Section 2: Yelp Data Challenge - NLP

Yiting Luo | Data Science Applied Research - 2

May 2018

In [1]:
```
import pandas as pd
```

In [2]:
```
df = pd.read_csv('dataset/last_2_years_restaurant_reviews.csv')
```

In [3]:
```
df.head()
```

```
Out[3]:
```

	business_id	name	categories	avg_stars	cool	date	funny	
0	--9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	['Cajun/Creole', 'Steakhouses', 'Restaurants']	4.0	0	2016-03-31	0	6SgvNWJl
1	--9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	['Cajun/Creole', 'Steakhouses', 'Restaurants']	4.0	0	2015-06-29	0	iwx6s6yQx
2	--9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	['Cajun/Creole', 'Steakhouses', 'Restaurants']	4.0	0	2015-03-16	0	UVUMu_b
3	--9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	['Cajun/Creole', 'Steakhouses', 'Restaurants']	4.0	0	2016-02-10	0	UxFpgng8
4	--9e1ONYQuAa-CB_Rrw7Tw	Delmonico Steakhouse	['Cajun/Creole', 'Steakhouses', 'Restaurants']	4.0	0	2017-02-14	0	Xp3ppynE

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

```
In [4]: # Take the values of the column that contains review text data, save t
        o a variable named "documents"
        documents = df['text'].values  # Make results to Numpy array
```

```
In [5]: # inspect your documents, e.g. check the size, take a peek at elements
        of the numpy array
        documents.dtype, documents.shape

Out[5]: (dtype('O'), (515752,))
```

```
In [6]: documents[3]
```

```
Out[6]: 'Truly Fantastic!  Best Steak ever. Service was awesome and timely.
        They knew right when you needed something. The root beer float for d
        essert was great.'
```

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

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

```
In [7]: # Make a column and take the values, save to a variable named "target"
        df['favorable'] = (df['stars'] > 4)
```

```
In [8]: target = df['favorable'].values
```

```
In [9]: target[:10]
```

```
Out[9]: array([ True, False,  True,  True,  True, False,  True,  True,  True
        ,
                False])
```

Look at the statistic of the target variable

```
In [10]: # To be implemented
         target.mean(), target.std(), documents.shape, target.shape
```

```
Out[10]: (0.46397299477268145, 0.49870036584541505, (515752,), (515752,))
```

Let's create training dataset and test dataset

```
In [11]: from sklearn.cross_validation import train_test_split
```

```
/Users/luoyiting/anaconda/lib/python3.5/site-packages/sklearn/cross_
validation.py:44: DeprecationWarning: This module was deprecated in
version 0.18 in favor of the model_selection module into which all t
he refactored classes and functions are moved. Also note that the in
terface of the new CV iterators are different from that of this modu
le. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

```
In [12]: # Documents is X, target is y
         # Now split the data to training set and test set
```

In [13]:
```
# 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 = 0.8, random_state = 42
)
```

Let's get NLP representation of the documents

In [14]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [15]:
```
# Create TfidfVectorizer, and name it vectorizer
vectorizer = TfidfVectorizer(stop_words = 'english', max_features = 5000)
```

In [16]:
```
# Train the model with training data
vectors_train = vectorizer.fit_transform(documents_train).toarray() # toarray avoid sparse matrix
```

```
/Users/luoyiting/anaconda/lib/python3.5/site-packages/sklearn/featur
e_extraction/text.py:1059: FutureWarning: Conversion of the second a
rgument of issubdtype from `float` to `np.floating` is deprecated. I
n future, it will be treated as `np.float64 == np.dtype(float).type`
.
  if hasattr(X, 'dtype') and np.issubdtype(X.dtype, np.float):
```

In [17]:
```
# Get the vocab of tfidf
words = vectorizer.get_feature_names()
```

In [18]:
```
vectors_train.shape
```

Out[18]:
```
(103150, 5000)
```

In [19]:
```
# Use the trained model to transform your test data
vectors_test = vectorizer.transform(documents_test).toarray()
```

```
/Users/luoyiting/anaconda/lib/python3.5/site-packages/sklearn/featur
e_extraction/text.py:1059: FutureWarning: Conversion of the second a
rgument of issubdtype from `float` to `np.floating` is deprecated. I
n future, it will be treated as `np.float64 == np.dtype(float).type`
.
  if hasattr(X, 'dtype') and np.issubdtype(X.dtype, np.float):
```

Similar review search engine

In [20]:

```
import numpy as np

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.argsor
t by default sorts values in ascending order

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"]
    '''
    return [labels[i] for i in np.argsort(lst)[:n]]
```

In [21]:

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

In [22]:

```
# Draw an arbitrary review from test (unseen in training) documents
some_random_number = 42
search_query = documents_test[some_random_number]
search_queries = [search_query]
print (search_query)
print (search_queries)
```

```
Great food and great price, the only thing is the waiting time. Duri
ng lunch and dinner hours, they are extremely busy. Call in ahead to
get your order and pick up for faster service :)
['Great food and great price, the only thing is the waiting time. Du
ring lunch and dinner hours, they are extremely busy. Call in ahead
to get your order and pick up for faster service :)']
```

In [23]:

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

```
/Users/luoyiting/anaconda/lib/python3.5/site-packages/sklearn/featur
e_extraction/text.py:1059: FutureWarning: Conversion of the second a
rgument of issubdtype from `float` to `np.floating` is deprecated. I
n future, it will be treated as `np.float64 == np.dtype(float).type`
.
  if hasattr(X, 'dtype') and np.issubdtype(X.dtype, np.float):
```

In [24]:

```
# Calculate the similarity score(s) between vector(s) and training vec
tors
similarity_score = cosine_similarity(vector_search_queries, vectors_tr
ain)
```

In [25]:

```
similarity_score.shape
```

Out[25]:

```
(1, 103150)
```

In [26]:

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

In [27]:

```
print ('Our search query:')
print  (search_queries[0])
```

```
Our search query:
Great food and great price, the only thing is the waiting time. Duri
ng lunch and dinner hours, they are extremely busy. Call in ahead to
get your order and pick up for faster service :)
```

In [28]:

```
print ('Most %s similar reviews:' % n)
for review in returned_reviews:
    print  (review )
```

```
Most 5 similar reviews:
Great buffet and a great price. Ive eaten here 4 times. It was great
each time. It can get busy though so i would suggest call ahead to f
ind out the wait time
Well I'm updating my last review. As previously stated gave them 2 s
tars because the food is somewhat good. I don't like dealing with th
e depressed people inside so I decided moving forward I would place 
the order online and pick up.
When I arrived a few minutes after my pick up time they told me sorr
y your order is not ready, it will be faster if you stand in line. H
ow is it faster if I stand in line when have had my order for 45minu
tes?! They told me it would be an additional 20 minutes to make my o
rder but they could magically make it faster if I stood in line in 1
0 min before ordering the exact same thing they had on the receipt i
n front of them. No sense in coming back. I have tried time and time
again, and every time it gets worse. This will be my last trip to pl
ace I at one point in my life came to weekly. So with this, so long 
cafe rio.
For lunch or dinner, get the BBQ burger. One of the best I ever had.
Great food, service, price and location.
I had a great time today going to the barbecue restaurant. It is the
first time of that I have been there, and the service is excellent m
uch faster than I expected, and the food was delicious!
Great place, Great Food, EXCELLENT CUSTOMER SERVICE!! We came in wit
h 8 ppl they fixed us up a table and served us faster than Mc Donald
s serves Fries! Great quality food... Everything is Great Great Grea
t!!! If you haven't ate here its a MUST!!
```

The search enginem makes sense.

Classifying positive/negative review

Naive-Bayes Classifier

In [29]:

```
# Build a Naive-Bayes Classifier

from sklearn.naive_bayes import MultinomialNB

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

Out[29]:

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

In [30]:

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

Out[30]:

```
0.8092195831313621
```

In [31]:

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

Out[31]:

```
0.8031250454433085
```

Logistic Regression Classifier

In [32]:

```
# Build a Logistic Regression Classifier

from sklearn.linear_model import LogisticRegression

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

Out[32]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_interce
pt=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jo
bs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0
.0001,
          verbose=0, warm_start=False)
```

In [33]:

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

Out[33]:

```
0.8433446437227339
```

In [34]:

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

Out[34]:

```
0.8256940102083848
```

Key features(words) that make the positive prediction?

```
In [35]: # find it out by ranking
         n = 20
         get_top_values(model_lrc.coef_[0], n, words)
```

```
Out[35]: ['amazing',
          'best',
          'incredible',
          'delicious',
          'awesome',
          'thank',
          'perfection',
          'perfect',
          'phenomenal',
          'fantastic',
          'favorite',
          'great',
          'perfectly',
          'excellent',
          'highly',
          'die',
          'heaven',
          'gem',
          'love',
          'bomb']
```

Q: Key features(words) that make the negative prediction?

```
In [36]: # find it out by ranking
         n = 20
         get_bottom_values(model_lrc.coef_[0], n, words)
```

```
Out[36]: ['worst',
          'ok',
          'horrible',
          'rude',
          'bland',
          'terrible',
          'okay',
          'mediocre',
          'slow',
          'disappointing',
          'dry',
          'meh',
          'lacking',
          'unfortunately',
          'overpriced',
          'wasn',
          'awful',
          'average',
          'poor',
          'decent']
```

Random Forest Classifier

```
In [37]: # Build a Random Forest Classifier

         from sklearn.ensemble import RandomForestClassifier

         model_rfc = RandomForestClassifier(max_depth = None,
                                           n_estimators = 5,
                                           min_samples_leaf = 10)
         model_rfc.fit(vectors_train, target_train)
```

```
Out[37]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion=
         'gini',
                     max_depth=None, max_features='auto', max_leaf_nodes=None
         ,
                     min_impurity_split=1e-07, min_samples_leaf=10,
                     min_samples_split=2, min_weight_fraction_leaf=0.0,
                     n_estimators=5, n_jobs=1, oob_score=False, random_state=
         None,
                     verbose=0, warm_start=False)
```

```
In [38]: # Get score for training set
         model_rfc.score(vectors_train, target_train)
```

```
Out[38]: 0.8138826951042172
```

```
In [39]: # Get score for test set
         model_rfc.score(vectors_test, target_test)

Out[39]: 0.7711305325713399
```

What features (words) are important by inspecting the RFC model?

```
In [41]: n = 20
         get_top_values(model_rfc.feature_importances_, n, words)

Out[41]: ['amazing',
          'great',
          'love',
          'best',
          'awesome',
          'delicious',
          'bad',
          'like',
          'friendly',
          'didn',
          'horrible',
          'favorite',
          'highly',
          'excellent',
          'perfect',
          'definitely',
          'wasn',
          'average',
          'thank',
          'worst']
```

Use cross validation to evaluate classifiers

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

```
In [42]: from sklearn.model_selection import cross_val_score

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

```
In [44]: cv_scores.mean(), cv_scores.std()

Out[44]: (0.826136643215901, 0.003967095258931436)
```

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.Gric](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.Gric)

In [45]:

```
# Tune Logistic Regression Regularization parameter C and different penalty
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report

param_grid = [{'penalty':['l1'], 'C':[0.1, 100]},
              {'penalty':['l2'], 'C':[0.1, 100]}]

scores = ['accuracy']

for score in scores:
    print("# Tuning hyper-parameters for %s" % score + "\n\n")
    clf = GridSearchCV(LogisticRegression(),
                       param_grid,
                       cv=5,
                       scoring=score)
    clf.fit(vectors_train[:500,:], target_train[:500])
    print("Best parameters set found on development set:\n\n")
    print(clf.best_params_)
    print("\nGrid scores on development set:\n\n")
    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("%0.3f (+/-%0.03f) for %r"
              % (mean, std * 2, params))

    print("\nDetailed classification report:\n")
    print("The model is trained on the full development set.")
    print("The scores are computed on the full evaluation set.")
    print("\n")
    y_true, y_pred = target_test, clf.predict(vectors_test)
    print(classification_report(y_true, y_pred))
    print("\n")
```

```
# Tuning hyper-parameters for accuracy


Best parameters set found on development set:


{'penalty': 'l2', 'C': 100}

Grid scores on development set:


0.554 (+/-0.004) for {'penalty': 'l1', 'C': 0.1}
0.718 (+/-0.083) for {'penalty': 'l1', 'C': 100}
0.552 (+/-0.012) for {'penalty': 'l2', 'C': 0.1}
0.752 (+/-0.073) for {'penalty': 'l2', 'C': 100}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.


             precision    recall  f1-score   support

      False       0.75      0.78      0.76    221181
       True       0.73      0.69      0.71    191421

avg / total       0.74      0.74      0.74    412602
```

In [ ]: