In [140]:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn import ensemble
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import numpy as np
from sklearn import datasets
import pandas as pd


from math import sin, cos, sqrt, atan2, radians

#sentiment packages
from textblob import TextBlob
```

In [142]:

```
# load business_df dataframe with ALL additional columns
# run instead of cells below
business_df = pd.read_json('business_df.json', lines=False)

# for saving business_df to json file
# business_df.to_json(r'business_df.json')
```

In [144]:

```
#load business data
business_df = pd.read_json('business.json', lines=True)

#load tip data
tip_df = pd.read_json('tip.json', lines=True)
```

Add Chain Column

In [152]:

```
# Create a dictionary where key=business_name, value=count of that business
business_names = {}
for index, tip in business_df.iterrows():
    business_name = tip['name']
    if business_name in business_names:
        business_names[business_name] += 1
    else:
        business_names[business_name] = 1

# Add a Boolean column 'chain' to business_df
# True if there are more than one business by the same name
business_df['chain'] = False
for index, business in business_df.iterrows():
    business_name = business['name']
    if business_names[business_name] > 1:
        business_df.at[index, 'chain'] = True
```

Add Tip_Count Column

In [153]:

```
# create a dictionary of tips matched to business IDs
bzn_tips = {}
for index, tip in tip_df.iterrows():
    business_id = tip['business_id']
    if business_id in bzn_tips:
        bzn_tips[business_id] += 1
    else:
        bzn_tips[business_id] = 1

# Add a 'tip_count' column to businesses_df dataframe
business_df['tip_count'] = 0

for index, business in business_df.iterrows():
    business_id = business['business_id']
    if business_id in bzn_tips:
        business_df.at[index, 'tip_count'] = bzn_tips[business_id]
```

Sentiment Analysis of Tips.json

Perform Sentiment Analysis with TextBlob

In [ ]:

```
import json

data=[]
for l in open("tip.json").readlines():
    data.append(json.loads(l))
tips_sentiment_df = pd.DataFrame.from_records(data[0:1000000])[['business_id','text','date']]

tips_sentiment_df['sentiment'] = df[['text']].applymap(lambda x: TextBlob(x).sentiment.polarity)
tips_sentiment_df.to_json(r'tips_with_sentiment.json',orient='records')
```

Add mean_tip_sentiment column

In [155]:

```
# aggregate mean sentiments by 'business_id'
mean_tips_sentiment = tips_sentiment_df.groupby('business_id').mean()[['sentiment']]

# Join/Append 'sentiment' column to business_df
business_df = business_df.join(mean_tips_sentiment, on='business_id')

# Fill NaNs with mean_sentiment
mean_sentiment = business_df['sentiment'].mean()
business_df = business_df.fillna(value=mean_sentiment)
business_df = business_df.rename(columns={"sentiment": "mean_tip_sentiment"})
```

Add Neighbor Columns to Illinois data

- Begin using illinois_business df instead of business_df

In [157]:

```
def get_distance(lat1, lon1, lat2, lon2):
    # approximate radius of earth in km
    R = 6373.0
    lat1 = radians(lat1)
    lon1 = radians(lon1)
    lat2 = radians(lat2)
    lon2 = radians(lon2)

    dlon = lon2 - lon1
    dlat = lat2 - lat1

    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    distance = R * c
    return distance


illinois_business = business_df[business_df['state'] == 'IL']

bus_loc = [[] for i in range(illinois_business.shape[0])]
count = 0
for index, row in illinois_business.iterrows():
    bus_loc[count].append(row['business_id'])
    bus_loc[count].append(row['latitude'])
    bus_loc[count].append(row['longitude'])
    count += 1

il_neighbors_close = [[] for i in range(len(bus_loc))]
il_neighbors_far = [[] for i in range(len(bus_loc))]

for i, biz1 in enumerate(bus_loc):
    for j, biz2 in enumerate(bus_loc):
        if i == j:
            continue
        distance = get_distance(biz1[1], biz1[2], biz2[1], biz2[2])
        if distance < 0.3:
            il_neighbors_far[i].append([biz2[0], distance])
        if distance < 0.1:
            il_neighbors_close[i].append([biz2[0], distance])


illinois_business['.1_km'] = il_neighbors_close
illinois_business['.3_km'] = il_neighbors_far

# number_neighbors_close = [0 for i in range(len(il_neighbors_close))]
# number_neighbors_far = [0 for i in range(len(il_neighbors_far))]


# for i, bzn in enumerate(il_neighbors_close):
#     number_neighbors_close[i] = len(bzn)

# for i, bzn in enumerate(il_neighbors_far):
#     number_neighbors_far[i] = len(bzn)

illinois_business['.1_count'] = illinois_business['.1_km'].apply(lambda x: len
(x))
```

```
illinois_business['.3_count'] = illinois_business['.3_km'].apply(lambda x: len
(x))
```

```
/home/david/.local/lib/python3.6/site-packages/ipykernel_launcher.py:42: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
/home/david/.local/lib/python3.6/site-packages/ipykernel_launcher.py:43: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
/home/david/.local/lib/python3.6/site-packages/ipykernel_launcher.py:55: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
/home/david/.local/lib/python3.6/site-packages/ipykernel_launcher.py:56: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
```

1) Logistic Regression of Illinois Businesses with neighbors

In [160]:

```
il_LogReg_data = illinois_business[['stars', 'review_count', 'chain', 'tip_count', '.1_count', '.3_count', 'mean_tip_sentiment']]
il_LogReg_targets = illinois_business['is_open']
il_LogReg_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1932 entries, 289 to 192521
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   stars               1932 non-null   float64
 1   review_count        1932 non-null   int64  
 2   chain               1932 non-null   bool   
 3   tip_count           1932 non-null   int64  
 4   .1_count            1932 non-null   int64  
 5   .3_count            1932 non-null   int64  
 6   mean_tip_sentiment  1932 non-null   float64
 7   mean_tip_sentiment  1932 non-null   float64
dtypes: bool(1), float64(3), int64(4)
memory usage: 122.6 KB
```

In [159]:

```
clf = LogisticRegression()
scores = cross_val_score(clf, il_LogReg_data, il_LogReg_targets, cv=5)
print('Logistic Regression Scores', scores)
print('Logistic Regression Mean Score', np.mean(scores))
```

```
/home/david/.local/lib/python3.6/site-packages/sklearn/linear_model/_logisti
c.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
sion
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/home/david/.local/lib/python3.6/site-packages/sklearn/linear_model/_logisti
c.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
sion
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
Logistic Regression Scores [0.79328165 0.79844961 0.80569948 0.79533679 0.792
74611]
Logistic Regression Mean Score 0.797102729913912
```

```
/home/david/.local/lib/python3.6/site-packages/sklearn/linear_model/_logisti
c.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
sion
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Gradient Boosting Classifier of Illinois Businesses with neighbors

In [161]:

```
clf = ensemble.GradientBoostingClassifier()
scores = cross_val_score(clf, il_LogReg_data, il_LogReg_targets, cv=5)
print('Illinois Businesses Boosting Classifier Scores', scores)
print('Illinois Businesses Boosting Classifier Mean Score', np.mean(scores))
```

```
Illinois Businesses Boosting Classifier Scores [0.80620155 0.79328165 0.77720
207 0.78756477 0.79015544]
Illinois Businesses Boosting Classifier Mean Score 0.7908810967854227
```

Gradient Boosting Classifier Parameter Grid Search of Illinois Businesses

In [162]:

```
tuned_parameters = [{'max_depth': [2, 3],
                     'n_estimators': [50, 100],
                     'learning_rate': [0.01, 0.05]}]
clf = ensemble.GradientBoostingClassifier()
clf = GridSearchCV(clf, tuned_parameters)
clf.fit(il_LogReg_data, il_LogReg_targets)

print("Scores for parameter grid search:")
print()
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))
```

```
Scores for parameter grid search:

0.800 (+/-0.002) for {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators':
50}
0.800 (+/-0.002) for {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators':
100}
0.800 (+/-0.002) for {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators':
50}
0.800 (+/-0.002) for {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators':
100}
0.800 (+/-0.002) for {'learning_rate': 0.05, 'max_depth': 2, 'n_estimators':
50}
0.801 (+/-0.005) for {'learning_rate': 0.05, 'max_depth': 2, 'n_estimators':
100}
0.798 (+/-0.008) for {'learning_rate': 0.05, 'max_depth': 3, 'n_estimators':
50}
0.798 (+/-0.012) for {'learning_rate': 0.05, 'max_depth': 3, 'n_estimators':
100}
```

Logistic Regression of National Businesses, No Neighbors

In [166]:

```
national_data = business_df[['stars', 'review_count', 'chain', 'tip_count', 'm
ean_tip_sentiment']]
national_targets = business_df['is_open']
```

In [167]:

```
clf = LogisticRegression()
scores = cross_val_score(clf, national_data, national_targets, cv=5)
print('Logistic Regression Scores, National', scores)
print('Logistic Regression Scores, National Mean Score', np.mean(scores))
```

```
Logistic Regression Scores, National [0.82241317 0.82256892 0.82267276 0.8225
17   0.82266816]
```

Gradient Boosting Classifier of National Businesses, No Neighbors

In [177]:

```
clf = ensemble.GradientBoostingClassifier()
scores = cross_val_score(clf, national_data, national_targets, cv=5)
print('National Business Boosting Classifier Scores', scores)
print('National Business Boosting Classifier Scores Mean Score', np.mean(score
s))
```

```
National Business Boosting Classifier Scores [0.8239188  0.82446394 0.8238409
2 0.82397072 0.82381039]
National Business Boosting Classifier Scores Mean Score 0.8240009543138422
```

Create neighbor data for national set

In [169]:

```
state_dict = {}
for index, row in business_df.iterrows():
    state = row['state']
    if row['state'] in state_dict:
        state_dict[state] += 1
    else:
        state_dict[state] = 1
```

In [172]:

```
print('business count per state', state_dict)
```

```
business count per state {'AZ': 56686, 'ON': 33412, 'NC': 14720, 'AB': 8012,
'NV': 36312, 'OH': 14697, 'PA': 11216, 'QC': 9219, 'WI': 5154, 'IL': 1932, 'N
Y': 22, 'SC': 1162, 'TX': 6, 'UT': 1, 'NM': 1, 'FL': 4, 'CA': 19, 'VA': 2, 'B
AS': 1, 'NE': 2, 'AK': 2, 'XGM': 4, 'WA': 3, 'XWY': 2, 'CON': 1, 'BC': 1, 'G
A': 2, 'VT': 2, 'CT': 3, 'AL': 3, 'DUR': 1, 'TN': 1, 'NJ': 1, 'AR': 1, 'XGL':
1, 'DOW': 1}
```

In [173]:

```
large_states = ["IL", 'PA', 'AZ', 'ON', 'NC', 'AB', 'NV', 'OH', "QC", "WI", "S
C"]
large_state_df = business_df[business_df.state.isin(large_states)]
```

In [174]:

```
completed_distance_df = pd.DataFrame()
```

In [176]:

```
for state in large_states:
#for i in [1]:
    current_state_df = large_state_df[large_state_df.state == state]
    if current_state_df.shape[0] < 15000:
        bus_loc = [[] for i in range(current_state_df.shape[0])]
        count = 0
        for index, row in current_state_df.iterrows():
            bus_loc[count].append(row['business_id'])
            bus_loc[count].append(row['latitude'])
            bus_loc[count].append(row['longitude'])
            count += 1

        #initialize empty neighbor dict
        current_neighbors_close = [[] for i in range(len(bus_loc))]
        current_neighbors_far = [[] for i in range(len(bus_loc))]

        for i, biz1 in enumerate(bus_loc):
            if i % 1000 == 0:
                print(state, i)

            for j, biz2 in enumerate(bus_loc):
                if i == j:
                    continue
                distance = get_distance(biz1[1], biz1[2], biz2[1], biz2[2])
                if distance < 0.3:
                    current_neighbors_far[i].append([biz2[0], distance])
                if distance < 0.1:
                    current_neighbors_close[i].append([biz2[0], distance])

        current_state_df['.1_km'] = current_neighbors_close
        current_state_df['.3_km'] = current_neighbors_far


        completed_distance_df = completed_distance_df.append(current_state_df,
ignore_index = True)
        current_state_df = pd.DataFrame()
```

```
IL 0
IL 1000

/home/david/.local/lib/python3.6/site-packages/ipykernel_launcher.py:30: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
/home/david/.local/lib/python3.6/site-packages/ipykernel_launcher.py:31: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
PA 0
PA 1000
PA 2000
PA 3000
PA 4000
PA 5000
PA 6000
PA 7000
PA 8000
PA 9000
PA 10000
PA 11000
NC 0
NC 1000
NC 2000
NC 3000
NC 4000
NC 5000
NC 6000
NC 7000
NC 8000
NC 9000
NC 10000
NC 11000
NC 12000
NC 13000
NC 14000
AB 0
AB 1000
AB 2000
AB 3000
AB 4000
AB 5000
AB 6000
AB 7000
AB 8000
OH 0
OH 1000
OH 2000
OH 3000
OH 4000
OH 5000
OH 6000
OH 7000
OH 8000
OH 9000
OH 10000
OH 11000
OH 12000
OH 13000
OH 14000
QC 0
QC 1000
QC 2000
QC 3000
QC 4000
QC 5000
QC 6000
QC 7000
QC 8000
QC 9000
WI 0
WI 1000
WI 2000
WI 3000
WI 4000
WI 5000
SC 0
SC 1000
```

In [178]:

```python
completed_distance_df['.1_count'] = completed_distance_df['.1_km'].apply(lambda x: len(x))
completed_distance_df['.3_count'] = completed_distance_df['.3_km'].apply(lambda x: len(x))
```

Logistic Regression of National Businesses, With Neighbors

In [179]:

```python
national_neighbor_data = completed_distance_df[['stars', 'review_count', 'chain', 'tip_count', 'mean_tip_sentiment', '.1_count', '.3_count']]
national_neighbor_targets = completed_distance_df['is_open']
```

In [180]:

```python
clf = LogisticRegression()
scores = cross_val_score(clf, national_neighbor_data, national_neighbor_targets, cv=5)
print('Logistic Regression Scores, National with neighbors', scores)
print('Logistic Regression Scores, National with neighbors Mean Score', np.mean(scores))
```

```
Logistic Regression Scores, National with neighbors [0.83748015 0.83354761 0.83792165 0.83845107 0.83769475]
Logistic Regression Scores, National with neighbors Mean Score 0.837019043597721

/home/david/.local/lib/python3.6/site-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Logistic Regression of National Businesses, With Neighbors

In [181]:

```
clf = ensemble.GradientBoostingClassifier()
scores = cross_val_score(clf, national_neighbor_data, national_neighbor_target
s, cv=5)
print('National Business Boosting Classifier Scores', scores)
print('National Business Boosting Classifier Scores Mean Score', np.mean(score
s))
```

```
National Business Boosting Classifier Scores [0.83846328 0.84005143 0.8398124
3 0.8399637  0.83761912]
National Business Boosting Classifier Scores Mean Score 0.8391819919118255
```

Gradient Boosting Classifier Parameter Grid Search, national data with neighbors

In [138]:

```
tuned_parameters = [{'max_depth': [2, 3],
                     'n_estimators': [50, 100],
                     'learning_rate': [0.01, 0.05]}]
clf = ensemble.GradientBoostingClassifier()
clf = GridSearchCV(clf, tuned_parameters)
clf.fit(national_neighbor_data, national_neighbor_targets)

print("Scores for parameter grid search:")
print()
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))
```

```
Scores for parameter grid search:

0.837 (+/-0.000) for {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators':
50}
0.837 (+/-0.000) for {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators':
100}
0.837 (+/-0.000) for {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators':
50}
0.837 (+/-0.000) for {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators':
100}
0.837 (+/-0.000) for {'learning_rate': 0.05, 'max_depth': 2, 'n_estimators':
50}
0.837 (+/-0.000) for {'learning_rate': 0.05, 'max_depth': 2, 'n_estimators':
100}
0.837 (+/-0.000) for {'learning_rate': 0.05, 'max_depth': 3, 'n_estimators':
50}
0.837 (+/-0.000) for {'learning_rate': 0.05, 'max_depth': 3, 'n_estimators':
100}
```