DS III Final Project Report

Due Thursday May 4th 2023

Yelp Data Challenge

Authorship Table

Data Cleaning	Jennifer Kimball
Coding for Task 1	Jennifer Kimball Madelyn Marcotte Matthew Suyer
Coding for Task 2	Matthew Suyer
Paper	Pegah Emdad Kendall Haddigan Madelyn Marcotte Jennifer Kimball Matthew Suyer
Presentation	Pegah Emdad Kendall Haddigan

Introduction

Yelp is an online platform that relies on input from its users to provide reviews of local businesses. The platform features dedicated pages for specific establishments, including restaurants, where users can post reviews and give ratings ranging from one to five stars for the products or services offered. People can use Yelp to search for businesses based on their preferences. The company can use information about user preferences to help optimize their system and improve the quality of user experience.

The Yelp dataset can be found on Kaggle and is composed of five distinct datasets. These datasets represent businesses, reviews, users, check ins, and tips, and are saved as JSON files. We used the business, reviews, users, and tips datasets for our applications. In the business dataset, we utilized information regarding a business' ID, name, star rating, number of reviews, different attributes (for example if it accepts credit cards, business delivery, if it offers parking, bike parking, etc.), and business categories such as 'bubble tea', 'mexican', 'burgers', and 'coffee and tea.' In the review dataset we used the columns for the review ID, user ID, business ID, stars given, and the written review provided by the user. Additionally, the dataset quantifies the number of cool, useful and funny votes on the review. In the user dataset, we used the columns for the user's ID, name, and the number of reviews. Finally, in the tip dataset we used user ID, business ID, and the text of the tip.

Research Tasks and Proposed Solutions

There are many research questions that can be answered with this data, but we decided to focus on two tasks. Our first task was to predict business attributes by using review and tip textual information. We chose a multi-label classification approach to this task. Our second task was to predict how a user would rate a restaurant based on their previous behavior and ratings. For this task, we chose two main approaches - collaborative filtering and latent factorization. For both tasks, we chose to only work with businesses within the "Restaurant" and "Food" categories to narrow the focus of our research and to save computational resources.

Task 1: Predict Business Attributes

For our first task, we predicted business attributes based on review and tip textual information. After preprocessing our data, we ran a multi-label classification model on our

subset of the review and tip data to predict a set of business attributes for each review and tip separately. Our multi-label classification model was composed of a neural network created using the keras library. Our model parameters, such as the number of epochs, were dependent on the computational limits that we faced.

Data Preprocessing - One-Hot Encoding

In order to complete this task, we first needed to preprocess the data to make it suitable for the model. We primarily utilized the pandas library for these steps. To clean the business dataframe, we removed restaurants that had less than 3 attributes. The most difficult part of the preprocessing was extracting the individual attributes from the attribute column of the refined business dataframe. The column values were originally of the object data type, so we first needed to convert them into strings. To gather a list of attributes, we combined the column data into one extensive string, selected the unique attributes, and performed additional text processing to remove the punctuation and true false values. Once we successfully separated the individual attributes, we then created a dictionary of the 50 most common attributes. We wanted to further reduce the computational time of our models, so we selected a subset of 29 attributes that contained only true and false values.

Our next preprocessing step was to one-hot encode the attributes. We created an attribute dataframe and filled this with zeros. We then merged this dataframe with the business dataframe and wrote a for-loop to look through the attribute column and encode a 1 if the attribute was present and true. The column value remained zero if the attribute was either missing or false.

Data Preprocessing - Text Preprocessing

Next we had to preprocess the textual data to prepare it for multi-label classification. This included removing the numbers, punctuation, single characters, and single spaces. Additionally, we embedded the word vectors, and converted text inputs to their numeric counterparts using GloVe 100d.

GloVe stands for “global vectors for word representation.” It is an unsupervised learning algorithm that is used to generate word embeddings. It does this by aggregating global word-word co-occurrence statistics from a corpus, the representations for which showcase interesting linear substructures of the word vector space.

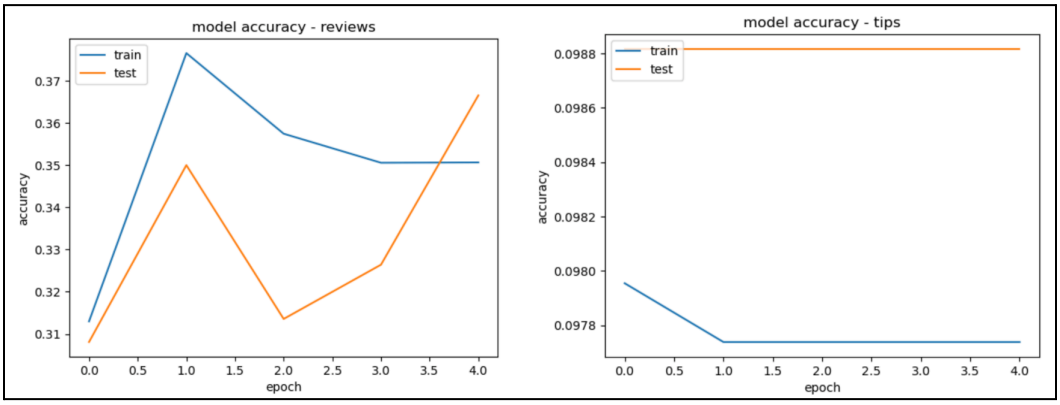
GloVe works by deriving the relationship between the words from statistics. Each value in the co-occurrence matrix represents how often a particular word pair occurs together. The GloVe 100d file is a pre-trained word vector that we used to create an embeddings dictionary that we could apply to our textual data to represent it numerically.

Multi-Label Classification Model

With preprocessing complete, we then began initializing our multi-label classification model. To start, we separated the data into X (textual data) and Y (attribute labels). We then split it into training data (64%), validation data (16%) and testing data (20%). Our multi-label classification model consisted of a long short-term memory neural network, which we created using the keras library. We used a sigmoid activation function, a batch size of 128, and trained the network for 5 epochs. For our loss function we used binary cross-entropy, and we used Adam as our optimizer.

Task 1 Results

Accuracy Results



Reviews

Test Accuracy: ~0.366

Tips

Test Accuracy: ~0.0975

Figure 1. Task 1 model accuracy by epoch.

We can see in both graphs of the model accuracy that there is a significant difference in the accuracy of the training and the test data. For reviews, the test accuracy eclipsed the training accuracy when the last epoch ran. The testing accuracy was still trending up at the conclusion of

model fitting, so if we had the computational resources to run more epochs, it is likely that the test accuracy would have been greater than 0.366. The testing accuracy for tips flatlined at 0.0988, so we can be fairly confident that the true testing accuracy hovers around this point.

Loss Results

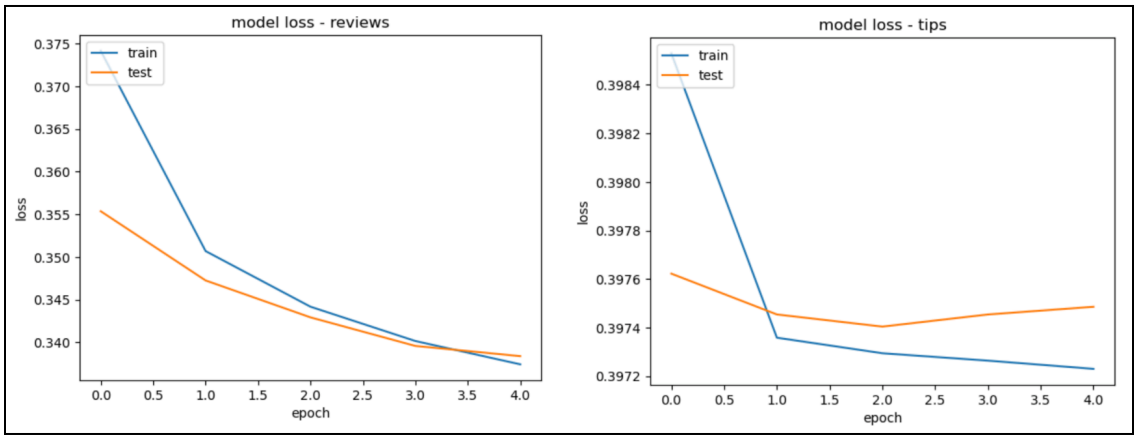


Figure 2. Task 1 model loss by epoch.

We can see in both graphs that the model loss decreases overall as the number of epochs increases. For both review and tip we can see that for the training data, the loss decreases sharply from the first epoch, whereas the testing data sees a more gradual decrease across epochs.

From the graphs, it is evident that reviews are a much better predictor of attributes than tips. This makes sense, as reviews contain more text for the neural network to work with. One reason we think the test accuracy was not very high is because some of the attributes relate to things that are not often discussed in reviews, such as whether or not a business accepts credit cards. Additionally, many of the attributes were related to a restaurant's ambience, so it is likely that many of the reviews did not describe the ambience well enough for the model to classify it.

Multi-Label Classification Model Limitations

There were multiple limitations that we encountered with the dataset and our model approach. In terms of the dataset, we found that the formatting of some of the attributes as dictionaries made it hard to work with. We also encountered time limitations and memory failures as a result of the size of the dataset files. Because of the size of the data we were

required to use a subset, however we were still limited by the model run time, which is why we only used 5 epochs.

In terms of our model, the predictions we were making were one-to-one, as opposed to many-to-one. This is referring to the fact that we were predicting a business' attributes for each of the reviews written about the business, as opposed to making one prediction of the business attributes based on all of the reviews. We also did not distinguish between attributes that were not present in the business data, and attributes that were present but labeled false. This means we essentially assumed that all attributes were false unless proven true, but in reality, there is an important distinction between an attribute being false and there being no data about an attribute.

Future Work

To improve upon our proposed model, we would recommend optimizing the model hyperparameters, such as batch size, learning rate, and number of training epochs. The goal of this optimization would be to train a number of networks, each with a different combination of hyperparameter values, and observe which one has the best accuracy. Another thing we would like to try would be using stochastic gradient descent as our optimizer as opposed to Adam to see if our results improve. We also recommend modifying the model to use all of the reviews and tips for a given restaurant to make a prediction about its attributes, rather than predicting attributes based on each individual review or tip. Lastly, we would like to be able to differentiate between attributes that have false values and attributes that are missing.

Conclusion

We can conclude that reviews are a significantly better predictor of restaurant attributes than tips with the accuracy of 36.6% and 9.75% respectively. This makes sense because we know that the reviews are much longer than the tips, and contain more information that can be used to make predictions. Neither of the models have an extremely high accuracy. There is potential to increase the model performance if we used a combination of the two predictors or optimized the hyperparameters of the model.

Task 2 - Predict User Ratings

Our second task was to predict how a user would rate a restaurant based on their previous ratings. This would allow Yelp to recommend users restaurants that they are likely to rate highly based on their similarity to other users, which in the long run, may lead to increased profit. The group decided to use two main approaches - collaborative filtering and latent factorization - to tackle this task. For the collaborative filtering, we tried both a user-based and an item-based approach, and used two different methods of weighting the similarities for each.

Approach 1: Collaborative Filtering

Collaborative filtering is a technique used in recommendation systems to make predictions or recommendations about items of interest to a user based on the similarities between users or items (restaurants). The ideas behind collaborative filtering are that users that have rated many restaurants similarly in the past will continue to do so in the future, and that restaurants that are similar to each other will likely get similar ratings by users. To gain a notion of the similarities between users and restaurants, we could construct a user-restaurant matrix (in which each entry is how a specific user rated a specific restaurant), and could use the rows and columns of this matrix to represent users and restaurants as vectors of ratings. While the user-restaurant matrix would be extremely sparse, the vector representations within the matrix would allow us to compute similarity between users and restaurants. As such, we decided it was more computationally efficient to represent each user as a vector of ratings of restaurants, and each restaurant as a vector of ratings received by users. For collaborative filtering, the measure of similarity between vectors we chose was cosine similarity. Since this metric only cares about the overlapping nonzero elements in each vector, calculating it for each pair of users or restaurants was easy on our computational resources.

We selected the user, business, and rating data as it was cleaned for Task 1 to prepare to run our models. Within each method of user- and item-based collaborative filtering, we had two approaches - the first approach was to weigh the sum of all similar vectors by their similarity, and the second was to average the top 3 most similar vectors. We used Pandas to handle the input data and dictionaries to build vectors. The first method used all vectors with a nonzero similarity, and weights their rating of the restaurant according to that similarity. The second method only considered the top 3 most similar vectors and averaged their ratings of the restaurant.

Approach 2: Latent Factorization

Latent factorization, or matrix factorization, is a technique used to discover underlying relationships within a dataset. We sought to "factor" our sparse user-restaurant matrix into two dense matrices such that, when they are multiplied together, the known values in our user-restaurant matrix match the corresponding values in the product matrix as closely as possible. As mentioned above, we used vector representations of users and restaurants instead of a sparse matrix, but the methodology remained the same. Then, we created a feedforward neural network to learn an embedding matrix for users and for restaurants. The network seeks to minimize the "reconstruction error" when the matrices are multiplied back together. We varied many hyperparameters to find the network that achieves the best loss, such as batch size, learning rate, number of training epochs, and the dimension of the user and restaurant embedding matrices. We implemented this network using PyTorch, and manipulated the data using Pandas.

Task 2 Results

Task 2 Results Summary

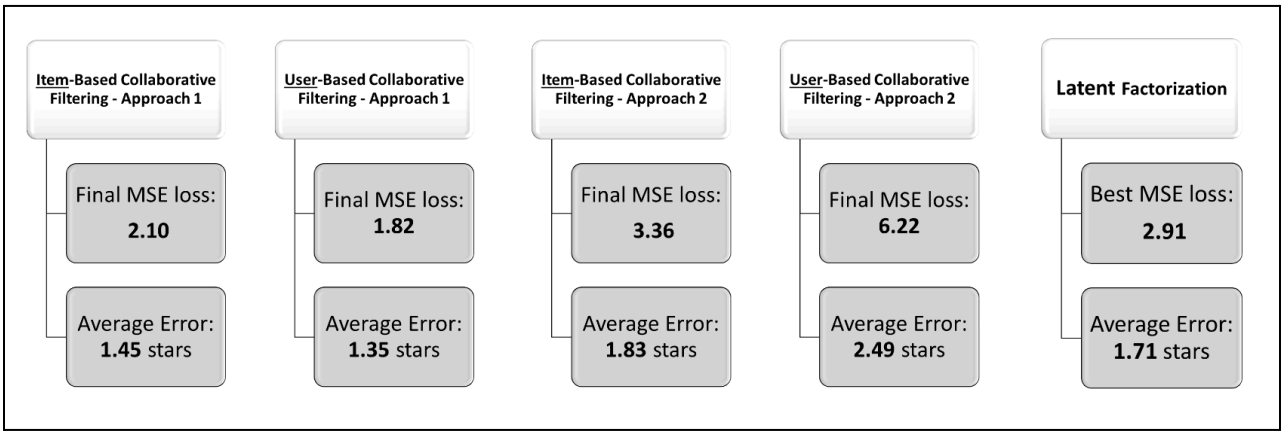


Figure 3. Summary of results for task 2 approaches.

The results of our five total approaches of collaborative filtering and latent factorization can be seen in the diagram above. We used MSE loss to determine model performance and computed our average error using RMSE loss. We chose MSE loss because PyTorch does not have a built-in RMSE loss function, but we reported RMSE as well as MSE to get the unit of the loss back to stars (from stars squared).

As is shown in the diagram, the approach with the lowest MSE loss and average error is the weighted-sum approach to user-based collaborative filtering. It has a final MSE loss of 1.82, and an RMSE of 1.35 stars. This means that the model had an average prediction error of 1.35 stars when predicting a user/restaurant rating. 1.35 stars is not a terrible average error, in the sense that if we predicted a user would rate a restaurant with 5 stars, and they only rate it 3.5, they still presumably had a somewhat positive experience because of our recommendation. However, if the model predicts a user would rate a restaurant 3.5 stars, and they only rate it 2 stars, then we have given them a bad recommendation.

The approach that performed the worst based on these metrics is the second approach at user-based collaborative filtering. It has a final MSE loss of 6.22, and an RMSE of 2.49 stars. This error is too large to be reliable in almost any situation, so we would not want to use this method to recommend restaurants to users.

One reason why we think the weighted-sum approach to user-based collaborative filtering is the best approach is because it deals with the limited amount of input data the best. While 1 million data points is typically sufficient for a deep neural network of this scale, these data points are scattered across over half a million users and restaurants. This means that in addition to users and restaurants each having fewer ratings, the overlaps between the rating vectors is decreased. This lack of data - or more specifically, the lack of overlaps between user-restaurant pairs - means that even “similar” vectors likely won’t have high similarity scores with each other. The first method accounts for this lack of similarity by weighting the vectors based on their similarity, but the second method inherently assumes that the top 3 vectors have high similarity with the given vector. An average between three dissimilar vectors will yield a prediction that is farther from the ground-truth than a weighted sum by similarity of dissimilar vectors. One reason latent factorization performs worse than collaborative filtering is that non-deep (“shallow”) methods will inherently have an advantage with small amounts of input data. Latent factorization especially suffers from the sparseness of user-restaurant pairs. Because we have so few points, the network is seeking to minimize error on a very small number of ratings. This means the network has many possible combinations of latent values it can use to reconstruct the known values, which leads to large errors. For example, if the matrix is seeking to reconstruct a row with eight known rating values and two missing values, the latent values it will use must be very specific to minimize error - this would give us an increased confidence that the network has

learned something useful about the relationships between ratings, and that the missing values it fills in are close to accurate. If the row instead has two known values and eight missing values, the latent values the network can learn to reconstruct those two ratings become more varied.

Collaborative Filtering and Latent Factorization Limitations

One limitation of the collaborative filtering model is that it suffers from the Cold-Start Problem. This refers to the fact that users or restaurants without any ratings in their vectors (if they are brand new to Yelp, for example) were unable to contribute to the model. Using this method, without ratings, we have no way to gauge similarity between this user/restaurant and other users/restaurants in the dataset, which means it won't be incorporated in rating predictions at all. In addition, we only used 1 million data points in our model, which meant the number of ratings per user and per restaurant was relatively low, which affects the similarity measure. Another limitation is that both models have a high computational cost. Running the models requires lots of time for collaborative filtering and GPU support for latent factorization.

Future Work

In the future, model performance could be improved by looking at other user/restaurant features to determine similarity, as this would help to mitigate the Cold-Start Problem. It may also help to use more data when running the models and to look into additional GPU resources for all three models.

Conclusion

After predicting user ratings of a restaurant using Item-Based and User-Based collaborative filtering as well as latent factorization, we concluded that the best model was the weighted-sum approach to user-based collaborative filtering, as it had the smallest RMSE loss of 1.35 stars. This model had somewhat decent accuracy, although we would not feel confident about our ability to consistently recommend restaurants that users will enjoy. Data limitations, time constraints, and computing costs severely impacted the performance of these models, but based on our preliminary results, we are fairly confident that our approach would yield better results if these issues were addressed.