CSC 180-01 Intelligent Systems (Fall 2023)

Project 1: Yelp Business Rating Prediction using Tensorflow

William Moosakhanian, ID# 302537495

Xiong Moua, ID# 220278332

Due: September 29, 2023

Problem Statement

The problem that we are solving with this project is to train a neuron network and accurately predict the rating of a business based on the reviews and any other information that we believe to be relevant.

Methodology

The first thing we did was create a dataframe for the businesses and another for the reviews. This was where we implemented our first feature which was obtaining each business' city and category. Our reasoning behind this was because we believe that the city the business is in and its categories would be good features for the model later on. Then we renamed the column "stars" in each respective dataframe to "business_stars" and "review_stars" because we intended to perform an inner join later during the filtering process. Afterwards we grouped all of the reviews to its relative business id to ensure all of the reviews were uniquely tied to its respective business as well as calculating the mean for the ratings. Then we implemented our second feature which was the ability to change the minimum number of reviews required per business for prediction purposes.

After filtering the data, we one-hot encoded the "categories" and "city" columns, and then we vectorized the "all_reviews" column with the TFIDF. Our stop word count was set to 2500, with a minimum frequency of 30 and a maximum of 200. In order to remove any numbers or unnecessary punctuations, as they held zero value in creating a better performing model, we utilized the following token pattern, r'\b[^\d\W_]{4,}\b', which was documented in the *sklearn.feature_extraction.text.TfidfVectorizer* documentation under *token_pattern*. Essentially, the token, r'\b[^\d\W_]{4,}\b', is broken down like so; the 'r' stands for a raw literal string, the '\b' defines boundaries for the target string, '[^\d\W_]' is a set of characters to exclude from the string because '^' acts as a negator in the set, while '\d', '\W', and '_' all stand for digits, punctuation, and underscores, respectively, and finally, the '{4,}' means to only allow words with 4 or more characters. Subsequently, we created a 2-D matrix, converted it to a numpy array, and merged it with another numpy array which contained all of the normalized data. Before we can start to get our numpy array ready for the model with the function to_xy, we had to check if there was any more normalization that we needed to do. Out of the three columns that we had left, 'business_stars', 'review_stars', and 'review_count', the only one that made sense was review_count because these values are numerical. After we finished normalizing this column and were happy with our dataframe, we used the to_xy function to prep the dataframe for the model. After creating the numpy data array for our dataframe, we concatenated that numpy array with the one we received from the TfidfVectorizer. At this point we are ready to split our newly created numpy array into testing and training data.

Finally, we created our first model using RELU and Adam as the optimizer. Utilizing local optimum, we selected our best model, predicted the values, and calculated the root mean squared error, (RMSE). Our second model used Sigmoid and Adam while the third model used Tanh and SGD as the optimizer. Each model had a local optimum of 5, the same number of layers, although the neurons in the second layer for model one was 50 unlike the other models which used 75. Additionally, early stopping was implemented with a minimum delta of 1e-2 and

an epoch of 1000. However, on most occasions, the model would stop before reaching 20 epochs. Afterwards, we chose the RMSE which was closest to zero and created the chart based on it.

Experimental Results and Analysis

During this project, we experimented quite a bit with the models and how they interacted with the data. We observed an interesting trend: as we increased the volume of data initially written to the dataframe and expanded our training dataset, our model's performance significantly improved. For example, when we were debugging the program trying to make sure the models would perform correctly from a logical standpoint, we would only use the first 100,000 values in the JSON data set. We quickly noticed that our margin of error was around 22%, but when we ran the full 6,100,000 values, (filtered down from 6,700,000), our margin of error dropped to around 7%.

Additionally, we found that the RELU and Sigmoid based activation layers performed very similarly to one another, with RELU pulling ahead slightly. Both models utilized the Adam optimizer, while model three used the Tanh activation with the SGD optimizer. What was interesting was that model three would consistently outperform the other two by a margin of 0.5% which we think was mainly attributed to the mathematical properties of Tanh. As a function, Tanh squashes all the values closer to zero and has a steeper curve than RELU which is a piecewise linear function and Sigmoid which has a curve that is not as steep as Tanh.

Furthermore, SGD optimizes the model towards moving in the direction where the loss decreases the most, doing so with a fixed or manually adjusted learning rate, whereas Adam adaptively adjusts using momentum and by taking dynamic step sizes. On top of that, SGD has a slower learning rate, because of the nature of searching for the biggest loss decrease, while Adam fine-tunes its learning rate progressively, which in turn impacts outliers and can often decrease stability. One thing we noticed is that with SGD we had less outliers because due to its slower learning rate, it would effectively remain more stable and not produce as many outliers. With all this being said, the differences were around 1-4% which in the grand scheme of things, is not much.

Tabulated Results

Model No.	One	Two	Three
Activation	Relu	Sigmoid	Tanh
Layers	4	4	4
Neuron Count	Input Layer 1: 100 Hidden Layer 2: 50 Hidden Layer 3: 25 Output Layer: 1	Input Layer 1: 100 Hidden Layer 2: 75 Hidden Layer 3: 25 Output Layer: 1	Input Layer 1: 100 Hidden Layer 2: 75 Hidden Layer 3: 25 Output Layer: 1
Optimizers	Adam	Adam	SGD
Results (100,000 values)	0.5661546 ≈ ± 22.646%	0.5358583 ≈ ± 21.434%	0.5697864 ≈ ± 22.791%
Results (6,100,000 values)	0.18307644 ≈ ± 7.32%	0.16891135 ≈ ± 6.76%	0.16505583 ≈ ± 6.60%
Prediction Improvement	67.63%	68.44%	70.89%

Predictions Without Names

	ground_truth	predicted
0	5.0	4.488310
1	4.0	3.505982
2	4.5	4.216613
3	3.5	3.638973
4	3.0	2.251703

Task Division and Project Reflection

The two of us split the project evenly. Xiong primarily handled the data preprocessing and merging the data frames together to ensure our data was in the correct format for models. On the other hand, William created the models, tuning them for hours on end, and created the prediction results, charts and RMSE outputs. With all this being said, the two of us assisted each other with every portion of the project to ensure we could meet our deadline.

The two biggest challenges that we faced was setting up the environment and the learning curve for data preprocessing and modeling the tensorflow model. Setting up the environment was extremely time consuming and frustrating. At first we installed the newest version of anaconda. Installing tensorflow with the command : conda install tensorflow, was extremely time consuming because it was examining all of its modules quite slowly. At the end it threw out an error code saying that tensorflow is not compatible with anaconda's version of python. We tried downgrading the version of python that anaconda used but we still received the same error code.

Ultimately we decided to install an older version of anaconda that was compatible and installed tensorflow with: pip install tensorflow instead. As for the learning curve on data preprocessing and tensorflow. Learning pandas and its different function calls with the data frames was tougher than we expected, but nothing we couldn't handle. Initially, It was quite confusing to connect the dots as to how tensorflow interacted with the dataframe, but as we progressed through the project, we examined the labs provided for us and learned from them and how they functioned. Over time, we became comfortable enough to understand what the parameters actually did, and how they interacted with our dataframe, thus allowing us to select what we felt the most optimum parameters were for the project.

Some insights we gained from this project was the importance of cleaning the data before we send it to our model for testing and training, the amount of ram needed to use large data files, and understanding how long our code can take to run. We ran through multiple versions of the project. For our first version we did not clean our data thoroughly. This resulted in our model being incredibly inaccurate. As we refactored our code, and did more preprocessing the model became more accurate. Both our laptops have 16 gb of ram but that was not enough if we wanted to use the entire size of the file. Luckily Will has 32 gb of ram on his desktop. The most time consuming aspect of the project was writing to a file. At first we wanted to write to a file to debug, but once we increased the size of our data, writing to a file was too time consuming. We did all our debugging on a smaller data size. Once we were happy with our results, we used the entire dataset.