BitTiger DS501

Week 5 & 6 HW Meina Wang

Question 1

Complete Yelp Data Challenge project preprocessing code: "Yelp_Data_Challenge_Project/Yelp*Dataset*-_Data_Preprocessing.ipynb"

- Load, visualize, filter data
- **I only saved ~ one year of data, since using two years of data is still too much for my laptop (took a really long time for later clustering processes).**

Attached in following documents.

Question 2

reviews. Complete Yelp Data Challenge project NLP code: "Yelp_Data_Challenge_Project/Yelp*Dataset*-_NLP.ipynb". Tasks:

- Load, visualize data
- Define positive/negative reviews
- Extract Tf-Idf feature vectors from review data
- Build review classifiers using supervised ML models
- Use cross-validation and grid search to tune parameters and select models
- Question: Think about the use case for this work, and answer why you want to do this in the first when asked by interviewers.

Attached in following documents.

Question 3

Yelp data clustering. Complete Yelp Data Challenge project code (only clustering part): "Yelp_Data_Challenge_Project/Yelp*Dataset*-_Clustering_and_PCA.ipynb". Tasks:

- Load, visualize data
- Extract Tf-Idf feature vectors from review data
- Perform K-Means clustering of the reviews, we will be limiting to positive reviews (since we don't want to cluster good vs bad)
- Question: What does the clustering result tell us, look for answers from the centroid and examples, and how this work can help the business (interviewer may ask this type of questions)?
- **Answer: The clustering results tell us the type of business of restaurants. This info can help business owners to know more about how they are doing from a customer point of view, and extract info to improve their business.**
- Extra credits: there are 5 optional extra credits at the end of ipynb, can you complete any of them?

Attached in following documents.

Question 1

Yelp data PCA. Complete Yelp Data Challenge code: "Yelp_Data_Challenge_Project/Yelp*Dataset*-_Clustering_and_PCA.ipynb" (only PCA part). Tasks:

- Load, visualize data
- Extract Tf-Idf feature vectors from review data
- Perform PCA on review feature vectors
- Visualize variance explained
- Perform PCA review classification
- Questions: Will PCA help improve performance? What is the disadvantage of PCA regression/classification?
- **Answer: PCA helped reduce the overfitting problem. However, by using PCA, the model loses its interpretability since each PC doesn't hold physical meanings.**

Attached in following documents.

Question 2

Yelp recommender system. Complete Yelp Data Challenge code: "Yelp*Dataset*-_Restaurant_Recommender.ipynb". Tasks:

- Prepare utility matrix
- Use item-item collaborative filtering to build recommender system
- Use matrix factorization to build recommender system

Attached in following documents.

Yelp Data Challenge - Data Preprocessing

BitTiger DS501

Jun 2017

Dataset Introduction

Yelp Dataset Challenge (https://www.yelp.com/dataset_challenge)

The Challenge Dataset:

```
4.1M reviews and 947K tips by 1M users for 144K businesses
1.1M business attributes, e.g., hours, parking availability, ambience.
Aggregated check-ins over time for each of the 125K businesses
200,000 pictures from the included businesses
```

Cities:

```
U.K.: Edinburgh
Germany: Karlsruhe
Canada: Montreal and Waterloo
U.S.: Pittsburgh, Charlotte, Urbana-Champaign, Phoenix, Las Vegas, Madison,
Cleveland
```

Files:

```
yelp_academic_dataset_business.json
yelp_academic_dataset_checkin.json
yelp_academic_dataset_review.json
yelp_academic_dataset_tip.json
yelp_academic_dataset_user.json
```

Notes on the Dataset

```
Each file is composed of a single object type, one json-object per-line.
Take a look at some examples to get you started: https://github.com/Yelp/dat
aset-examples.
```

Read data from file and load to Pandas DataFrame

Warning: Loading all the 1.8 GB data into Pandas at a time takes long time and a lot of memory!

In [1]:

```
import json
import pandas as pd
```

In [4]:

```
file_business, file_checkin, file_review, file_tip, file_user = [
    'dataset/business.json',
    'dataset/checkin.json',
    'dataset/review.json',
    'dataset/tip.json',
    'dataset/user.json'
]
```

Business Data

In [6]:

```
with open(file_business) as f:
    df_business = pd.DataFrame(json.loads(line) for line in f)
```

In [7]:

```
df_business.head(2)
```

Out[7]:

	address	attributes	business_id	categories	city	ho
0	4855 E Warner Rd, Ste B9	{u'AcceptsInsurance': True, u'ByAppointmentOnl...	FYWN1wneV18bWNgQjJ2GNg	[Dentists, General Dentistry, Health & Medical...	Ahwatukee	{u'Tuesc u'7 17: u'Fric u'7:3
1	3101 Washington Rd	{u'BusinessParking': {u'garage': False, u'stre...	He-G7vWjzVUysIKrfNbPUQ	[Hair Stylists, Hair Salons, Men's Hair Salons...	McMurray	{u'Monc u'9 20: u'Tuesc u'9:0

In [8]:

```
df_business.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 174567 entries, 0 to 174566
Data columns (total 15 columns):
address         174567 non-null object
attributes      174567 non-null object
business_id     174567 non-null object
categories      174567 non-null object
city            174567 non-null object
hours           174567 non-null object
is_open         174567 non-null int64
latitude        174566 non-null float64
longitude       174566 non-null float64
name            174567 non-null object
neighborhood    174567 non-null object
postal_code     174567 non-null object
review_count    174567 non-null int64
stars           174567 non-null float64
state           174567 non-null object
dtypes: float64(3), int64(2), object(10)
memory usage: 20.0+ MB
```

Checkin Data

In [9]:

```
with open(file_checkin) as f:
    df_checkin = pd.DataFrame(json.loads(line) for line in f)
df_checkin.head(2)
```

Out[9]:

	business_id	time
0	7KPBkxAOEtb3QeIL9PEErg	{u'Monday': {u'19:00': 1, u'14:00': 1, u'17:00...
1	kREVIrSBbtqBhIYkTccQUg	{u'Sunday': {u'19:00': 1}, u'Saturday': {u'16:...

Review Data

In [ ]:

```
with open(file_review) as f:
    df_review = pd.DataFrame(json.loads(line) for line in f)
df_review.head(2)
```

Tip Data

In [ ]:

```
# with open(file_tip) as f:
#     df_tip = pd.DataFrame(json.loads(line) for line in f)
# df_tip.head(2)
```

User Data

In [ ]:

```
# with open(file_user) as f:
#     df_user = pd.DataFrame(json.loads(line) for line in f)
# df_user.head(2)
```

Filter data by city and category

Create filters/masks

- create filters that selects business
  - that are located in "Las Vegas"
  - that contains "Restaurants" in their category (You may need to filter null categories first)

In [10]:

```
# Create Pandas DataFrame filters
cond_city = df_business['city'] == "Las Vegas"
cond_category_not_null = -df_business['categories'].isnull()
con_category_restaurant = df_business['categories'].apply(str).str.contains("Restau
```

In [12]:

```
# Create filtered DataFrame, and name it df_filtered
df_filtered = df_business[cond_city & cond_category_not_null & con_category_restaura
```

Keep relevant columns

- only keep some useful columns
  - business_id
  - name
  - categories
  - stars

In [13]:

```
selected_features = [u'business_id', u'name', u'categories', u'stars']
```

In [14]:

```
# Make a DataFrame that contains only the abovementioned columns, and name it as df_
df_selected_business = df_filtered[selected_features]
```

In [17]:

```
# Rename the column name "stars" to "avg_stars" to avoid naming conflicts with revie
df_selected_business.rename(columns={"stars":"avg_stars"}, inplace=True)
```

```
/Users/meinawang/anaconda2/lib/python2.7/site-packages/pandas/core/fra
me.py:3027: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
(http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-vi
ew-versus-copy)
  return super(DataFrame, self).rename(**kwargs)
```

In [19]:

```
# Inspect your DataFrame
df_selected_business.info()
df_selected_business.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5899 entries, 52 to 174469
Data columns (total 4 columns):
business_id    5899 non-null object
name           5899 non-null object
categories     5899 non-null object
avg_stars      5899 non-null float64
dtypes: float64(1), object(3)
memory usage: 230.4+ KB
```

Out[19]:

	business_id	name	categories	avg_stars
52	Pd52CjgyEU3Rb8co6QfTPw	Flight Deck Bar & Grill	[Nightlife, Bars, Barbeque, Sports Bars, Ameri...	4.0
53	4srfPk1s8nlm1YusyDUbjg	Subway	[Fast Food, Restaurants, Sandwiches]	2.5
54	n7V4cD-KqqE3OXk0irJTyA	GameWorks	[Arcades, Arts & Entertainment, Gastropubs, Re...	3.0
91	F0fEKpTk7gAmuSFI0KW1eQ	Cafe Mastrioni	[Italian, Restaurants]	1.5
122	Wpt0sFHcPtV5MO9He7yMKQ	McDonald's	[Restaurants, Fast Food, Burgers]	2.0

Save results to csv files

In [20]:

```
# Save to ./data/selected_business.csv for your next task
df_selected_business.to_csv("dataset/selected_business.csv", index=False, encoding='
```

In [21]:

```
# Try reload the csv file to check if everything works fine
pd.read_csv("dataset/selected_business.csv", encoding="utf-8").head()
```

Out[21]:

	business_id	name	categories	avg_stars
0	Pd52CjgyEU3Rb8co6QfTPw	Flight Deck Bar & Grill	[Nightlife, Bars, Barbeque, Sports Bars, Ameri...	4.0
1	4srfPk1s8nlm1YusyDUbjg	Subway	[Fast Food, Restaurants, Sandwiches]	2.5
2	n7V4cD-KqqE3OXk0irJTyA	GameWorks	[Arcades, Arts & Entertainment, Gastropubs, Re...	3.0
3	F0fEKpTk7gAmuSFI0KW1eQ	Cafe Mastrioni	[Italian, Restaurants]	1.5
4	Wpt0sFHcPtV5MO9He7yMKQ	McDonald's	[Restaurants, Fast Food, Burgers]	2.0

Use the "business_id" column to filter review data

- We want to make a DataFrame that contain and only contain the reviews about the business entities we just obtained

Load review dataset

In [22]:

```
with open(file_review) as f:
    df_review = pd.DataFrame(json.loads(line) for line in f)
df_review.head(2)
```

Out[22]:

	business_id	cool	date	funny	review_id	stars	text
0	0W4lkclzZThpx3V65bVgig	0	2016-05-28	0	v0i_UHJMo_hPBq9bxWvW4w	5	Love the staff, love the meat, love the place....
1	AEx2SYEUJmTxVVB18LlCwA	0	2016-05-28	0	vkVSCC7xljjrAI4UGfnKEQ	5	Super simple place but amazing nonetheless. It...

Prepare dataframes to be joined, - on business_id

In [23]:

```
# Prepare the business dataframe and set index to column "business_id", and name it
df_left = df_selected_business.set_index('business_id')
```

In [24]:

```
# Prepare the review dataframe and set index to column "business_id", and name it as
df_review = df_review[df_review['date']>'2016-02-01'].set_index('business_id')
```

Join! and reset index

In [25]:

```
# Join df_left and df_right. What type of join?
df_final = df_left.join(df_review,how='inner')
```

In [26]:

```
# You may want to reset the index
df_final = df_final.reset_index()
```

We further filter data by date, e.g. keep comments from last 2 years

- Otherwise your laptop may crush on memory when running machine learning algorithms
- Purposefully ignoring the reviews made too long time ago

In [ ]:

```
# Make a filter that selects date after 2015-01-20
pass
```

In [ ]:

```
# Filter the joined DataFrame and name it as df_final
pass
```

Take a glance at the final dataset

- Do more EDA here as you like!

In [27]:

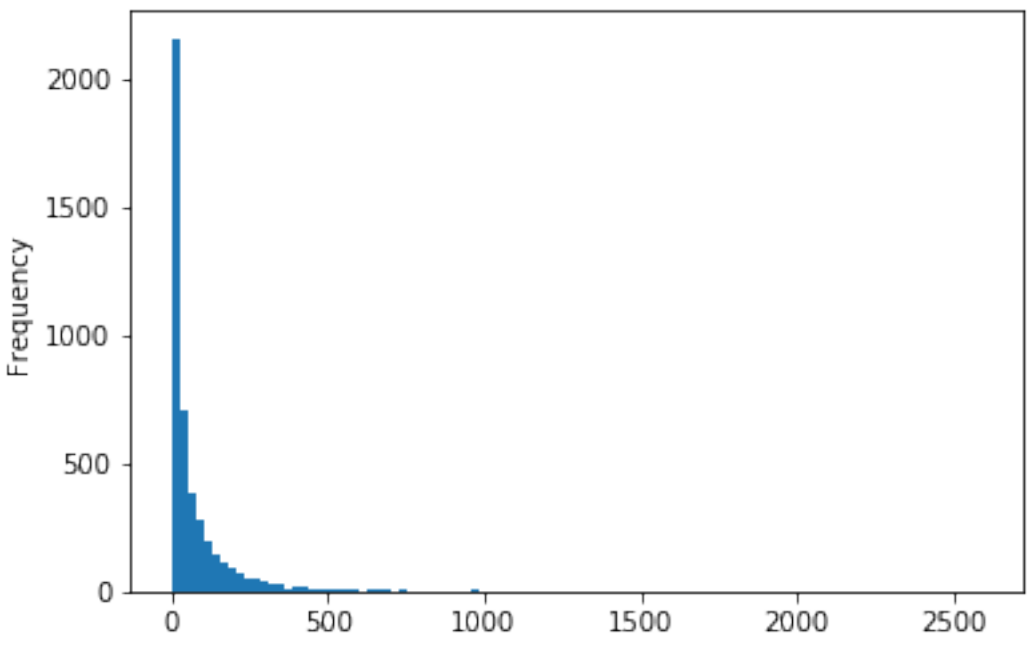
```
import matplotlib.pyplot as plt

% matplotlib inline
```

In [28]:

```
# e.g. calculate counts of reviews per business entity, and plot it
df_final['business_id'].value_counts().plot.hist(bins=100)
plt.show
```

Out[28]:

```
<function matplotlib.pyplot.show>
```



Save your preprocessed dataset to csv file

- Respect your laptop's hard work! You don't want to make it run everything again.

In [29]:

```
# Save to ./data/last_2_years_restaurant_reviews.csv for your next task
df_final.to_csv('dataset/last_2_years_restaurant_reviews.csv',index=False,encoding=
```

In [ ]:

```

```

Yelp Data Challenge - Data Preprocessing

BitTiger DS501

Jun 2017

Dataset Introduction

Yelp Dataset Challenge (https://www.yelp.com/dataset_challenge)

The Challenge Dataset:

```
4.1M reviews and 947K tips by 1M users for 144K businesses
1.1M business attributes, e.g., hours, parking availability, ambience.
Aggregated check-ins over time for each of the 125K businesses
200,000 pictures from the included businesses
```

Cities:

```
U.K.: Edinburgh
Germany: Karlsruhe
Canada: Montreal and Waterloo
U.S.: Pittsburgh, Charlotte, Urbana-Champaign, Phoenix, Las Vegas, Madison,
Cleveland
```

Files:

```
yelp_academic_dataset_business.json
yelp_academic_dataset_checkin.json
yelp_academic_dataset_review.json
yelp_academic_dataset_tip.json
yelp_academic_dataset_user.json
```

Notes on the Dataset

```
Each file is composed of a single object type, one json-object per-line.
Take a look at some examples to get you started: https://github.com/Yelp/dat
aset-examples.
```

Read data from file and load to Pandas DataFrame

Warning: Loading all the 1.8 GB data into Pandas at a time takes long time and a lot of memory!

In [1]:

In [4]:

Business Data

In [6]:

In [7]:

Out[7]:

	address	attributes	business_id	categories	city	ho
0	4855 E Warner Rd, Ste B9	{u'AcceptsInsurance': True, u'ByAppointmentOnl...	FYWN1wneV18bWNgQjJ2GNg	[Dentists, General Dentistry, Health & Medical...	Ahwatukee	{u'Tuesc u'7 17: u'Fric u'7:3
1	3101 Washington Rd	{u'BusinessParking': {u'garage': False, u'stre...	He-G7vWjzVUysIKrfNbPUQ	[Hair Stylists, Hair Salons, Men's Hair Salons...	McMurray	{u'Monc u'9 20: u'Tuesc u'9:0

In [8]:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 174567 entries, 0 to 174566
Data columns (total 15 columns):
address         174567 non-null object
attributes      174567 non-null object
business_id     174567 non-null object
categories      174567 non-null object
city            174567 non-null object
hours           174567 non-null object
is_open         174567 non-null int64
latitude        174566 non-null float64
longitude       174566 non-null float64
name            174567 non-null object
neighborhood    174567 non-null object
postal_code     174567 non-null object
review_count    174567 non-null int64
stars           174567 non-null float64
state           174567 non-null object
dtypes: float64(3), int64(2), object(10)
```

Checkin Data

In [9]:

Out[9]:

	business_id	time
0	7KPBkxAOEtb3QeIL9PEErg	{u'Monday': {u'19:00': 1, u'14:00': 1, u'17:00...
1	kREVIrSBbtqBhIYkTccQUg	{u'Sunday': {u'19:00': 1}, u'Saturday': {u'16:...

Review Data

In [ ]:

Tip Data

In [ ]:

User Data

In [ ]:

Filter data by city and category

Create filters/masks

- create filters that selects business
  - that are located in "Las Vegas"
  - that contains "Restaurants" in their category (You may need to filter null categories first)

In [10]:

In [12]:

Keep relevant columns

- only keep some useful columns
  - business_id
  - name
  - categories
  - stars

In [13]:

In [14]:

In [17]:

```
/Users/meinawang/anaconda2/lib/python2.7/site-packages/pandas/core/fra
me.py:3027: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
(http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-vi
ew-versus-copy)
  return super(DataFrame, self).rename(**kwargs)
```

In [19]:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5899 entries, 52 to 174469
Data columns (total 4 columns):
business_id    5899 non-null object
name           5899 non-null object
categories     5899 non-null object
avg_stars      5899 non-null float64
dtypes: float64(1), object(3)
memory usage: 230.4+ KB
```

Out[19]:

	business_id	name	categories	avg_stars
52	Pd52CjgyEU3Rb8co6QfTPw	Flight Deck Bar & Grill	[Nightlife, Bars, Barbeque, Sports Bars, Ameri...	4.0
53	4srfPk1s8nlm1YusyDUbjg	Subway	[Fast Food, Restaurants, Sandwiches]	2.5
54	n7V4cD-KqqE3OXk0irJTyA	GameWorks	[Arcades, Arts & Entertainment, Gastropubs, Re...	3.0

Save results to csv files

In [20]:

In [21]:

Out[21]:

	business_id	name	categories	avg_stars
0	Pd52CjgyEU3Rb8co6QfTPw	Flight Deck Bar & Grill	[Nightlife, Bars, Barbeque, Sports Bars, Ameri...	4.0
1	4srfPk1s8nlm1YusyDUbjg	Subway	[Fast Food, Restaurants, Sandwiches]	2.5
2	n7V4cD-KqqE3OXk0irJTyA	GameWorks	[Arcades, Arts & Entertainment, Gastropubs, Re...	3.0
3	F0fEKpTk7gAmuSFI0KW1eQ	Cafe Mastrioni	[Italian, Restaurants]	1.5
4	Wpt0sFHcPtV5MO9He7yMKQ	McDonald's	[Restaurants, Fast Food, Burgers]	2.0

Use the "business_id" column to filter review data

- We want to make a DataFrame that contain and only contain the reviews about the business entities we just obtained

Load review dataset

In [22]:

Out[22]:

	business_id	cool	date	funny	review_id	stars	text
0	0W4lkclzZThpx3V65bVgig	0	2016-05-28	0	v0i_UHJMo_hPBq9bxWvW4w	5	Love the staff, love the meat, love the place....
1	AEx2SYEUJmTxVVB18LlCwA	0	2016-05-28	0	vkVSCC7xljjrAI4UGfnKEQ	5	Super simple place but amazing nonetheless. It...

Prepare dataframes to be joined, - on business_id

In [23]:

In [24]:

Join! and reset index

In [25]:

In [26]:

We further filter data by date, e.g. keep comments from last 2 years

- Otherwise your laptop may crush on memory when running machine learning algorithms
- Purposefully ignoring the reviews made too long time ago

In [ ]:

In [ ]:

Take a glance at the final dataset

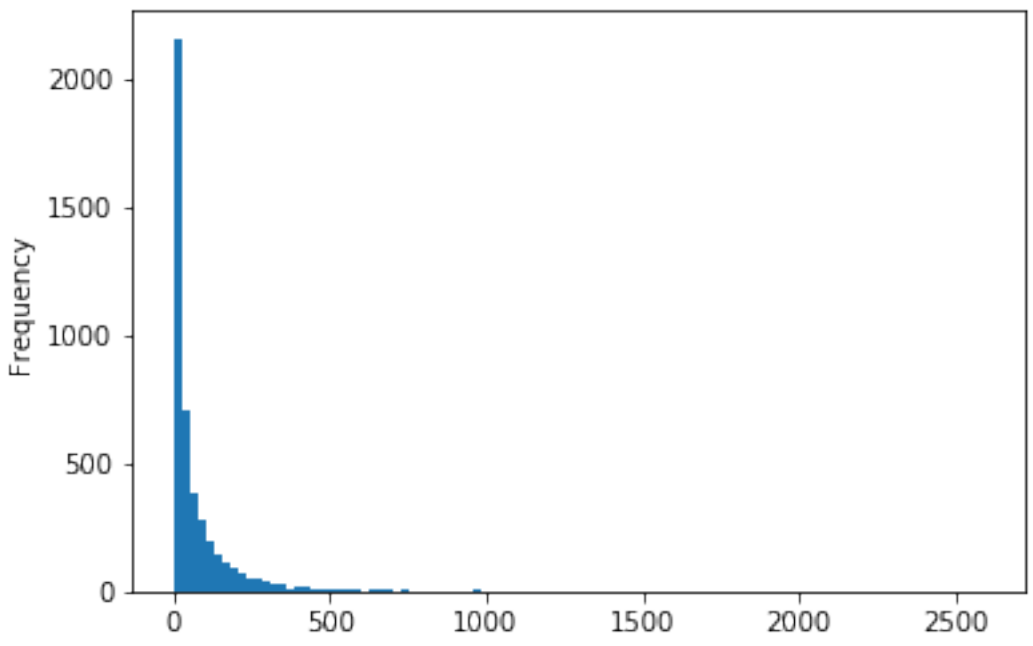
- Do more EDA here as you like!

In [27]:

In [28]:

Out[28]:

```
<function matplotlib.pyplot.show>
```



Save your preprocessed dataset to csv file

- Respect your laptop's hard work! You don't want to make it run everything again.

In [29]:

In [ ]:

Yelp Data Challenge - NLP

BitTiger DS501

Jun 2017

In [1]:

```
import pandas as pd
```

In [3]:

```
df = pd.read_csv('dataset/last_2_years_restaurant_reviews.csv')
```

In [4]:

```
df.head()
```

Out[4]:

	business_id	name	categories	avg_stars	cool	date	funny	revi
0	-9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	[Cajun/Creole, Steakhouses, Restaurants]	4.0	0	2016-03-31	0	6SgvNWJltnZhW7duJ
1	-9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	[Cajun/Creole, Steakhouses, Restaurants]	4.0	0	2016-02-10	0	UxFpgng8dPMWOj996
2	-9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	[Cajun/Creole, Steakhouses, Restaurants]	4.0	0	2017-02-14	0	Xp3ppynEvVu1KxDHQ
3	-9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	[Cajun/Creole, Steakhouses, Restaurants]	4.0	1	2017-05-28	0	LEzphAnz0vKE32PUC
4	-9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	[Cajun/Creole, Steakhouses, Restaurants]	4.0	0	2017-08-25	0	4e-cxYVdIlu2ZDxVJ

Define your feature variables, here is the text of the review

In [5]:

```
# Take the values of the column that contains review text data, save to a variable
documents = df['text'].values
```

In [7]:

```
# inspect your documents, e.g. check the size, take a peek at elements of the numpy
documents.dtype, documents.shape
documents[100]
```

Out[7]:

```
'Still my favorite steakhouse so far!  Ribeye amazing, spinach and au
gratin well done.  Great service and they welcomed us back.  Still des
erves a 5 star on all points from food to service to ambiance.'
```

Define your target variable (any categorical variable that may be meaningful)

For example, I am interested in perfect (5 stars) and imperfect (1-4 stars) rating

In [8]:

```
# Make a column and take the values, save to a variable named "target"
df['favorable'] = df['stars'] > 4
target = df['favorable'].values
target[:10]
```

Out[8]:

```
array([ True,  True,  True, False,  True,  True,  True, False, False,
       False])
```

You may want to look at the statistic of the target variable

In [9]:

```
# To be implemented
target.mean(), target.std
```

Out[9]:

```
(0.4782396579185261, <function std>)
```

In [10]:

```
documents.shape, target.shape
```

Out[10]:

```
((348455,), (348455,))
```

Let's create training dataset and test dataset

In [13]:

```
from sklearn.cross_validation import train_test_split
```

```
/Users/meinawang/anaconda2/lib/python2.7/site-packages/sklearn/cross_v
alidation.py:41: DeprecationWarning: This module was deprecated in ver
sion 0.18 in favor of the model_selection module into which all the re
factored classes and functions are moved. Also note that the interface
of the new CV iterators are different from that of this module. This m
odule will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [ ]:

```
# Documents is your X, target is your y
# Now split the data to training set and test set
```

In [14]:

```
# Split to documents_train, documents_test, target_train, target_test
documents_train, documents_test, target_train, target_test = train_test_split(
    documents,
    target,
    test_size = 0.3,
    random_state = 7
)
```

Let's get NLP representation of the documents

In [15]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [21]:

```
# Create TfidfVectorizer, and name it vectorizer
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
```

In [24]:

```
# Train the model with your training data
vectors_train = vectorizer.fit_transform(documents_train).toarray()
```

In [25]:

```
# Get the vocab of your tfidf
words = vectorizer.get_feature_names()
```

In [26]:

```
# Use the trained model to transform your test data
vectors_test = vectorizer.transform(documents_test).toarray()
```

Similar review search engine

In [27]:

```
import numpy as np

# We will need these helper methods pretty soon

def get_top_values(lst, n, labels):
    '''
    INPUT: LIST, INTEGER, LIST
    OUTPUT: LIST

    Given a list of values, find the indices with the highest n values.
    Return the labels for each of these indices.

    e.g.
    lst = [7, 3, 2, 4, 1]
    n = 2
    labels = ["cat", "dog", "mouse", "pig", "rabbit"]
    output: ["cat", "pig"]
    '''
    return [labels[i] for i in np.argsort(lst)[::-1][:n]]  # np.argsort by default s

def get_bottom_values(lst, n, labels):
    '''
    INPUT: LIST, INTEGER, LIST
    OUTPUT: LIST

    Given a list of values, find the indices with the lowest n values.
    Return the labels for each of these indices.

    e.g.
    lst = [7, 3, 2, 4, 1]
    n = 2
    labels = ["cat", "dog", "mouse", "pig", "rabbit"]
    output: ["mouse", "rabbit"]
    '''
    pass  # To be implemented
```

In [28]:

```
# Let's use cosine similarity
from sklearn.metrics.pairwise import cosine_similarity
```

In [29]:

```
# Draw an arbitrary review from test (unseen in training) documents
some_random_number = 7
search_query = documents_test[some_random_number]
search_queries = [search_query]
```

In [30]:

```
# Transform the drawn review(s) to vector(s)
vector_search_queries = vectorizer.transform(search_queries).toarray()
```

In [31]:

```
# Calculate the similarity score(s) between vector(s) and training vectors
similarity_scores = cosine_similarity(vector_search_queries, vectors_train)
```

In [32]:

```
# Let's find top 5 similar reviews
n = 5
returned_reviews = get_top_values(similarity_scores[0], n, documents_train)
```

In [33]:

```
print('Our search query:')
print(search_queries[0]) # To be added
```

```
Our search query:
Seriously the best Japanese Steakhouse this fat boy has ever been to.
If you're lucky enough to sit at the table when the Owner is cooking y
ou're in for a real treat. All the chefs make custom sauces for your m
eal and each are a highlight. Must stop destination in Las Vegas.
```

In [34]:

```
print('Most %s similar reviews:' % n)
print(returned_reviews[0])  # To be added
```

```
Most 5 similar reviews:
Best local destination hands down in Las Vegas. Great food, great atmo
sphere, great service, and the best great drinks! From the staple lobs
ter pho to the dinosaur bone barrow soup dish this place has it all. D
on't think this is your regular pho destination! This place has so man
y more to offer! Chef Khai's cooking and presentations are one of the
most creative I've seen here in Las Vegas. Also great place to watch y
our favorite NBA, Football, baseball, or basketball games! Stop on in.
..You won't be disappointed!
```

Q: Does the result make sense to you?

A: Yes, it makes sense.

Classifying positive/negative review

Naive-Bayes Classifier

In [35]:

```
# Build a Naive-Bayes Classifier

from sklearn.naive_bayes import MultinomialNB

model_nb = MultinomialNB()
model_nb.fit(vectors_train, target_train)
```

Out[35]:

```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

In [36]:

```
# Get score for training set
model_nb.score(vectors_train, target_train)
```

Out[36]:

```
0.8117318115104256
```

In [37]:

```
# Get score for test set
model_nb.score(vectors_test, target_test)
```

Out[37]:

```
0.8108899241416915
```

Logistic Regression Classifier

In [38]:

```
# Build a Logistic Regression Classifier

from sklearn.linear_model import LogisticRegression

model_lrc = LogisticRegression()
model_lrc.fit(vectors_train, target_train)
```

Out[38]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept
=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs
=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0
001,
          verbose=0, warm_start=False)
```

In [39]:

```
# Get score for training set
model_lrc.score(vectors_train, target_train)
```

Out[39]:

```
0.8421887683565789
```

In [40]:

```
# Get score for test set
model_lrc.score(vectors_test, target_test)
```

Out[40]:

```
0.8358667266135436
```

Q: What are the key features(words) that make the positive prediction?

In [47]:

```
# Let's find it out by ranking
n = 20
get_top_values(model_lrc.coef_[0], n, words)
```

Out[47]:

```
[u'amazing',
 u'best',
 u'awesome',
 u'perfection',
 u'thank',
 u'perfect',
 u'incredible',
 u'delicious',
 u'phenomenal',
 u'fantastic',
 u'heaven',
 u'highly',
 u'excellent',
 u'great',
 u'favorite',
 u'gem',
 u'impeccable',
 u'love',
 u'perfectly',
 u'outstanding']
```

A: Listed as above.

Q: What are the key features(words) that make the negative prediction?

In [51]:

```
# Let's find it out by ranking
n = 20
get_bottom_values(model_lrc.coef_[0], n, words)
```

A: (insert your comments here)

Random Forest Classifier

In [54]:

```
# Build a Random Forest Classifier

from sklearn.ensemble import RandomForestClassifier

model_rfc = RandomForestClassifier(max_depth=20,
                                   n_estimators = 50,
                                   min_samples_leaf = 10,
                                   n_jobs = -1)
model_rfc.fit(vectors_train, target_train)
```

Out[54]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='g
ini',
            max_depth=20, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=10, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=50, n_jobs=-1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [55]:

```
# Get score for training set
model_rfc.score(vectors_train, target_train)
```

Out[55]:

```
0.7973704277667085
```

In [56]:

```
# Get score for test set
model_rfc.score(vectors_test, target_test)
```

Out[56]:

```
0.7893664444168094
```

Q: What do you see from the training score and the test score?

A: The model performances on training and test data are comparable, with test accuracy slightly lower than training accuracy.

Q: Can you tell what features (words) are important by inspecting the RFC model?

In [58]:

```
n = 20
get_top_values(model_rfc.feature_importances_, n, words)
```

Out[58]:

```
[u'amazing',
 u'best',
 u'great',
 u'delicious',
 u'ok',
 u'wasn',
 u'didn',
 u'awesome',
 u'vegas',
 u'bad',
 u'rude',
 u'love',
 u'highly',
 u'minutes',
 u'worst',
 u'pretty',
 u'friendly',
 u'excellent',
 u'good',
 u'asked']
```

Extra Credit #1: Use cross validation to evaluate your classifiers

sklearn cross validation (http://scikit-learn.org/stable/modules/cross_validation.html)

In [59]:

```
# To be implemented
from sklearn.model_selection import cross_val_score

cv_scores = cross_val_score(model_lrc,
                            vectors_train,
                            target_train,
                            cv = 5,
                            scoring = 'accuracy')
cv_scores
```

Out[59]:

```
array([0.83257149, 0.83314201, 0.83367157, 0.83678741, 0.83311809])
```

Extra Credit #2: Use grid search to find best predictable classifier

[sklearn grid search tutorial (with cross validation) (http://scikit-learn.org/stable/modules/grid_search.html#grid-search)](http://scikit-learn.org/stable/modules/grid_search.html#grid-search)

[sklearn grid search documentation (with cross validation) (http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridS](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridS)

In [63]:

```
# To be implemented
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report

param_grid = [{'penalty':['l1'], 'C':[0.01, 0.1, 1, 5, 10, 100]},
              {'penalty':['l2'], 'C':[0.01, 0.1, 1, 5, 10, 100]}]
scores = ['accuracy']

for score in scores:
    clf = GridSearchCV(LogisticRegression(),
                       param_grid,
                       cv=5,
                       scoring=score)
    clf.fit(vectors_train[:500,:], target_train[:500])
    print(clf.best_params_)
    means = clf.cv_results_['mean_test_score']
    stds = clf.cv_results_['std_test_score']

    for mean, std, params in zip(means, stds, clf.cv_results_['params']):
        print(mean, std * 2, params)

    y_true, y_pred = target_test, clf.predict(vectors_test)
```

```
{'penalty': 'l2', 'C': 10}
(0.546, 0.004080204015301296, {'penalty': 'l1', 'C': 0.01})
(0.546, 0.004080204015301296, {'penalty': 'l1', 'C': 0.1})
(0.688, 0.0781342184680282, {'penalty': 'l1', 'C': 1})
(0.75, 0.08530134601495568, {'penalty': 'l1', 'C': 5})
(0.76, 0.03603556376949889, {'penalty': 'l1', 'C': 10})
(0.738, 0.042431564238952835, {'penalty': 'l1', 'C': 100})
(0.546, 0.004080204015301296, {'penalty': 'l2', 'C': 0.01})
(0.554, 0.015725874548432304, {'penalty': 'l2', 'C': 0.1})
(0.758, 0.07043329954622668, {'penalty': 'l2', 'C': 1})
(0.762, 0.07403246369376118, {'penalty': 'l2', 'C': 5})
(0.764, 0.07976788007380635, {'penalty': 'l2', 'C': 10})
(0.764, 0.08092493611074689, {'penalty': 'l2', 'C': 100})
```

In [ ]:

```
```

Yelp Data Challenge - Clustering and PCA

BitTiger DS501

Nov 2017

In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
% matplotlib inline
plt.style.use("ggplot")
```

In [3]:

```
df = pd.read_csv('dataset/last_1_years_restaurant_reviews.csv')
```

In [4]:

```
df.head()
```

Out[4]:

	business_id	name	categories	avg_stars	cool	date	funny	review_id
0	--9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	[Cajun/Creole, Steakhouses, Restaurants]	4.0	0	2017-02-14	0	Xp3ppynEvVu
1	--9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	[Cajun/Creole, Steakhouses, Restaurants]	4.0	1	2017-05-28	0	LEzphAnz0vK
2	--9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	[Cajun/Creole, Steakhouses, Restaurants]	4.0	0	2017-08-25	0	4e-cxYVdIlu2Z
3	--9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	[Cajun/Creole, Steakhouses, Restaurants]	4.0	1	2017-02-12	1	heZd0W3HuP
4	--9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	[Cajun/Creole, Steakhouses, Restaurants]	4.0	0	2017-12-10	0	exzXjy7Y2ICX

1. Cluster the review text data for all the restaurants

Define your feature variables, here is the text of the review

In [5]:

```
# Take the values of the column that contains review text data, save to a variable n
documents = df['text'].values
```

Define your target variable (any categorical variable that may be meaningful)

For example, I am interested in perfect (5 stars) and imperfect (1-4 stars) rating

In [6]:

```
# Make a column and take the values, save to a variable named "target"
df['favorable'] = (df['stars'] > 4)
target = df['favorable'].values
```

You may want to look at the statistic of the target variable

In [7]:

```
# To be implemented
target.mean()
```

Out[7]:

```
0.49041002371746206
```

Create training dataset and test dataset

In [8]:

```
from sklearn.cross_validation import train_test_split
```

In [9]:

```
# documents is your X, target is your y
# Now split the data to training set and test set
# You may want to start with a big "test_size", since large training set can easily
documents_train, documents_test, target_train, target_test = train_test_split(
    documents,
    target,
    test_size = 0.4,
    random_state = 7
)
```

Get NLP representation of the documents

Fit TfidfVectorizer with training data only, then tranform all the data to tf-idf

In [10]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [11]:

```
# Create TfidfVectorizer, and name it vectorizer, choose a reasonable max_features,
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
```

In [12]:

```
# Train the model with your training data
vectors_train = vectorizer.fit_transform(documents_train).toarray()
```

In [13]:

```
# Get the vocab of your tfidf
words = vectorizer.get_feature_names()
```

In [16]:

```
# Use the trained model to transform all the reviews
vectors_documents = vectorizer.transform(documents).toarray()
```

Cluster reviews with KMeans

Fit k-means clustering with the training vectors and apply it on all the data

In [18]:

```
# To be implemented
from sklearn.cluster import KMeans
kmeans = KMeans()
kmeans.fit(vectors_train)
```

Out[18]:

```
KMeans(copy_x=True, init='k-means++', max_iter=300, n_clusters=8, n_in
it=10,
    n_jobs=1, precompute_distances='auto', random_state=None, tol=0.00
01,
    verbose=0)
```

Make predictions on all your data

In [19]:

```
# To be implemented
assigned_cluster = kmeans.predict(vectors_documents)
```

Inspect the centroids

To find out what "topics" Kmeans has discovered we must inspect the centroids. Print out the centroids of the Kmeans clustering.

These centroids are simply a bunch of vectors. To make any sense of them we need to map these vectors back into our 'word space'. Think of each feature/dimension of the centroid vector as representing the "average" review or the average occurances of words for that cluster.

In [20]:

```
# To be implemented
print kmeans.cluster_centers_
```

```
[[  2.08424407e-03   8.32311606e-05   2.27168982e-04 ...,   1.01968652
e-03
    3.06647217e-04   1.17583552e-03]
 [  2.60949419e-03   1.22430731e-04   1.26984636e-04 ...,   1.25357273
e-03
    1.44531080e-04   1.12667491e-03]
 [  1.80534989e-03   1.31422903e-04   1.63891484e-04 ...,   7.67102778
e-04
    1.54226075e-04   4.75496262e-04]
 ...,
 [  5.03630474e-04   6.23060977e-05   6.49397645e-05 ...,   2.23595464
e-04
    2.39801519e-05   3.02701603e-04]
 [  1.99855298e-03  -8.21283146e-18   3.29575078e-04 ...,   4.06401675
e-04
    5.43791709e-05   1.20185028e-04]
 [  3.73793049e-03   1.57879092e-04   6.18984318e-04 ...,   3.22530075
e-03
    1.27320208e-04   3.39336619e-04]]
```

Find the top 10 features for each cluster.

For topics we are only really interested in the most present words, i.e. features/dimensions with the greatest representation in the centroid. Print out the top ten words for each centroid.

- Sort each centroid vector to find the top 10 features
- Go back to your vectorizer object to find out what words each of these features corresponds to.

In [21]:

```
# To be implemented
n = 10
top_centroids = kmeans.cluster_centers_.argsort()[:,-1:-n:-1]

a = np.random.randn(10)
a
```

Out[21]:

```
array([ 0.32383537, -1.00993548, -0.60647155, -0.19658069, -0.7812461
,
       -1.91354056,  0.55716966,  1.5811195 , -0.08272222,  1.17011805
])
```

In [22]:

```
for num, centroid in enumerate(top_centroids):
    print (num,",".join(words[i] for i in centroid))
```

```
(0, u'chicken,fried,food,rice,good,ordered,place,great,delicious')
(1, u'pizza,crust,good,place,great,cheese,slice,order,best')
(2, u'food,place,best,vegas,delicious,amazing,service,love,like')
(3, u'good,food,really,place,service,nice,pretty,like,great')
(4, u'burger,fries,burgers,good,place,cheese,great,food,shake')
(5, u'great,food,service,place,amazing,friendly,definitely,awesome,sta
ff')
(6, u'sushi,rolls,place,roll,ayce,great,good,fresh,fish')
(7, u'order,food,minutes,time,just,service,came,didn,said')
```

Try different k

If you set k == to a different number, how does the top features change?

In [ ]:

```
# To be implemented
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=6)
kmeans.fit(vectors_train)
```

Print out the rating and review of a random sample of the reviews assigned to each cluster to get a sense of the cluster.

In [26]:

```
# To be implemented
np.unique(assigned_cluster)
```

Out[26]:

```
array([0, 1, 2, 3, 4, 5, 6, 7], dtype=int32)
```

In [27]:

```
for i in range(kmeans.n_clusters):
    cluster = np.arange(0, vectors_documents.shape[0])[assigned_cluster==i]
    sample_reviews = np.random.choice(cluster, 2, replace=False)
    for review_index in sample_reviews:
        print df.ix[review_index]['stars'],
        print df.ix[review_index]['text']
    print
```

```
5 The "Tasty Grill" is absolutely Tasty! Everything I tried was phenom
enal! We ordered the "chicken lula kabob plate" it was delicious!!! Th
e Sampler appetizer was good as well! This restaurant really does live
up to its name! It was quick and easy delivery and our delivery guy wa
s so nice! I will definitely be eating here again soon! Maybe today! L
ol & what other restaurant gives you a free piece of cheesecake for ch
ecking in! It's dine in only so I will be eating there next time, just
for some cheesecake!
5 The yellow curry chicken bowl was awesome. The rice was perfectly co
oked and nice and fluffy.

I am very sensitive to spicy food which poses a challenge because I lo
ve Thai food. The menu states that the yellow curry chicken is mild an
d I would agree with that. It has just a little bit of heat but with t
he nice fluffy rice to go along with it even someone as sensitive as m
e can enjoy it.

At $8 for a bowl of delicious yellow curry chicken with potatoes and o
nions on top of fluffy white rice I will definitely be back.

****Thursday night food truck gathering by Fry's at Town Center.****

3 Actually it's a 3.5. Good quality, authentic to the East coast, crus
t crispy. Losses .5 because of cost. A small pizza (no matter what the
y call it) with 1.5 toppings to go north of $18. just isn't worth it.
3 I've always have been a big fan of Metro Pizza. This is the first ti
me I've ever been to this location. Considering the circumstances, I t
hought that this location was just ok.

Came here for their Three Square event which all the proceeds of their
1 dollar slice of pizza special went to this great charity. This is a
great cause and this review has nothing to do with that charity. Comin
g in here, the place had sort of a retro look to it, sort of a 'Saved
by the Bell' type of vibe. Super casual place.
```

On to the pizza. I thought it was ok, the pizza dough was good. I also thought that the sauce on the pizza was good as well. I wasn't a big fan of the cheese that they used. It seemed to separate a lot, and there was a pile of grease that had accumulated on top of the pizza. Now I have been to other Metro locations over the years. This was probably the first time I had seen this on their pizza. But other than that, it was fine.

I will continue to patronize Metro. They could have had just an off day or something. But generally, the pizza is excellent, probably some of the best I've had ever.

4 German food first time ever! Authentic and what an experience not only tasty food but when u order a shot you get a spank with a paddle. Lol! Live band and fun environment.
4 I love their drinks and their decors and also their shaved snow. I like how they have a variety of teas, and their salted cream is actually really good compared to the other places. There are plenty of tables, and their snow is fluffy and different to other places. I like their green tea the most since it has that bitterness that other places try to avoid- I personally love that taste.

The only complaint I have towards Snow White is their staff. They are definitely not the friendliest or welcoming workers. They never smile at you nor greet you when you come in. They would walk around the cafe and tell us that our drink is ready after awhile it has been out. I dont really understand why they would bring out the snow but not the drinks. That I do find odd. Else than that, I like coming here if I am nearby.

4 I moved here from Oahu and have tried many poke place n since I moved to rainbow. This is the to go poke place. Good portion fresh ingredient and I ll say it's actually really good in taste. Brought friends a couple times here n they all like it, my favorite is still Hawaiian sauce. Strongly recommend this for the price range. Just mind that there will be wait time during meal period.... n I will say the food line is not moving fast enough so be patient...
5 The food was good and juicy. The meats were meaty and not just bone/fat. The corn nuggets were a delicious surprise. The sides were fresh. Definitely coming back

5 Ate lunch and loved it. I had the Mac n cheese bacon burger and fries. The fries are delicious, all cut differently and that made it fun! Seasoning salt was awesome. The burger was so good! No wait and staff was super friendly. I highly recommend it. The table next to me had the ribs and they looked amazing!
5 Best dam burger on the Vegas strip, a little pricey but well worth it. The service is impeccable fast, friendly and polite. There is a wait on Friday and Saturday so get there early or just be prepared for a little wait. The food is sooo good especially the tater tots!!

4 Quick breakfast stop - we wished we had more time to enjoy. HUGE portions beware but oh so good. If you're thinking of sharing, share.

```
Omelets with homemade chips instead of hashbrowns and the pumpkin and
another type of bread (both great) were served first.  No Amex accepte
d, but great service.
5 This place is wonderful!  I wanted to enjoy a wonderful steak dinner
with a great atmosphere,  and this place delivered.  The staff was ver
y friendly and well educated on their menu.   I will be back to eat th
e Cesar Salad again!  They make the Cesar tableside by scratch, and wa
s plate licking good. Come prepared with an empty stomach.
```

2. Cluster all the reviews of the most reviewed restaurant

Let's find the most reviewed restaurant and analyze its reviews

In [28]:

```
# Find the business who got most reviews, get your filtered df, name it df_top_resta
df_unique = df.business_id.value_counts()
df_unique
df_top_restaurant = df[df['business_id'] == "RESDUcs7fIiihp38-d6_6g"]
```

We can also load restaurant profile information from the business dataset (optional)

In [29]:

```
# Load business dataset (optional)
# Take a look at the most reviewed restaurant's profile (optional)
pass
```

Vectorize the text feature

In [30]:

```
# Take the values of the column that contains review text data, save to a variable n
documents_top_restaurant = df_top_restaurant['text'].values
documents_top_restaurant.shape
```

Out[30]:

```
(1293,)
```

Define your target variable (for later classification use)

Again, we look at perfect (5 stars) and imperfect (1-4 stars) rating

In [31]:

```
# To be implemented
target_top_restaurant = (df_top_restaurant['stars'] > 4).astype(int).values
target_top_restaurant.shape
```

Out[31]:

```
(1293,)
```

Check the statistic of the target variable

In [32]:

```
# To be implemented
print('statistics of the target variable')
print('median value: %f' % (np.median(target_top_restaurant)))
print('mean value: %f' % (np.mean(target_top_restaurant)))
print('standard deviation value: %f' % (np.std(target_top_restaurant)))
```

```
statistics of the target variable
median value: 0.000000
mean value: 0.381284
standard deviation value: 0.485702
```

Create training dataset and test dataset

In [33]:

```
from sklearn.cross_validation import train_test_split
```

In [34]:

```
# documents_top_restaurant is your X, target_top_restaurant is your y
# Now split the data to training set and test set
# Now your data is smaller, you can use a typical "test_size", e.g. 0.3-0.7
x_train, x_test, y_train, y_test = train_test_split(documents_top_restaurant, target
```

Get NLP representation of the documents

In [35]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [36]:

```
# Create TfidfVectorizer, and name it vectorizer
vectorizer_top = TfidfVectorizer(stop_words = 'english', max_features = 3000)
```

In [37]:

```
# Train the model with your training data
train_vector_top = vectorizer_top.fit_transform(x_train).toarray()
```

In [38]:

```
# Get the vocab of your tfidf
vocab_top = vectorizer_top.get_feature_names()
```

In [39]:

```
# Use the trained model to transform the test data
test_vector_top = vectorizer_top.fit_transform(x_test).toarray()
```

In [40]:

```
# Use the trained model to transform all the data
doc_vector_top = vectorizer_top.transform(documents_top_restaurant).toarray()
```

In [41]:

```
doc_vector_top.shape
```

Out[41]:

```
(1293, 3000)
```

Cluster reviews with KMeans

Fit k-means clustering on the training vectors and make predictions on all data

In [42]:

```
# To be implemented

kmeans = KMeans(random_state=42)
kmeans.fit(train_vector_top)
```

Out[42]:

```
KMeans(copy_x=True, init='k-means++', max_iter=300, n_clusters=8, n_in
it=10,
    n_jobs=1, precompute_distances='auto', random_state=42, tol=0.0001
,
    verbose=0)
```

Make predictions on all your data

In [43]:

```
# To be implemented
target_labels = kmeans.predict(doc_vector_top)
```

Inspect the centroids

In [44]:

```
# To be implemented

print kmeans.cluster_centers_
```

```
[[  5.35289646e-03   1.49919837e-03   2.35014887e-02 ...,   3.89252654
e-03
    9.75781955e-19  -4.87890978e-19]
 [  4.33680869e-18  -5.42101086e-19   6.69845027e-03 ...,   2.31996705
e-03
    3.79470760e-19   2.71050543e-19]
 [  5.85279815e-03   2.92117650e-03   4.84695523e-03 ...,   1.22192404
e-03
    9.75781955e-19   1.13326592e-03]
 ...,
 [  1.30104261e-18   3.25260652e-19   1.05872536e-02 ...,   7.70419573
e-03
   -3.79470760e-19   1.62630326e-19]
 [  4.18240206e-03  -1.30104261e-18   8.89000401e-03 ...,   1.00789408
e-03
    1.12268849e-03   7.11146641e-04]
 [  3.46944695e-18  -3.25260652e-19   9.58750608e-03 ...,   2.99839321
e-03
    0.00000000e+00   2.71050543e-19]]
```

Find the top 10 features for each cluster.

In [45]:

```
# To be implemented
n = 10
top_centroids = kmeans.cluster_centers_.argsort()[:,-1:-n:-1]

a = np.random.randn(10)
a
```

Out[45]:

```
array([-0.34048605, -1.74348922,  1.27579183,  0.53686884,  0.05638433
,
        0.37382413,  0.60994877, -0.62734361,  0.41091988,  0.03312756
])
```

In [46]:

```
for num, centroid in enumerate(top_centroids):
    print (num,",".join(words[i] for i in centroid))
```

```
(0, u'drivers,mmm,concern,duty,magnificent,decently,courses,incredible
,event')
(1, u'available,million,bazaar,bbq,concern,dollars,gnocchi,ingredients
,croutons')
(2, u'bus,lava,dr,concern,bazaar,incident,carving,country,discount')
(3, u'industrial,appear,country,carving,bazaar,concern,mason,bus,incid
ent')
(4, u'bazaar,garlic,mmm,country,concern,magnificent,bus,network,caused
')
(5, u'ahead,concern,clock,intimate,bazaar,crack,luckily,acai,million')
(6, u'bazaar,concern,directed,country,fork,drinker,group,basil,chang')
(7, u'crack,concern,intimate,network,gas,bazaar,country,ingredients,co
ns')
```

Print out the rating and review of a random sample of the reviews assigned to each cluster to get a sense of the cluster.

In [47]:

```
# To be implemented
for i in range(kmeans.n_clusters):
    cluster = np.arange(0, vectors_documents.shape[0])[assigned_cluster==i]
    sample_reviews = np.random.choice(cluster, 2, replace=False)
    for review_index in sample_reviews:
        print df.ix[review_index]['stars'],
        print df.ix[review_index]['text']
    print
```

```
4 Sit down, but has a fast food joint that satisfies.  Yes, the hot fo
od is either on the spit or already sitting under warmers, but it's so
good.  I'm not doing a good job selling this place, but you have to tr
y it.

This tiny restaurant on Pecos, in a massive parking lot has a variety
to satisfy the taste buds.  I guess you either like it or you don't.
The kebobs are not sitting out, but in it's own refrigerated section.
The skewer is a decent size and I would consider it a generous portion
for the price point.

I'm totally into their chicken curry here.  It's a bowl of coconut mil
k, spiciness with thin strips of chicken and all the fresh parsley my
bowl can hold.  The basmati rice is so fluffy, I'm glad it's all soake
d in my curry sauce to make it that much more satisfying.  My husband
really enjoys their chicken shawarma bowl, with some free added ingred
ients and some for an up charge.

Their nachos are made with fried pita bread and their seasoning for th
```

3. Use PCA to reduce dimensionality

Stardardize features

Your X_train and X_test

In [48]:

```
from sklearn.preprocessing import StandardScaler

# To be implemented
ss = StandardScaler()
doc_ss = ss.fit_transform(doc_vector_top)
x_train_scale, x_test_scale, y_train, y_test = train_test_split(doc_ss, target_top_1
                                                                random_state = 42)
```

Use PCA to transform data (train and test) and get princial components

In [49]:

```
from sklearn.decomposition import PCA

# Let's pick a n_components
n_components = 50

# To be implemented
pca = PCA(n_components=n_components)
train_components = pca.fit_transform(x_train_scale)
test_components = pca.transform(x_test_scale)
explained_variance = np.sum(train_components.T.dot(train_components))
print(train_components.T.dot(train_components).shape)
```

```
(50, 50)
```

See how much (and how much percentage of) variance the principal components explain

In [50]:

```
# To be implemented

print pca.explained_variance_[:10]
```

```
[ 28.32172053  28.22278384  21.57801673  20.0753514   19.20636374
  19.171516    17.91174146  16.8330347   16.05805784  15.27252625]
```

In [51]:

```
# To be implemented
print pca.explained_variance_ratio_[:10]
```

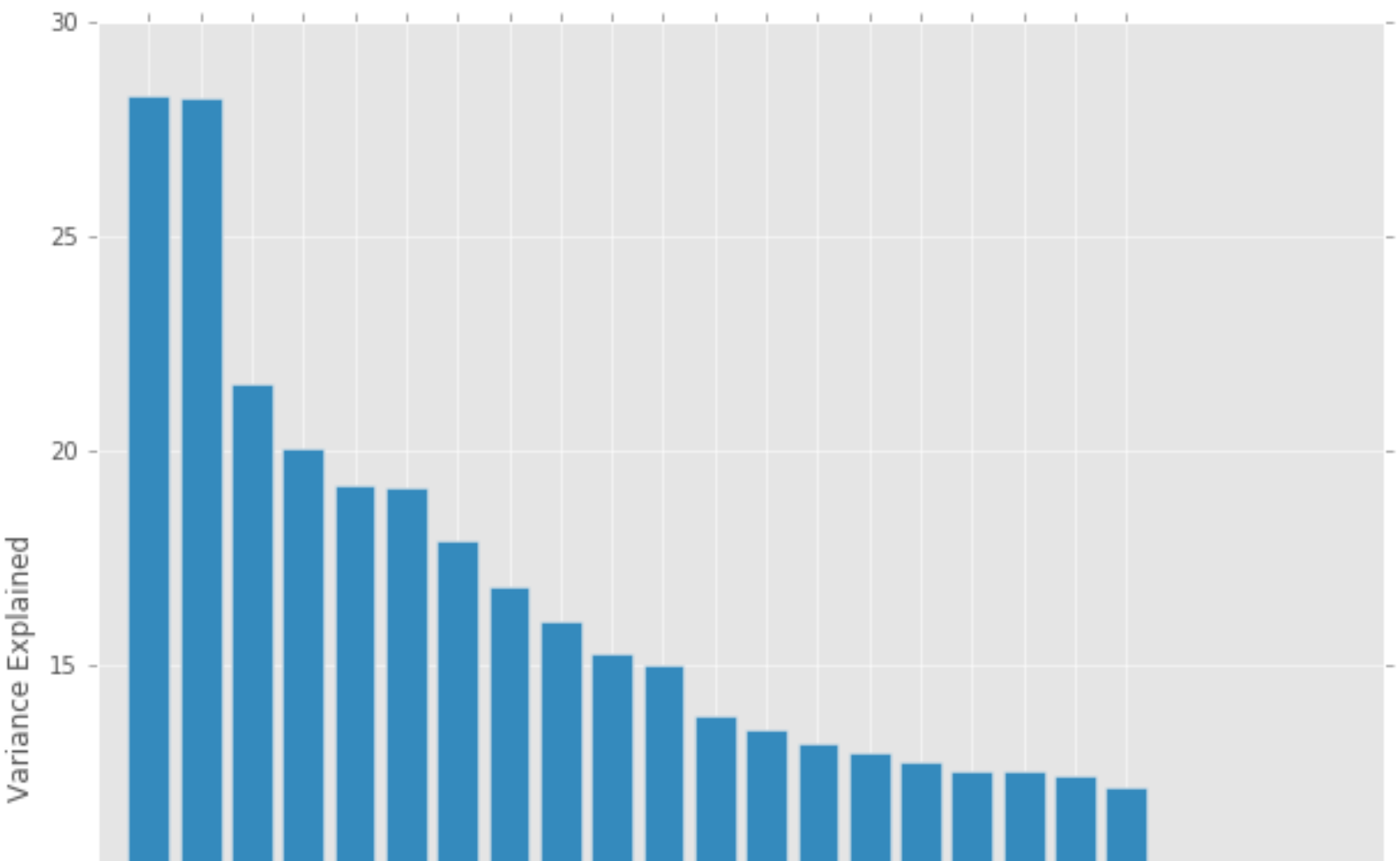
```
[ 0.00961856  0.00958496  0.00732828  0.00681794  0.00652282  0.006510
99
  0.00608314  0.0057168   0.0054536   0.00518682]
```

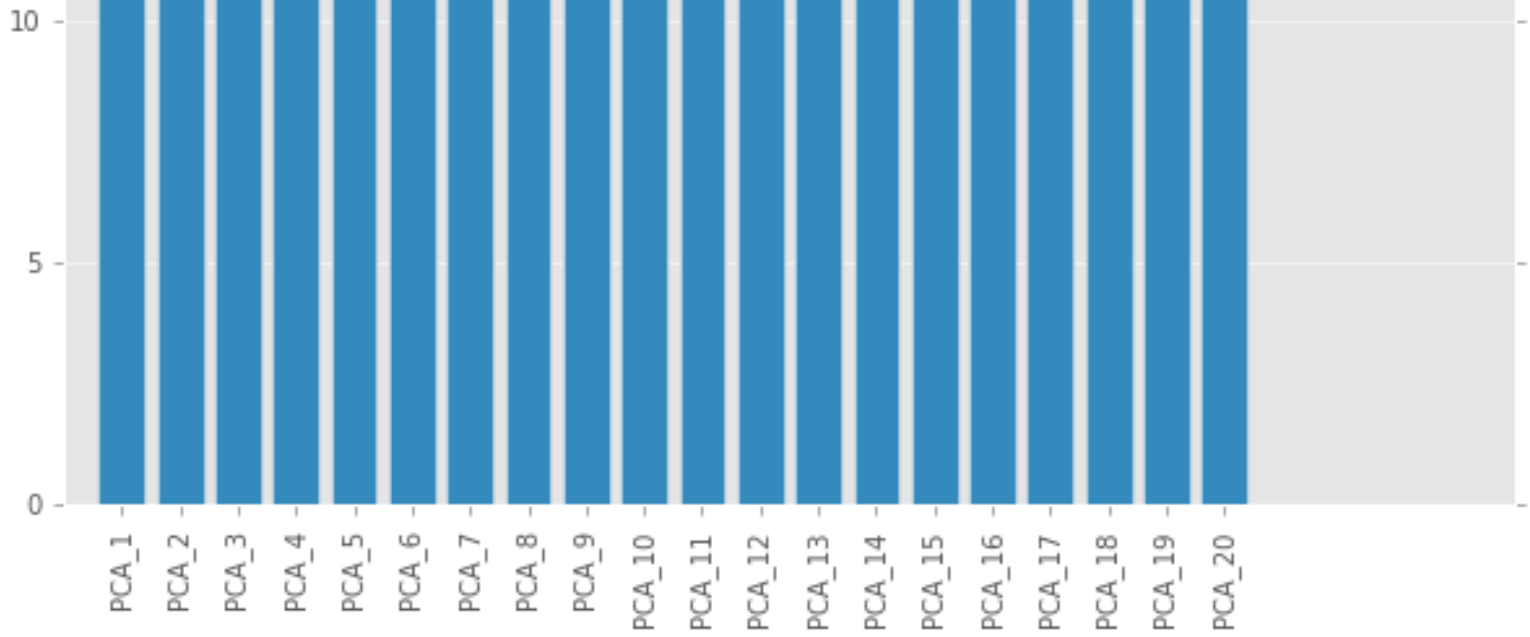
Viz: plot proportion of variance explained with top principal components

For clear display, you may start with plotting <=20 principal components

In [52]:

```
# To be implemented
n_col_to_display = 20

pca_range = np.arange(n_col_to_display) + 1
pca_names = ['PCA_%s' % i for i in pca_range]

plt.figure(figsize=(10, 10))
plt.bar(pca_range,
        pca.explained_variance_[:n_col_to_display],
        align='center')
xticks = plt.xticks(pca_range,
                    pca_names,
                    rotation=90)
plt.ylabel('Variance Explained')
plt.show()
```





Classifying positive/negative review with PCA preprocessing

Logistic Regression Classifier

Use standardized tf-idf vectors as features

In [53]:

```
# Build a Logistic Regression Classifier, train with standardized tf-idf vectors

from sklearn.linear_model import LogisticRegression

# To be implemented

model_lrc = LogisticRegression()

model_lrc.fit(x_train_scale, y_train)
```

Out[53]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept
=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs
=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0
001,
          verbose=0, warm_start=False)
```

In [54]:

```
# Get score for training set
model_lrc.score(x_train_scale, y_train)
```

Out[54]:

```
1.0
```

In [55]:

```
# Get score for test set

model_lrc.score(x_test_scale, y_test)
```

Out[55]:

```
0.716494845360824 75
```

Very overfitting!

Use (Stardardized + PCA) tf-idf vectors as features

In [56]:

```
# Build a Logistic Regression Classifier, train with PCA tranformed X

from sklearn.linear_model import LogisticRegression

# To be implemented

model_lrc_pca = LogisticRegression()

model_lrc_pca.fit(train_components, y_train)
```

Out[56]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept
=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs
=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0
001,
          verbose=0, warm_start=False)
```

In [57]:

```
# Get score for training set
model_lrc_pca.score(train_components, y_train)
```

Out[57]:

```
0.759116022099447 51
```

In [58]:

```
# Get score for test set, REMEMBER to use PCA-transformed X!

model_lrc_pca.score(test_components, y_test)
```

Out[58]:

```
0.71907216494845361
```

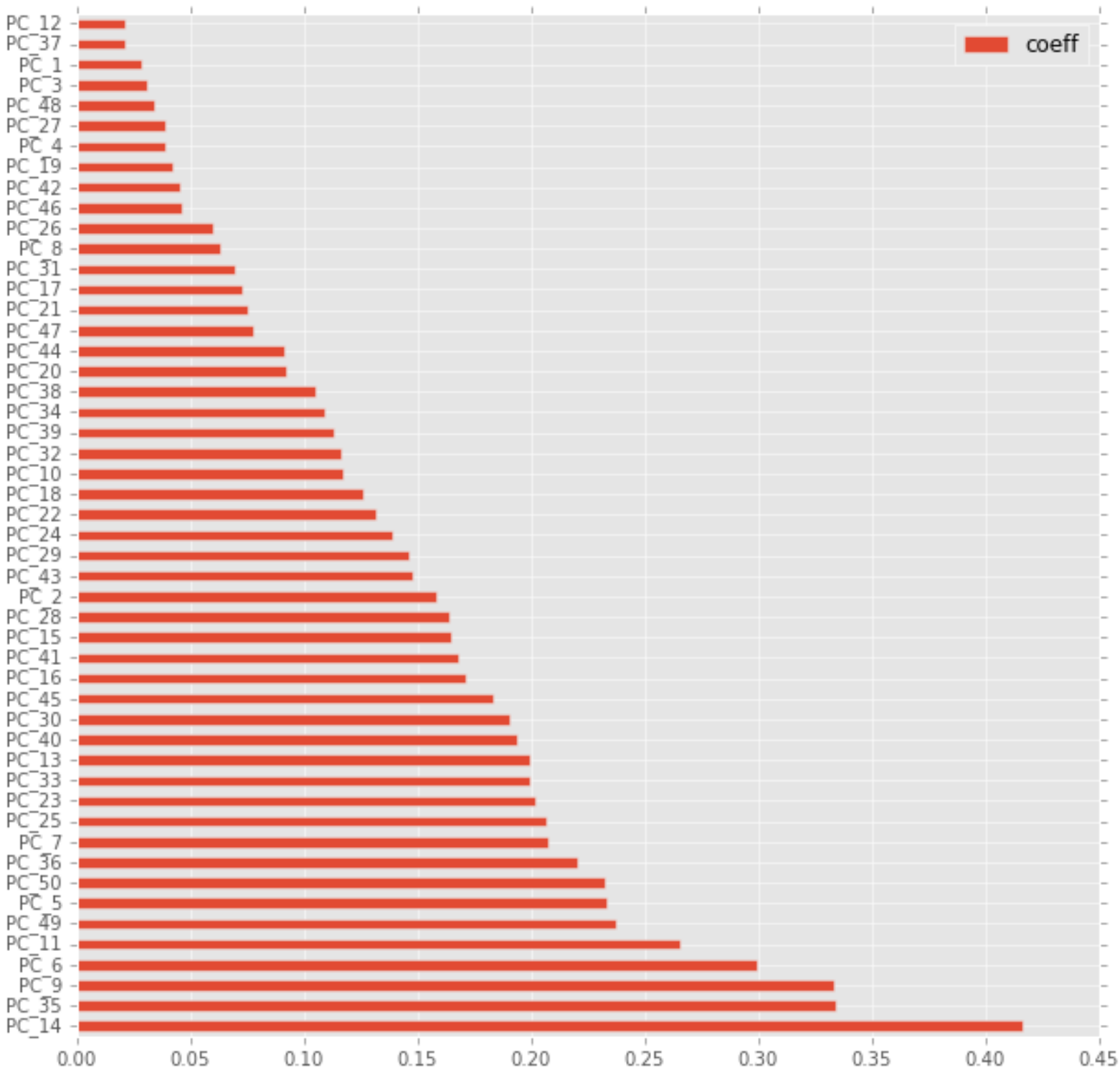
Q: What do you see from the training score and the test score? How do you compare the results from PCA and non-PCA preprocessing?

A: By comparing the results from PCA and non-PCA, we can see that the Stardardized + PCA has much reduced the overfitting probelm.

You can plot the coefficients against principal components

In [59]:

```
# To be implemented
pca_range = np.arange(pca.n_components_) + 1
pca_names = ['PC_%s' % i for i in pca_range]

df_coeffs = pd.DataFrame(list(zip(pca_names, abs(model_lrc_pca.coef_.flatten())))).s
df_coeffs.columns = ['PCs', 'coeff']

ax = df_coeffs.plot.barh(figsize=(10, 10))
t = np.arange(pca.n_components_)
ax.set_yticks(t)
ax.set_yticklabels(df_coeffs['PCs'])
plt.show()
```



Random Forest Classifier

Use standardized tf-idf vectors as features

In [60]:

```
# Build a Random Forest Classifier

from sklearn.ensemble import RandomForestClassifier

# To be implemented
model_rf = RandomForestClassifier()

model_rf.fit(x_train_scale, y_train)
```

Out[60]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='g
ini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [61]:

```
# Get score for training set
model_rf.score(x_train_scale, y_train)
```

Out[61]:

```
0.98342541436464093
```

In [62]:

```
# Get score for test set
model_rf.score(x_test_scale, y_test)
```

Out[62]:

```
0.70618556701030932
```

Also overfitting!

Use (Stardardized + PCA) tf-idf vectors as features

In [63]:

```
# Build a Random Forest Classifier

from sklearn.ensemble import RandomForestClassifier

# To be implemented
model_rf_pca = RandomForestClassifier()

model_rf_pca.fit(train_components, y_train)
```

Out[63]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='g
ini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [64]:

```
# Get score for training set
model_rf_pca.score(train_components, y_train)
```

Out[64]:

```
0.9823204419889503
```

In [65]:

```
# Get score for test set, REMEMBER to use PCA-transformed X!
model_rf_pca.score(test_components, y_test)
```

Out[65]:

```
0.672680412371134O7
```

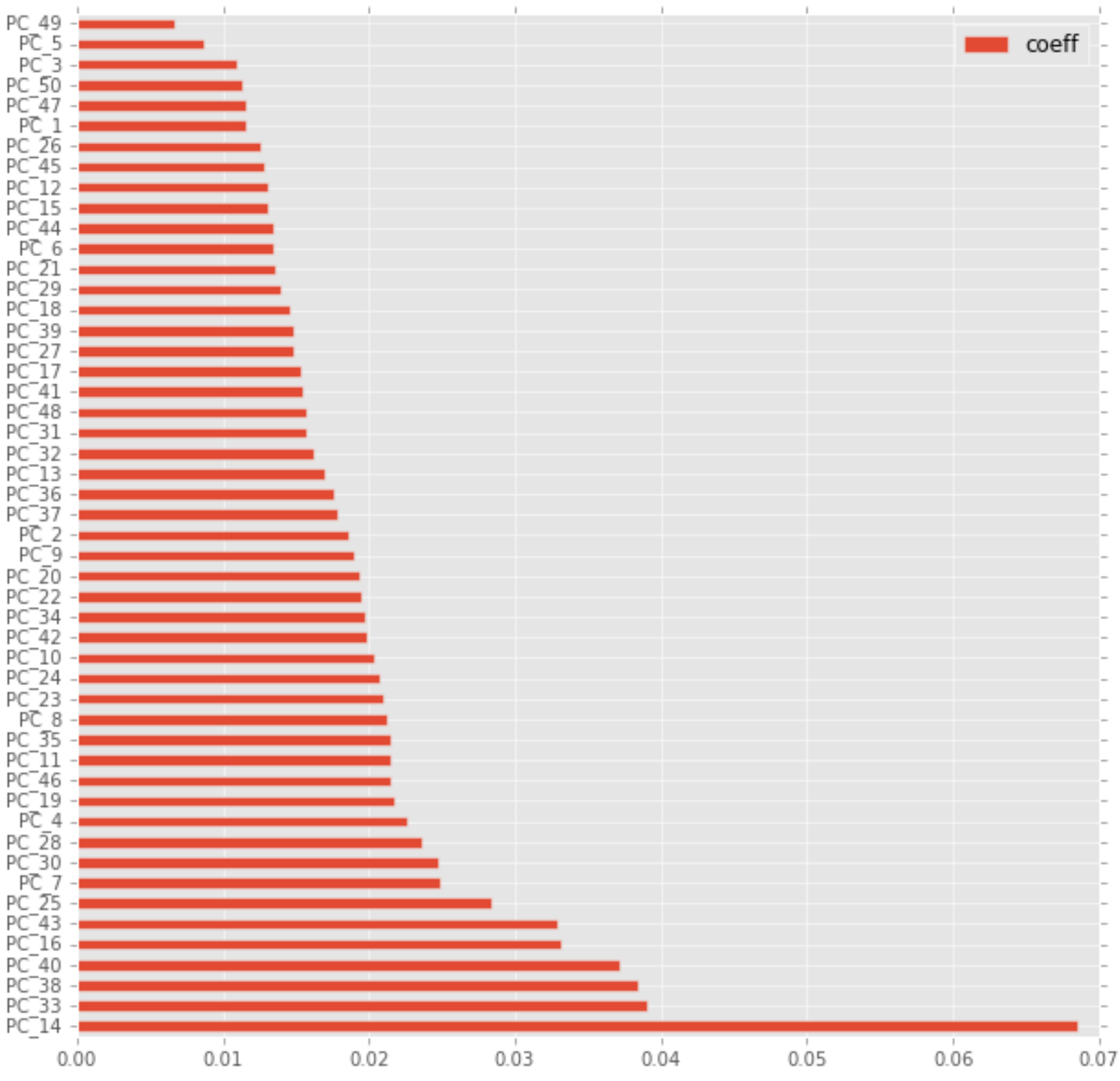
Q: What do you see from the training result and the test result?

A: With the random forest model, applying PCA didn't help the overfitting problem.

You can plot the feature importances against principal components

In [66]:

```
# To be implemented
pca_range = np.arange(pca.n_components_) + 1
pca_names = ['PC_%s' % i for i in pca_range]

df_coeffs = pd.DataFrame(list(zip(pca_names, abs(model_rf_pca.feature_importances_))
df_coeffs.columns = ['PCs', 'coeff']

ax = df_coeffs.plot.barh(figsize=(10, 10))
t = np.arange(pca.n_components_)
ax.set_yticks(t)
ax.set_yticklabels(df_coeffs['PCs'])
plt.show()
```



Extra Credit #1: Can you cluster restaurants from their category information?

Hint: a business may have mutiple categories, e.g. a restaurant can have both "Restaurants" and "Korean"

In [67]:

```
# To be implemented
category = df['categories'].values
category
```

Out[67]:

```
array(['[Cajun/Creole, Steakhouses, Restaurants]',
       '[Cajun/Creole, Steakhouses, Restaurants]',
       '[Cajun/Creole, Steakhouses, Restaurants]', ...,
       '[Vegetarian, Restaurants, Mexican]',
       '[Vegetarian, Restaurants, Mexican]',
       '[Vegetarian, Restaurants, Mexican]'], dtype=object)
```

In [68]:

```
df['favorable'] = df['stars']>4
target = df['favorable'].values
```

In [69]:

```
from sklearn.cross_validation import train_test_split
cate_train, cate_test, y_train, y_test = train_test_split(category, target, test_si
```

In [70]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(stop_words = 'english', max_features = 3000)
vectors_train_category = vectorizer.fit_transform(cate_train).toarray()
words_category = vectorizer.get_feature_names()
vectors_category = vectorizer.transform(category).toarray()
```

In [71]:

```
from sklearn.cluster import KMeans
kmeans = KMeans(random_state=42)
kmeans.fit(vectors_train_category)
```

Out[71]:

```
KMeans(copy_x=True, init='k-means++', max_iter=300, n_clusters=8, n_in
it=10,
    n_jobs=1, precompute_distances='auto', random_state=42, tol=0.0001
,
    verbose=0)
```

In [72]:

```
target_labels = kmeans.predict(vectors_category)
```

In [73]:

```
print kmeans.cluster_centers_
```

```
[[ -2.37440276e-17   7.40471211e-03   8.18333989e-03 ...,   1.00119294
e-18
    1.33401708e-02  -3.94215910e-16]
 [  1.48535698e-16   9.28077060e-17  -5.52943108e-17 ...,   1.64154985
e-18
    3.80121282e-16   1.39645240e-16]
 [  2.59883261e-16  -5.16947596e-16  -6.26668856e-17 ...,  -3.56600871
e-18
    5.54027310e-16   4.54063870e-16]
 ...,
 [  2.31910845e-16   1.53312201e-03  -6.09321621e-17 ...,  -2.15823995
e-18
    2.07937240e-03   3.74700271e-16]
 [  2.88289358e-16   8.82546405e-03  -6.46184495e-17 ...,  -4.99241219
e-18
    3.39572120e-16   5.33861150e-16]
 [  7.42678488e-17   9.15066634e-17  -5.03069808e-17 ...,   1.44503821
e-18
    2.64328490e-16  -7.02563008e-17]]
```

In [74]:

```
n = 10
top_centroids = kmeans.cluster_centers_.argsort()[:,-1:-n:-1]

a = np.random.randn(10)
a
```

Out[74]:

```
array([ 0.20231625, -0.02692704, -0.96572502, -1.76942099, -1.77597561
,
        1.36803138, -2.07970511,  0.99877004, -0.06498306, -1.65802386
])
```

In [75]:

```
for num, centroid in enumerate(top_centroids):
    print (num,",".join(words[i] for i in centroid))
```

```
(0, u'advantage,asian,aware,65,bazaar,bagel,america,agree,attentivenes
s')
(1, u'banana,anniversary,27,attentiveness,aioli,2016,asking,avocados,a
loha')
(2, u'36,3pm,13,attentiveness,available,bartender,agree,3x,added')
(3, u'attentiveness,asia,animal,8pm,bardot,avocados,26,bakery,anticipa
ted')
(4, u'agree,advice,attentiveness,bao,400,ad,absolute,baguette,availabl
e')
(5, u'13,arcade,bartender,400,attentiveness,bakery,avocados,bag,availa
ble')
(6, u'27,areas,13,bed,able,attentiveness,arcade,baja,bartender')
(7, u'appointment,attentiveness,bang,advice,areas,27,agree,apples,barb
ecue')
```

In [76]:

```
for i in range(kmeans.n_clusters):
    cluster = np.arange(0, vectors_documents.shape[0])[assigned_cluster==i]
    sample_reviews = np.random.choice(cluster, 2, replace=False)
    for review_index in sample_reviews:
        print df.ix[review_index]['stars'],
        print df.ix[review_index]['text']
    print
```

```
2 Maybe our expectations were higher than they should have been  .. th
e Tuna pizza was tasty.
The large $200 chefs choice Sushi was presented well but for that much
money it was lack luster at best. The sauce they put on the whole frie
d fish was disgusting! And trying to get a chicken dish for my mother-
in-law with out being all curry sauced up proved to cause us to be ser
ved a mediocre chicken and potatoes dish. The medallion potatoes were
dry and flavorless. We didn't eat them, nor did we eat much of the chi
cken. Oh and the specialty cocktail tasted horrible, we didn't drink i
t either.
For such an upscale restaurant you'd think they would notice what we d
idn't eat or drink... nope!
5 This is my favorite dining restaurant.
I really like the decorations
Spicy chicken wing is a must!!!!
All staff is so nice and gentle.
I surely visit again and again.
```

```
1 Horrible pizza. They only give you 1 little pack of  parmesan cheese
```

Extra Credit #2: Can you try different distance/similarity metrics for clusterings, e.g. Pearson correlation, Jaccard distance, etc.

Hint: You can take a look at scipy (http://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.pdist.html#scipy.spatial.distance.pdist documentations to use other distances

Q: How do you compare with Cosine distance or Euclidean distance?

In [77]:

```
# To be implemented
#kmeans = KMeans(random_state=42, )

#kmeans.fit(vectors_train)
```

In [78]:

```
import nltk
from scipy.spatial.distance import cdist
from nltk.cluster.kmeans import KMeansClusterer
```

In [80]:

```
NUM_CLUSTERS = 5
data = doc_vector_top

# clustering using cosine_distance
kclusterer_cos = KMeansClusterer(NUM_CLUSTERS, distance=nltk.cluster.util.cosine_dis
assigned_clusters_cos = kclusterer_cos.cluster(data, assign_clusters=True)
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

Extra Credit #3: Can you cluster categories from business entities? What does it mean by a cluster?

Hint: Think the example where words can be clustered from the transposed tf-idf matrix.

What is the difference between "business entities" and the "category" of a business? Are they the same thing?
I assume they are not, otherwise this question would be the same as Extra Credit #1.
Here, from looking at the df, the columns that representing business are "bueiness_id", "name", and "category".
So, the below cluster will base on the info from "bueiness_id", "name", and "category" columns, representing "business entities".

In [81]:

```
# To be implemented
entities = df[['business_id', 'name', 'categories']]
entities = entities.drop_duplicates()
category = entities['categories'].values
```

In [82]:

```
vectorizer_top = TfidfVectorizer(stop_words = 'english', max_features = 5000)
```

In [83]:

```
train_vector_top = vectorizer_top.fit_transform(category).toarray()
```

In [84]:

```
category_vector = vectorizer.transform(category).toarray()
```

In [85]:

```
category_vector
```

Out[85]:

```
array([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       ...,
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.]])
```

In [86]:

```
vocab_top = vectorizer_top.get_feature_names()
```

In [87]:

```
train_vector_top
```

Out[87]:

```
array([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       ...,
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.]])
```

In [88]:

```
kmeans = KMeans(n_clusters=5, random_state=42)
kmeans_cluster = kmeans.fit(train_vector_top)
```

In [89]:

```
print kmeans.cluster_centers_
```

```
[[  4.77048956e-18   4.33680869e-18   1.35525272e-18 ...,  -2.43945489
e-19
    2.60208521e-18   3.94432752e-03]
 [  3.79470760e-18  -3.68628739e-18   1.24683250e-18 ...,   6.77626358
e-19
    1.69135539e-17   3.55618313e-17]
 [  1.88121503e-03   2.94737738e-03   4.02791495e-04 ...,   3.01788490
e-04
    4.15251660e-03   8.95745367e-03]
 [ -1.35525272e-17   9.52881047e-04  -1.89735380e-18 ...,  -1.97866896
e-18
    1.43871107e-03  -5.72458747e-17]
 [  2.92734587e-18  -1.12757026e-17   1.19262239e-18 ...,   3.25260652
e-19
    1.12757026e-17   3.20923843e-17]]
```

In [90]:

```
import numpy as np

# We will need these helper methods pretty soon

def get_top_values(lst, n, labels):
    '''
    INPUT: LIST, INTEGER, LIST
    OUTPUT: LIST

    Given a list of values, find the indices with the highest n values.
    Return the labels for each of these indices.

    e.g.
    lst = [7, 3, 2, 4, 1]
    n = 2
    labels = ["cat", "dog", "mouse", "pig", "rabbit"]
    output: ["cat", "pig"]
    '''
    return [labels[i] for i in np.argsort(lst)[::-1][:n]]  # np.argsort by default s

def get_bottom_values(lst, n, labels):
    '''
    INPUT: LIST, INTEGER, LIST
    OUTPUT: LIST

    Given a list of values, find the indices with the lowest n values.
    Return the labels for each of these indices.

    e.g.
    lst = [7, 3, 2, 4, 1]
    n = 2
    labels = ["cat", "dog", "mouse", "pig", "rabbit"]
    output: ["mouse", "rabbit"]
    '''
    pass  # To be implemented
```

In [91]:

```
n = 10
top_centroids = kmeans.cluster_centers_.argsort()[:,-1:-n:-1]

a = np.random.randn(10)
a
```

Out[91]:

```
array([ 0.54230895,  0.2082081 , -0.73081438,  0.0727607 , -0.55991212
,
       -0.44756505,  0.06370506,  0.20935079,  2.27106035, -1.29210237
])
```

In [92]:

```
for index, point in enumerate(kmeans_cluster.cluster_centers_):
    print('%d center has the top prominent words:' % (index))
    print(get_top_values(point, 20, words))
```

```
0 center has the top prominent words:
[u'advice', u'agree', u'400', u'attitude', u'average', u'8pm', u'8am',
u'beers', u'13', u'bartenders', u'acting', u'addition', u'ambience', u
'af', u'asian', u'authenticity', u'area', u'basil', u'24', u'avoid']
1 center has the top prominent words:
[u'appreciate', u'attitude', u'advice', u'agree', u'bardot', u'appoint
ment', u'banh', u'avoid', u'aren', u'3pm', u'36', u'27', u'anytime', u
'basil', u'13', u'ad', u'baked', u'bathrooms', u'bathroom', u'400']
2 center has the top prominent words:
[u'attitude', u'asian', u'agree', u'average', u'8pm', u'animal', u'8am
', u'beers', u'avoid', u'acting', u'away', u'authenticity', u'49', u'2
6', u'bare', u'bar', u'advantage', u'36', u'3pm', u'absolute']
3 center has the top prominent words:
[u'13', u'27', u'bartenders', u'aren', u'area', u'attitude', u'36', u'
3pm', u'bake', u'atleast', u'400', u'baklava', u'beef', u'added', u'av
oid', u'agree', u'average', u'able', u'app', u'admittedly']
4 center has the top prominent words:
[u'anniversary', u'bananas', u'2016', u'aioli', u'27', u'attitude', u'
8pm', u'aloha', u'atop', u'agree', u'anticipated', u'avoid', u'arena',
u'bare', u'3x', u'asks', u'aren', u'26', u'bao', u'baklava']
```

In [ ]:

```

```

In [ ]:

```

```

Extra Credit #4: What are the characteristics of each of the clustered ? For each cluster, which restaurant can best represent ("define") its cluster?

Hint: how to interpret "best"?

Here I define the "best" representation of a cluster as the value point that is closest to its cluster center, i.e. the distance between the point and the cluster center is the smallest. That point is defined as the most representative of its cluster.

In [235]:

```
# To be implemented
restaurant = df.name.unique()
best_rep = kmeans.cluster_centers_.argsort()[:, -1]
# argsort orders from small to big, -1 is used to select the one with highest weigh
for i, rep in enumerate(best_rep):
    print "Cluster %i: Best represent Restaurant %s" % (i, restaurant[i] )
```

```
Cluster 0: Best represent Restaurant Delmonico Steakhouse
Cluster 1: Best represent Restaurant Bavette's Steakhouse & Bar
Cluster 2: Best represent Restaurant Michael Mina
Cluster 3: Best represent Restaurant Sin City Thai Restaurant
Cluster 4: Best represent Restaurant Fresh Buffet
```

Extra Credit #5: Can you think of other use cases that clustering can be used?

Hint: of course you can make use of other yelp dataset. You can try anything you want as long as you can explain it.

This clustering + TF-IDF technique can also be used for book/ artical recommendation.
Clustering along can also be used to find pattern in data. For example, subset user data into subgroups.

Yelp Data Challenge - Restaurant Recommender

BitTiger DS501

Nov 2017

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
% matplotlib inline
plt.style.use("ggplot")
```

In [2]:

```
df = pd.read_csv('dataset/last_1_years_restaurant_reviews.csv')
```

In [3]:

```
df.head()
```

Out[3]:

	business_id	name	categories	avg_stars	cool	date	funny	review_id
0	--9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	[Cajun/Creole, Steakhouses, Restaurants]	4.0	0	2017-02-14	0	Xp3ppynEvVu
1	--9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	[Cajun/Creole, Steakhouses, Restaurants]	4.0	1	2017-05-28	0	LEzphAnz0vK
2	--9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	[Cajun/Creole, Steakhouses, Restaurants]	4.0	0	2017-08-25	0	4e-cxYVdIlu2Z
3	--9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	[Cajun/Creole, Steakhouses, Restaurants]	4.0	1	2017-02-12	1	heZd0W3HuP
4	--9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	[Cajun/Creole, Steakhouses, Restaurants]	4.0	0	2017-12-10	0	exzXjy7Y2ICX

1. Clean data and get rating data

Select relevant columns in the original dataframe

In [4]:

```
# Get business_id, user_id, stars for recommender

df_rec = df[['business_id', 'user_id', 'stars']]
df_rec.head()
```

Out[4]:

	business_id	user_id	stars
0	--9e1ONYQuAa-CB_Rrw7Tw	KC8H7qTZVPlEnanw9fG43g	5
1	--9e1ONYQuAa-CB_Rrw7Tw	3RTesI_MAwct13LWm4rhLw	4
2	--9e1ONYQuAa-CB_Rrw7Tw	EAOt1UQhJD0GG3I_jv7rWA	5
3	--9e1ONYQuAa-CB_Rrw7Tw	OtKA03ALQQ1CBhtaJod_Jw	2
4	--9e1ONYQuAa-CB_Rrw7Tw	Ymtd4cQypep_QZJ-qJsHuA	5

In [5]:

```
df_rec['stars'].value_counts()
```

Out[5]:

```
5    83329
4    33537
1    22348
3    17546
2    13157
Name: stars, dtype: int64
```

In [6]:

```
df_user_count = df_rec['user_id'].value_counts()
df_user_count.head()
df_user_count.sum()
```

Out[6]:

```
169917
```

In [7]:

```
df_user_count.index
```

Out[7]:

```
Index([u'bLbSNkLggFnqwNNzzq-Ijw', u'JaqcCU3nxReTW2cBLHounA',
       u'YE54kKTuqJJPNYWIKIpOEQ', u'keBv05MsMFBd0Hu98vXThQ',
       u'P0rGN5mDue55uIiAzkrRlw', u'U4INQZOPSUaj8hMjLlZ3KA',
       u'bvzwsK8u5i0Kvvyfy7aT1Q', u'OXSJCjKtvZPf-YPDCXcWZg',
       u'8dxkcmAXY4ttrVFD1GhbdQ', u'sCelgwFoaNLMC_A7Y8usCw',
       ...
       u'Wh5DF9NfhhQzgGNIyhxDAA', u'DTL1ZIqnXlxE5gwWWLi8qw',
       u'EXLQpXJDGmcN-7_douwqZw', u'Rpqj6raY0tgxQuyu16NG2Q',
       u'Md5L1KjZfxTgVgq_Ions5A', u'uV74b2fN1xBkQVPHaY51CQ',
       u'aWJRNycIBB9rXqUniSAA9w', u'7KYnhbHu1Rr2HzFhFLGXxg',
       u'ekfX9Zb5VPYCIG-DAvQSWg', u'GIaCC4TfsHufCa66YW57Pg'],
      dtype='object', length=92274)
```

In [8]:

```
df_user_count.describe()
```

Out[8]:

```
count    92274.000000
mean         1.841440
std          2.670796
min          1.000000
25%          1.000000
50%          1.000000
75%          2.000000
max        156.000000
Name: user_id, dtype: float64
```

In [9]:

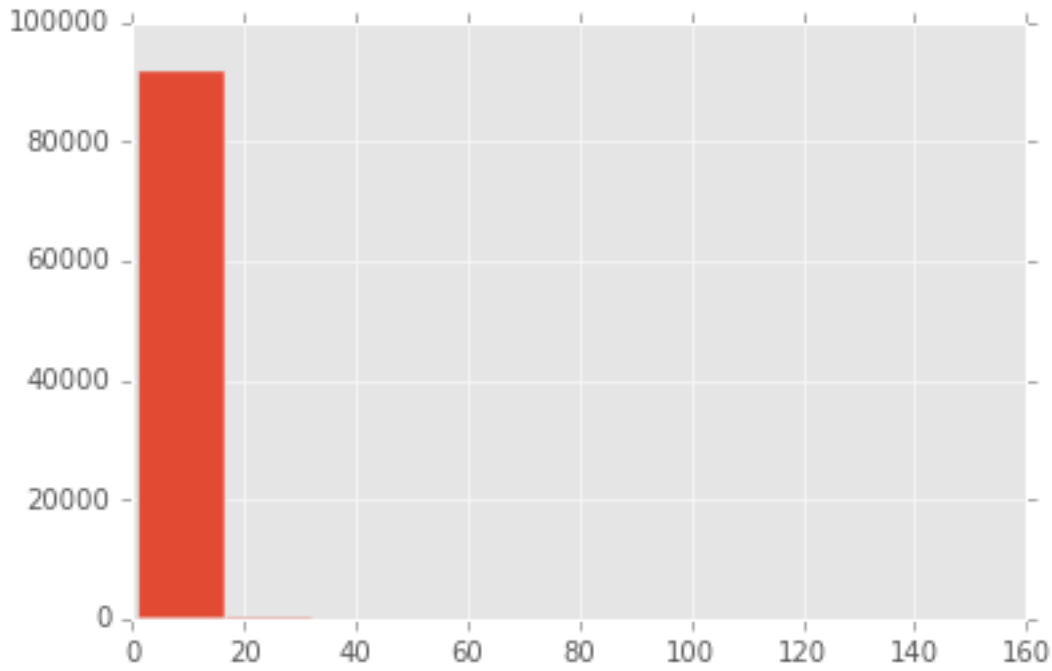
```
len([value for value in df_user_count if value == 1])
```

Out[9]:

```
63971
```

In [10]:

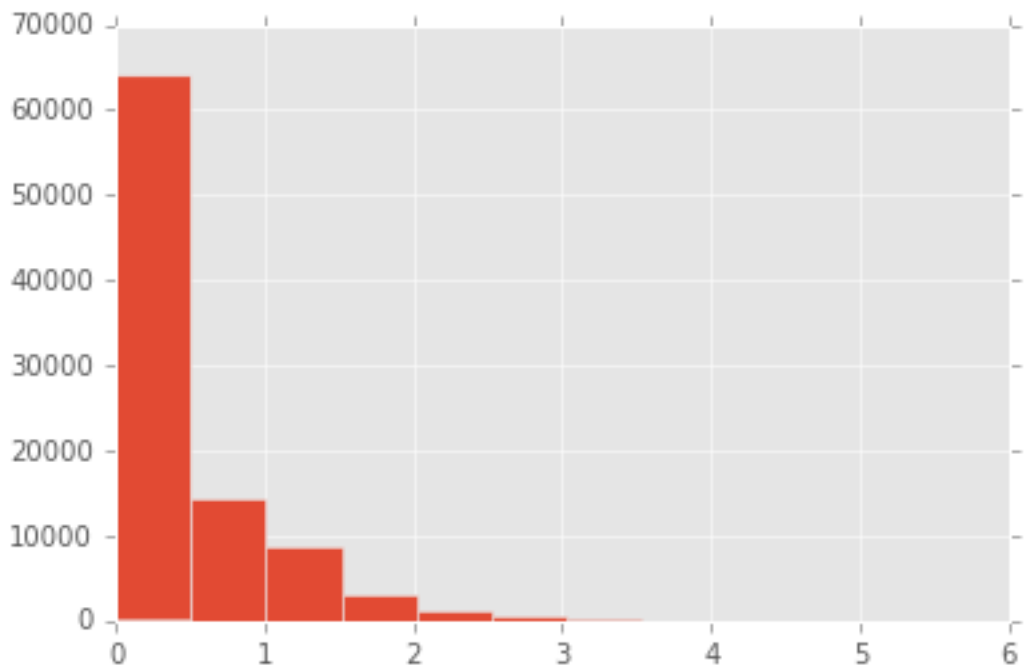
```
df_user_count.hist()
plt.show()
```



We can see the data is highly skewed - most people give only a few reviews. So we take log transform to data.

In [11]:

```
df_user_count.apply(np.log).hist()
plt.show()
```



There are many users that haven't given many reviews, exclude these users from the item-item similarity recommender

Q: How do we recommend to these users anyways?

In [12]:

```
# To be implemented
# users comment over 3 times
df_users = df_user_count[df_user_count > 3]
df_users.head()
```

Out[12]:

```
bLbSNkLggFnqwNNzzq-Ijw    156
JaqcCU3nxReTW2cBLHounA    100
YE54kKTuqJJPNYWIKIpOEQ     86
keBv05MsMFBd0Hu98vXThQ     82
P0rGN5mDue55uIiAzkrRlw     82
Name: user_id, dtype: int64
```

In [13]:

```
df_rec.set_index('user_id').head()
```

Out[13]:

	business_id	stars
user_id		
KC8H7qTZVPlEnanw9fG43g	--9e1ONYQuAa-CB_Rrw7Tw	5
3RTesl_MAwct13LWm4rhLw	--9e1ONYQuAa-CB_Rrw7Tw	4
EAOt1UQhJD0GG3l_jv7rWA	--9e1ONYQuAa-CB_Rrw7Tw	5
OtKA03ALQQ1CBhtaJod_Jw	--9e1ONYQuAa-CB_Rrw7Tw	2
Ymtd4cQypep_QZJ-qJsHuA	--9e1ONYQuAa-CB_Rrw7Tw	5

In [14]:

```
df_rec.set_index('user_id').loc[df_users.index].head()
```

Out[14]:

	business_id	stars
user_id		
bLbSNkLggFnqwNNzzq-Ijw	-WLrZPzjKfrftLWaCi1QZQ	4
bLbSNkLggFnqwNNzzq-Ijw	0G83H_zoum-Q4bWU2olTqQ	4
bLbSNkLggFnqwNNzzq-Ijw	0i9S0BejjRv0ZDwdO9XymA	3
bLbSNkLggFnqwNNzzq-Ijw	0wW9PasC8pw8SY7rlY3ZKw	4
bLbSNkLggFnqwNNzzq-Ijw	1CaM8elvl41l4f3V-V-cAw	4

In [15]:

```
df_rec_users = df_rec.set_index('user_id').loc[df_users.index].reset_index()
```

In [16]:

```
df_rec_users.head()
```

Out[16]:

	user_id	business_id	stars
0	bLbSNkLggFnqwNNzzq-Ijw	-WLrZPzjKfrftLWaCi1QZQ	4
1	bLbSNkLggFnqwNNzzq-Ijw	0G83H_zoum-Q4bWU2olTqQ	4
2	bLbSNkLggFnqwNNzzq-Ijw	0i9S0BejjRv0ZDwdO9XymA	3
3	bLbSNkLggFnqwNNzzq-Ijw	0wW9PasC8pw8SY7rlY3ZKw	4
4	bLbSNkLggFnqwNNzzq-Ijw	1CaM8elvl41l4f3V-V-cAw	4

Create utility matrix from records

In [17]:

```
# To be implemented
df_utility = pd.pivot_table(data=df_rec_users,
                            values='stars', # fill with stars
                            index='user_id', # rows
                            columns='business_id', # columns
                            fill_value=0) # fill missings
```

In [18]:

```
df_utility.head()
```

Out[18]:

business_id	--9e1ONYQuAa-CB_Rrw7Tw	-1m9o3vGRA8lBPNvNqKLmA	-3zffZUHoY
user_id			
--ZNfWKj1VyVElRx6-g1fg	0	0	0
--i9JYrfaKvCpl60nSyP3Q	0	0	0
-0D44Oa5eolEFmaIM4MbWA	0	0	0
-16aKjco1c0RJ7c4U-q_Kw	0	0	0
-1jAjdMaT7bi1GjUptJCIw	0	0	0

5 rows × 3643 columns

2. Item-Item similarity recommender

Let's reuse the ItemItemRecommender class derived from previous exercise

Hint: we need to make modification to accommodate the dense numpy array

In [19]:

```
# To be implemented
import graphlab
sf_rec = graphlab.SFrame(df_rec)

sf_rec_users = graphlab.SFrame(df_rec_users)
```

```
[INFO] graphlab.cython.cy_server: GraphLab Create v2.1 started. Loggin
g: /tmp/graphlab_server_1522448020.log

This non-commercial license of GraphLab Create for academic use is ass
igned to mnawang@ucdavis.edu and will expire on April 11, 2018.
```

In [20]:

```
sf_rec_users
```

Out[20]:

user_id	business_id	stars
bLbSNkLggFnqwNNzzq-Ijw	-WLrZPzjKfrftLWaCi1QZQ	4
bLbSNkLggFnqwNNzzq-Ijw	0G83H_zoum-Q4bWU2oITqQ	4
bLbSNkLggFnqwNNzzq-Ijw	0i9S0BejjRv0ZDwdO9XymA	3
bLbSNkLggFnqwNNzzq-Ijw	0wW9PasC8pw8SY7rlY3ZKw	4
bLbSNkLggFnqwNNzzq-Ijw	1CaM8elvl41l4f3V-V-cAw	4
bLbSNkLggFnqwNNzzq-Ijw	2QznyHGF0PuiYkrl5RU66A	4
bLbSNkLggFnqwNNzzq-Ijw	2WiMyg8-DoXQ8_R5qFc2iQ	3
bLbSNkLggFnqwNNzzq-Ijw	2vr2yGIzSehe_ITFamNpyw	3
bLbSNkLggFnqwNNzzq-Ijw	30TxDt2V8G7FWXsKaEKA9A	4
bLbSNkLggFnqwNNzzq-Ijw	3281FKql3HVOzSVxymjJRg	4

[59867 rows x 3 columns]

Note: Only the head of the SFrame is printed.

You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.

In [21]:

```
# item-item recommender
item_item_rec = graphlab.recommender.item_similarity_recommender.create(sf_rec_users
                                                                        user_id = 'us
                                                                        item_id = 'bu
                                                                        target = 'sta
```

```
Recsys training: model = item_similarity

Preparing data set.

    Data has 59867 observations with 8134 users and 3643 items.

    Data prepared in: 0.078378s

Training model from provided data.

Gathering per-item and per-user statistics.

+--------------------------------+------------+

| Elapsed Time (Item Statistics) | % Complete |

+--------------------------------+------------+
```

```
| 1.772ms                          | 12.25      |

| 3.377ms                          | 100        |

+----------------------------------+------------+

Setting up lookup tables.

Processing data in one pass using dense lookup tables.

+-------------------------------------+-----------------+-----------
-----+

| Elapsed Time (Constructing Lookups) | Total % Complete | Items Proce
ssed |

+-------------------------------------+-----------------+-----------
-----+

| 60.404ms                            | 0                | 0
|

| 135.185ms                           | 100              | 3643
|

+-------------------------------------+-----------------+-----------
-----+

Finalizing lookup tables.

Generating candidate set for working with new users.

Finished training in 1.15455s
```

In [22]:

```
item_item_rec_result = item_item_rec.recommend(k=4, verbose=False)
```

In [23]:

```
item_item_rec_result.head()
```

Out[23]:

user_id	business_id	score	ra
bLbSNkLggFnqwNNzzq-Ijw	Ef5P6C2yHAv08FPif5Rdtg	0.00521167501425	
bLbSNkLggFnqwNNzzq-Ijw	33Tr0eRki1Yamzleu4GMdw	0.00450817820353	
bLbSNkLggFnqwNNzzq-Ijw	_eDVVS8wwXcdllaW6nfLyg	0.004405325804	
bLbSNkLggFnqwNNzzq-Ijw	CoyeXg8FBsS_d20QzNIy-A	0.00440176671896	
JaqcCU3nxReTW2cBLHounA	e3PBfbtXUjMziX6XGsOMoA	0.00844675123692	
JaqcCU3nxReTW2cBLHounA	_eDVVS8wwXcdllaW6nfLyg	0.00782769858837	
JaqcCU3nxReTW2cBLHounA	7wHLFohwCw8l6WS-feLjeg	0.00772170186043	
JaqcCU3nxReTW2cBLHounA	gOOfBSBZlffCkQ7dr7cpdw	0.00755578696728	
YE54kKTuqJJPNYWIKlpOEQ	iFCz-xI7CV98fcaB4Chh3g	0.00912879164829	
YE54kKTuqJJPNYWIKlpOEQ	jaSowNITPRRCYpPb3_pjdA	0.00742893232856	

[10 rows x 4 columns]

3. Matrix Factorization recommender

Take a look at Graphlab Create examples

In [24]:

```
matrix_rec = graphlab.recommender.factorization_recommender.create(sf_rec_users,
                                                                   user_id='user_id',
                                                                   item_id='business_id',
                                                                   target='stars',
                                                                   solver='als',
                                                                   side_data_factorization=
```

```
Recsys training: model = factorization_recommender

Preparing data set.

    Data has 59867 observations with 8134 users and 3643 items.

    Data prepared in: 0.089258s

Training factorization_recommender for recommendations.

+-------------------------------+-------------------------------------
------------+----------+

| Parameter                     | Description
| Value    |

+-------------------------------+-------------------------------------
------------+----------+

| num_factors                   | Factor Dimension
```

In [25]:

```
matrix_rec_result = matrix_rec.recommend(k=4, verbose=False)
```

In [26]:

```
matrix_rec_result.head()
```

Out[26]:

user_id	business_id	score	ra
bLbSNkLggFnqwNNzzq-Ijw	RUsoxvu4HH0fcGW14hbPhg	15.9931679171	1
bLbSNkLggFnqwNNzzq-Ijw	HzGRl7ERUjPpBA_rOT3E-w	11.8415551585	2
bLbSNkLggFnqwNNzzq-Ijw	N-L7ISdndWsIWdK0fn29wg	11.0508175295	3
bLbSNkLggFnqwNNzzq-Ijw	TrN8HBHBL4-Tu7cXMDoopQ	10.9185638827	4
JaqcCU3nxReTW2cBLHounA	VOAO6ip7GwK4McQZbRa60g	18.553114931	1
JaqcCU3nxReTW2cBLHounA	Rxz2YpeLr9Ek54Lt5z6h0A	13.389845888	2
JaqcCU3nxReTW2cBLHounA	gsjxrwdHqKdTeNx3GeDKNQ	13.335700075	3
JaqcCU3nxReTW2cBLHounA	bG8OeTrW5T7pRaDgKKUNAQ	12.8748321932	4
YE54kKTuqJJPNYWIKIpOEQ	VOAO6ip7GwK4McQZbRa60g	29.6831932467	1
YE54kKTuqJJPNYWIKIpOEQ	6bgjcFOy4WHMyw62_1V9Pw	21.571275751	2

[10 rows x 4 columns]

In [27]:

```
matrix_rec['coefficients']
```

Out[27]:

```
{'business_id': Columns:
        business_id     str
        linear_terms    float
        factors array

 Rows: 3643

 Data:
 +------------------------+-------------+---------------------------
---+
 |      business_id       | linear_terms |            factors
|
 +------------------------+-------------+---------------------------
---+
 | -WLrZPzjKfrftLWaCi1QZQ |     0.0      | [-0.0548494234681, 0.10962.
.. |
 | 0G83H_zoum-Q4bWU2olTqQ |     0.0      | [-0.0234339553863, -0.0872.
.. |
 | 0i9S0BejjRv0ZDwdO9XymA |     0.0      | [-0.193892806768, -0.25042.
.. |
 | 0wW9PasC8pw8SY7rlY3ZKw |     0.0      | [-0.0471955537796, 0.07418.
```

```
.. |
 | 1CaM8eIvl41l4f3V-V-cAw |     0.0      | [0.10519913584, 0.14528679.
.. |
 | 2QznyHGF0PuiYkrl5RU66A |     0.0      | [-0.00890613719821, -0.063.
.. |
 | 2WiMyg8-DoXQ8_R5qFc2iQ |     0.0      | [0.062996044755, 0.1057540.
.. |
 | 2vr2yGIzSehe_ITFamNpyw |     0.0      | [0.336221724749, -0.225041.
.. |
 | 30TxDt2V8G7FWXsKaEKA9A |     0.0      | [-0.0569087639451, -0.0328.
.. |
 | 3281FKql3HVOzSVxymjJRg |     0.0      | [0.325046658516, -0.759718.
.. |
 +------------------------+--------------+----------------------------
---+
 [3643 rows x 3 columns]
 Note: Only the head of the SFrame is printed.
 You can use print_rows(num_rows=m, num_columns=n) to print more rows
and columns.,
 'intercept': 3.886598626956423,
 'user_id': Columns:
        user_id str
        linear_terms    float
        factors array

 Rows: 8134

 Data:
 +------------------------+--------------+----------------------------
---+
 |        user_id         | linear_terms |            factors
|
 +------------------------+--------------+----------------------------
---+
 | bLbSNkLggFnqwNNzzq-Ijw |     0.0      | [-2.13566470146, 0.0771686.
.. |
 | JaqcCU3nxReTW2cBLHounA |     0.0      | [0.179353043437, 0.8527637.
.. |
 | YE54kKTuqJJPNYWIKIpOEQ |     0.0      | [1.73792254925, 1.68259167.
.. |
 | keBv05MsMFBd0Hu98vXThQ |     0.0      | [-1.63466000557, -6.036939.
.. |
 | P0rGN5mDue55uIiAzkrRlw |     0.0      | [-1.86762309074, -1.932832.
.. |
 | U4INQZOPSUaj8hMjLlZ3KA |     0.0      | [-3.75838422775, 1.5965961.
.. |
 | bvzwsK8u5i0Kvvyfy7aT1Q |     0.0      | [4.07703018188, -4.5877771.
.. |
 | OXSJCjKtvZPf-YPDCXcWZg |     0.0      | [1.64814603329, -4.5759048.
.. |
 | 8dxkcmAXY4ttrVFD1GhbdQ |     0.0      | [0.904299557209, -2.567059.
.. |
 | sCelgwFoaNLMC_A7Y8usCw |     0.0      | [-10.6837100983, 3.8347885.
```

```
.. |
 +----------------------+-------------+--------------------------
---+
 [8134 rows x 3 columns]
 Note: Only the head of the SFrame is printed.
 You can use print_rows(num_rows=m, num_columns=n) to print more rows
and columns.}

In [ ]:
```

4. Other recommenders (optional)

What are other ways you can build a better recommender?

- Other features (have you noticed there are other features in the Yelp dataset, e.g. tips, etc.?)
- Popularity-based
- Content-based
- Hybrid

- **Content-based**

In [28]:

```
df.head()
```

Out[28]:

	business_id	name	categories	avg_stars	cool	date	funny	review_id
0	--9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	[Cajun/Creole, Steakhouses, Restaurants]	4.0	0	2017-02-14	0	Xp3ppynEvVu
1	--9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	[Cajun/Creole, Steakhouses, Restaurants]	4.0	1	2017-05-28	0	LEzphAnz0vK
2	--9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	[Cajun/Creole, Steakhouses, Restaurants]	4.0	0	2017-08-25	0	4e-cxYVdIlu2Z
3	--9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	[Cajun/Creole, Steakhouses, Restaurants]	4.0	1	2017-02-12	1	heZd0W3HuP
4	--9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	[Cajun/Creole, Steakhouses, Restaurants]	4.0	0	2017-12-10	0	exzXjy7Y2ICX

In [30]:

```
# group the df by busniess_id
df_business = df.groupby(['business_id']).mean()
df_business.head()
```

Out[30]:

	avg_stars	cool	funny	stars	useful
business_id					
--9e1ONYQuAa-CB_Rrw7Tw	4.0	0.613139	0.401460	4.124088	0.788321
-1m9o3vGRA8IBPNvNqKLmA	4.5	1.000000	0.631579	4.736842	1.315789
-3zffZUHoY8bQjGfPSoBKQ	4.0	0.395833	0.312500	4.208333	0.937500
-8R_-EkGpUhBk55K9Dd4mg	3.5	1.111111	0.925926	3.962963	1.259259
-9YyInW1wapzdNZrhQJ9dg	2.5	0.242424	0.121212	2.939394	0.848485

In [31]:

```
# categorize different groups by its category
categories_business = df.groupby(['business_id']).categories.apply(np.unique)

categories_business.head()
```

Out[31]:

```
business_id
--9e1ONYQuAa-CB_Rrw7Tw           [[Cajun/Creole, Steakhouses, Restaura
nts]]
-1m9o3vGRA8IBPNvNqKLmA    [[African, Restaurants, Nightlife, Bars, Ste
ak...
-3zffZUHoY8bQjGfPSoBKQ    [[Seafood, Bars, Nightlife, American (New),
Re...
-8R_-EkGpUhBk55K9Dd4mg                                  [[Thai, Restaura
nts]]
-9YyInW1wapzdNZrhQJ9dg                               [[Buffets, Restaura
nts]]
Name: categories, dtype: object
```

In [32]:

```
# convert data to string and remove `[]`
categories_business = categories_business.str.join('').apply(lambda x: x[1:-1])
categories_business.head()
```

Out[32]:

```
business_id
--9e1ONYQuAa-CB_Rrw7Tw                Cajun/Creole, Steakhouses, Restau
rants
-1m9o3vGRA8IBPNvNqKLmA    African, Restaurants, Nightlife, Bars, Steak
ho...
-3zffZUHoY8bQjGfPSoBKQ    Seafood, Bars, Nightlife, American (New), Re
st...
-8R_-EkGpUhBk55K9Dd4mg                                   Thai, Restau
rants
-9YyInW1wapzdNZrhQJ9dg                                Buffets, Restau
rants
Name: categories, dtype: object
```

In [34]:

```
# create a table with business_id and category.
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
categories_mat = vectorizer.fit_transform(categories_business).toarray()
categories = vectorizer.get_feature_names()
df_categories = pd.DataFrame(categories_mat,
                             columns=categories,
                             index=categories_business.index)
df_categories.head()
```

Out[34]:

	acai	active	activities	acupuncture	adoption	adult	afgha
business_id							
--9e1ONYQuAa-CB_Rrw7Tw	0	0	0	0	0	0	0
-1m9o3vGRA8IBPNvNqKLmA	0	0	0	0	0	0	0
-3zffZUHoY8bQjGfPSoBKQ	0	0	0	0	0	0	0
-8R_-EkGpUhBk55K9Dd4mg	0	0	0	0	0	0	0
-9YyInW1wapzdNZrhQJ9dg	0	0	0	0	0	0	0

5 rows × 451 columns

In [35]:

```
df_categories.shape
```

Out[35]:

```
(4051, 451)
```

The above matrix is sparse, so we want to reduce the dimentionality first.

In [36]:

```
# use svd to reduce dimension
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=200, random_state=42)

svd.fit(categories_mat)
categories_svd = svd.transform(categories_mat)
df_categories_svd = pd.DataFrame(categories_svd,
                                 index=categories_business.index)

print(svd.explained_variance_ratio_.sum())
# As to the result, 98% of the total variance can be explained
df_categories_svd.head()
```

```
0.983614693345
```

Out[36]:

	0	1	2	3	4	5
business_id						
--9e1ONYQuAa-CB_Rrw7Tw	0.727720	0.025981	-0.436491	-0.447002	0.021092	-0.00
-1m9o3vGRA8lBPNvNqKLmA	1.857303	1.992566	0.566649	-0.009360	-0.116075	-0.18
-3zffZUHoY8bQjGfPSoBKQ	1.509998	1.338844	-0.020165	0.153496	-0.042445	-0.21
-8R_-EkGpUhBk55K9Dd4mg	0.701672	-0.013322	-0.428213	-0.468336	0.009372	0.040
-9YyInW1wapzdNZrhQJ9dg	0.699882	-0.003055	-0.439044	-0.458892	0.008921	0.035

5 rows × 200 columns

In [37]:

```
df_business_join = df_average.join(df_categories_svd)  # use join() function, '+' do
df_business_join.head()
```

Out[37]:

	avg_stars	cool	funny	stars	useful	0
business_id						
--9e1ONYQuAa-CB_Rrw7Tw	4.0	0.613139	0.401460	4.124088	0.788321	0.72772
-1m9o3vGRA8lBPNvNqKLmA	4.5	1.000000	0.631579	4.736842	1.315789	1.85730
-3zffZUHoY8bQjGfPSoBKQ	4.0	0.395833	0.312500	4.208333	0.937500	1.50999
-8R_-EkGpUhBk55K9Dd4mg	3.5	1.111111	0.925926	3.962963	1.259259	0.70167
-9YyInW1wapzdNZrhQJ9dg	2.5	0.242424	0.121212	2.939394	0.848485	0.69988

5 rows × 205 columns

In [39]:

```
# prepare data
data_sf = graphlab.SFrame(df_business_join.reset_index())
data_sf.head()
```

Out[39]:

business_id	avg_stars	cool	funny
--9e1ONYQuAa-CB_Rrw7Tw	4.0	0.613138686131	0.401459854015
-1m9o3vGRA8lBPNvNqKLmA	4.5	1.0	0.631578947368
-3zffZUHoY8bQjGfPSoBKQ	4.0	0.395833333333	0.3125
-8R_-EkGpUhBk55K9Dd4mg	3.5	1.11111111111	0.925925925926
-9YyInW1wapzdNZrhQJ9dg	2.5	0.242424242424	0.121212121212
-AD5PiuJHgdUcAK-Vxao2A	3.5	0.277777777778	0.0555555555556
-ADtl9bLp8wNqYX1k3KuxA	4.0	0.333333333333	0.121212121212
-AGdGGCeTS-njB_8GkUmjQ	4.0	2.0	2.0
-Bf8BQ3yMk8U2f45r2DRKw	4.0	0.666666666667	0.606060606061
-Bv-HHUs8aHzDrdWcZHn8w	3.0	0.321428571429	0.214285714286

0	1	2	3
0.727719914811	0.0259805873653	-0.436490705052	-0.44700238296
1.85730296782	1.99256579828	0.56664888495	-0.0093595249493

1.50999753279	1.33884361173	-0.0201651804608	0.153495657788	
0.701672384145	-0.013322089252	-0.428212961183	-0.468335884732	
0.69988166844	-0.00305522030528	-0.439044430856	-0.458892009091	
0.772191017214	-0.0422744174686	-0.47492677879	-0.378180516922	
1.11941617783	0.642164300222	0.214325270758	-0.993733288651	
0.697406159516	-0.00762761122546	-0.42478731544	-0.465093575987	
1.40423921878	-0.84630656044	-0.00303974436216	-0.0512841397339	
0.772191017214	-0.0422744174686	-0.47492677879	-0.378180516922	

6	7	8	9	
-0.0346233622853	0.124480951181	-0.03506121911	0.0346592184836	
-0.14353993614	0.165172922994	0.12982210791	0.457680382149	
-0.140856846729	0.264023983798	0.144032656104	0.449828291191	-
-0.0299197485714	0.120198306604	-0.0572896441148	0.032088100312	
-0.0240951364842	0.0894132089777	-0.0385985892952	0.017694493249	
0.0319316429159	-0.19652197056	-0.30591040197	-0.131954470702	
-0.31136981379	0.81967444367	-1.01068341689	-0.126703441761	
-0.0304596798828	0.111387317001	-0.0621140736346	0.0258560959824	
-0.0275330970336	-0.46415168928	-0.485965253521	-0.233921039122	
0.0319316429159	-0.19652197056	-0.30591040197	-0.131954470702	

In [40]:

```
content_rec = graphlab.recommender.item_content_recommender.create(data_sf, "busines
```

```
WARNING: The ItemContentRecommender model is still in beta.
WARNING: This feature transformer is still in beta, and some interpret
ation rules may change in the future.
('Applying transform:\n', Class             : AutoVectorizer

Model Fields
------------
Features          : ['avg_stars', 'cool', 'funny', 'stars', 'useful',
'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '1
3', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24',
'25', '26', '27', '28', '29', '30', '31', '32', '33', '34', '35', '36'
, '37', '38', '39', '40', '41', '42', '43', '44', '45', '46', '47', '4
8', '49', '50', '51', '52', '53', '54', '55', '56', '57', '58', '59',
'60', '61', '62', '63', '64', '65', '66', '67', '68', '69', '70', '71'
, '72', '73', '74', '75', '76', '77', '78', '79', '80', '81', '82', '8
3', '84', '85', '86', '87', '88', '89', '90', '91', '92', '93', '94',
'95', '96', '97', '98', '99', '100', '101', '102', '103', '104', '105'
, '106', '107', '108', '109', '110', '111', '112', '113', '114', '115'
, '116', '117', '118', '119', '120', '121', '122', '123', '124', '125'
```

In [41]:

```
# make recommendation for a single item
sample_item = [df_rec.iloc[7].business_id]   # select the third business_id as the s
content_rec.recommend_from_interactions(sample_item) # recommendation from single it
```

Out[41]:

business_id	score	rank
uWECX6-Uq9n8v5ipk9R29A	0.942015171051	1
TT658qQinO6MBHP9q7rJ8w	0.941025078297	2
p3YqOYELqXtLyHz9T49p_w	0.910501003265	3
zcScEL0WEdFkROcnz5379g	0.909703612328	4
L2W0QLXIIR5MEmhQwZk-iA	0.908503472805	5
5TY6bUT3bbl9aHItiIXXqw	0.905634880066	6
UNI1agsPX2k3eJSJVB91nw	0.885742247105	7
VPO8pBUwYz1u6GoG0d2U-Q	0.877820253372	8
6uV3KMOo8YQofuNUs4D9pA	0.864193558693	9
L_ZLtfHvfzfoNVQ0-okTXg	0.863706588745	10

[10 rows x 3 columns]

In [42]:

```
# similar items per item (recommend top 10 for each)
similar_items_df = content_rec.get_similar_items().to_dataframe()
similar_items_df.head()
```

Out[42]:

	business_id	similar	score	rank
0	--9e1ONYQuAa-CB_Rrw7Tw	uWECX6-Uq9n8v5ipk9R29A	0.942015	1
1	--9e1ONYQuAa-CB_Rrw7Tw	TT658qQinO6MBHP9q7rJ8w	0.941025	2
2	--9e1ONYQuAa-CB_Rrw7Tw	p3YqOYELqXtLyHz9T49p_w	0.910501	3
3	--9e1ONYQuAa-CB_Rrw7Tw	zcScEL0WEdFkROcnz5379g	0.909704	4
4	--9e1ONYQuAa-CB_Rrw7Tw	L2W0QLXIIR5MEmhQwZk-iA	0.908503	5

In [ ]:

In [ ]:

In [44]:

```
# make recommendation for a sample user
df_favored = df_rec[df_rec.stars > 4] # select favored restuarants for a sample use
bid_favored = df_favored[df_favored.user_id == df_favored.user_id.iloc[0]] # select
```

In [57]:

```
# first select favored restaurants' similar items
# second sort those restaurants, then got top 5
similar_items_df[similar_items_df['business_id'].isin(bid_favored.business_id)].sort
```

Out[57]:

```
27740    fZM-vnMe00UbFbbo03uSDQ
27741    4oJeSkScjPxuDC_LeK3KiQ
0        uWECX6-Uq9n8v5ipk9R29A
1        TT658qQinO6MBHP9q7rJ8w
27742    5QNXZcclknB2PkfeN7FJWQ
Name: similar, dtype: object
```