Problem Set 3: Text As Data
IMT 575: Scaling, Applications, and Ethics

Instructions

All problem sets must be submitted as a Jupyter notebook. You must organize your analysis and utilize the notebook's markdown capabilities to present your solutions. Partial credit will be awarded for each question for which a serious attempt at finding an answer has been shown. Please see the course syllabus for more instructions/information. Unless otherwise noted, this assignment is due on **May 15, 2023 at 11:59pm. Submit a Jupyter notebook and its pdf for grading purposes.**

Overview

In this problem set, you will perform a supervised text regression task. The data for the task will consist of student essays from The William and Flora Hewlett Foundation. The dataset is meant to be used to develop fast, effective and affordable solutions for automated grading of student-written essays. You will be using a subset of this dataset and predict the scores of the essays. You may not use external data to make predictions.

You will be provided with two compressed folders: train.tsv and test.tsv. Both files contain the text of the essay and the score of each essay.

Part 1: Set-up

1. For reproducible purpose, you will set the random seeds for numpy and tensorflow as 1234 with the following codes:

```
from numpy.random import seed
import tensorflow as tf
seed(1234)
tf.set_random_seed(seed = 1234)
```

2. To begin, you will first work through processing the data. Start by loading in the training data and test data. Keep each in their own dataframe/list (i.e. a dataframe/list for all training observation text and a separate one for all test observations). How many training and test observations do you have?

3. Create a vector of training labels. What does the distribution of labels look like for the training set and the test set? Comment on what you see.

4. Convert the training data to lower case. Remove punctuation from the training data. Also remove stop words from the training data using the NLTK package's English stop word list. In addition to NLTK's stopwords, are there words specific to this dataset that may be worthwhile to treat as stop words? What are these words and why would you exclude them? Remove these additional stop words from the training data as well (note: you can also remove this secondary set of stop words after tokenizing. However, keep in mind that this may cause feature alignment issues with your test dataset when you tokenize it.)

cats -> cat   saw -> see

5. Stem/lemmatize your training data using a stemmer/lemmatizer of your choosing. Show a before and after using a few observations and comment on what you see.

6. Tokenize your training data using unigrams (hint: see sklearn's CountVectorizer). If you set upper and lower limits on word frequency, what are they? How many unique tokens are in your vocabulary? the training set and test set should contain the same features

the test and training dimensions should be aligned

7. Process the test data in a manner identical to the training data. Note that you will need to have the same dimensions for your training and test data. One way in which this can be done is using sklearn's CountVectorizer is to fit on the training data and transform the test data. Show that the number of features for your training and test data are identical.

Part 2: Supervised Learning

7. The primary objective of this problem set is to build classifiers to predict the scores of the essays. Having the right features are essential for this (and any) prediction task. In addition to the features you were tasked with generating in Part 1, design/build additional features of your choosing. Please note that this process can be iterative in a cycle of designing/adding features and evaluating model performance - this is completely reasonable and you can come back to this question after completing the remainder of Part 2. Describe the features that you ultimately used in your final models and articulate your reasoning for including the features that you did. Also describe any feature manipulation/scaling you did and your reasoning for doing so. Ensure your test set has identical features as your training set. Some potential ideas for additional features include bi-/trigrams, topic model weights, TF-IDF weights, sentiments, hand-engineered word co-occurrences (not adding any/all of these is completely fine as well).

8. You will create two different models for this problem set: a regularized linear regression model (Ridge), and any other model of your choice (regardless of whether it was discussed in class or not). You do not need to implement anything from scratch and can use out-of-the-box models/pipelines. To start, build a regularized linear regression model. What was your regularization coefficient and how well does this model perform? Are there any score ranges where it performs particularly well/poorly? Use graphs/tables where appropriate and contextualize results.

9. Next, build any model other than the ones you've already built. Which model did you use and why? Did you tune hyperparameters? If so, which ones and how? Are there any scores where it performs particularly well? Are there any scores where it performs particularly poorly? Use graphs/tables where appropriate and contextualize results.

10. Compare the performance of the regressors you built. Which is the strongest? Which is the weakest?

Part 3: Deep Sentence Embedder

13. For this part, we will implement a deep sentence embedder to replace the feature selection process. As a first step, select which of the regressors from part 2 you will use. State which model you chose and your thought process for choosing it versus the other models. You may want to copy this model to another variable so you can refer to it in its current form later.

14. You will be using Universal Sentence Encoder to embed the text data and get the dense feature vectors for the essays. First you need to import the model with the following codes:

```
module_url = "https://tfhub.dev/google/universal-sentence-encoder/4"
model = hub.load(module_url)
```

transformer encoder combines and averages contextualized embeddings that contain word order information

15. Next, You will embed the data with the imported model. Universal Sentence Encoder takes a list of strings and generates the embeddings of the sentence list. Assuming x_train is a list of the strings, here's how you get the embeddings of the strings:

```
emb = model(x_train)
```

16. Next, feed the embeddings to the model you chose from the part 2. How does your model perform? Does it perform better than using TF-IDF features?