Homework 5: Regression

Due October 23rd

1 Deep Learning Math

1.1 Computing Loss in Binary Regression

Compute the cross-entropy loss for a negative review classified as positive with a probability of 0.55 by a binary logistic regression model. Treat the positive class as class 1.

1.2 Computing Loss in Multinomial Regression

Now consider a multinomial logistic regression model with three classes: positive, negative, and neutral. Compute the cross-entropy loss for a negative review given the following probabilities by the model: $\hat{y}_{pos}$ = 0.55, $\hat{y}_{neg}$ = 0.25, and $\hat{y}_{neutral}$ = 0.20.

2 Song Classification

In the second part of this assignment, we will build a regression model to classify songs by three artists: Taylor Swift, Drake, and Beyoncé.

I have already scraped the lyrics from each of the artists' studio albums, and split the data into two datasets: one for testing and one for training.

2.1 Data Preprocessing

The input to our model will be a vector of numbers. For training and evaluation, we will also need to provide a list of *labels*, the correct classes for each lyric. We will need to code the labels as numbers; the starter code contains a hard-coded dictionary mapping artist names to class numbers.

Write a function called **`make_data`**. Your function should take the name of a dataset file and the label map as arguments. It should read in the contents of the file. The song lyric should be **lower-cased** and then tokenized using the **`tokenize`** function. The artist name should be mapped to a class number.

Your function should return two lists: a nested list, where the inner lists are the result of calling **`tokenize`** on a lyric; and a list of labels.

2.2 Computing Features

Our song data is mapped to feature representations in the **`compute_features`** function. This function does three things: 1) it computes feature representations for songs, 2) it **scales** those

features, and 3) converts the data to a Tensor representation.

Computing features Currently, the only feature that is implemented is song length. Later, you will add other features by adding more calls to **`add_feature`** within **`compute_features`**. **`add_feature`** takes the current list of feature representations, the song data, a feature function, a list of feature names, and the name of a new feature. It applies the function to each song line to create a new feature and augments the list of feature representations and the list of feature names to reflect this.

Scaling features Because regression fits a linear equation, there can be issues if some feature values are much larger than others. I've given you code to scale features so that each feature representation is scaled between 0 and 1.

The normalization is computed on the training dataset, and reused for the test data set. When you call **`compute_features`** on the training data, **you should save the set of norms** it returns and pass them into **`compute_features`** when you call it on the test data.

Making a tensor The machine learning library that we will use, PyTorch, requires input to the model to be in the form of a Tensor, an n-dimensional generalization of a matrix. The function that I've given you turns the list of feature representations into a list of Tensors.

Without adding any more features, **add calls to main** to process your testing and training datasets using **`make_data`** and then **`compute_features`**. Make sure to save the norms returned from processing the training data and pass them in to normalize your test data.

2.3 Data Loaders

Next, create data loaders for each dataset. You will need to zip together your features and labels before passing them to DataLoader:

```
train_dataset = list(zip(train_feats,train_labels))
test_dataset = list(zip(test_feats,test_labels))
```

2.4 Building a Regression Model

In class, I showed how to build a regression model. You can start building your regression model by copying over the model class that we wrote in class, plus the functions for training and testing.

Make these changes and test out your model, using only song length as a feature. It should not perform well, but it should run without error.

3 Crafting Features

How should we turn our song lyrics into numbers? In this part of the assignment, you will do some **feature engineering**. You will write functions that take song lyrics as input and output a number. Hopefully, this number will capture something meaningful about the lyric.

As you construct new features, you will add more calls to **`add_feature`** in **`compute_features`**. **`add_feature`** takes five arguments: the current list of feature representations, the song data, a feature function, a list of feature names, and the name of a new feature. It applies the function to each song line to create a new feature and augments the list of feature representations and the list of feature names to reflect this.

`add_feature` **is a *higher-order function***: it expects a function as its second argument.

3.1 Word Count Helper Function

To start with, write yourself a helper function for calculating average number of word occurrences. This function should take two arguments: a lyric and a word. It should return the average number of words in the lyric that are identical to the word.

Call this function **`avg_word`**. You can now write feature functions that are simply wrappers around this.

3.2 Wordlist Helper Function

Write yourself another helper function. This one calculates the average number of words in the lyric that occur in a list of search words. It should take two arguments: a lyric and a list of words.

Call this function **`avg_words`**. You can now write feature functions that are simply wrappers for this.

3.3 Add Some Features

Add a few features using these helper functions. You should do this by adding additional calls to **`add_feature`** into the **`compute_features`** function. Each call will take the name of your feature construction function as its second argument.

4 Evaluation

As you add features, you may find yourself wondering how much they are really helping. We will take a short break from feature-engineering to write some evaluation functions.

4.1 Evaluation Metrics

So far we have looked only at overall model performance. But our lyric dataset suffers from **class imbalance**: there are twice as many Taylor Swift songs as Beyoncé songs.

Write a function called print_performance_by_class that prints a summary of accuracy by category. Your function should take two parameters: a set of labels (such as test_labels) and a set of model predictions.

Your function should print a summary like the one below. Note that I am rounding to 3 places to make the output more readable.

```
Accuracy by Category:
Category 0 : 1.0
Category 1 : 0.0
Category 2 : 0.0
```

You can use the test() function as an example of how to retrieve the model predictions and compare them to the correct labels.

4.2 How Much Does a Feature Help?

I have given you a helper function called **print_coefficients**. This function prints out a table of regression weights for a given model. The columns correspond to different categories; the rows correspond to different features.

A negative weight for a particular feature/category pair means that as the value of the feature increases, the likelihood of the category decreases. A positive weight indicates a correlation (positive linear relationship) between the feature and the category.

4.2.1 Questions

1. Use this function to analyze three of your features. What do the coefficients tell you about how useful this feature is for predicting each artist?
2. Do you think that looking at model coefficients is sufficient for determining whether a feature is useful? Why or why not?

5 Feature Engineering

Your main challenge is to build features that will improve model performance. You are required to experiment with several feature formats:

- At least one feature that uses punctuation
- At least one feature that identifies words in a *syntactic* group: pronouns, verbs, negation words, etc.
- At least one feature that identifies words in a *semantic* group: positive words, music words, feelings, etc.
- At least one feature that uses regular expressions

To receive full credit, your model must achieve above 15% accuracy on all classes. You will receive substantial partial credit if you achieve above 15% accuracy on two classes, and partial credit if you meet the requirements above, but do not achieve above 15% accuracy on more than one class.

Note: you should NOT base your features on what you observe in the test data. We use a separate test set so that we can evaluate how well our model generalizes to unseen data— if you peek at the test dataset while feature-building, it lets our model cheat!

5.1 Questions

1. Describe the features you have constructed. How did you decide on them?
2. Which features did you find to be most useful?
3. What is the best performance that you achieved?
4. Can you think of any other ways you might have been able to improve performance?

6 Intellectual Curiosity (10pts)

As usual, you will receive 90 points for implementing everything described above. To increase your score further, you can extend your investigation in some way. **If you choose to do this, please briefly describe what you've done in your report.**

If you would like to use data from another artist, I have provided some scripts for webscraping lyrics and post-processing them into the same format as the Swift, Beyoncé, and Drake lyrics.

6.1 Wrapping Up

When you're finished, you should submit all of your Python files and your answers to the questions (as a PDF) to Gradescope.