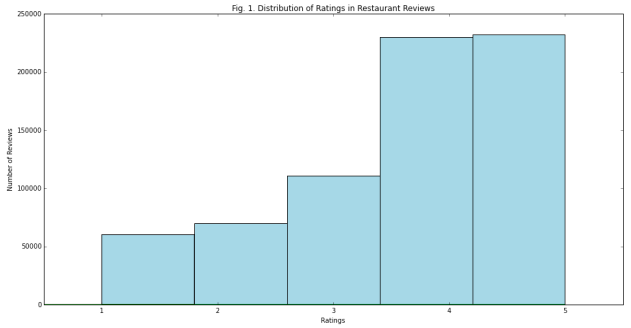
Data Mining Capstone Report
(Analysis of Yelp dataset)

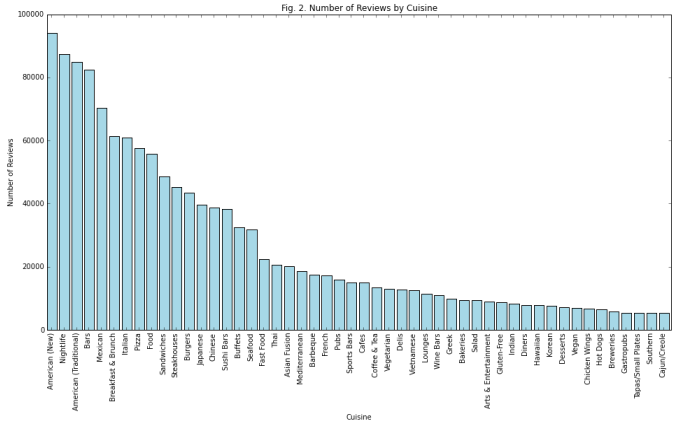
1. Summary

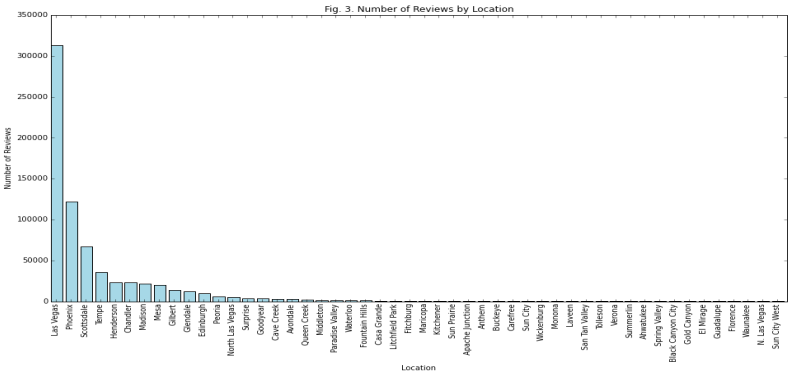
Note: this summary only shows the final charts for each task. For charts with intermediate results, please follow the links to individual task reports at the end of this document.

1.1. Task 1. Exploratory Analysis

The dataset contains 42,153 businesses, of which 14,303 are restaurants (identified by presence of 'Restaurants' category). After removing all businesses without 'Restaurants' category, the dataset has 1,125,458 reviews in total with 706,646 reviews for the restaurants.
Initial review of the categories for restaurants showed 240 categories with some of these categories assigned very few businesses, most with irrelevant labels like "Dry Cleaning & Laundry" and "Sporting Goods". Filtering out any categories with less than 10 businesses, leaves 123 categories and 14,057 restaurants with 702,816 reviews. Fig 1 shows the distribution of ratings ('stars' attribute) among reviews.





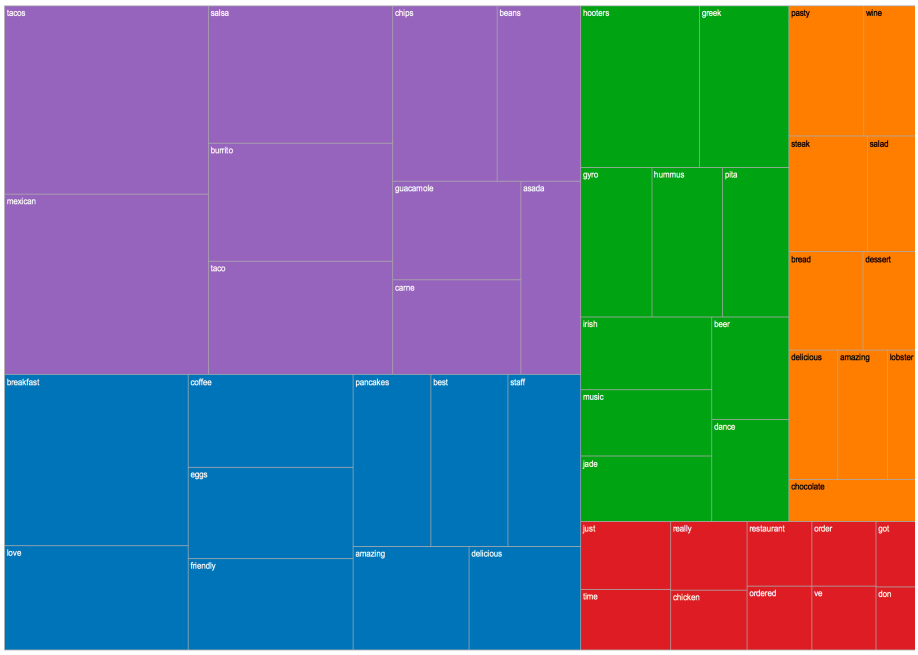


We see a very sharp drop in number of reviews after 15-17 most popular categories, which is to be expected – majority of restaurants stick to few most popular cuisines. Fig. 2 shows the distribution of reviews by category for top 50 categories.

Geographically, most of the reviews in the dataset are in Nevada and Arizona, with overwhelming majority in Las Vegas, NV, as shown in Fig. 3.

I performed topic extraction on the full set of reviews, using LDA. Words appearing in more than 30% of the reviews, words in Pyhon scikitlearn 'english' stopword list, and words appearing fewer than 5 times in all reviews were ignored.
LDA model was built for 5 topics, doing 10 passes and 1,000 iterations. Fig. 4 shows top 10 words for each of the topics. Words belonging to the same topic are grouped by color, and size of each word's container is proportional to its weight within the topic.
We can see several very clear topics here: Mexican cuisine (purple) - tacos, mexican, salsa, burrito, taco, beans, guacamole, chips, carne, asada; Breakfast (blue) - breakfast, pancakes, coffee, eggs; American cuisine (orange) - wine, steak, salad, pasty, bread, dessert; Mediterranean/Bars (green) - greek, gyro, hummus, pita, hooters, irish, beer, music, dance, and Other (red).

Fig. 4. Top 10 words for review topics



I did several runs of LDA to tune the selection. I tried cutoffs of words appearing in more than 50% of the reviews and words appearing less than 2 times in all reviews. I used 10 topics, on random subsets of the reviews, which gave much worse results. I also initially ran LDA with fewer iterations – also much worse results, as the number of iterations was insufficient for LDA to converge. However, even increasing number of iterations to 3,000 with 10 topics did not significantly improve results. I got the best improvement by reducing the number of topics to 5 and changing cutoffs to 30% and 5 occurrences respectively.

Next, I extracted from the full set of reviews only those for Mexican cuisine (70,406), and split them into two sets: positive reviews (ratings 4 and 5 – 44,157 reviews) and negative reviews (ratings 1 and 2 – 15047 reviews). I dropped reviews rated 3 (11,202 reviews) as neutral to get better separation between positive and negative ratings. On these two sets, I again used LDA for topic extraction with 3 topics, word cutoff at 50% and minimum 5 occurrences, and 1,000 iterations.

Fig. 5. Top 10 words for Mexican restaurants positive review topics

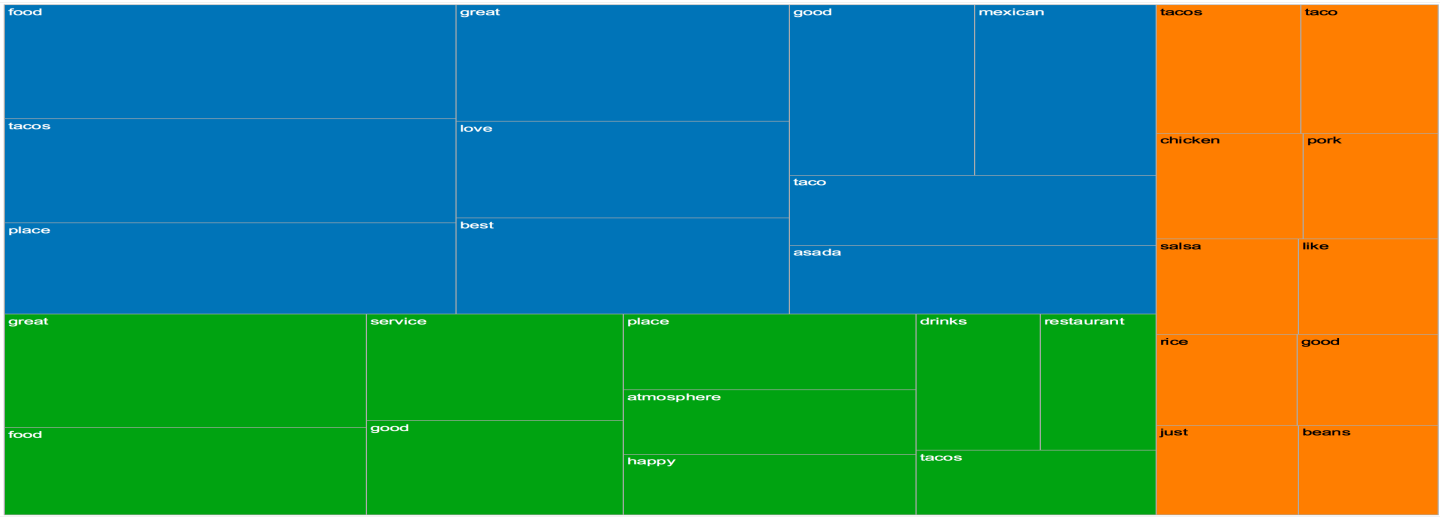
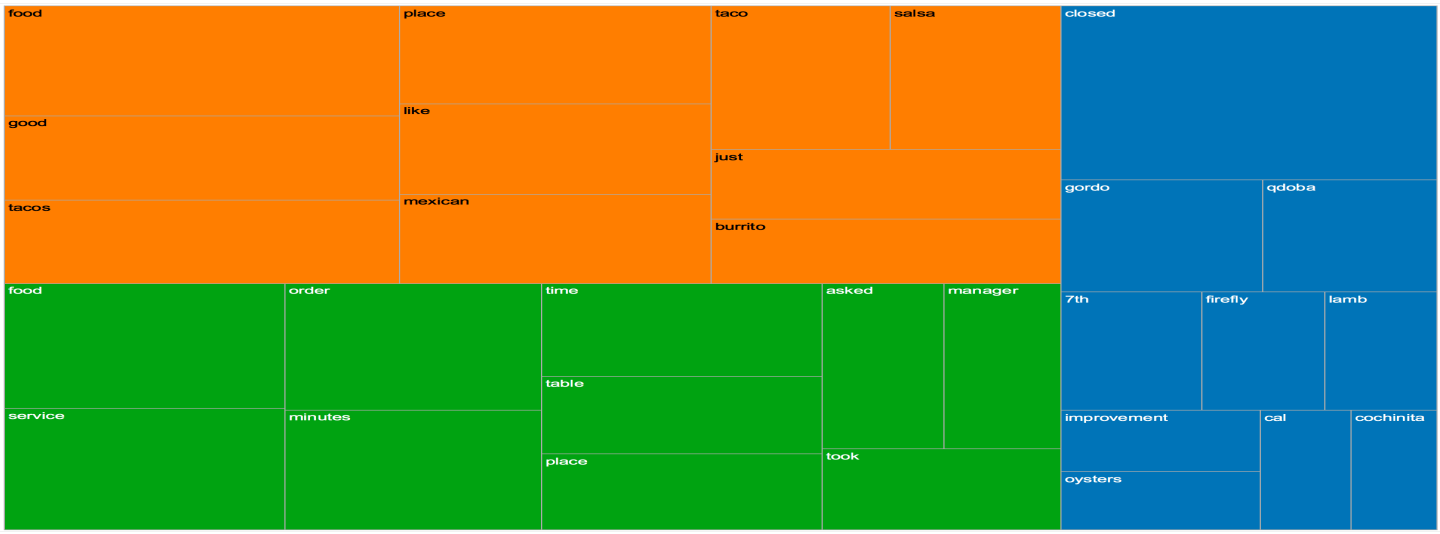


Fig. 6. Top 10 words for Mexican restaurants negative review topics



The results are in Fig. 5 and 6. In the positive reviews, it's pretty clear that one topic (blue) is about food quality, the other (green) is about service quality, and the third (orange) is "other". In the negative reviews, the split is similar: quality of service and wait times (green), quality of food (orange), which has words like "good" and "like", demonstrating the downside of 1-gram tokenization – they came out of "not good" and "don't like" in the review texts. Interestingly the third (blue) topic in negative reviews captures not just "other", but also reviews related to work hours and/or possibly restaurants that went out of business (highest weighted word is "closed").
Similar to previous topic extraction, I tried running this comparison with different parameters. I started with 5 topics, which turned out to be too many, as there was no clear distinction between topics.

1.2. Task 2. Cuisine Clustering

For similarity analysis I took top 50 categories (based on number of restaurants per category). From this point I will refer to these categories as cuisines. This left 13,647 restaurants with 681,590 reviews. Note that since some restaurants list more than one cuisine, when projecting reviews onto cuisines, reviews for those restaurants end up in more than one cuisine, which brings the grand total number of reviews across all cuisines to 1,332,178. It's important to note that while in previous section

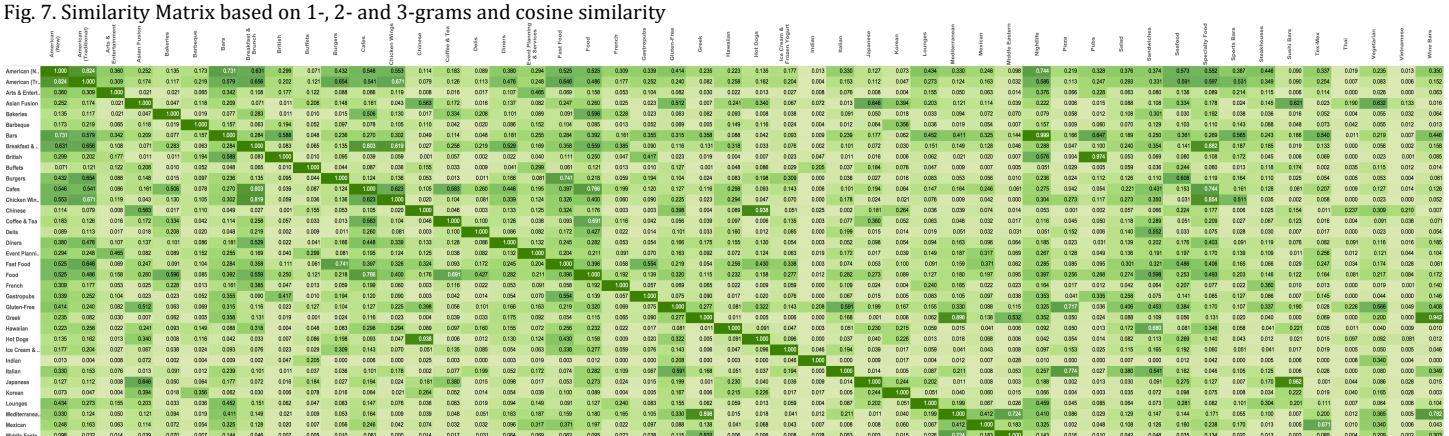
(1.1) we needed to avoid these duplicates to build a topic model across all reviews, here we are computing a similarity model between cuisines. Thus, same review present for more than one cuisine actually helps us identify similar cuisines, so this is a benefit, not a problem.

On import I converted each review to lower case, removed punctuation and stopwords, and applied stemming (using Porter algorithm).

I first concatenated all reviews for each category, ending up with 50 very large documents. Then I used a TF-IDF vectorizer to project these documents onto vector space (using up to 10,000 features, and ignoring words that appear in more than 50% of the documents or less than 2 times). Then I used LDA with 100 topics, running 10 passes with 1,000 iterations, and computed similarity matrix using Euclidean distance, converted to similarity as 1/(1+distance) to give measures in [0,1] range, on the results. There results were not very telling, so I looked at 3 possible improvements:

1. Using 1-, 2- and 3-grams for tokenization, when projecting documents onto vector space (varying text representation)
2. Using cosine similarity (varying similarity function)
3. Instead of 50 large documents, keeping each review as individual document, computing similarities between individual reviews, then aggregating from review-review similarities to cuisine-cuisine similarities (varying both text representation and similarity calculation)

Fig. 7. Similarity Matrix based on 1-, 2- and 3-grams and cosine similarity



Combining TF-IDF representation of unigrams, bigrams and trigrams with cosine similarity produced good results (Fig. 7). As expected, cosine similarity proved a much better measure. Euclidean measure is not very good for text-based vectors, because it focuses on magnitude of the vectors, which in case of text comparison is less important than the direction of the vectors. And direction is what we capture with cosine similarity. While using Euclidean similarity, going to n-grams gave us worse results than 1-grams, producing less pronounced groupings. Going to n-grams with cosine similarity gave the best result of the four.

For example, let's consider Indian cuisine (which rarely gets grouped with anything else). In 1-gram Euclidean, it got very high similarity with Hot Dogs (very strange), and somewhat smaller with Buffets and Vegetarian (probably the only 2 categories that make sense, since many Indian restaurants in US are buffet-style and there's a large selection of vegetarian dishes in Indian cuisine). In n-gram Euclidean, it's marked as similar to almost every category. In 1-gram cosine, it's not much different from 1-gram Euclidean, but in n-gram cosine similarity for Indian cuisine, Hot Dog is close to 0 and Vegetarian is the top similarity, with Buffet as 2nd, which looks quite good. To help better identify top similar cuisine pairs, I produced a different visualization (Fig. 8), showing cuisine pairs with similarity above 0.8.

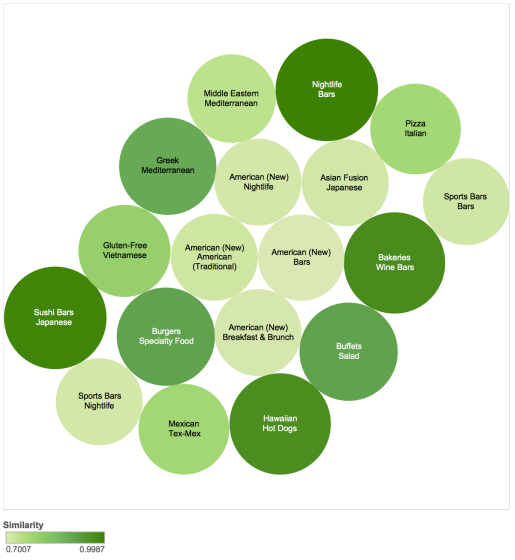
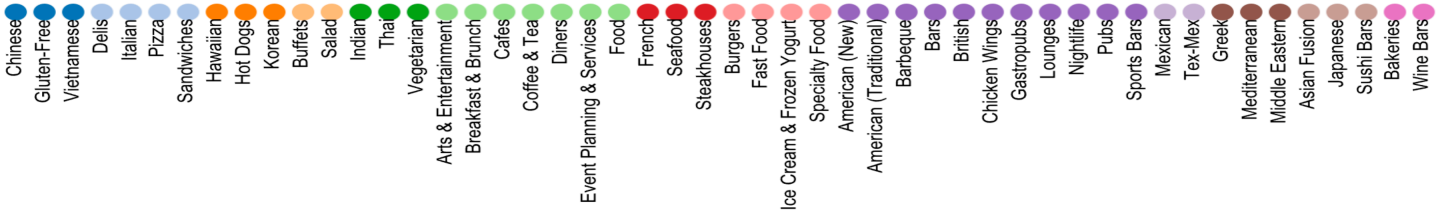


Fig. 8. Bubble chart of most similar cuisine pairs (n-grams and cosine similarity)

Producing similarity matrix between individual reviews, to aggregate similarities to category level from individual review pairs, has proven computationally prohibitive (1,332,178 reviews would mean n*(n-1)/2 = 887,348,445,753 similarities to compute. I ran it for 20 hours on my laptop before having to kill it. This is a good calculation to run on a distributed cluster, since it's easy to parallelize, but I don't have one at hand).

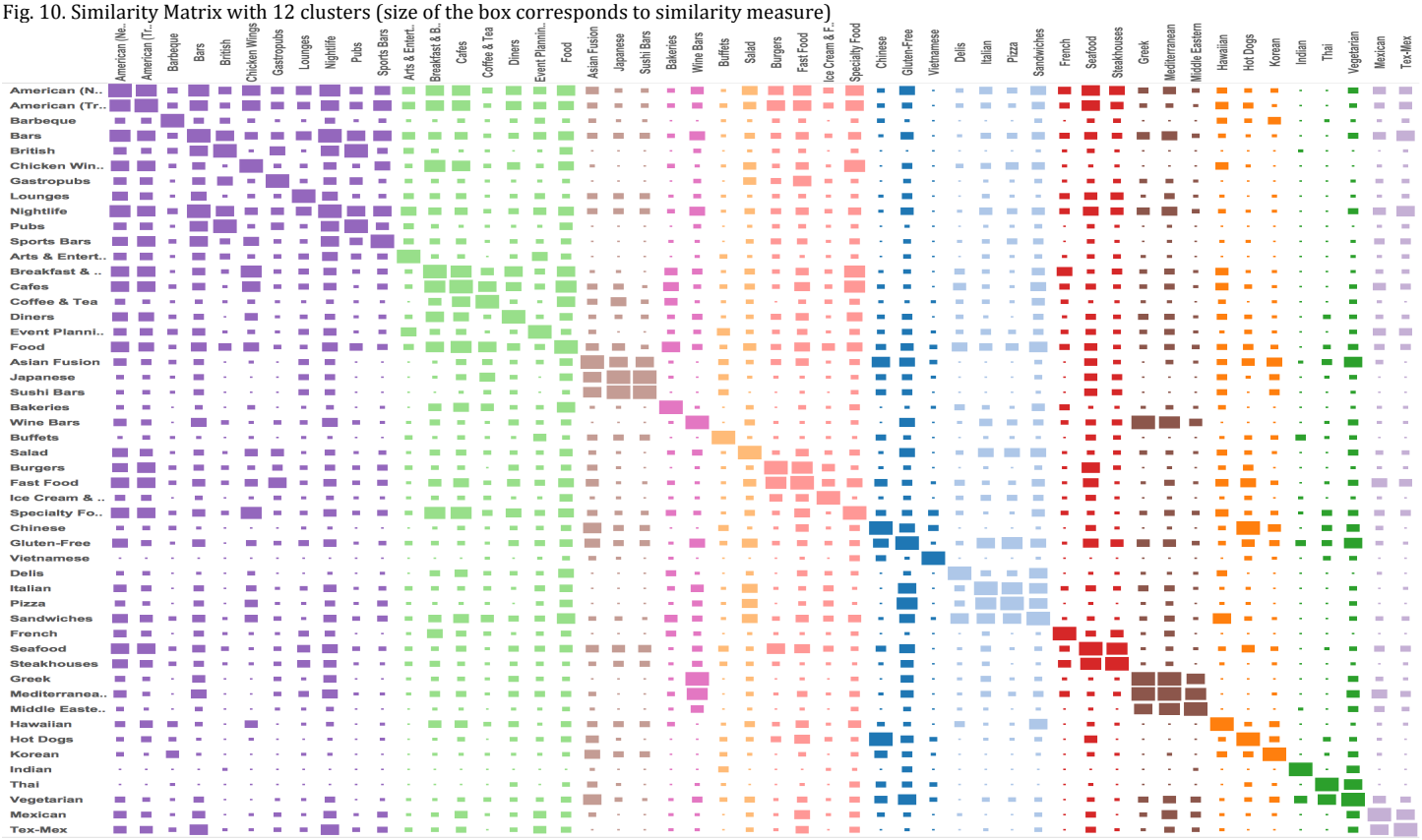
Having a pre-computed similarity matrix is a good input for clustering. First I used Affinity Propagation algorithm, which automatically determines the number of clusters by iteratively examining each pair of cuisines to determine if one is a good representation of the other, until convergence. The algorithm resulted in 12 clusters, as follows (Fig. 9). It's not a bad split, but I also wanted to try forcing the number of clusters. So, I used KMeans (by way of passing my pre-computed similarity matrix to Spectral Clustering algorithm), with 5, 6, 7 and 10 clusters.

Fig. 9. 12-cluster grouping with Affinity Propagation



The most interesting observation here is that Greek/Mediterranean/Middle-Eastern is the most stable cluster that gets identified early on and stays a separate cluster. And both algorithms identify it. Comparing the clusters, the 12-cluster split made by Affinity Propagation made the most sense to me. You can see very clearly identified groups (from right to left on Fig. 7): Asian, Lunch/Pizza, Lunch, Indian, Generic Service Categories, Dinner, Lunch/Snack, Bars, Mexican, Mediterranean, Japanese, Other.

Fig. 10. Similarity Matrix with 12 clusters (size of the box corresponds to similarity measure)



Application of cluster colors to similarity matrix is shown in Fig. 10. In this chart, the cuisines are sorted by cluster affinity and we can clearly see the difference in groupings along the diagonal.

How is doing this analysis useful? Well, we could use it for improving a search system, for example. When someone searches for a restaurant for specific cuisine, and there are not enough restaurants meeting the search criteria (not enough close by, or not enough high-rated restaurants), we could suggest restaurants with cuisines from the same cluster that meet other search criteria. Looking at the clusters helps us see that someone looking for a Pub is much more likely to find a place they like, if we also show Bars, or that someone looking for Middle-Eastern cuisine is much more likely to find a place, if we also show Mediterranean.

1.3. Task 3. Dish Name Detection

Dish name detection was performed on Indian cuisine. The dataset contains 202 Indian restaurants. After restricting businesses only to 'Indian' category, the dataset has 8,230 reviews for the Indian cuisine restaurants.

Starting with list of dishes that was provided to us, I decided to extend it by combining results of several methods into two stages: phrase detection (ToPMine, SegPhrase and sidekick) and phrase cleanup (common phrases detection and removal).

First, I used ToPMine algorithm (based on *Scalable Topical Phrase Mining from Text Corpora* paper by Ahmed El-Kishky, Yanglei Song, Chi Wang, Clare R. Voss and Jiawei Hanto) to create a bag-of-phrases. The algorithm combines frequent phrase mining stage that collects statistics for candidate phrases, weeding out phrases that fail minimum frequency criteria with phrase construction phase that re-builds phrases from bottom up, using most significant candidates (phrases of shorter length). This is a very interesting approach, giving higher performance than regular n-gram approach, and potentially more meaningful results due to phrase construction from highest significance candidates rather than just frequency of the outcome. There is also topic modeling part of the algorithm, which I will not cover, since for this exercise I am only interested in bag-of-phrases creation. Application of ToPMine resulted in a set of 1,588 phrases. Review of the phrases showed that some phrases were related more to quality than to dishes ("pretty good", "good food", etc.). I constructed a special set of stopwords for words that should not appear in dish names: ['indian', 'india', 'food', 'lunch', 'dinner', 'we', 'like', 'love', 'good', 'great', 'be', 'is', 'was', 'to', 'are', 'recommend', 'recommended', 'ordered', 'have', 'had', 'back', 'staff', 'service']. I excluded every phrase that contained any of these words, and merged the resulting list with the list from Annotations file (removing any duplicates), ending up with 838 phrases, which contained dish names with very few exceptions. Submitting this file initially gave me a score of 8.

Next, I used SegPhrase tool (https://github.com/shangjingbo1226/SegPhrase, based on *Mining Quality Phrases from Massive Text Corpora* paper by Jialu Liu, Jingbo Shang, Chi Wang, Xiang Ren and Jiawei Han). The algorithm is also multi-phase: first it parses the corpus, computing phrase frequencies and generating candidate phrases; then it estimates phrase quality by calculating a number of features for the phrases, based on concordance (features quantifying probability that elements of the phrase occurred together as a phrase rather than as parts of different adjacent phrases) and informativeness (occurrence of stopwords within a phrase, average IDF of words in the phrase, and punctuation of the phrase – whether it's in quotes, brackets or capitalized), and uses a predictive model to calculate phrase quality; then it re-parses the corpus, assigning each frequent word occurrence to only one candidate phrase (segmentation), evaluating for segmentation that results in highest phrase quality, and re-calculating phrase frequencies according to segmentation (rectified frequencies); rectified frequencies result in new features that can be used in previously constructed predictive model to adjust phrase quality and perform new segmentation – this can be done for a number of iterations. This resulted in a set of 1,513 phrases. Review of the phrases showed similar issues to those in ToPMine output. Additionally, SegPhrase included phrases with both high-quality and low-quality scores in the output. Review has shown that quality score of 0.3 was a reasonable threshold.

I processed SeqPhrase output, removing phrases with scores below 0.3 and phrases that contained any stopwords from my special set, and merged the resulting list with the combined ToPMine/Annotations list (removing any duplicates), ending up with 1115 dishes. Submitting this file raised the score to 9.

Next I decided to use an approach that is not strictly an algorithm, but more of a design pattern for solving problems with dirty data (based on Strata talk *The Sidekick Pattern: Using Small Data to Increase the Value of Big Data* by Abe Gong). The main point of the pattern is to find an external, small, well-curated dataset that can be used in conjunction with the dataset one is trying to analyze. Well, what better source of Indian cuisine names than Wikipedia? Scraping the dish names from Wikipedia pages on Indian cuisine, I got a curated set of of 521 dish names. However, I needed to be sure that these dish names actually occur in the reviews. Examining some of the reviews, I saw that many dish names in the reviews had extra words inserted in the middle of dish name, some 2- and 3-word dish names were mentioned in the reviews with just one word (and not always the same word), some words in dish names were misspelt in some of the reviews. So, just matching full dish names to reviews to find which dishes from curated data set are mentioned in the reviews clearly would not work.

Instead, I created a bag-of-words from the reviews, and considered each dish in the curated data set matched if every word in the name of that dish was present in the bag-of-words. This left 201 dish names.

I merged these 201 dishes with the combined SegPhrase/ToPMine/Annotations list (removing any duplicates), ending up with 1259 dishes. Submitting this file gave me a score of 12.

The most interesting thing at this point was the difference between the curated dataset of 521 dish names and the mined dataset of 1259 dish names.

There are several reasons for this I identified:

- a lot of duplication comes from incomplete dish names in the reviews ("naan" and "naan bread", "chicken tikka" and "chicken tikka masala" and "tikka masala").
- dish variations contribute as well ("garlic naan")
- many non-dish names that got through my filtering ("huge fan", "main courses", "las vegas")

So for second stage (phrase cleanup), the first impulse may be to use only the intersection between curated dataset and mined dataset (201 dish names) and consider that to be the resulting list of dishes. However, that would be a mistake, because it would get rid of dish variations, and we cannot consider this to be a complete data set - there are many more dishes in Indian cuisine that are missing from the curated data set.

The second impulse is to try tuning ToPMine and SegPhrase to get rid of non-dish names. Both tools can be tuned, but we have to remember that neither one is designed specifically for dish-name mining. Both are designed for phrase mining and both will turn up phrases commonly used in the reviews.

So, we need a method to eliminate phrases commonly used in reviews that are not dish names. SegPhrase's ability to use wordnet corpus to limit candidate phrases to noun-based ones is a big help, but I did use it, and still SegPhrase's output contained fare share of non-dish names.
Using stop-words also helped, but manually constructing a stop-word list is a tedious exercise, and making such list exhaustive is harder than generating a full list of cuisines manually.
So, I needed to find a way to identify non-dish names (phrases related to food quality, getting a good table, service quality, etc.) automatically. It is reasonable to assume that these phrases would be common across multiple cuisines. So, similar to how IDF helps us get rid of terms that are frequent across entire corpus, we can use the rest of our corpus (reviews of other cuisines) to get rid of these common phrases. I chose a cuisine as distant as possible from Indian, so that there would be no dish names in common (Italian), and ran ToPMine and SegPhrase tools again, this time for Italian cuisine. Now, removing the phrases that occurred in both Indian and Italian sets of phrases from the Indian set, reduced the Indian cuisine dish list to 693 dishes, a set with much higher quality than previous one.

1.4. Task 4. Dish Popularity

The dataset contains examples of negative reviews that still mention specific dish very positively, while being negative about service quality. It also contains examples of positive reviews that mention specific dish negatively, while speaking positively of other dishes. While these examples do not represent majority of reviews, they occur frequently enough to be considered. Looking only at review rating would miss these cases and misclassify those specific mentions of a dish. So, we need to do sentiment analysis of individual sentences, to account for these cases.
Continuing to work with Indian cuisine, as a first step, I used a tokenizer to split each review into individual sentences, resulting in a set of 70,367 sentences.
I also looked at using POS tagger to identify simple parts of complex sentences, and split sentences like "The service was terrible, but the food was great" into separate sentences. I used OpenNLP toolkit for this, but the model in the toolkit was trained on news articles (with good grammar), and it did not work very well on yelp reviews. So, I had to abandon the idea of splitting complex sentences and stuck with individual sentences I already tokenized. Still, I think POS approach could work, given a good set of training data to train grammar model on.
To identify sentiment for each sentence, I used word2vec, combined with logistic regression:
Treating each sentence as a document, I trained a word2vec model on my set of sentences, projecting each word on 10,000 - dimensional vector space. Then, I created a feature vector for each sentence by averaging word vectors for every word in the sentence (w1+w2+…wn)/n, generating 10,000 features for each sentence.
To train logistic regression model, I needed a labeled set. Given that in most cases sentence sentiment is the same as review sentiment (as expressed by the review rating), I assigned initial sentiment label (0 for negative sentiment, 1 for positive) to each sentence, based on reviews (0 for 1- and 2-star reviews, 1 for 4- and 5-star reviews, and None for 3-star reviews), and used sentences with non-empty labels as a training set, training logistic regression model with sentiment label as the outcome.
Given fairly low accuracy of sentiment detection, I wanted to overturn rating-based sentiment labels only when the model was giving very high probability for opposite classification. So, I applied my logistic regression model to entire set of sentences and re-assigned sentiment labels as follows:

- If initial (rating-based) label was 1, and model classified the sentence as 1 with probability less than 30%, label was changed to 0
- If initial label was 1, and model classified the sentence as 1 with probability greater than 30%, label remained 1
- If initial label was 0, and model classified the sentence as 0 with probability less than 30%, label was changed to 1
- If initial label was 0, and model classified the sentence as 0 with probability greater than 30%, label remained 0
- If initial label was empty (3-star review), label was set to 1 if model classified the sentence as 1 with probability at or greater than 50%, and to 0 otherwise.

Based on review of a number of sentences where initial rating was overturned, this has made an improvement in sentence classification compared to initial rating-based assignment.
Now that I had a list of sentences with sentiment label assigned, I filtered out only the sentences that contained dish names (based on my list of dishes, created as described in 1.3), creating a mapping between a dish and its mentions. I removed dishes with fewer than 50 mentions, ending up with a list of 58 dishes, that I further filtered down to 53, removing last remaining non-dish names ('meats', 'look forward', 'looking forward', 'name', 'vegetarian dish')
There are several ways to rank dish popularity: by number of reviews where it is mentioned, by number of positive sentiment reviews where it's mentioned, by ratio of positive/negative sentiment mentions, etc. Of course, while every one of these is useful, every one is problematic in some way:

- Number of mentions would give undeserved high rating to the dishes with large number of negative mentions
- Number of positive sentiment mentions would rate highly dishes that have high number of reviews in general (and therefore high number of positive reviews), even if they have more negative reviews than positive
- Ratio of positive/negative mentions would not give us any indication of actual number of reviews.

We can combine several of these in visualization, e.g. showing number of reviews as size of the bar, and ratio of positive/negative sentiment (for actual calculation I added 1 to denominator to avoid division by 0, and also added 1 to enumerator to make the ratio equal 1, when the numbers of positive and negative reviews are equal) as color gradient (Fig. 11), or we can try to design a different popularity index metric that combines several items together:

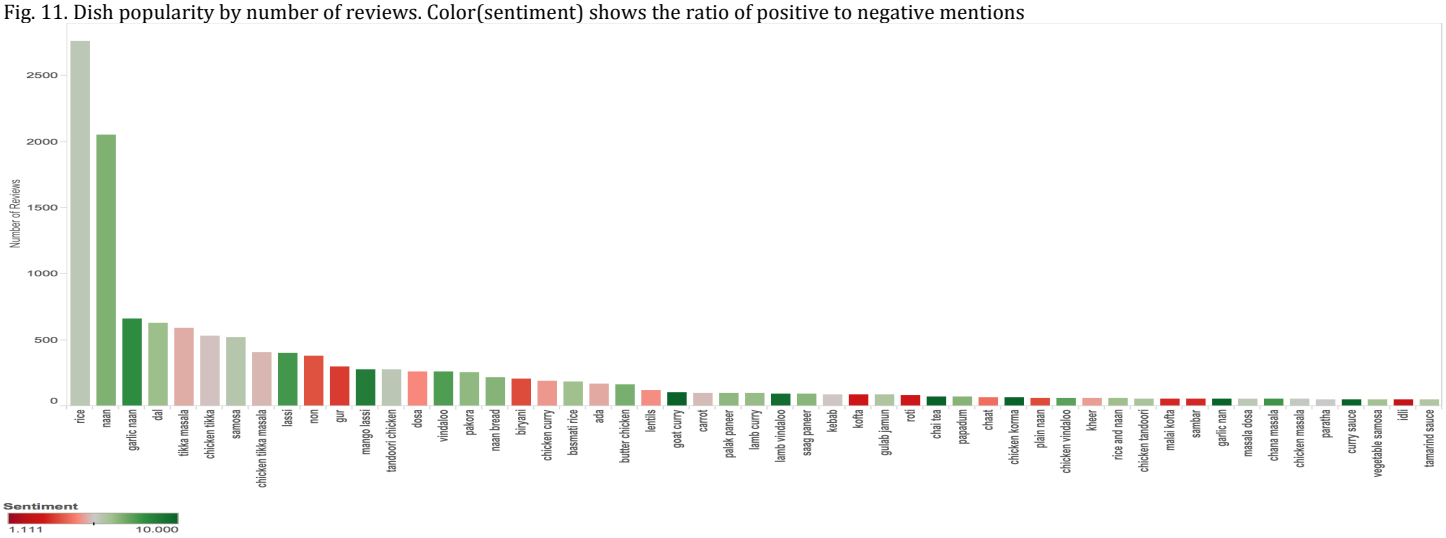
- We want to reward positive mentions (M_{pos}) and punish negative mentions (M_{neg}), but we also want to reward negative mentions (though in a lesser way), because any mention of the dish means that it's popular. This gives us a

component like (w_{pos}*M_{pos} + M_{neg})/(M_{neg}+1), where w_{pos} is the weight we assign to positive mentions. After some experimenting, w_{pos}=2 got good results. We add 1 to denominator to avoid division by zero.

- We want to reward higher dishes that get mentioned in more restaurants (R_{total}) than dishes that get mentioned a lot but in fewer restaurants. At the same time, we don't want to reward for negative mentions (R_{neg}) too much, similar to previous component. This gives us a component like R_{total}/R_{neg}.
- We want the measure to be between 0 and 1, so we divide by total number of restaurants (R) and total number of reviews (M_{total}). Note that R is different from R_{total} – they can be equal only if every restaurant in the data set has a review mentioning this dish.

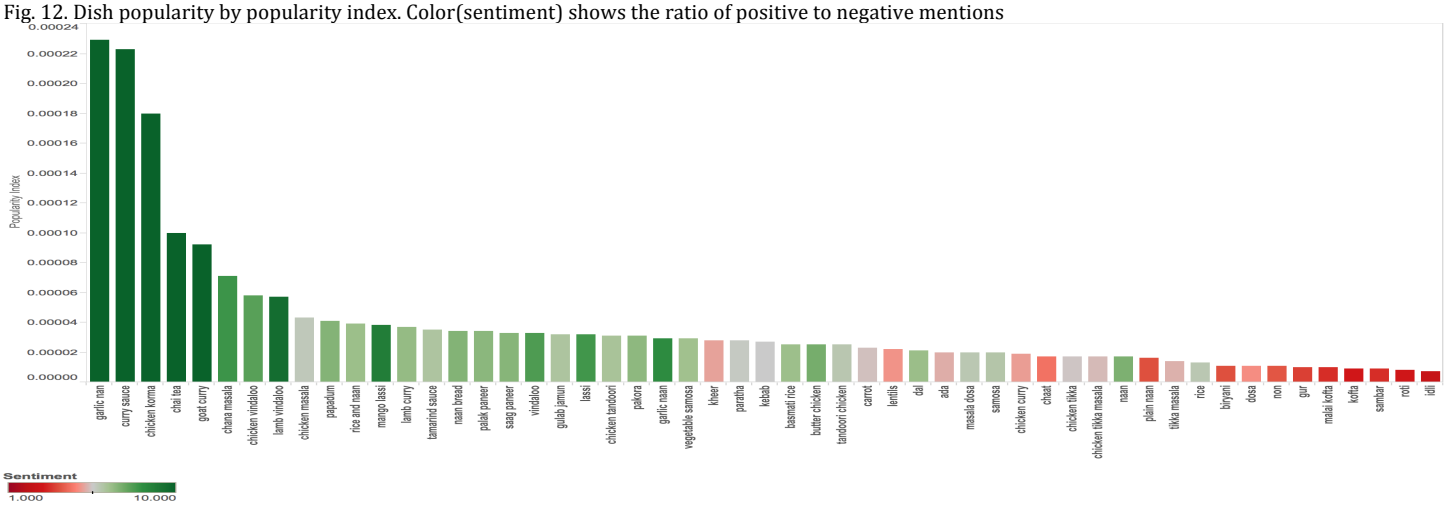
$$\frac{w_{pos} * M_{pos} + M_{neg}}{M_{neg} + 1} * \frac{R_{total}}{R_{neg}} * \frac{1}{R} * \frac{1}{M_{total}}$$

Fig. 11. Dish popularity by number of reviews. Color(sentiment) shows the ratio of positive to negative mentions



Now making a similar visualization, but using our popularity index instead of the number of reviews, we can see that it gives us a better view on dish popularity than either of the single metrics above did (Fig. 12). Note, I used same ratio for color as in Fig. 11 to allow easier comparison between the two visualizations.

Fig. 12. Dish popularity by popularity index. Color(sentiment) shows the ratio of positive to negative mentions



Looking at practical use for discovered dish popularity, we can look at dish combinations (similar to what we did in 1.2, looking at cuisine similarity). I looked at dishes that get mentioned together in the same review in more than 10 reviews (Fig. 13), and which dishes get mentioned together in the same restaurant in more than 10 reviews (Fig. 14).
We can use these to recommend a dish to try together with the dish someone is ordering. While some combinations are due to same dish getting spelled as partial name vs complete name (chicken tikka / chicken tikka masala, naan/garlic naan), some are actual combinations (naan/rice, chicken tikka/garlic naan, samosa/dal).

1.5. Task 5. Restaurant Recommendations

For each of the 53 selected dishes, I built a list of restaurants where this dish was mentioned, producing the data set with the following columns:

- Dish

Fig. 13. Dish co-occurrence in the same reviews

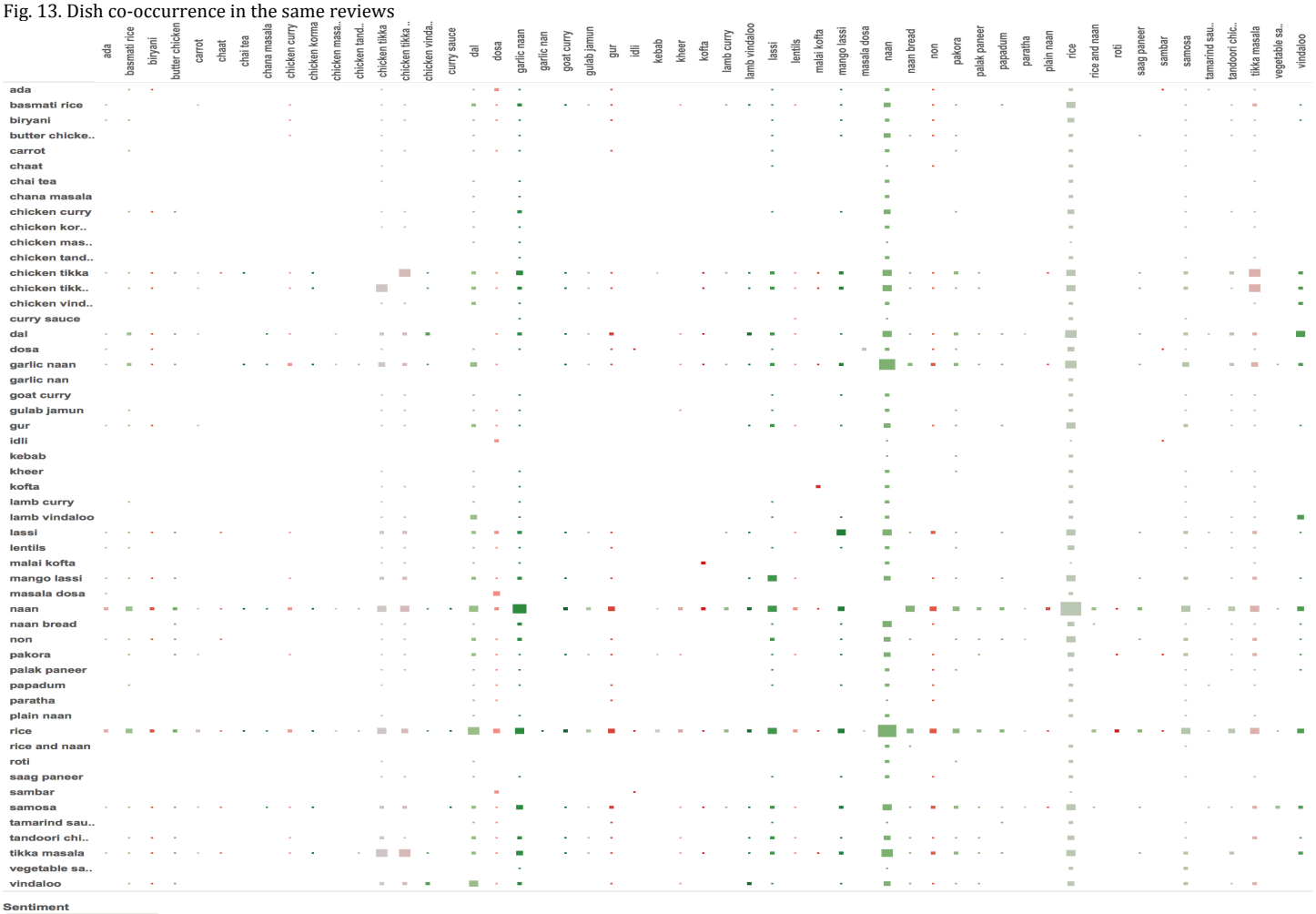
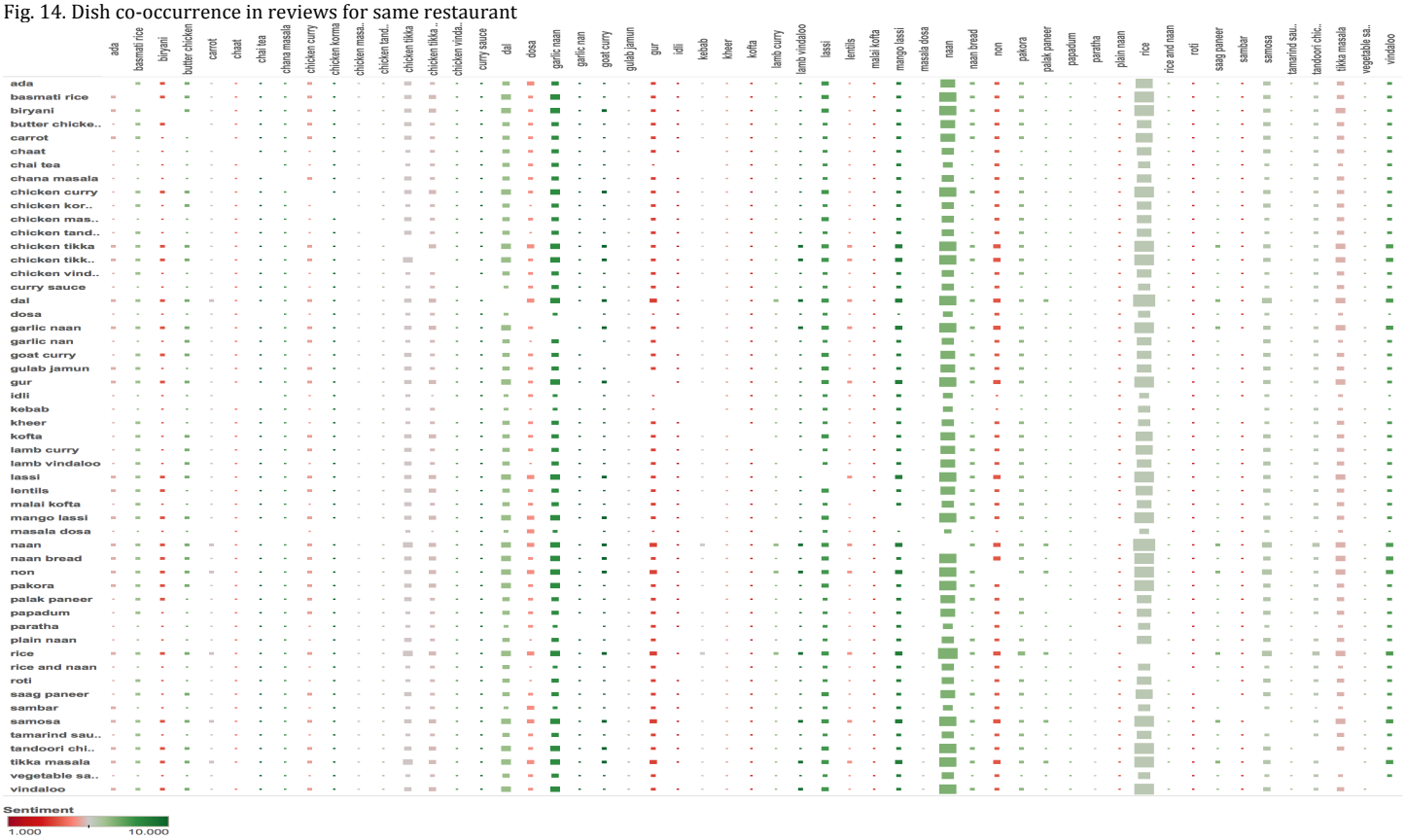
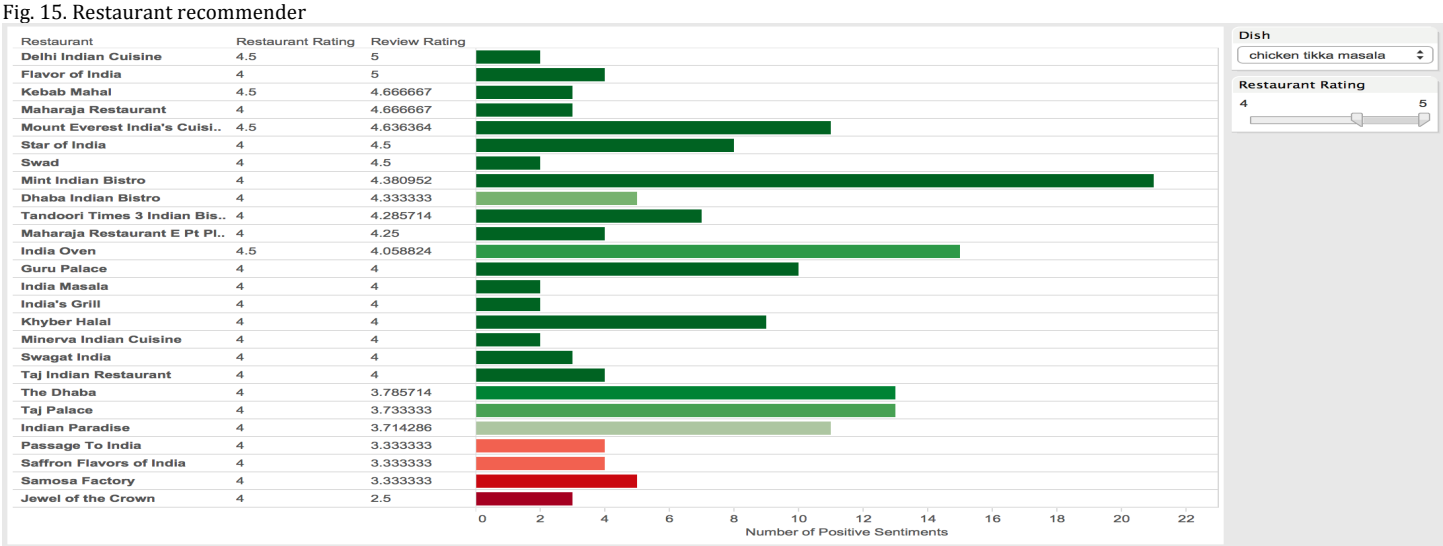


Fig. 14. Dish co-occurrence in reviews for same restaurant



- Restaurant
- Restaurant Rating
- Average Review Rating (of reviews that mention this dish)
- Average Positive Sentiment – % of number of sentences that mention this dish with positive sentiment out of total number of sentences that mention this dish
- Total Number of Reviews (for this restaurant with mention of this dish)
- Number of Positive Mentions (for this restaurant for this dish)

Fig. 15. Restaurant recommender



Since we only want to make positive recommendations, and we don't want to make them based on a single review, I filtered out any restaurant/dish combinations where Average Positive Sentiment was below 0.5 and any combinations that only had a single review. I then built an interactive visualization (Fig. 15), where the user can pick up a dish, and will see a list of recommended restaurants, with information about the restaurant rating, average rating for the selected dish in each restaurant and number of positive mentions for the dish, colored by average positive sentiment for the dish. Further, the user can narrow down selection of restaurants by selecting a range of restaurant ratings. Try it : https://public.tableau.com/profile/ml7356#!/vizhome/shared/9KP6FYQJC

1.6. Task 6. Hygiene Prediction

The dataset for this task contained full, concatenated text of the reviews. To convert them into features, I combined 4 text representation techniques: unigrams, bigrams, trigrams and 4-grams. First, I used a tokenizer to split text into sentences (to avoid creating n-grams across sentences). Then I removed all punctuation, removed stopwords, and applied Porter stemmer to each word. After that, I generated the 4 text representations in the form of document-term matrix. For training data, this resulted in 822,800 features. After removing n-grams that were too frequent (in more than 80% of the cases), and not frequent enough (single occurrences), I ended up with 58,192 features (where each feature is the number of occurrences of corresponding n-gram in reviews of given restaurant).

I also used additional features in the following way

- Cuisines

Training data set contains 68 cuisines. I added to the feature matrix additional feature corresponding to each of these cuisines, with value 1 if the restaurant offered this cuisine and 0 otherwise

- Reviews and Rating

I also added two additional features – number of reviews and average review rating

- Zip codes

For zip codes I used a different approach. Test data contained zip codes not present in training data, but adjacent to zip codes in training data. So, instead of creating features containing 1 for matching zip code and 0 for non-matching, I encoded distances: I found public data with latitude and longitude of each zip code, calculated distance between each pair of zip codes, and created a feature for each zip code in the data. The feature was equal either to 0 (if the restaurant was in this zip code), or to a distance to feature's zip code from the restaurant's zip code.

If I had latitude and longitude of each restaurant (like we had in the data set we used for previous tasks), then instead of calculating geographical distance I would have used Google Maps API to calculate driving and walking distances between restaurants instead, since you can often have situations when two restaurants in the same zip code are actually far apart, two restaurants in adjacent zip codes are actually across the street from each other, and two restaurants with small geographical distance between them are actually separated by a river, ravine, etc. and should not be considered close to each other. Missing this data, I stuck with geographical distances between zip codes.

I tried the following classification methods:

- Logistic Regression
- Support Vector Machines
- K Nearest Neighbors
- Multinomial Naïve Bayes
- Random Forest
- Gradient Boosting (aka Gradient Boosting Regression Trees)

Additionally, I used PCA to reduce the number of features to 500 principal components, and repeated a subset of the set of methods above on this new dataset (Logistic Regression has shown such poor results that I decided not to repeat it, and the implementation of Multinomial Naïve Bayes that I used did not work with negative values in some of the principal components).
For SVM, KNN, and Multinomial Naïve Bayes I used grid search with parameter variation and 3-fold cross-validation to find the best set of parameters for the model.
For all methods, I used 10-fold cross-validation to make preliminary estimate F1 on the testing data, additional test on 10% of the training data I held out as test set, and actual F1 from submission of prediction on test data as the final grade for the method.

Method	without PCA	with PCA
Logistic Regression	0.540943369886	N/A
SVM	0.546466897674	0.549156176476
KNN (k=10)	0.541819757481	0.543406876048
Multinomial Naïve Bayes	0.546978898362	N/A
Random Forest	0.557377544221	0.529792996762
GB	0.554302238913	0.53728487075

Logistic regression performed the worst, and that's not a surprise: it's not a good method for detecting anomalies, with no clear boundary, which is what we're doing in this task.
KNN did a little better, but still not too well – the more dimensions (features) there are for distance calculation, the more data you need to train KNN well, and it has to be fairly dense data too, otherwise it will have a problem differentiating distances within same class from distances between classes. Not surprisingly, with dimension reduction from PCA, KNN showed an improvement.
SVM and Multinomial Naïve Bayes have shown very comparable results. For SVM, after a grid search to optimize parameters, I used rbf (radial basis function) kernel with c=100 and gamma=0.001. Using a non-linear kernel clearly helped. In MNB, to help the algorithm perform on unbalanced test set, I set a prior with probability 90% for passing an inspection and 10% for failing. SVM is somewhat sensitive to situations when number of features is greater than number of cases, so there is an expected improvement after applying PCA.
Random Forest and GB are my go-to methods for classification problems and I expected them to perform best – exactly what happened. Given that both methods have built-in identification of most important features (random forest does feature pruning for every tree it builds, based on subset of cases and features selected for that tree, and GB also prunes features as it builds a shallow tree at every iteration to optimize the loss function). For that reason, both did better with the original set of features than with PCA-reduced set of features.

I looked at actual cases that got misclassified in my held out test set, and compared them to cases that got classified correctly. What I noticed was that most correctly classified cases had either at least one review that mentioned something negative about food quality or cleanliness of the place, or had a combination of the restaurant's cuisine being among several cuisines that had high percentage of places that did not pass inspection, and the restaurant was located in the zipcode with high percentage of places that did not pass inspection (or in zipcode adjacent to it). The cases that got misclassified seemed to have no indication in the reviews or in cuisine and location that there was a risk of failing inspection.

Just to see importance of text features compared to cuisines, ratings and geographical location, I built additional Random Forest model using only the additional features (no text features), and got F1 of 0.531039068474; I also built additional Random forest model using only the text features, and got F1 of 0.547831189377.

I see some possibilities for improving the results further from using LDA on n-grams to create clustering-based cuisine group features, and also to do co-occurrence frequency calculation for cuisines (which cuisines are offered together in the same restaurants) to create additional cuisine group features.

2. Project Highlights

This section is relatively small, because it refers to items already described in section 1, rather than repeating them.

2.1 Usefulness of results

- Results of topic detection (1.1) are useful, for example for yelp application designers. Splitting review rating into subcategories (service quality, food quality) automatically, based on detected topics allows for simpler user experience when rating a review: For users who want to get done with a review quickly, there's no need to set separate ratings for many different topics. But we can still show topic-specific ratings to users who are searching for a restaurant, identifying topics automatically. Optionally, identified topics can be labeled and offered as separate rating elements for users who are willing to provide topic-based ratings rather than a single one.

- Cuisine clustering results (1.2) could be used for improving a search system, for example. When someone searches for a restaurant for specific cuisine, and there are not enough restaurants meeting the search criteria (e.g., if there are not enough restaurants close by, or not enough high-rated restaurants), we could suggest restaurants with cuisines from the same cluster that meet other search criteria. Looking at the clusters helps us see that someone looking for a Pub is much more likely to find a place they like, if we also show Bars, or that someone looking for Middle-Eastern cuisine is much more likely to find a place, if we also show Mediterranean.

- Looking at practical use for discovered dish popularity (1.4), we can look at dish combinations (dishes that get mentioned together in the same review in more than 10 reviews (Fig. 13), and dishes that get mentioned together in the same restaurant in more than 10 reviews (Fig. 14). We can use these to recommend a dish to try together with the dish someone is ordering.

- Restaurant recommendation (1.5) has obvious practical use, and I built an interactive visualization (Fig. 15), to demonstrate it.

- Hygiene prediction results also have practical use. I see three possibilities: We can give early warning to restaurant owners that they are likely to have a hygiene problem in their restaurant that they need to deal with; we can downgrade the restaurant where there's a likely problem to help our users avoid such places; we can alert regulators (I have to say, this one seems a bit creepy, but it's still a possible use, so I'll mention it) about places where they may want to schedule a hygiene inspection with higher priority.

2.2 Novelty of exploration

Brief summary of non-standard techniques I used:

- Cuisne pairs visualization (1.2, Fig. 8) as a different way to visualize most similar cuisines
- Use of Sidekick pattern (1.3) – using curated dataset of dish names to help mine dish names in the main uncurated data set
- Use of dish names mined from a dissimilar cuisine (1.3) to eliminate common phrases from target set dish names
- A way to compensate for poor quality of sentiment analysis tools by combining review-level rating as a prior with sentence-level sentiment detection as a way to overturn the prior only when sentiment is established with high enough probability (1.4)
- Design of new popularity index metric (1.4)
- Replacement of zip codes with distances (1.6) to compensate for zip codes that were present in test data, but not training data, and to take into zipcode adjacency rather than only colocation in the same zipcode.

2.3 Contribution of new knowledge

I believe this work generated multiple new items, most of which are already mentioned in previous sections. The ones I consider most interesting are:

- Dish co-occurrence analysis, and its possible uses for dish recommendations (1.4)
- Prototype of interactive restaurant recommender (1.5)
- Use of dish names mined from a dissimilar cuisine (1.3) to eliminate common phrases from target set dish names
- Approach to compensating sentiment analysis quality (1.4)
- Approach to designing a metric, with specific example of dish popularity index (1.4)

3. How to make these results useful

This goes right back to section 2.1, except now we look at practical steps:

- Apply topic mining to positive/negative reviews on regular basis, periodically review generated set of topics, give them names manually (e.g., Food Quality, Service Quality, etc.).
    - Automatically assign ratings to these discovered topics by mapping each review to a topic and averaging ratings of all reviews assigned to same topic
    - Offer users an option to manually specify a rating for each of discovered topics

- Include restaurants with cuisines from same cluster as user-selected cuisine into search results, if the number of restaurants that offer user-selected cuisine in the search area is too small.

- Offer new "dish to try" option in the application, where users can specify dish(es) they like, and the application can make recommendation of other dishes they may want to try in the same cuisine, based on dish popularity and dish co-occurrence in reviews.

- Offer "where can I eat this?" option in the application, where users can specify their favorite dish and get a list of high-rated restaurants in their area where this dish is getting good reviews.

- Offer "do I have a problem?" service to restaurant owners, where they get a notification if the likelihood of their restaurant failing hygiene inspection increases past certain threshold, according to review-based predictive model

4. Tools Used

Dish name extraction: ToPMine, Segphrase, Python
Visualization: Python matplotlib library (Fig. 1-3), Tableau (all others)
Data processing: Python with nltk, gensim and sklearn libraries. Specifically:

- Stemming: nltk.stem.porter.PorterStemmer
- Sentence tokenization: nltk English punkt tokenizer
- N-gram tokenization: sklearn.feature_extraction.text.CountVectorizer,
- TF-IDF: sklearn.feature_extraction.text.TfidfVectorizer
- Zip code coordinates lookup and distance calculation: https://github.com/cmhulett/zipcode_distance
- Holdout test set: sklearn.cross_validation.StratifiedShuffleSplit
- Model parameter search: sklearn.grid_search.GridSearchCV
- LDA: gensim.models.ldamodel.LdaModel
- Word2Vec: gensim.models.Word2Vec
- Clustering: sklearn.cluster.AffinityPropagation, sklearn.cluster.SpectralClustering
- Logistic regression: sklearn.linear_model.LogisticRegression
- SVM: sklearn.svm.SVC
- KNN: sklearn.neighbors.KNeighborsClassifier
- Multinomial Naïve Bayes: sklearn.naive_bayes.MultinomialNB
- Random Forest: sklearn.ensemble.RandomForestClassifier
- GB: sklearn.ensemble.GradientBoostingClassifier
- PCA: sklearn.decomposition.PCA

5. Task reports

Task 1: https://github.com/mlyubinin/misc/blob/master/Task1.pdf
Task 2: https://github.com/mlyubinin/misc/blob/master/Task2.pdf
Task 3: https://github.com/mlyubinin/misc/blob/master/Task3.pdf
Tasks 4 and 5: https://github.com/mlyubinin/misc/blob/master/Task4_5.pdf
Task 6: https://github.com/mlyubinin/misc/blob/master/Task6.pdf