the two different input processing methodologies do not create that much of a difference. My guess is that the feature embeddings generated from Word2Vec aren't differentiable/distinct enough (see the comparison between our specific embedding model from our corpus vs the word2vec one) to create a descriptive enough emebdding on which to train our data. I would definetely like to try these out with our own home-brewed embedding

4. Feedforward Neural Networks

Part a: good ol fashioned input from above and training an MLP on dat

- reference: https://www.kaggle.com/code/mishra1993/pytorch-multi-layer-perceptron-mnist/notebook

```
# Imports ofc
import torch
from torch.utils.data import DataLoader, Dataset
import torchvision
import torchvision.transforms as transforms
from torch.utils.data.sampler import SubsetRandomSampler
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from tqdm import tqdm
```

I realize that I need to do some data transformations here to make it work, so I retroactively did some mods to the data

Let's expand those columns and write it out to a file WITH THE RATINGS for testing and training

```
temp_train = pd.DataFrame(train['review_body'].to_list())
temp_train['star_rating'] = train['star_rating']

temp_test = pd.DataFrame(test['review_body'].to_list())
temp_test['star_rating'] = test['star_rating']

full_expanded_data = pd.concat([temp_train, temp_test])
full_expanded_data=full_expanded_data.reset_index(drop=True)
```

Let's write these to files for reading purposes for next steps.

```
full_expanded_data.to_csv('full_expanded_data.csv', index=False)
```

OK, now for 4a, we're gonna build up the MLP for Multiclass classification

```
# imports, ofc
from numpy import vstack
from numpy import argmax
import pandas as pd
# from pandas import read_csv
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from torch import Tensor
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from torch.utils.data import random_split
from torch.nn import Linear
from torch.nn import ReLU
from torch.nn import Softmax
from torch.nn import Module
from torch.optim import SGD
from torch.nn import CrossEntropyLoss
from torch.nn.init import kaiming_uniform_
from torch.nn.init import xavier_uniform_

# dataset definition
class readinData(Dataset):
    # load the dataset
    def __init__(self, path):
        df = pd.read_csv(path, skiprows=1, header=None)
        self.X = df.values[:, :-1]
        self.y = df.values[:, -1] - 1
        self.X = self.X.astype('float32')
        self.y = self.y.astype('int')
        self.y = LabelEncoder().fit_transform(self.y)

    # number of rows in the dataset
    def __len__(self):
        return len(self.X)

    # get a row at an index
    def __getitem__(self, idx):
        return [self.X[idx], self.y[idx]]

    # get indexes for train and test rows
    def split_data(self, n_test=0.20):
        # determine sizes
        test_size = round(n_test * len(self.X))
        train_size = len(self.X) - test_size
```

```python
        # calculate the split
        return random_split(self, [train_size, test_size])

class MLP(Module):
    def __init__(self, input_size):
        super(MLP, self).__init__()

        self.input_layer = Linear(input_size, 50)
        kaiming_uniform_(self.input_layer.weight, nonlinearity='relu')
        self.input_activation = ReLU()

        self.inner_layer = Linear(50, 10)
        kaiming_uniform_(self.inner_layer.weight, nonlinearity='relu')
        self.inner_activation = ReLU()

        self.output_layer = Linear(10, 5)
        xavier_uniform_(self.output_layer.weight)
        self.output_activation = Softmax(dim=1)

    def forward(self, X):
        X = self.input_layer(X)
        X = self.input_activation(X)

        X = self.inner_layer(X)
        X = self.inner_activation(X)

        X = self.output_layer(X)
        X = self.output_activation(X)
        return X

def prepare_data(path):
    data_set = readinData(path)
    train, test = data_set.split_data()

    train_dl = DataLoader(train, batch_size=32, shuffle=True)
    test_dl = DataLoader(test, batch_size=1024, shuffle=False)
    return train_dl, test_dl

def train_model(train_dl, model):
    criterion = CrossEntropyLoss()
    optimizer = SGD(model.parameters(), lr=0.01, momentum=0.9)

    for epoch in range(50):
        for i, (xs, y_targets) in enumerate(train_dl):
            optimizer.zero_grad()

            y_med_pred = model(xs)
            loss = criterion(y_med_pred, y_targets)
            loss.backward()
```

```
            optimizer.step()

def evaluate_model(test_dl, model):
    predictions = list()
    actuals = list()
    for i, (xs, y_targets) in enumerate(test_dl):
        y_preds = model(xs)

        y_preds = y_preds.detach().numpy()
        actual = y_targets.numpy()

        y_preds = argmax(y_preds, axis=1)

        actual = actual.reshape((len(actual), 1))
        y_preds = y_preds.reshape((len(y_preds), 1))

        predictions.append(y_preds)
        actuals.append(actual)

    predictions, actuals = vstack(predictions), vstack(actuals)
    acc = accuracy_score(actuals, predictions)
    return acc, actuals, predictions

def predict(row, model):
    row = Tensor([row])
    y_pred = model(row)
    y_pred = y_pred.detach().numpy()
    return y_pred

path = "./full_expanded_data.csv"
train_dl, test_dl = prepare_data(path)

print(len(train_dl.dataset), len(test_dl.dataset))

79954 19989

model = MLP(300)

train_model(train_dl, model)

acc, actuals, predictions  = evaluate_model(test_dl, model)
print('Accuracy: %.3f' % acc)

Accuracy: 0.494

from sklearn.metrics import confusion_matrix
matrix = confusion_matrix(actuals, predictions)
acc_array = matrix.diagonal()/matrix.sum(axis=1)
for idx, acc_val in enumerate(acc_array):
    print("Class", idx+1, "accuracy:", acc_val)
```

```
Class 1 accuracy: 0.7286762844849405
Class 2 accuracy: 0.3472117083017916
Class 3 accuracy: 0.2662192393736018
Class 4 accuracy: 0.332245102963335
Class 5 accuracy: 0.7904176904176904
```

Not too bad for accuracy; some things to improve would be higher epochs and potentially better weight initializations to allow for faster convergence, and ofc more training examples in other categories. But we can move onto the next example now.

4b

Read in the train and test from before, concat them, and reset indices just to get the full, pre-converted string reviews

```
train = pd.read_csv('./train.csv', header=0)
test = pd.read_csv('./test.csv', header=0)
```

Concat the vectors to make a 3000 column monstrosity. and yes, pad with 0's apparently

```
def concatEmbeddings(model, sentence):
        words = sentence.split()
        words = [word for word in words if word in model]
        return_vec = list()
        counter = 0

        if len(words) >= 1:
            emb_words = model[words]
            for idx, word in enumerate(emb_words):
                if idx == 10:
                    break
                return_vec = return_vec + word.tolist()
                counter = idx
            padding = [0] * (3000 - len(return_vec))
            return_vec = return_vec + padding
            return return_vec
        else:
            return []

full_ = pd.concat([train, test])
full_ =full_.reset_index(drop=True)
full_['review_body'] = full_.apply(lambda x: concatEmbeddings(wordvec,
x['review_body']), axis=1)
```

Again, same issue as before where there's gonna be empty values, but that's a minor issue I'll take a look at if time permits

```
full_ = full_[full_['review_body'].map(lambda d: len(d)) > 0]

full_=full_.reset_index(drop=True)
```

```python
expanded_thiq = pd.DataFrame(full_['review_body'].to_list())
expanded_thiq['star_rating'] = full_['star_rating']

expanded_thiq
```

```
              0         1         2         3         4         5
6  \
0      0.109375  0.140625 -0.031738  0.166016 -0.071289  0.015869 -
0.003113
1      0.071777  0.208008 -0.028442  0.178711  0.132812 -0.099609
0.096191
2      0.044189  0.106934  0.097168 -0.006042 -0.078125  0.212891
0.160156
3      0.071777  0.208008 -0.028442  0.178711  0.132812 -0.099609
0.096191
4      0.080078  0.104980  0.049805  0.053467 -0.067383 -0.120605
0.035156
...         ...       ...       ...       ...       ...       ...
...
99938  0.016602  0.045654 -0.119141  0.069824 -0.143555  0.104980 -
0.030151
99939  0.080078  0.104980  0.049805  0.053467 -0.067383 -0.120605
0.035156
99940 -0.225586 -0.019531  0.090820  0.237305 -0.029297  0.093262 -
0.058838
99941  0.200195  0.154297  0.103027  0.008667  0.001183 -0.162109
0.023438
99942  0.109375  0.140625 -0.031738  0.166016 -0.071289  0.015869 -
0.003113

              7         8         9  ...      2991      2992      2993
\
0     -0.084961 -0.048584  0.055664  ...  0.112305 -0.041016  0.093262

1     -0.116699 -0.008545  0.148438  ...  0.000000  0.000000  0.000000

2     -0.092773 -0.103027 -0.130859  ... -0.022827 -0.060547 -0.045166

3     -0.116699 -0.008545  0.148438  ... -0.021729 -0.066406  0.055420

4     -0.118652  0.043945  0.030151  ...  0.134766 -0.003601  0.079590

...         ...       ...       ...  ...       ...       ...       ...

99938 -0.119141  0.126953 -0.005981  ...  0.000000  0.000000  0.000000

99939 -0.118652  0.043945  0.030151  ...  0.034424 -0.005707 -0.033203

99940 -0.041016  0.052246  0.020020  ...  0.000000  0.000000  0.000000
```

```
99941 -0.124512  0.034180 -0.142578  ...  0.000000  0.000000  0.000000

99942 -0.084961 -0.048584  0.055664  ...  0.000000  0.000000  0.000000


           2994      2995      2996      2997      2998      2999
star_rating
0     -0.202148 -0.145508  0.082520  0.131836  0.181641 -0.093750
5
1      0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
5
2      0.046143 -0.051758 -0.049072 -0.046875  0.161133 -0.199219
5
3     -0.102051 -0.021729  0.149414 -0.171875 -0.029297 -0.206055
5
4     -0.052979 -0.133789  0.194336  0.112305 -0.052490 -0.016357
5
...         ...       ...       ...       ...       ...       ...
...
99938  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
1
99939 -0.000938  0.031982  0.063477 -0.108887  0.048828 -0.130859
1
99940  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
1
99941  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
1
99942  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
1

[99943 rows x 3001 columns]
```

```
expanded_thiq.to_csv('expanded_thiq.csv', index=False)
```

Gonna read in the data now, then go from there. could also have just...like...not done this and just...used the damn data frame...but fuck it lol

```
path = "./expanded_thiq.csv"
exp_train_dl, exp_test_dl = prepare_data(path)
```

```
print(len(exp_train_dl.dataset), len(exp_test_dl.dataset))
```

```
79954 19989
```

```
exp_model = MLP(3000)
```

```
train_model(exp_train_dl, exp_model)
```

```
exp_acc, exp_actual, exp_preds = evaluate_model(exp_test_dl,
exp_model)
print('Accuracy: %.3f' % exp_acc)
```

```
Accuracy: 0.415
```

```
from sklearn.metrics import confusion_matrix
matrix = confusion_matrix(exp_actual, exp_preds)
acc_array = matrix.diagonal()/matrix.sum(axis=1)
for idx, acc_val in enumerate(acc_array):
    print("Class", idx+1, "accuracy:", acc_val)
```

```
Class 1 accuracy: 0.5233714569865738
Class 2 accuracy: 0.3436188595408541
Class 3 accuracy: 0.31132554596241746
Class 4 accuracy: 0.33208582834331335
Class 5 accuracy: 0.565743073047859
```

Accuracies are either the same or slightly better. However, the feed-forward neural net approach, while being slightly more difficult to code up, has the most room for fine-tuning regarding hyperparameters. Slight tweaks to epochs or batch size created massive differences in testing accuracy. Given the right model for feature embedding generation, I think the accuracy could really go up from here.

RNN part

5a

Our favorite place to start: dataproc

```
train_dset = pd.read_csv('./train.csv', header=0)
test_dset = pd.read_csv('./test.csv', header=0)
```

ok, we've got our data boi back, with just the reviews as words. I believe the idea will be to generate a 20 x 1 x 300 tensor, where 20 is the number of words from a review we will be getting, 1 is the batch size, and 300 is the dimension of each word embedding vector.

So first, a function that gets the first 20 words of the review and convert them into tensors of words embeddings

```
def convert_review_to_tensor(sentence, model):
    words = sentence.split()
    words = [word for word in words if word in model]
    tensor_20 = torch.zeros(20, 1, 300)

    for idx, word in enumerate(words):
        if idx == 20:
            break
        tensor_20[idx][0] = torch.from_numpy(wordvec[word])

    return tensor_20
```

Ok, we have a pandas dataframe with 2 columns now; star rating that is an int and a review that is a 20x1x300 tensor

Let's take an aside to actually define the RNN...

```
import torch.nn as nn

class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()

        self.hidden_size = hidden_size

        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(input_size + hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        combined = torch.cat((input, hidden), 1)
        hidden = self.i2h(combined)
        output = self.i2o(combined)
        output = self.softmax(output)
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, self.hidden_size)

n_hidden = 20
rnn = RNN(300, n_hidden, 5)
```

Getting a random sample...

```
all_categories = [1,2,3,4,5]

def randomTrainingExample(df):
    row = df.sample().values.tolist()
    rating = row[0][0]
    review = row[0][1]
    review_tensor = convert_review_to_tensor(review, wordvec)
    rating_tensor = torch.tensor([all_categories.index(rating)],
dtype=torch.long)
    return rating_tensor, review_tensor, review, rating

def categoryFromOutput(output):
    top_n, top_i = output.topk(1)
    category_i = top_i[0].item()
    return all_categories[category_i], category_i
```

how to train now...

```
criterion = nn.NLLLoss()
learning_rate = 0.002

def train(category, line_tensor):
    hidden = rnn.initHidden()
```

```
    rnn.zero_grad()

    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)

    loss = criterion(output, category)
    loss.backward()

    # Add parameters' gradients to their values, multiplied by
learning rate
    for p in rnn.parameters():
        p.data.add_(p.grad.data, alpha=-learning_rate)

    return output, loss.item()
```

test out our training...

```
import time
import math

n_iters = 100000
print_every = 5000

current_loss = 0
all_losses = []

for iter in range(1, n_iters + 1):
    category_tensor, review_tensor, review, rating  =
randomTrainingExample(train_dset)
    # print(category, review_tensor.size())
    output, loss = train(category_tensor, review_tensor)
    # print(output)
    guess, guess_i = categoryFromOutput(output)

    if iter % print_every == 0:
        guess, guess_i = categoryFromOutput(output)
        correct = '✓' if guess == rating else '✗ (%s)' % rating
        print('%d %d%%  %.4f %s / %s %s' % (iter, iter / n_iters *
100, loss, review, guess, correct))
```

```
5000 5%  1.6398 i never regret buying anything from amazon this ring
is amazing so beautiful  / 5 ✗ (4)
10000 10%  1.6297 terrible ankle bracelet broke immediately  / 5 ✗ (1)
15000 15%  1.6703 sent this ring back i thought it was a real diamond
the mark down was and some odd dollars for cubic zirconia it was white
gold but not worth that much sorry / 4 ✗ (1)
20000 20%  1.6382 i have a double conch piercing and have been looking
for rings that would fit around my ear these are wonderfully made
durable comfortable and the seams for the closure are not visible they
```

are droopy but i ended up loving that look even though i was super against it at first / 1 ✗ (5)
25000 25% 1.6590 the price was little less than most alex and ani bracelets and for good reason when i received the bracelet the quality of the metal was garbarge it looked like raw brushed steel and to top it off the lock has a chip out of the bottom of the heart the packaging was fine but the quality of the metal and the bite taking out of the heart were my biggest complains save your money / 5 ✗ (2)
30000 30% 2.0254 my fiance and i wanted to have a low budget wedding since were so young and plan on having a real one a few years down the line so i wanted a beautiful but inexpensive ring with his birthstone on it rather than a diamond this one is perfect i have worn it every day for the past months and the peridot is still shinning brilliantly not a single gem has fallen out a problem i have had with other low cost rings including the very small ones on the sides of the ring i love this ring and i highly recommend it to anyone who is interested / 2 ✗ (5)
35000 35% 1.4533 beautiful earrings lots of compliments if you care about that sort of thing / 5 ✓
40000 40% 1.6679 this is a very pretty sterling silver ring it looks nice on the hand and i have been wearing it for a while now and it still looks shiny and new / 2 ✗ (5)
45000 45% 1.3307 product looked fine from the front but the backside showed a fatal flaw which caused the stone to break off when attempting to wire wrap / 1 ✗ (2)
50000 50% 1.5878 broke in weeks / 5 ✗ (1)
55000 55% 1.7095 thick than i expected but ok / 5 ✗ (3)
60000 60% 2.1458 it looks thin in the picture was not expecting it to be so wide but i love it anyways thank you / 1 ✗ (4)
65000 65% 1.5698 these earrings are much prettier in person than on screen they shine and appear far more expensive than they are until you get to the clasp it is a little crude and is fragile when using i had to return one pair due to clasp failure but the earrings are so goodlooking that i took a chance on another pair i am glad i did even though the clasp leaves something to be desired / 3 ✗ (4)
70000 70% 0.8757 i love amber and will buy a piece with a stylist look and well made design these earring fit the bill the color is warm and matches other amber pieces the design is solid along with easy clasp one earring loosened up and i had to attach to the clasp but it is fine and i wear them about once a week i would recommend / 4 ✓
75000 75% 1.2152 i love this item i just wished th chain was a little thicker i had been looking for a item like this for some time very beautiful item http www amazon com gp product b f uee ref cm cr rev prod title / 4 ✓
80000 80% 1.4467 this bracelet came with scratches and has continued to lose it is finish as i have gently worn it also while i have gained benefit or my minor arthritis from other titanium bracelets call me crazy the phiten stuff works for me and so do some other titanium things i have received no such benefit from this bracelet if it had not scratched further from casual wear i would return it i will say

```
that the bracelet appears to be reasonably sturdy and the clasp works
well that is what got it the two stars / 3 ✗ (2)
85000 85%  1.2018 straps to hold necklaces broke st time i tried to
use it outside is great need to fix the cheap snaps and straps to hold
necklaces and this would be a great items outside looks great but the
inside snaps and straps for necklaces are a joke  / 2 ✓
90000 90%  1.4841 forget the rave reviews you get what you pay for ok
for i guess / 1 ✗ (3)
95000 95%  1.2399 thought they would be a little bigger but i just
love them / 5 ✓
100000 100%  1.3168 the flower i received was bigger than i would like
it to be i think when they are too big it looks rather gaudy  / 4 ✗
(3)
```

```python
def evaluate(review_tensor):
    hidden = rnn.initHidden()

    for i in range(review_tensor.size()[0]):
        output, hidden = rnn(review_tensor[i], hidden)

    return output

def rnn_predict(input_line, n_preds=1):

    line = input_line.tolist()[0]
    with torch.no_grad():
        output = evaluate(convert_review_to_tensor(line, wordvec))

        topv, topi = output.topk(n_preds, 1, True)
        predictions = []

        for i in range(n_preds):
            value = topv[0][i].item()
            category_index = topi[0][i].item()
            predictions.append([value,
all_categories[category_index]])

        return all_categories[category_index]

real_preds = test_dset['star_rating']
data_lines = pd.DataFrame(test_dset['review_body'])

data_lines['review_body'] = data_lines.apply(lambda x: rnn_predict(x,
n_preds=1), axis=1)
print(data_lines)
```

```
       review_body
0                4
1                5
2                5
```

```
3                5
4                5
...            ...
19995            3
19996            1
19997            5
19998            3
19999            1

[20000 rows x 1 columns]
```

```
from sklearn.metrics import accuracy_score
test_preds = data_lines['review_body']
acc = accuracy_score(real_preds, test_preds)
print("Avg acc: ", acc)
```

```
Avg acc:  0.332
```

```
from sklearn.metrics import confusion_matrix
matrix = confusion_matrix(real_preds, test_preds)
acc_array = matrix.diagonal()/matrix.sum(axis=1)
for idx, acc_val in enumerate(acc_array):
    print("Class", idx+1, "accuracy:", acc_val)
```

```
Class 1 accuracy: 0.36625
Class 2 accuracy: 0.07425
Class 3 accuracy: 0.22275
Class 4 accuracy: 0.10825
Class 5 accuracy: 0.8885
```

Woah! Accuracy is...alright...but hell it's better than 0 I guess?

5b

- Now, let's try a gated RNN?

```
import torch.nn as nn


class GRNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(GRNN, self).__init__()

        self.hidden_size = hidden_size

        self.i2h = nn.GRUCell(input_size + hidden_size, hidden_size)
        self.i2o = nn.GRUCell(input_size + hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        combined = torch.cat((input, hidden), 1)
        hidden = self.i2h(combined)
```

```
        output = self.i2o(combined)
        output = self.softmax(output)
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, self.hidden_size)

n_hidden = 20
grnn = GRNN(300, n_hidden, 5)
```

train da grnn

```
def grnn_train(category, line_tensor):
    hidden = grnn.initHidden()

    grnn.zero_grad()

    for i in range(line_tensor.size()[0]):
        output, hidden = grnn(line_tensor[i], hidden)

    loss = criterion(output, category)
    loss.backward()

    for p in grnn.parameters():
        p.data.add_(p.grad.data, alpha=-learning_rate)

    return output, loss.item()

import time
import math

n_iters = 100000
print_every = 5000

current_loss = 0
all_losses = []

for iter in range(1, n_iters + 1):
    category_tensor, review_tensor, review, rating  =
randomTrainingExample(train_dset)
    output, loss = grnn_train(category_tensor, review_tensor)
    guess, guess_i = categoryFromOutput(output)

    if iter % print_every == 0:
        guess, guess_i = categoryFromOutput(output)
        correct = '✓' if guess == rating else '✗ (%s)' % rating
        print('%d %d%%  %.4f %s / %s %s' % (iter, iter / n_iters *
100, loss, review, guess, correct))
```

5000 5% 1.5079 just as expected / 5 ✓
10000 10% 1.6036 this was pricey for the type of jewelry it is but it is artsy and different and i like that the picture would have you believe the closure is directly degrees across from the pendant drop but it is not the pendant drop is actually about to the left of center i have a neck and there is not a lot of play for this necklace to hang it fits pretty snugly when i wear this i arrange the pendant to be center front on me and just wear the closure off center in the back the guitar pick is positioned too high up near my collarbone to hang noticeably despite its size i like this piece for the uniqueness of it it is great quality based on the durability of it and the charms used are securely attached and not one of the charms are flimsy at all / 3 ✗ (4)
15000 15% 1.5221 beautiful and exactly as advertised / 5 ✓
20000 20% 1.7157 this owl cuff is great i bought this item as a birthday gift the cuff is smaller than it appears in the picture width wise other than that its great / 3 ✗ (4)
25000 25% 1.7103 appearance looks exactly like the picture fits perfectly too durability i have had this for about a month or so and the ring has not become tarnished or discolored it is still the same color and i wear this product every day when i wash my hands i take it off to take a shower though i have this problem when i buy rings that my finger turns green at the area where the ring sits but this ring has not made my finger turn green at all if you have the same problem and are worried about that do not be i would definitely recommend this product to others / 2 ✗ (5)
30000 30% 1.8227 purchased as gift sized as shown only wanted bible no crosses or other symbols so it suited my purpose sufficiently / 3 ✗ (4)
35000 35% 1.6173 i like it its big but i am ok with it the material is more plastic than anything / 5 ✗ (2)
40000 40% 1.5055 did not like them as like other reviewers have said these are uncomfortable to wear they are rather pretty but not quite what i like also agree that the closing of the hoop part is poorly made / 3 ✗ (2)
45000 45% 1.5856 one eyes was missing from one of the owls i need to return them other than that they are really cute / 4 ✗ (1)
50000 50% 1.6081 the earrings are a little ti short an pinch my ear when i try to wear them an it took longer to recive them than it should have i was not pleased with that at all / 1 ✗ (3)
55000 55% 1.4911 after reading the description i misunderstood that the clear in the description meant the actual ring i thought the stud was clear and the actual ring was silver as it looks in the picture i was disappointed and will not wear this in public as it does look fake and could never be mistaken as a real piercing if that what you are aiming for i had my nose pierced but had to take it out for interviews and it closed so i was looking for something more realistic / 2 ✓
60000 60% 1.5526 only returned for non opal flatness appeared to be an opal defect where it was not flat but an indentation on one earring otherwise solid and as described wonderful was a fluke defect in my

```
order  / 3 ✗ (2)
65000 65%  1.5252 it is really a brown bead not at all red as stated
in description it is still a very nice bracelet however  / 1 ✗ (4)
70000 70%  1.4610 this was almost impossible to put on by yourself the
charms although pretty all keep falling under the wrist because there
is nothing to keep them in place so what stays on the top of your
wrist is the plain bracelet band not only does not it look so good
with the charms under the wrist but it is really uncomfortable i just
do not get who could wear this  / 2 ✓
75000 75%  1.5578 i really like the size of these earrings and the
clasp happens to be my favorite type of hoop clasp but the stationary
part of the clasp was bent back and the space for the hinged piece to
fit between has too wide of an opening and therefore does not stay
closed i have to be creative and use a post back to keep from losing
the earrings for the price the earrings were too much trouble to send
back but works well with my creative way of keeping them on my ear  /
4 ✗ (2)
80000 80%  1.7279 it a bit small but it will do i can used it where i
want it for it has to be longer maybe inch more you have a bless day
ps the chain is made well  / 3 ✗ (5)
85000 85%  1.9319 these earrings are what i have been looking for for
a long time could always find the single or double ball but never the
three balls and they are beautiful although they are very inexpensive
they look stunning  / 3 ✗ (5)
90000 90%  1.6574 love love love this ring only dislike is that the
stone is a lot lighter than pictured still beautiful though  / 5 ✗ (4)
95000 95%  1.5534 very beautiful but broke when adjusting size / 5 ✗
(1)
100000 100%  1.6725 bought this for my husband and it loves it great
true to size fit  / 5 ✗ (4)
```

```
def grnn_evaluate(review_tensor):
    hidden = grnn.initHidden()

    for i in range(review_tensor.size()[0]):
        output, hidden = grnn(review_tensor[i], hidden)

    return output

def grnn_predict(input_line, n_preds=1):
    line = input_line.tolist()[0]
    with torch.no_grad():
        output = grnn_evaluate(convert_review_to_tensor(line,
wordvec))

        topv, topi = output.topk(n_preds, 1, True)
        predictions = []

        for i in range(n_preds):
            value = topv[0][i].item()
```

```
            category_index = topi[0][i].item()
            predictions.append([value,
all_categories[category_index]])

        return all_categories[category_index]
```

```
real_preds = test_dset['star_rating']
g_data_lines = pd.DataFrame(test_dset['review_body'])

g_data_lines['review_body'] = g_data_lines.apply(lambda x:
grnn_predict(x, n_preds=1), axis=1)
print(g_data_lines)
```

```
       review_body
0                2
1                5
2                5
3                5
4                3
...            ...
19995            5
19996            2
19997            5
19998            5
19999            5

[20000 rows x 1 columns]
```

```
from sklearn.metrics import accuracy_score
g_test_preds = g_data_lines['review_body']
acc = accuracy_score(real_preds, g_test_preds)
print("Avg GRNN acc: ", acc)
```

```
Avg GRNN acc:  0.2287
```

```
from sklearn.metrics import confusion_matrix
matrix = confusion_matrix(real_preds, g_test_preds)
acc_array = matrix.diagonal()/matrix.sum(axis=1)
for idx, acc_val in enumerate(acc_array):
    print("Class", idx+1, "accuracy:", acc_val)
```

```
Class 1 accuracy: 0.09825
Class 2 accuracy: 0.313
Class 3 accuracy: 0.13475
Class 4 accuracy: 0.0985
Class 5 accuracy: 0.499
```

Overall, it seems our accuracy got worse. However, there are a few reasons this might be the case. the main issue could be the training hyperparameters, ie number of examples of each category, the embedding method, the learning rate(s), the batch size, etc... I think if we tried tuning these parameters we might have some luck with this!