Large-scale data analysis
2024

Petya Ivanova Petrova

BSc in Data Science
IT University of Copenhagen
pety@itu.dk

Table of Contents

A) Scalable data processing

A.1) Businesses reviewed by more than 5 influencers

The Yalp dataset consists of 3 entities: businesses(bu), users(us), and reviews(re), which are all linked by a common attribute such as the *business_id*, *users_id* and *reviews_id*. Through these attributes, it was possible to establish connections between them and create data frames. To find all businesses that were reviewed by more than 5 influencers, the very first step is to define who an influencer is from the user's data frame. From the Yalp assignment, an influencer was defined as someone who has written more than 1000 reviews. Therefore, I found all the *users_ids*, who wrote more than 1000 reviews, by writing this code as shown in Figure1:

```
influencers = us.filter(col("review_count") > 1000).select("user_id")

# Count the influencers who have written more than 1000 reviews.
count_influencers = influencers.count()

# Print the result
print("Number of influencers who have written more than 1000 reviews:", count_influencers)
```

```
Number of influencers who have written more than 1000 reviews: 1949
```

FIGURE 1 INFLUENCERS

Then I joined the influencers data frame with the reviews one on user_id. This way I filtered all the users with more than 1000 reviews and their actual reviews. Then I joined all the 3 tables, which resulted in a new data frame called *business_reviews_users*. Then I counted how many unique users have reviewed each business, which results in a new data frame called: *business_count*, having a new column called *infl_count*. Then this data frame was filtered to get all the businesses with more than 5 influencers. Then I counted them, which resulted in 4365. Figure 2 shows the code of how I have achieved that.

```
business_reviews_users = bu.join(re, "business_id").join(influencers, "user_id")
business_count = business_reviews_users.groupby("business_id").agg(f. countDistinct("user_id").alias("infl_count"))
bus_5_infl = business_count.filter(business_count.infl_count > 5).select("business_id")

bus_5_infl.count()
```

FIGURE 2 BUSINESSES WITH MORE THAN 5 INFLUENCERS

A.2) Language Authenticity Comparison

The dataset includes 17 million restaurant reviews, from more than 30 different countries. (Kay)To answer the question if there is a difference in the amount of authenticity language used in the different areas, the very first step was to create a data frame, that filters out all the restaurant reviews that include the following words “legitimate” and “authentic”, since these words are considered the authenticity language, as shown on Figure 3.

```
#Filter out all the restaurants
all_df = re.join(restaurants , re['business_id'] == restaurants ['business_id'], 'inner') \
         .join(us, re['user_id'] == us['user_id'], 'inner') \
         .select(re['stars'], re['text'],restaurants['categories'],restaurants['state'],restaurants['city'])
```

FIGURE 3 JOINING THE TABLES

Then I grouped by state and city and I counted the the number of reviews that contained the authenticity language, as seen in Figure 4.

```
cube_df = filtered_df.cube("state", "city").agg(count("*").alias("count"))
```

FIGURE 4 GROUP BY STATE AND CITY

Then I ordered the data frame based on the number of reviews (count) in descending order, to get the top 20. My goal was to find the areas, where the authenticity language is used the most and compare them. Figure 5 shows the result of that:

```
+-----+---------------+-----+
|state|           city|count|
+-----+---------------+-----+
|   PA|   Philadelphia|18054|
|   FL|          Tampa| 9581|
|   LA|    New Orleans| 7500|
|   TN|      Nashville| 6743|
|   AZ|         Tucson| 6320|
|   IN|   Indianapolis| 6258|
|   NV|           Reno| 4380|
|   CA|  Santa Barbara| 3980|
|   MO|    Saint Louis| 3924|
|   AB|       Edmonton| 1918|
|   FL|     Clearwater| 1722|
|   ID|          Boise| 1533|
|   FL|Saint Petersburg| 1303|
|   LA|       Metairie|  961|
|   NV|         Sparks|  956|
|   DE|     Wilmington|  895|
|   FL|        Brandon|  893|
|   MO|      St. Louis|  871|
|   FL| St. Petersburg|  822|
|   TN|       Franklin|  765|
+-----+---------------+-----+
```

FIGURE 5 TOP 20 CITIES

As Figure 5 suggests, the area, where the authentic language is used the most is in Philadelphia, having 18,054 reviews, whereas in Tampa, the second usage of that language is used almost two times less. The northern parts of the USA include the following states: Philadelphia, Wilmington, Boise, and Indiana. By looking at the top 20, the total number of reviews containing the authenticity

language in the Northern States sums to 26,740. The southern states include Florida, Louisiana, Tennessee, Arizona, Nevada, California, and Missouri, where the total number of reviews results at 39,118. From these numbers, it is obvious that the Southern states have a significantly higher total count of reviews compared to the Northern States, even though the highest number of reviews for a single city is in Philadelphia located in Pennsylvania, which is a Northern state.

A.3) Hypothesis Testing

According to a study of the Yalp dataset, the word authentic has different meanings across different types of cuisines. For example, for Asian cuisines such as Thai, Japanese, Chinese, Korean, Indian, Mexican, and South Latin America, this term is believed to be used with a negative context. For instance, the term relates to characteristics such as dirt floors, plastic stools, and others. In contrast, Western and European cuisines such as French, Italian, Mediterranean, and others are related with a positive association such as charming service, creative menu, and others (Kay).To investigate this question, I have created the following hypothesis:

H0: There is NO significant difference in the use of authentic language and negative words between Asian and European cuisines.

H1: There is a significant difference in the use of authentic language and negative words between Asian and European cuisines.

To investigate this, first, I have filtered all the restaurants and the reviews containing the authenticity language. Then I split the data frame into two: one for European and one for Asian cuisines, based on the category, as shown in Figures 6 and 7 below.

```
european_df = filtered_df.filter((col("categories").contains("French")) |    (col("categories").contains("Italian")) |    (col("categories").contains("Mediterranean")) |
    (col("categories").contains("European")))
```

FIGURE 6 EUROPEAN DATA FRAME

```
asian_df = filtered_df.filter(
    (col("categories").contains("Thai")) |
    (col("categories").contains("Japanese")) |
    (col("categories").contains("Chinese")) |
    (col("categories").contains("Korean")) |
    (col("categories").contains("Indian")) |
    (col("categories").contains("Mexican")) |
    (col("categories").contains("Soul")) |
    (col("categories").contains("Latin American")) |
    (col("categories").contains("Asian"))
)
```

FIGURE 7 ASIAN DATA FRAME

After that, I took a random subset of both types of cuisines (both data frames), by taking 50 reviews of each. Figure 8 shows the code I have used to achieve that.

```
sampled_european_df = european_df.sample(False, 0.1)
sampled_european_df = sampled_european_df.limit(50)

sampled_asian_df = asian_df.sample(False, 0.1)
sampled_asian_df = sampled_asian_df.limit(50)
```

FIGURE 8 SAMPLING

Then I used a library called TextBlob for sentiment analysis, which assigns positive, negative, or neutral scores to each review. The average sentiment scores are as follows:

```
Average sentiment scores for Asian cuisines
Positive score: 0.20496368239812873
Negative score: -0.017286704808065025

Average sentiment scores for European cuisines:
Positive score: 0.27702170666841436
Negative score: -0.029861111111111116
```

FIGURE 9 AVERAGE SENTIMENT SCORES

The average positive overall score for European cuisines 0,27 is slightly higher compared to the Asian cuisines 0,20, as shown in Figure 9. On the other hand, the average negative score for the European cuisine -0,02 is slightly higher compared to the Asian cuisine -0,01. Based on these results, the difference in average negative scores between both types of cuisines is quite low, even though it is a bit higher for European cuisines. Therefore, I fail to reject the null hypothesis, meaning that there is no significant difference in the use of authentic language and negative words between Asian and European cuisines.

A.4) ML Prediction Model

The goal of this model is to predict the review rating based on the written text. Firstly, I filtered out my joined data frame to get only the restaurants, as seen in Figures 10 and 11.

```
all_df = re.join(restaurants , re['business_id'] == restaurants ['business_id'], 'inner') \
         .join(us, re['user_id'] == us['user_id'], 'inner') \
         .select(re['stars'], re['text'])
```

```
all_df.show()
```

```
+-----+--------------------+
|stars|                text|
+-----+--------------------+
|  4.0|Had a really enjo...|
|  3.0|The veggie tacos ...|
|  2.0|Weird place. It i...|
|  1.0|Walk another bloc...|
|  4.0|We went in there ...|
|  1.0|These wings were ...|
|  5.0|Always great food...|
```

FIGURE 10 JOINED TABLES

```
restaurants = bu.filter(bu.categories.rlike("Restaurants"))
restaurants.show()
```

FIGURE 11GET ONLY RESTAURANTS

The next step was to sample the dataset. I decided to take 1500 reviews. Then I tokenized the reviews; shown in Figure 12.

```
tokenizer = Tokenizer(inputCol="text", outputCol="tokens")
reviews_tokenized = tokenizer. transform(all_df)
```

FIGURE 12 TOKENIZER

The models do not understand text, so I used HashingTF to convert these tokens into numerical values. I have found two ways of doing that: by word2vec and hashingTF layer. I decided to use hashing because of its simplicity. The way that the hashingTF layer works is by converting each token into a number using a hash function and calculating how many times each word appeared in the whole dataset (Spark Apache). After transforming the data with IDF, I used a Vector Assembler to combine these features into a single vector, just so I could train the models with them; as shown in Figures 13,14 and 15.

```
# Token to Numerical Features
tokens2numb = HashingTF(inputCol="tokens", outputCol="NewFeatures", numFeatures=500)
#transformer = IDF(inputCol="NewFeatures", outputCol="NewFeatures1")
```

```
model = transformer.fit(tokens2numb.transform(reviews_tokenized))
transformed_data = model.transform(tokens2numb.transform(reviews_tokenized))
```

FIGURE 13 TURN TOKENS INTO NUMERICAL VALUES

```
assembler = VectorAssembler(inputCols=["NewFeatures"], outputCol="features_vector1")
transformed_data = assembler.transform(transformed_data)
```

```
transformed_data.select("features_vector1").show()
```

FIGURE 14 ASSEMBLER

```
# Show the transformed data
transformed_data.show()
```

```
+-----+--------------------+--------------------+--------------------+
|stars|                text|              tokens|         NewFeatures|
+-----+--------------------+--------------------+--------------------+
|  4.0|Had a really enjo...|[had, a, really, ...|(500,[7,8,17,23,4...|
|  3.0|The veggie tacos ...|[the, veggie, tac...|(500,[7,17,68,74,...|
|  2.0|Weird place. It i...|[weird, place., i...|(500,[17,19,22,24...|
|  1.0|Walk another bloc...|[walk, another, b...|(500,[12,17,33,35...|
|  4.0|We went in there ...|[we, went, in, th...|(500,[8,17,69,76,...|
|  1.0|These wings were ...|[these, wings, we...|(500,[17,18,29,43...|
|  5.0|Always great food...|[always, great, f...|(500,[0,11,17,53,...|
|  1.0|I was very disapp...|[i, was, very, di...|(500,[8,17,48,62,...|
+-----+--------------------+--------------------+--------------------+
```

FIGURE 15 TRANSFORMED DATA

Then I split my data into train, dev, and test data. Look at Figure 16.

```
# Split the data  (60%, 20%, 20%)
train_data, test_data, dev_data = transformed_data.randomSplit([0.6, 0.2, 0.2], seed=42)
```

FIGURE 16 DATA SPLIT

I have trained a Decision Tree model to see which features out of five thousand, are the most important as shown on the plot.

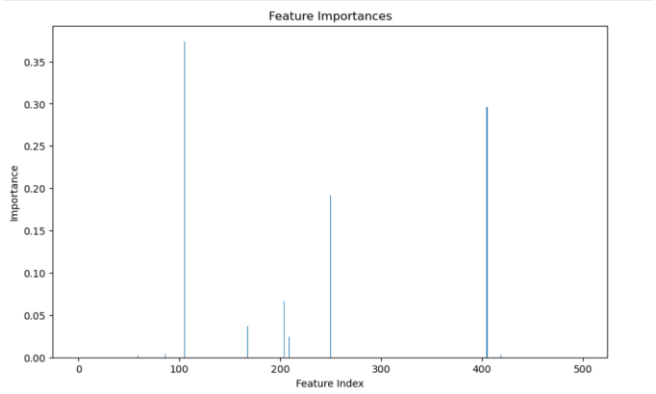

FIGURE 17 FEATURE IMPORTANCE

```
Sorted Features by Importance:
Feature Index: 105, Importance: 0.37363327436389016
Feature Index: 405, Importance: 0.29579557929147987
Feature Index: 250, Importance: 0.19189324200599797
Feature Index: 204, Importance: 0.06662988778194603
Feature Index: 168, Importance: 0.03731671568936724
Feature Index: 209, Importance: 0.024405505300101295
Feature Index: 86, Importance: 0.004076339648210463
Feature Index: 419, Importance: 0.0035657212871935177
Feature Index: 59, Importance: 0.0026837346318134827
```

FIGURE 18 FEATURE IMPORTANCE INDEXES

As the list suggests, the following features on idex:105,405,250,204 have the highest impurity, therefore I have decided to train my future models with the mentioned features.

```
feature_indices = [105, 405, 250, 204]
selected_feature_names = [f'feature_{idx}' for idx in feature_indices]
```

FIGURE 19 SELECT FEATURES

Then I trained Linear Regression and Random Forest models with these features. I decided to use the Mean Squared Error for this task since it is widely used to evaluate regression models. It indicates the difference between the predicted and the actual value squared, which can never be negative. The downside of that metric is that is sensitive to outliers, meaning that it gives more importance to them. However, in this case, since the MSE is quite low, I believe that due to its simplicity the compatible properties, it is suitable for this dataset.

Result: Random Forest dev data

```
Mean Squared Error on Dev Data for Random Forest: 1.5658551094031448
```

FIGURE 20 RANDOM FOREST DEV DATA

Result: Linear Regression dev data

```
Mean Squared Error on Dev Data for Linear Regression: 1.2698446482699537
```

FIGURE 21 LINEAR REGRESSION DEV DATA

Both models perform well on the dev data, but since Linear Regression slightly outperformed Random Forest, I have decided to test that model further on the test data.

Result: Linear Regression test data

```
Mean Squared Error on Test Data for Linear Regression: 1.2573355385510177
```

FIGURE 22 LINEAR REGRESSION TEST DATA

The results are satisfying because the MSE on both the dev and the train data are similar and quite low. However, the model was trained and tested on a small sample of the data, therefore a future improvement would be to train the model on the whole dataset.