Gabe Colton
Meredith Margulies
Ryan Vo

CIT 594 Final Project Menu Critic

Section 1 Requirements

Interfaces:

There were 3 main parts of this project, with a defined interface between each:

1. Query Locu to get Menu JSONs and filter them for valid restaurants with menus with descriptions
    - Steps 1 to 2 interface:
      DataInterface.java and RestaurantMenu.java these two classes abstracted away the format of the menu and allowed the second section to simply iterate through the menus, their items and their descriptions
2. Use NLP and Term Frequency to analyze menus and create vectors of independent attributes (number of adjectives/word, average length of word etc.)
    - Steps 2 to 3 interface:
      These vectors were written to a text file of the form label, features abstracting away how the data was created as well as its meaning.
3. The vectors were used as inputs into the Data Frame in R. This was then used to choose features for both the Naive Bayes Classifier as well as the Support Vector Machine.

Design Patterns: Factory, Singleton

We used Apache's ObjectPools and Multithreading library support, which required both Factory and Singleton Design Patterns. We wanted to limit the number of times a certain class (MenuAnalyzer) was created, as creation is relatively time intensive and it is needed for each thread we created. An object pool is a limited size pool of Singleton Objects and uses a Factory to create new objects when needed, in a controlled manner. This obviated the need to create new thousands of new MenuAnalyzer objects for each MenuDataObject, instead, limiting the number of new MenuAnalyzer objects to the total number of threads in the thread pool.

Section 2 Requirements

Data Structures: HashMap, TreeMap, Set, ArrayList, ConcurrentHashMap, Vector

Various types of storage, association between words and their frequency, between menu ids and their associated attribute vector etc.

MultiThreading/Advanced Topics

We multithreaded the analyzing of the menus, using a thread pool that handed out different menus to be analyzed, the results of which were combined into one overall ConcurrentHashMap <id, vector of attributes>. We also use MongoDB to store API calls made to Locu, and used

Apache Lucene and OpenNLP libraries to analyze the menus and used R to do statistical analysis (see section below).

Data Flow within Program

1) Data is taken from the Yelp dataset. Then using a local mongo instance, only restaurant data is returned into a JSON file.
2) Menu data is queried from Locu using the latitude and longitude coordinates given by yelp. The result is then parsed for the menu data and descriptions if available. Then the data is cleaned by removing any duplicate menus and deleting any empty menus that may have been created. The result of this is then printed to a JSON file.
3) The JSON file is parsed by the DataInterface class into RestaurantMenu objects that allow for storage and access of the menu fields. The class returns an ArrayList of RestaurantMenu objects.
4) For each RestaurantMenu object, we create a MenuDataObject that calculates and stores the metrics of each menu in a MenuAttributeVector. This vector contains the features we used to train our classifier model.
5) The MenuDataObjects are added to a thread pool to execute the calculations. The thread pool returns a List of MyTuple<Document, Document> objects. The Document objects contain the term frequencies for each menu. The first element in the tuple are the monogram term frequencies and the second element are the bigram term frequencies.
6) Each Document is organized into two ArrayLists, one for monograms and one for bigrams.
7) The ArrayList of monograms and bigrams are used to create two VectorSpaceModels. The VectorSpaceModel class generates the tfIdf weight in a Hashmap where the menu ID is mapped to a TreeMap of monogram/bigram mapped to their respective weight.
8) The weights of each monogram and bigram are then appended to the MenuAttributeVector for each menu. The MenuAttributeVector for each menu is stored in a MenuVectorMap data structure that maps the features vector for each menu to the menu's ID.
9) The data is finally output to a text file for statistical processing in R.

Summary

We set out to answer the question could tell if a restaurant was going to be good just by looking at the menu items and descriptions. At first, we could not exactly quantify what this meant, but we thought that metrics such as how "fancy" the words appeared or how long the item descriptions were would potentially indicate of the quality of the restaurant. We went about creating metrics to quantify a given menu, and decided on the following 8 metrics: for each Item and its Description, respectively, average word length, number of adjectives per word, number of adverbs per word, number of words per item/description. We also attempted to analyze the tf-idf

values for words within the menu as well as bigrams within each menu. Due to the number of unique bigrams, the computational time/power required to include this in the analysis was intractable without a distributed computing solution. We attempted to process our data with the Spark library on Amazon EC2, which uses Hadoop MapReduce to distribute the calculations in across clusters, but we found that it was harder than expected to carry-out. We decided to abandon the idea after incurring Amazon AWS costs, that literally made it expensive to play around with the data. Instead we decided to use R instead. To give a high-level overview of our statistical tests, our Naive Bayes Classifier was not very predictive, however if we allow restaurants to be "correct" if they are within 0.5 of their rating, the model is fairly predictive with 80% accuracy. Please see our statistical analysis section below for more details. We did not do any directional tests to say whether more or fewer adjectives, longer words or more words were suggestive of a higher rating

We thought it would be interesting in the future to be able to include a feature that allowed the user to give the name of a restaurant and for us to query locu for the menu, parse it, calculate its attributes, and then get the predicted rating based on our trained model. Given more time, I think this would be possible, but unfortunately we did not create a UI for this project.

Statistical Methods, Analysis, and Conclusions

We ran two different algorithms to create classifiers for our data. The first classifier was the Naive Bayes Classifier. We predicted rating based on nine factors: the average word length of words in the item, the number of adjectives per word in the item, the adverbs per word in the item description, the average number of words per item, the average word length of words in the description, the number of adjectives in the description, the adverbs per description, and the average number of words per description. Using the Naive Bayes Classifier, we predicted ratings for our entire dataset. The confusion matrix is shown here:

	Actual Ratings								
Predicted Rating		1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
	1.5	**4**	0	2	2	0	1	0	0
	2.0	0	**3**	0	0	2	1	1	0
	2.5	0	0	**7**	0	3	1	0	0
	3.0	1	1	11	**50**	30	23	2	0
	3.5	0	6	20	28	**109**	50	13	1
	4.0	0	4	12	13	33	**77**	4	0

	4.5	0	0	2	6	2	9	**18**	0
	5.0	0	0	0	1	2	0	0	**2**

This model ran with an accuracy of 48%. While this is not a great classifier, We found it interesting that most ratings were fairly close to their actual rating. If we count ratings within .5 rating away, this classifier is 80% accurate which is a significant predictor. We also used a Support Vector Machine to classify our data. The confusion matrix for the Support Vector Machine is:

	Actual Ratings								
Predicted Rating		1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
	1.5	**0**	0	0	0	0	0	0	0
	2.0	0	**0**	0	0	0	0	0	0
	2.5	0	0	**0**	0	0	0	0	0
	3.0	0	1	0	**6**	0	0	0	0
	3.5	4	10	42	84	**164**	104	23	3
	4.0	1	3	12	10	17	**58**	12	0
	4.5	0	0	0	0	0	0	**3**	0
	5.0	0	0	0	0	0	0	0	**0**

This model had 40% accuracy. However, this model has a wide range of wrong guesses because it almost exclusively classifies everything as either 3.5 or 4.0 making a less good model for our use.

Acknowledgements
Thank you to Swap, for his guidance and providing the code for TF-IDF analysis we modified for our purposes. Thank you to Theresa for helping with the early brainstorming stages of this project. And finally, thank you to Apache for seemingly having a library to do almost anything we needed to do.

User Guide

1) Add all .jar files to the java build path
2) Make sure En-pos-maxent.bin and FinalData.json are in the project directory
3) Run MainClass, which will create an output file called vector_output_R.txt
4) In R file, change file path to direct to vector_output_R.txt. Then run script file and output confusion matrices and accuracies will be output onto the console.

Files Included

.java
BigramGenerator
Corpus
DataInterface
Document
MainClass
MenuAnayzer
MenuAnalyzerFactory
MenuAttributeVector
MenuDataObject
MenuVectorMap
MyTuple
RestaurantMenu
VectorSpaceModel
RestaurantData
FileParser
MyFileParser

.R
NaiveBayes

.jar
Opennlp-uima
Opennlp-tools
Lucene-core
Lucene-analyzers-common
Lucene-analyzers-icu
Commons-lang
Commons-pool
Javax.json
Org.apache.http
org.json.simple

Required to Run Code

output.json
FinalData.json
En-pos-maxent.bin