Dish Popularity / Restaurant Recommendation (Analysis of Yelp dataset)

1. Data Preparation

For this task I continue to analyze Indian cuisine, using the set of dishes identified in previous task (see my Task 3 report for description of how the set of dish names was identified - https://github.com/mlyubinin/misc/blob/master/Task3.pdf). As a result of Task 3, I have a list of 693 dishes.

Going back to original Yelp dataset, it contains 42,153 businesses, of which 202 are Indian restaurants (identified by presence of 'Indian' category). After restricting businesses only to 'Indian' category, the dataset has 8,230 reviews for the Indian cuisine restaurants.

2. Mining Popular Dishes (Task 4)

2.1 Sentiment Analysis

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 at review rating would miss these cases and misclassify those specific mentions of a dish. So, we need to do sentiment analysis to account for these cases.

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.

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 $(w_1+w_2+...w_n)/n$, generating 10,000 features for each sentence. Next, 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 trained model to entire set of sentences and 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.

2.2 Filtering dish-related sentences and popular dishes

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), creating a mapping between a dish and its mentions, and 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')

2.2 Popularity metric design

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. 1), or we can try to design a different popularity index metric that combines several items together:

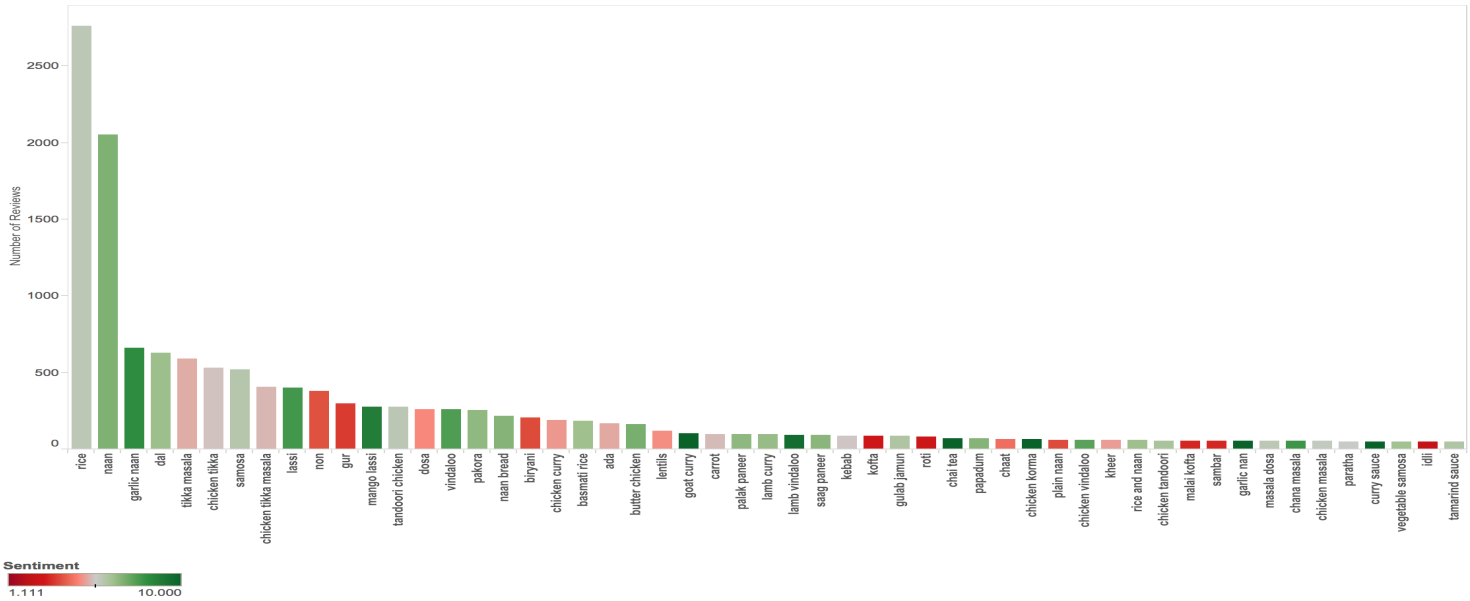

Fig. 1. Dish popularity by number of reviews. Color(Sentiment) shows the ratio of positive to negative mentions.

- 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 it 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.

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

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. 2). Note, I used same ratio for color as in Fig. 1 to allow easier comparison between the two visualizations.

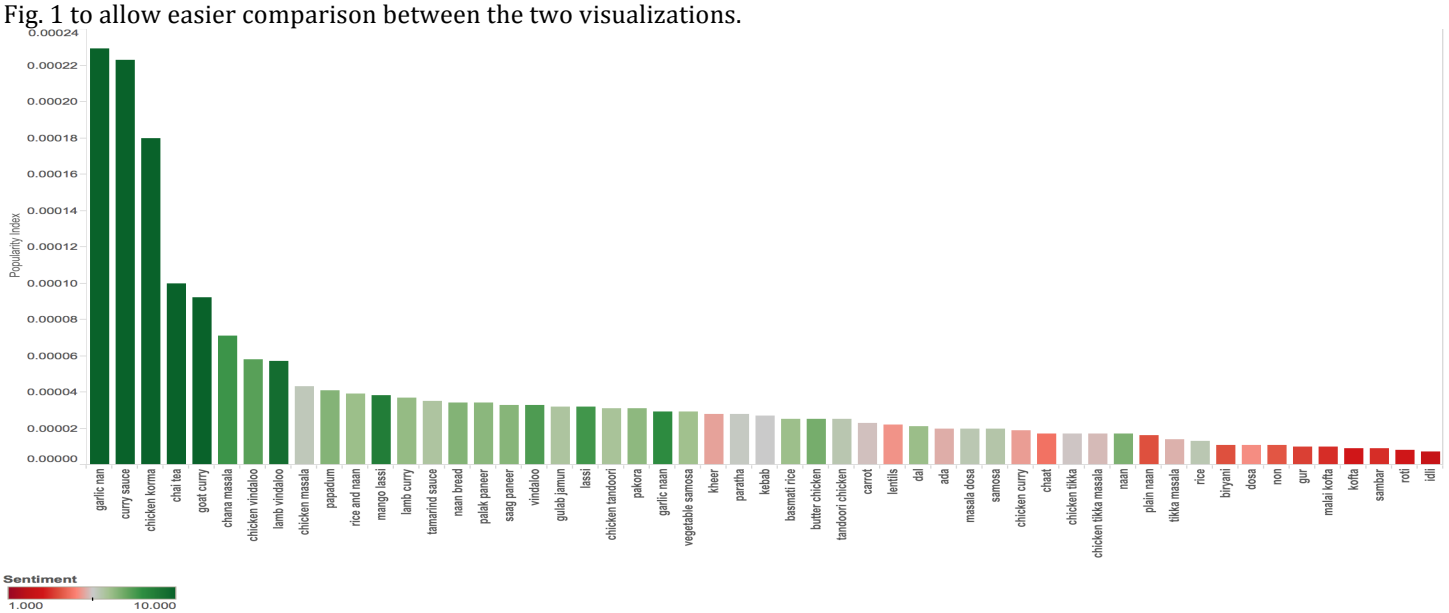

Fig. 2. Dish popularity by popularity index. Color(Sentiment) shows the ratio of positive to negative mentions.

2.3 Practical uses for dish popularity rankings

Similar to what we did in Task 2, looking at cuisine similarity, here we can look at dish combinations – which dishes get mentioned together in the same review in more than 10 reviews (Fig. 3), and which dishes get mentioned together in the same restaurant in more than 10 reviews (Fig. 4).

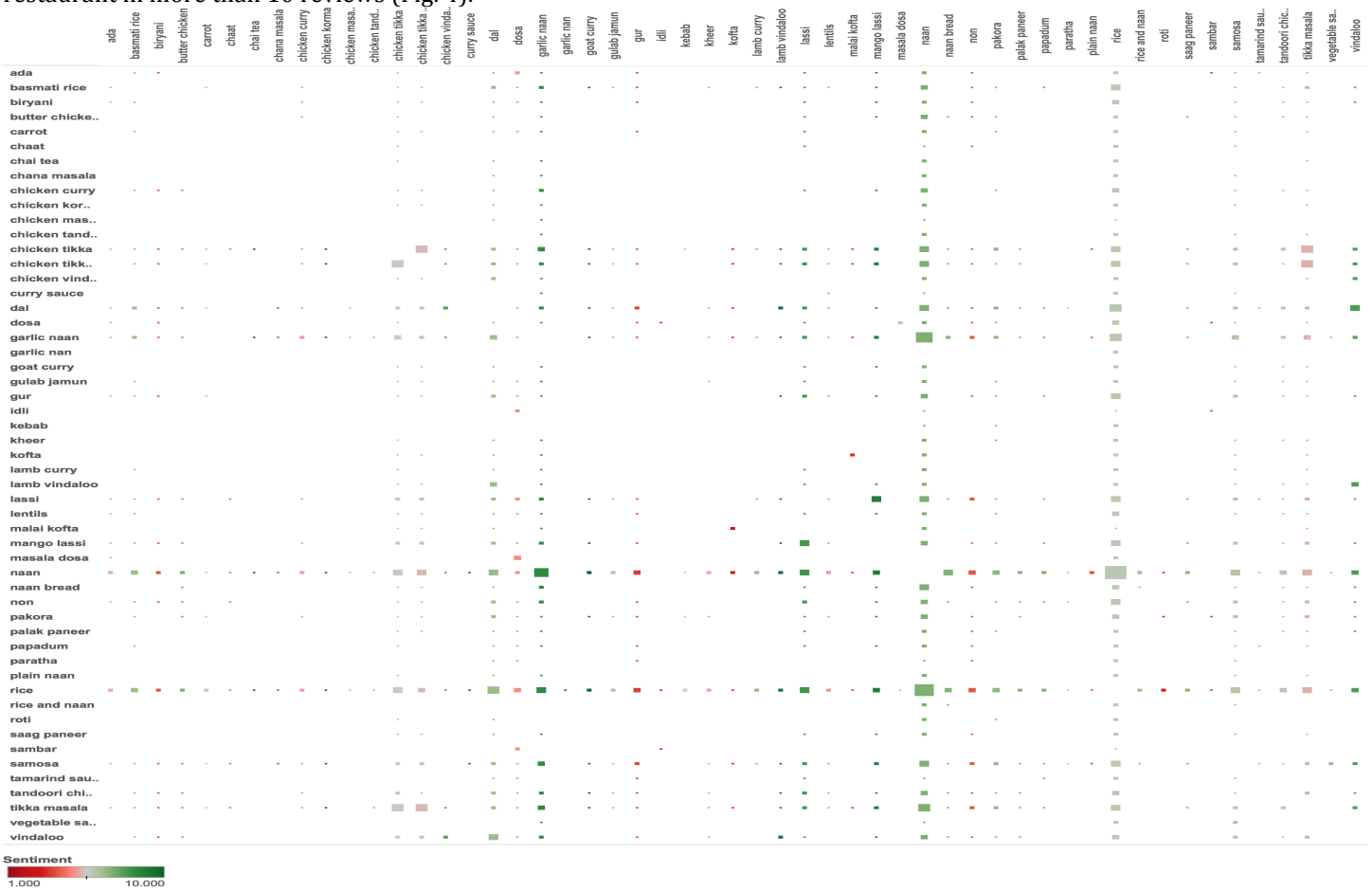

Fig. 3. Dish co-occurrence in same reviews.

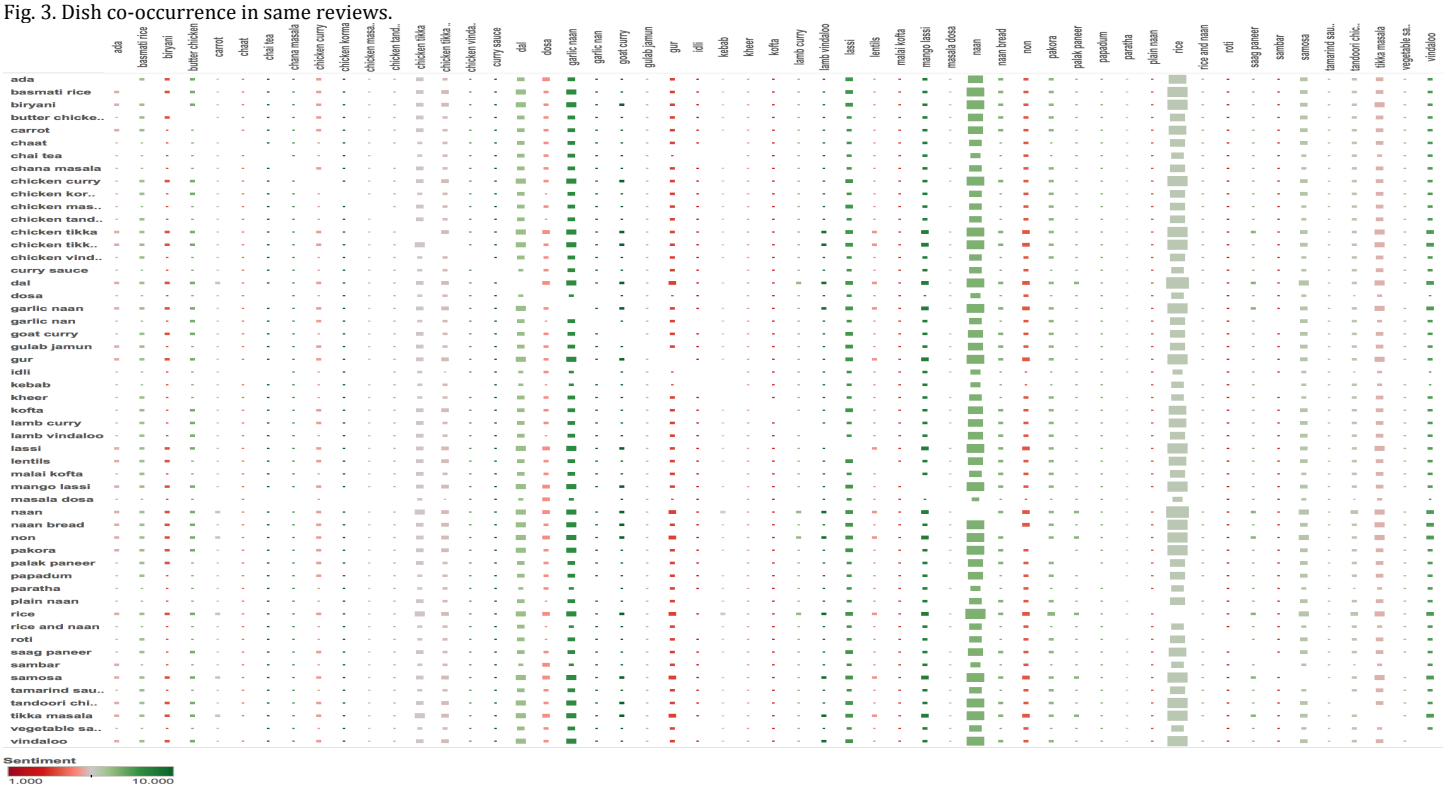

Fig. 4. Dish co-occurrence in reviews for same restaurant.

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).

3. Restaurant Recommendation (Task 5)

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
- 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)

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. 5), 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 – the link to interactive version is below, in section 4.

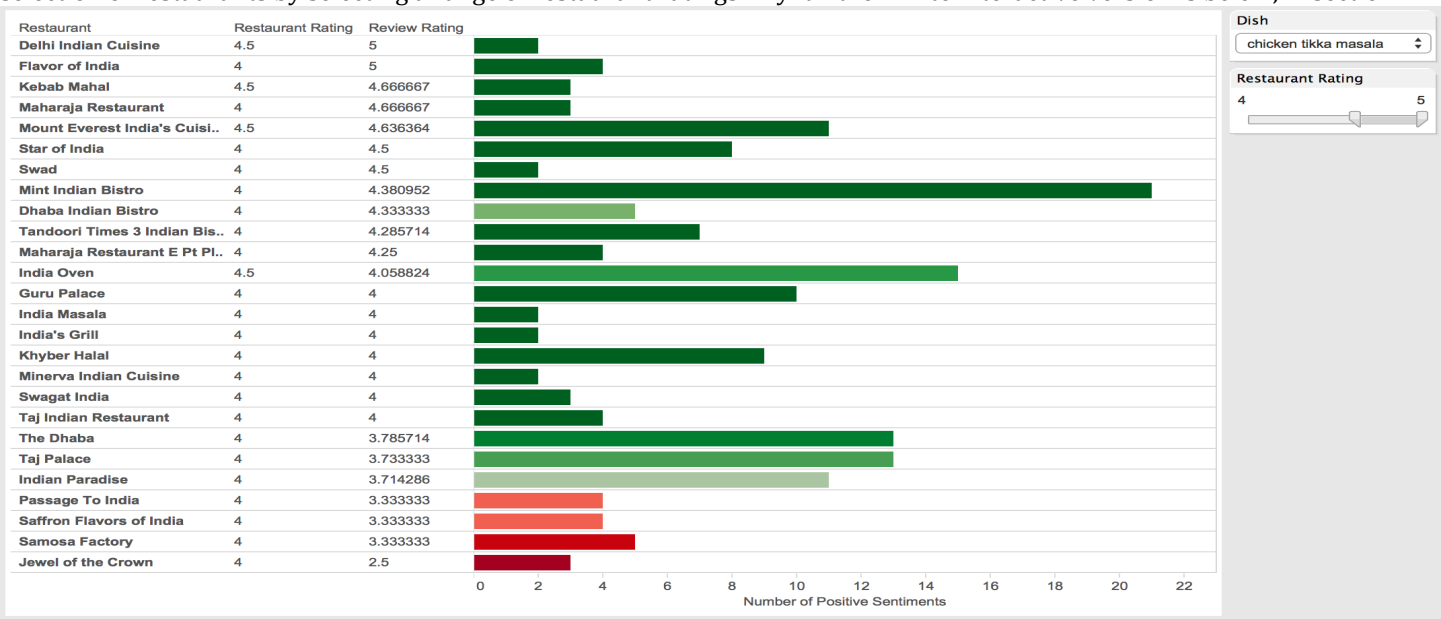

Fig. 5. Restaurant recommender.

4. Tools Used

Data processing: Python with nltk, gensim and sklearn libraries
Visualization: Tableau, with publishing to TableauPublic
Processing results were exported to CSV files from Python for visualization in Tableau.
Interactive version of the charts is available at:

- Fig. 1 - https://public.tableau.com/shared/QYM6RQ8W9?:toolbar=no&:display_count=yes
- Fig. 2 - https://public.tableau.com/shared/886XJSZ2W?:toolbar=no&:display_count=yes
- Fig. 3 - https://public.tableau.com/shared/YGRXWZHKD?:toolbar=no&:display_count=yes
- Fig. 4 - https://public.tableau.com/shared/T238DB6JD?:toolbar=no&:display_count=yes
- Fig. 5 - https://public.tableau.com/shared/9KP6FYQJC?:toolbar=no&:display_count=yes

5. Important code snippets

Filtering Indian cuisine:

```
restaurant_ids = []
restaurant_ratings = {}
restaurant_names = {}
with open(path2buisness, 'r') as f:
    for line in f.readlines():
        business_json = json.loads(line)
        if 'Indian' in business_json['categories']:
            restaurant_ids.append(business_json['business_id'])
            restaurant_names[business_json['business_id']] = business_json['name']
            restaurant_ratings[business_json['business_id']] = business_json['stars']
```

Importing reviews:

```
reviews_text = []
reviews_ids = []
reviews_ratings = []
reviews_restaurant_ids = []
with open (path2reviews, 'r') as f:
    for line in f.readlines():
        review_json = json.loads(line)
        if review_json['business_id'] in restaurant_ids:
            reviews_text.append(review_json['text'])
            reviews_ids.append(review_json['review_id'])
            reviews_restaurant_ids.append(review_json['business_id'])
            reviews_ratings.append(review_json['stars'])
```

Split reviews into sentences and map review ids, restaurant ids and ratings to sentences:

```
reviews_text = [x.replace('\n', ' ') for x in reviews_text]
reviews_text = [x.replace('\r', '') for x in reviews_text]
tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
reviews_text_sentences = [tokenizer.tokenize(x) for x in reviews_text]
reviews_sentence_text = []
reviews_sentence_ids = []
reviews_sentence_restaurant_ids = []
reviews_sentence_ratings = []
for i in range(0, len(reviews_ids)):
    for sentence in reviews_text_sentences[i]:
        if len(sentence) > 1:
            reviews_sentence_text.append(sentence)
            reviews_sentence_ids.append(reviews_ids[i])
            reviews_sentence_restaurant_ids.append(reviews_restaurant_ids[i])
            reviews_sentence_ratings.append(reviews_ratings[i])
```

Prepare sentences for word2vec and train the word2vec model:

```
reviews_sentence_wordlist = [nltk.tokenize.word_tokenize(x) for x in reviews_sentence_text]
reviews_sentence_wordlist = [[x.lower() for x in sentence if len(x)>1] for sentence in reviews_sentence_wordlist]
del_list = []
for i in range(0, len(reviews_sentence_text)):
    if len(reviews_sentence_wordlist[i]) == 0:
        del_list.append(i)
for index in reversed(del_list):
    del reviews_sentence_ids[index]
    del reviews_sentence_ratings[index]
    del reviews_sentence_restaurant_ids[index]
    del reviews_sentence_text[index]
    del reviews_sentence_wordlist[index]
model = models.Word2Vec(reviews_sentence_wordlist, size=10000, min_count=1)
```

Build the feature matrix:

```
reviews_sentence_features = []
for sentence in reviews_sentence_wordlist:
    feature_vec = model[sentence[0]]
    for i in range(1, len(sentence)):
        feature_vec += model[sentence[i]]
    reviews_sentence_features.append(feature_vec / len(sentence))
reviews_sentence_features = np.array(reviews_sentence_features)
```

Assign initial sentiment labels based on review ratings:

```
reviews_sentence_labels = []
for i in range(0,len(reviews_sentence_ratings)):
    if reviews_sentence_ratings[i] == 1 or reviews_sentence_ratings[i] == 2:
        label = 0
    elif reviews_sentence_ratings[i] == 4 or reviews_sentence_ratings[i] == 5:
        label = 1
    elif reviews_sentence_ratings[i] == 3:
        label = None
    else:
        print 'ERROR: ' + str(reviews_sentence_ratings[i])
    reviews_sentence_labels.append(label)
reviews_sentence_labels = np.array(reviews_sentence_labels)
```

Train logistic regression model:

```
training_index = [x for x in range(0,len(reviews_sentence_labels)) if reviews_sentence_labels[x] is not None]
train_features = reviews_sentence_features[training_index][:]
train_labels = reviews_sentence_labels[training_index]
logistic_model = sklearn.linear_model.LogisticRegression()
logistic_model.fit(train_features,train_labels)
```

Apply the model and assign sentiment labels:

```
reviews_sentence_yhat = logistic_model.predict_proba(reviews_sentence_features)
for i in range(0, len(reviews_sentence_ids)):
    if reviews_sentence_labels[i] == 1 and reviews_sentence_yhat[i][0] < 0.3:
        reviews_sentence_labels[i] = 0
    elif reviews_sentence_labels[i] == 0 and reviews_sentence_yhat[i][0] > 0.7:
        reviews_sentence_labels[i] = 1
    elif reviews_sentence_labels[i] is None:
        if reviews_sentence_yhat[i][0] >= 0.5:
            reviews_sentence_labels[i] = 1
        else:
            reviews_sentence_labels[i] = 0
```

Map sentences to dishes and filter out infrequent items and non-dish items:

```
dish_reviews = {x:[] for x in dishes}
for dish in dishes:
    for i in range(0, len(reviews_sentence_ids)):
        if dish in reviews_sentence_text[i]:
            dish_reviews[dish].append({'review_id':reviews_sentence_ids[i],
                                       'restaurant_id':reviews_sentence_restaurant_ids[i],
                                        'rating': reviews_sentence_ratings[i],
                                       'label':reviews_sentence_labels[i]})
dish_exclude_list = ['meats', 'look forward', 'looking forward', 'name', 'vegetarian dish']
dish_reviews_filtered = {x:dish_reviews[x] for x in dish_reviews.keys()
                         if len(dish_reviews[x])>50 and x not in dish_exclude_list}
```

Calculate popularity metrics:

```
dish_mensions_positive = {x:sum([1 for y in dish_reviews_filtered[x] if y['label']==1])
                          for x in dish_reviews_filtered.keys()}
dish_mensions_negative = {x:sum([1 for y in dish_reviews_filtered[x] if y['label']==0])
                          for x in dish_reviews_filtered.keys()}
dish_reviews_positive = {x:len(set([y['review_id'] for y in dish_reviews_filtered[x] if y['label']==1]))
                          for x in dish_reviews_filtered.keys()}
dish_reviews_negative = {x:len(set([y['review_id'] for y in dish_reviews_filtered[x] if y['label']==0]))
                          for x in dish_reviews_filtered.keys()}
dish_reviews_total = {x:len(set([y['review_id'] for y in dish_reviews_filtered[x]]))
                          for x in dish_reviews_filtered.keys()}
dish_restaurants_positive = {x:len(set([y['restaurant_id'] for y in dish_reviews_filtered[x] if y['label']==1]))
                          for x in dish_reviews_filtered.keys()}
dish_restaurants_negative = {x:len(set([y['restaurant_id'] for y in dish_reviews_filtered[x] if y['label']==0]))
                          for x in dish_reviews_filtered.keys()}
dish_restaurants_total = {x:len(set([y['restaurant_id'] for y in dish_reviews_filtered[x]]))
                          for x in dish_reviews_filtered.keys()}
dish_sentiment = {x:(dish_reviews_positive[x]+1)/float(dish_reviews_negative[x] + 1)
                  for x in dish_reviews_filtered.keys()}
dish_rating_index = {x:(dish_reviews_positive[x] + dish_reviews_total[x])*dish_restaurants_total[x]/
                     float((dish_reviews_negative[x] + 1)*(dish_restaurants_negative[x] + 1)*
                           len(restaurant_ids)*len(reviews_ids))
                     for x in dish_reviews_filtered.keys()}
```

Calculate dish review and restaurant co-occurrence:

```
dish_review_matrix = {(dish1,dish2):0 for dish1 in dish_reviews_filtered.keys()
                      for dish2 in dish_reviews_filtered.keys()}
for (dish1, dish2) in dish_review_matrix.keys():
    if dish1 != dish2:
        d1_reviews = set([sentence['review_id'] for sentence in dish_reviews_filtered[dish1] if sentence['label']==1])
        dish_review_matrix[(dish1,dish2)] = sum([1 for sentence in dish_reviews_filtered[dish2]
                                                 if sentence['label']==1 and sentence['review_id'] in d1_reviews])
dish_review_matrix = {x:dish_review_matrix[x] for x in dish_review_matrix.keys() if dish_review_matrix[x] > 10}
dish_restaurant_matrix = {(dish1,dish2):0 for dish1 in dish_reviews_filtered.keys()
                      for dish2 in dish_reviews_filtered.keys()}
for (dish1, dish2) in dish_restaurant_matrix.keys():
    if dish1 != dish2:
        d1_restaurants = set([sentence['restaurant_id'] for sentence in dish_reviews_filtered[dish1] if sentence['label']==1])
        dish_restaurant_matrix[(dish1,dish2)] = sum([1 for sentence in dish_reviews_filtered[dish2]
                                                 if sentence['label']==1 and sentence['restaurant_id'] in d1_restaurants])
dish_restaurant_matrix = {x:dish_restaurant_matrix[x] for x in dish_restaurant_matrix.keys()
                          if dish_restaurant_matrix[x] > 10}
```

Calculate restaurant ratings per dish:

```
dish_restaurant_ratings = {}
for dish, review_list in dish_reviews_filtered.iteritems():
    for review in review_list:
        rest_name = restaurant_names[review['restaurant_id']]
        rest_rating = restaurant_ratings[review['restaurant_id']]
        if (dish, rest_name) in dish_restaurant_ratings.keys():
            rev_rating_total = review['rating'] + dish_restaurant_ratings[(dish,rest_name)][1]
            rev_label_total = review['label'] + dish_restaurant_ratings[(dish,rest_name)][2]
            rev_rating_count = dish_restaurant_ratings[(dish,rest_name)][3] + 1
        else:
            rev_rating_total = review['rating']
            rev_label_total = review['label']
            rev_rating_count = 1
        dish_restaurant_ratings[(dish, rest_name)] = (rest_rating, rev_rating_total, rev_label_total, rev_rating_count)
# Structure:
# (dish, restaurant_name):(restaurant_rating, avg_review_rating, sentiment_pct, num_reviews, num_positive_sentiments)
dish_restaurant_ratings = {x:(t[0], t[1]/float(t[3]), t[2]/float(t[3]), t[3], t[2])
                           for x, t in dish_restaurant_ratings.iteritems()}
dish_restaurant_ratings = {x:t for x,t in dish_restaurant_ratings.iteritems() if t[2] >= 0.5 and t[3]>1}
```