Milestone #4

In this notebook we will look to fit a baseline model to the data. We first need severalfunctions to be able to split and analyze our data. First and foremost we will need to import sklearn packages in order to implement the model. We will also need to build some functions that will allow us to analyze the efficacy of particular models.

In [86]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn import preprocessing
from sklearn import cross_validation
from sklearn import metrics
from sklearn import ensemble
from sklearn import discriminant_analysis
from sklearn import linear_model
from sklearn.preprocessing import OneHotEncoder

import datetime

%matplotlib inline
```

In [20]:
```
def KFold_Score(folds, X, y, mdl, metric_func=metrics.accuracy_score):
    """
    Function to take in a single training set and check the efficacy
    of a particular model using `folds`-fold validation.

    The function returns the mean of the `folds` `metric` scores
    """
    # fold the dataset into `folds`
    kf = cross_validation.KFold(len(X), n_folds=folds)

    # array to store results
    kf_res = np.empty((folds,))

    for i, (train_ix, test_ix) in enumerate(kf):
        fold_model = mdl
        fold_model.fit(X[train_ix, :], y[train_ix])

        # inputs are y_true, y_pred
        kf_res[i] = metric_func(y[test_ix], fold_model.predict(X[test_ix, :]))

    # aggregate scores by averaging
    return np.nanmean(kf_res)
```

Let us load in our data.

In [88]:
```
FI = pd.read_csv('datasets/Food_Inspections.csv', index_col='Inspection ID')
```

In [83]:
```
FI.head()
```

Out[83]:

	DBA Name	AKA Name	License #	Facility Type	Risk	Address	City	State	Zip	Inspe Date
Inspection ID										
1967170	GOOSE ISLAND BAR - T1, B4	GOOSE ISLAND (T1-B4)	2477070.0	Restaurant	Risk 2 (Medium)	11601 W TOUHY AVE	CHICAGO	IL	60666.0	10/25
1967164	ERMEL'S	ERMEL'S	2484238.0	Restaurant	Risk 1 (High)	5729 N NORTHWEST HWY	CHICAGO	IL	60646.0	10/25
1967146	WENDY'S PROPERTIES, LLC	WENDY'S	2469194.0	Restaurant	Risk 1 (High)	6324 N WESTERN AVE	CHICAGO	IL	60659.0	10/25
1967133	LEARN TOGETHER GROW TOGETHER CHILD DEVELOPMENT...	LEARN TOGETHER GROW TOGETHER CHILD DEVELOPMENT C	2384887.0	Daycare Above and Under 2 Years	Risk 1 (High)	1126 W 99TH ST	CHICAGO	IL	60643.0	10/25
1967115	Porkchop	Porkchop	2373923.0	Restaurant	Risk 1 (High)	29 E ADAMS ST	CHICAGO	IL	60603.0	10/24

Data Cleaning

This data definitely needs to be cleaned. This will take several steps.

- Remove immediate non-predictor columns (`DBA Name`, `AKA Name`, `License #` (although this is useful later), `Address`)
- Remove uneccesary predictor columns (`City`, `State` (all in Chicago, IL), `Location` (already encapsulated in `Latitude/Longitude`)
- Remove (temporarily) inspection date. This will be useful when we add in data about weather
- Conversion of some columns into dummy variables easier for a computer to interpret.
  - `Facility Type` -> dummies (each separate)
  - `Risk` -> dummies (place on a scale, 1 highest etc.)
  - `Zip` -> dummies (each separate)
  - `Inspection Type` -> dummies (each separate)
  - `Violations` -> dummies (each separate),
- Extra Column for Number of `Violations`

We will need to abstract this whole process into a function so we can clean testing / OOS data.

We also need to consider the following:

In [23]:
```
len(FI['License #'].unique())
```

Out[23]: 31097

In [24]:
```
len(FI.index.unique())
```

Out[24]: 134192

The above means that in this dataset, many restaurants have been inspected more than once. What is unique to each restaurant is its `License #`.

The reason that this is a problem is that there would probably be some conditional distribution on past inspections and past inspection results. This will be a good thing to explore going forward. Does a failing grade on an inspection incentivize restaurants to clean up their act? Do restraunt who pass get complacent and relx their hygeine standards?

In [148]:
```
FI.groupby('License #').last().head(n=5)
```

Out[148]:

	DBA Name	AKA Name	Facility Type	Risk	Address	City	State	Zip	Inspection Date	Inspection Type	F
License #											
0.0	QuiteFrankly,Ltd.	UPS Cafeteria	Restaurant	Risk 1 (High)	1400 S JEFFERSON ST	CHICAGO	IL	60607.0	01/06/2010	Canvass	F
1.0	HARVEST CRUSADES MINISTRIES	HARVEST CRUSADES MINISTRIES	Special Event	Risk 2 (Medium)	118 N CENTRAL AVE	CHICAGO	IL	60644.0	06/04/2010	Special Events (Festivals)	F
2.0	COSI	COSI	Restaurant	Risk 1 (High)	230 W MONROE ST	CHICAGO	IL	60606.0	06/15/2010	Canvass	F
9.0	XANDO COFFEE & BAR / COSI SANDWICH BAR	XANDO COFFEE & BAR / COSI SANDWICH BAR	Restaurant	Risk 1 (High)	116 S MICHIGAN AVE	CHICAGO	IL	60603.0	07/15/2010	Suspected Food Poisoning	F
40.0	COSI	COSI	Restaurant	Risk 1 (High)	233 N MICHIGAN AVE	CHICAGO	IL	60601.0	08/23/2010	Canvass	F

However, for this baseline model, we will ignore these potential complications, although we undertand that delving into this will be an important part of future work.

Also, note that for our baseline model, we decompose the results into `Pass` or `Fail`. We hope to include multi-class classification into our final model too.

There are some further concerns too. For example, there are a great deal of Inspction Types. Let us see how many of them have been used. *NB:* A personal favourite is 'TWO PEOPLE ATE AND GOT SICK'.

In [26]:
```
FI['Inspection Type'].unique()
```

```
Out[26]: array(['License', 'Short Form Complaint', 'License Re-Inspection',
                'Complaint', 'Complaint Re-Inspection', 'Canvass',
                'Suspected Food Poisoning Re-inspection', 'Canvass Re-Inspection',
                'Suspected Food Poisoning', 'Tag Removal', 'Consultation',
                'Recent Inspection', 'Special Events (Festivals)', nan, 'Not Ready',
                'License-Task Force', 'Complaint-Fire Re-inspection',
                'Complaint-Fire', 'Short Form Fire-Complaint', 'Non-Inspection',
                'KITCHEN CLOSED FOR RENOVATION', 'O.B.', 'CORRECTIVE ACTION',
                'Package Liquor 1474', 'LICENSE CANCELED BY OWNER',
                'OWNER SUSPENDED OPERATION/LICENSE', 'LICENSE CONSULTATION',
                'License consultation', 'Task Force Liquor 1475',
                'Illegal Operation', 'fire complaint',
                'TWO PEOPLE ATE AND GOT SICK.', 'Pre-License Consultation',
                'CANVASS SPECIAL EVENTS', 'CANVASS SCHOOL/SPECIAL EVENT',
                'OUT OF BUSINESS', 'No entry', 'NO ENTRY', 'no entry',
                'TASK FORCE LIQUOR 1470', 'Sample Collection', 'license task 1474',
                'LICENSE REQUEST', 'FIRE/COMPLAIN', 'Task Force for liquor 1474',
                'Out of Business', 'ADDENDUM', '1315 license reinspection',
                'No Entry', 'Task force liquor inspection 1474',
                'Task Force Liquor Catering', 'SFP', 'CANVAS', 'SFP/COMPLAINT',
                'TASK FORCE NIGHT', 'SFP/Complaint', 'expansion',
                'SMOKING COMPLAINT', 'SFP RECENTLY INSPECTED', 'CANVASS',
                'TAVERN 1470', 'LICENSE RENEWAL INSPECTION FOR DAYCARE',
                'LICENSE RENEWAL FOR DAYCARE', 'CHANGED COURT DATE',
                'CANVASS RE INSPECTION OF CLOSE UP', 'TASKFORCE',
                'LICENSE TASK FORCE / NOT -FOR-PROFIT CLUB',
                'LICENSE TASK FORCE / NOT -FOR-PROFIT CLU', 'LICENSE/NOT READY',
                'NO ENTRY-SHORT COMPLAINT)', 'CITF', 'KIDS CAFE',
                'LICENSE DAYCARE 1586', 'task force(1470) liquor tavern',
                'LICENSE WRONG ADDRESS', 'error save', 'CANVASS/SPECIAL EVENT',
                'DAY CARE LICENSE RENEWAL', 'LIQUOR CATERING', 'Summer Feeding',
                'TASK FORCE PACKAGE LIQUOR', 'citation re-issued',
                'TASTE OF CHICAGO', 'LICENSE', 'HACCP QUESTIONAIRE',
                'out ofbusiness', 'CLOSE-UP/COMPLAINT REINSPECTION',
                'finish complaint inspection from 5-18-10', 'Duplicated',
                'sfp/complaint', 'license', 'RECALL INSPECTION',
                'TASK FORCE LIQUOR (1481)', 'Special Task Force',
                'REINSPECTION OF 48 HOUR NOTICE', 'REINSPECTION',
                'Business Not Located', 'CANVASS FOR RIB FEST',
                'RE-INSPECTION OF CLOSE-UP', 'task force', 'SPECIAL TASK FORCE',
                'LIQOUR TASK FORCE NOT READY', 'TASK FORCE NOT READY',
                'POSSIBLE FBI', 'TASK FORCE LIQUOR 1474', "Kids Cafe'",
                'TASK FORCE PACKAGE GOODS 1474'], dtype=object)
```

In [27]:
```
FI.groupby('Inspection Type').count().loc[:, 'License #'].sort_values(ascending=False).head(30)
```

Out[27]:
```
Inspection Type
Canvass                                    70424
License                                    17610
Canvass Re-Inspection                      12835
Complaint                                  12266
License Re-Inspection                       6631
Short Form Complaint                        5329
Complaint Re-Inspection                     5061
Suspected Food Poisoning                     649
Consultation                                 646
License-Task Force                           605
Tag Removal                                  603
Out of Business                              284
Task Force Liquor 1475                       254
Recent Inspection                            167
Complaint-Fire                               161
Suspected Food Poisoning Re-inspection       151
Short Form Fire-Complaint                    113
No Entry                                      60
Special Events (Festivals)                    56
Complaint-Fire Re-inspection                  44
Package Liquor 1474                           44
OUT OF BUSINESS                               22
LICENSE REQUEST                               19
Pre-License Consultation                      15
Not Ready                                     10
Non-Inspection                                10
NO ENTRY                                       7
Illegal Operation                              5
no entry                                       4
SFP                                            4
Name: License #, dtype: int64
```

As we can see from the data above, the vast majority of `Inspection Types` are confined to a small subset of the total number of types listed above. It would make sense to only use the most common data as columns for a dummy predictor and store the rest under the custom label of `misc`. The cutoff for this will be 100 registered inspection types. Although it would be more ideal to do this in a more rigorous automated, it is clear that we do not want information as to food poisoning reinspections masked by noise in the catch-all column. We do something very similar to the `Facility Type Column`

In [28]:
```
set(FI.groupby('Inspection Type').count().loc[:, 'License
#'].sort_values(ascending=False).iloc[:17].index)
```

Out[28]:
```
{'Canvass',
 'Canvass Re-Inspection',
 'Complaint',
 'Complaint Re-Inspection',
 'Complaint-Fire',
 'Consultation',
 'License',
 'License Re-Inspection',
 'License-Task Force',
 'Out of Business',
 'Recent Inspection',
 'Short Form Complaint',
 'Short Form Fire-Complaint',
 'Suspected Food Poisoning',
 'Suspected Food Poisoning Re-inspection',
 'Tag Removal',
 'Task Force Liquor 1475'}
```

In [122]:

```
import re

#Encode categorical variables using sklearn's one-hot encoder
def encode_categorical(array):
    if not array.dtype == np.dtype('float64'):
        return preprocessing.LabelEncoder().fit_transform(array)
    else:
        return array

# helper functions abstracted for clarity, adaptibility
def results_helper(x):
    """
    Helper for results column
    """
    return np.where(x == 'Pass', 1, 0)

def inp_type_helper(df, col):
    """
    Helper for inspection type column.
    Would be great to have a better metric than 17 arbitrarily for the future.
    """
    dummy_set = set(df.groupby('Inspection Type').count().loc[:, 'License #'].sort_values(ascending=F
alse).iloc[:17].index)
    return ['Misc' if x not in dummy_set else x for x in col]


def fac_type_helper(df, col):
    """
    Helper for inspection type column.
    Would be great to have a better metric than 17 arbitrarily for the future.
    """
    dummy_set = set(df.groupby('Facility Type').count().loc[:, 'License #'].sort_values(ascending=Fal
se).iloc[:21].index)
    return ['Misc' if x not in dummy_set else x for x in col]

def risk_helper(col):
    """
    Helper for risk column.
    Catch-all is 4
    """
    bad_set = ['All', np.nan]
    return [x.split(' ')[1] if x not in bad_set else 4 for x in col]
```

```python
def viols_helper(df):
    """
    Helper for violations column.
    Also creates a column for number of vioaltions
    """
    # cleaned data. will be inputted into DF after cleaning
    viol_list_of_lists = []

    for i, viol in enumerate(df['Violations']):
        # for each establishment
        viols = []

        # if nan, no complaints
        if pd.isnull(viol):
            viol_list_of_lists.append(viols)
        else:
            # split into separate complaints
            viols = viol.split(' | ')
            for j, complaint in enumerate(viols):
                complaint = complaint.split(' - Comments: ')[0]
                viols[j] = complaint
            viol_list_of_lists.append(viols)

    violations_df = pd.Series([item for sublist in viol_list_of_lists for item in sublist])
    no_viols = [len(x) for x in viol_list_of_lists]

    for lst in viol_list_of_lists:
        for i, viol in enumerate(lst):
            code = viol.split('. ')[0]
            lst[i] = int(code)

    return (no_viols, viol_list_of_lists)

def clean_and_split(df, multiclass = False):
    """
    Function to clean raw food inspection data and
    split this into predictor and label parts
    """
    df = df.drop(['DBA Name', 'AKA Name', 'Address', 'City', 'State', 'Location'], 1)
    df = df.drop('Inspection Date', 1) # NB will most likely be included in the final model

    # clean inspection types
```

```
    df['Inspection Type'] = inp_type_helper(df, df.loc[:, 'Inspection Type'])

    # clean facility types
    df['Facility Type'] = fac_type_helper(df, df.loc[:, 'Facility Type'])

    # clean risk types
    df['Risk'] = risk_helper(df['Risk'])

    # clean violations and add nnumber of violations
    no_viols, viols = viols_helper(df)
    df['# of Violations'] = no_viols
    df['Violations'] = viols

    # split columns into dummies
    viols_dummies_df = pd.get_dummies(pd.Series(df['Violations']).apply(pd.Series).stack()).sum(level=
)
    zip_dummies_df = pd.get_dummies(df['Zip'])
    inp_dummies_df = pd.get_dummies(df['Inspection Type'])
    fac_dummies_df = pd.get_dummies(df['Facility Type'])

    # drop columns that are now dummies
    df = df.drop(['Violations', 'Zip', 'Inspection Type', 'Facility Type'], 1)

    # add dummy columns
    df = pd.concat([df, viols_dummies_df, zip_dummies_df, inp_dummies_df, fac_dummies_df], axis=1)

    # drop last column
    df = df.drop('License #', 1)

    # drop nans, which will cause models to fail
    df = df.dropna(axis=0)

    # split off results and predictors and clean into Pass/Fail (if not doing multiclass)
    # (Note, we only consider 'Pass' as a true Pass, as 'Pass with Conditions' in some sense implies
 a failure in the current state.)
    dirty_y = df.loc[:, 'Results']
    if not multiclass:
        y = results_helper(dirty_y)
    else:
        y = preprocessing.LabelEncoder().fit_transform(dirty_y)

    df = df.drop('Results', 1)
    return (df, y)
```

In [120]:
```
FI.Results.shape[0]
```
Out[120]:
```
134192
```
In [126]:
```
fi, y = clean_and_split(FI, multiclass=False)
```

Model Creation

This being a classification problem, let us see if we can tune a logistic regression model to this data.

In [17]:
```
# baseline
KFold_Score(5, fi.as_matrix(), np.array(y), linear_model.LogisticRegression())
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-17-dada070bce53> in <module>()
      1 # baseline
----> 2 KFold_Score(5, fi.as_matrix(), np.array(y), linear_model.LogisticRegression())

NameError: name 'fi' is not defined
```

As we can see Logistic Regression is not necessarily the best model to use.

In [14]:
```
def plot_tuning_results(tuning_vals, tuning_res_1, two_plots, tuning_res_2, log_flag, lab1, lab2, tit
le):
    """
    Plot results for tuning parameters
    """
    plt.plot(tuning_vals, tuning_res_1, label=lab1, c='b')
    if two_plots:
        plt.plot(tuning_vals, tuning_res_2, label=lab2, c='g')

    plt.title(title)
    plt.xlabel('Tuning Values')
    plt.ylabel('Scores')

    if log_flag:
        plt.xscale('log')

    plt.ylim([0., 1.])
    plt.legend();
```

In [16]:
```
KFold_Score(5, fi.as_matrix(), np.array(y), ensemble.RandomForestClassifier())
```

Out[16]: 0.94850345759129517

Using the Random Forest Classifier, we achieve a slightly better score. This can be attributable to the fact that tree ensembles do not expect linear features, which may not be present in the inspection data.

Additional features (weather)

To improve the model further, we can aggregate external weather data to training data. We pulled data the daily max and daily minimum temperatures from weather stations in Chicago. After cleaning up the dataset, we appended the data to the entire inspection dataset.

In [76]:
```
weather_df = pd.read_csv('datasets/weather.csv')
```

In [77]:
```
weather_df.head()
```

Out[77]:

	STATION	ELEVATION	LATITUDE	LONGITUDE	DATE	TAVG	TMAX	TMIN
0	GHCND:USC00111550	180.4	41.86611	-87.61528	20100101	-9999	21	10
1	GHCND:USC00111550	180.4	41.86611	-87.61528	20100102	-9999	16	7
2	GHCND:USC00111550	180.4	41.86611	-87.61528	20100103	-9999	24	6
3	GHCND:USC00111550	180.4	41.86611	-87.61528	20100104	-9999	21	13
4	GHCND:USC00111550	180.4	41.86611	-87.61528	20100105	-9999	27	19

In [78]:
```
weather_df['DATE'] = pd.to_datetime(weather_df['DATE'], format="%Y%m%d")

weather_df['Inspection Date'] = weather_df['DATE'].dt.strftime('%m/%d/%Y')
```

In [79]:
```
weather_df.drop(['STATION','LATITUDE','DATE','TAVG','ELEVATION','LONGITUDE'],inplace=True,axis=1)
```

In [80]:
```
weather_df.head()
```

Out[80]:

	TMAX	TMIN	Inspection Date
0	21	10	01/01/2010
1	16	7	01/02/2010
2	24	6	01/03/2010
3	21	13	01/04/2010
4	27	19	01/05/2010

In [149]:

```
FI.set_index('Inspection Date').join(weather_df.set_index('Inspection Date')).head(n=5)
```

Out[149]:

	DBA Name	AKA Name	License #	Facility Type	Risk	Address	City	State	Zip	Inspection Type	Results	Violations
Inspection Date												
01/02/2013	NICK'S GYROS	NICK'S GYROS	1403378.0	Restaurant	Risk 1 (High)	2011 W 63RD ST	CHICAGO	IL	60636.0	Complaint	Pass	36. LIGHTING REQUIREI MINIMUM FOOT-CANDLES OF...
01/02/2013	NICK'S GYROS	NICK'S GYROS	1403378.0	Restaurant	Risk 1 (High)	2011 W 63RD ST	CHICAGO	IL	60636.0	Complaint	Pass	36. LIGHTING REQUIREI MINIMUM FOOT-CANDLES OF...
01/02/2013	NICK'S GYROS	NICK'S GYROS	1403378.0	Restaurant	Risk 1 (High)	2011 W 63RD ST	CHICAGO	IL	60636.0	Complaint	Pass	36. LIGHTING REQUIREI MINIMUM FOOT-CANDLES OF...
01/02/2013	NICK'S GYROS	NICK'S GYROS	1403378.0	Restaurant	Risk 1 (High)	2011 W 63RD ST	CHICAGO	IL	60636.0	Complaint	Pass	36. LIGHTING REQUIREI MINIMUM FOOT-CANDLES OF...

	DBA Name	AKA Name	License #	Facility Type	Risk	Address	City	State	Zip	Inspection Type	Results	Violations
Inspection Date												
01/02/2013	NICK'S GYROS	NICK'S GYROS	1403378.0	Restaurant	Risk 1 (High)	2011 W 63RD ST	CHICAGO	IL	60636.0	Complaint	Pass	36. LIGHTING REQUIREI MINIMUM FOOT-CANDLES OF...

In [82]:
```
fi, y = clean_and_split(FI)
```

In [73]:
```
KFold_Score(5, fi.as_matrix(), np.array(y), ensemble.RandomForestClassifier())
```

Out[73]:
```
0.94930075131680636
```

With the additional weather data, the accuracy score of our Random Forest model increases slightly. It should be noted that our weather data only takes the temperature from one weather station; to be even more accurate we can take the average of multiple weather station or perhaps indentify the one closest to the actual restaurant (using the latitude and longitude data).

In [134]:
```
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.cross_validation import KFold
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import roc_auc_score
```

Tuning our models

Now let's use grid search to optimize the hyper-parameters of our logistic regression model using cross-validation.

In [37]:
```
# param_grid = np.logspace(-6, -1, 10)
# clf = GridSearchCV(linear_model.LogisticRegression(penalty='l2'), param_grid)
grid = {
        'C':
         , 'solver': ['newton-cg']
    }
fold = KFold(len(y), n_folds=5, shuffle=True, random_state=777)
np.power(10.0, np.arange(-10, 10)
clf = linear_model.LogisticRegression(penalty='l2', random_state=777, max_iter=10000, tol=10)
gs = GridSearchCV(clf, grid, scoring='roc_auc', cv=fold)
gs.fit(fi.as_matrix(), np.array(y))

print ('gs.best_score_:', gs.best_score_)
# gs.predict
```

```
('gs.best_score_:', 0.95635889695346177)
```

In [52]:
```
X_train, X_test, y_train, y_test = train_test_split(fi.as_matrix(), np.array(y), test_size=0.3, rando
m_state=0)
y_preds = gs.predict(X_test)
print(classification_report(y_test, y_preds))
```

```
             precision    recall  f1-score   support

          0       0.94      0.89      0.92     10689
          1       0.95      0.97      0.96     21294

avg / total       0.94      0.94      0.94     31983
```

Random Forests

In [137]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import roc_auc_score
```

In [141]:
```
model = RandomForestRegressor(n_estimators = 100 , oob_score = True, random_state = 42)
model.fit(X_train, np.array(y_train))
```

Out[141]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_split=1e-07, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=100, n_jobs=1, oob_score=True, random_state=42,
           verbose=0, warm_start=False)
```

In [147]:
```
sample_leaf_options = [1,5,10,50,100,200,500]

for leaf_size in sample_leaf_options:
    model = RandomForestRegressor(n_estimators = 200, oob_score = True, n_jobs = -1, random_state
=50, max_features = "auto", min_samples_leaf = leaf_size)
    model.fit(X_train, np.array(y_train))
    print "AUC - ROC : " + roc_auc_score(y,model.oob_prediction)
```

Deciding on a *performance metric*

- For this project, we have in essence been hired by the city of Chicago to examine whether we can reduce the spread of foodborne illness by locating restaurants with a high probability of violating the health codes for their early inspection. Our goal is to avoid exposing these restaurants' patrons to unnecessary risk, by optimizing the use of the city's limited number of inspections.
- Given that our goal is to reduce the spread of foodborne illnesses, and therefore to correctly identify violators as swiftly as possible, our performance metric should be much more weighted towards reducing the false negative rate, and therefore maximizing the *sensitivity* (or true positive) percentage. In this case, where Chicagoans are better off safe than sorry, maximizing *sensitivity* is more important than minimizing *specificity*, or the false positive rate. While it's not ideal to misclassify a clean restaurant as violating a health code, especially given the heatlh department's limited resources, accidentally shutting down a well-run restaurant is not the end of the world given the plethora of dining options in the city, whereas not identifying a potential outbreak could result in unnecessary deaths, as well as undue strain on the health care system and mistrust in the local food industry. Thus, **for now we will focus on maximizing the sensitivity of our models when evaluating our predictions.**

In [69]:
```
score_df = pd.concat([pd.Series(y_preds, name='Predictions'), pd.Series(y_test, name='True Vals')], a
xis=1)

true_positives = score_df[score_df['Predictions'] == 1][score_df['True Vals'] == 1]
true_negatives = score_df[score_df['Predictions'] == 0][score_df['True Vals'] == 0]
false_positives = score_df[score_df['Predictions'] == 1][score_df['True Vals'] == 0]
false_negatives = score_df[score_df['Predictions'] == 0][score_df['True Vals'] == 1]
```

```
/Users/evanbrown/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:3: UserWarning: Boolean
 Series key will be reindexed to match DataFrame index.
  app.launch_new_instance()
/Users/evanbrown/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:4: UserWarning: Boolean
 Series key will be reindexed to match DataFrame index.
/Users/evanbrown/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:5: UserWarning: Boolean
 Series key will be reindexed to match DataFrame index.
/Users/evanbrown/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:6: UserWarning: Boolean
 Series key will be reindexed to match DataFrame index.
```

In [72]:
```
print "Our Tuned Model's Sensitivity is " + str(true_positives.shape[0] / float(true_positives.shape[(
 + false_positives.shape[0]))
print "The False Negative Rate is " + str(false_negatives.shape[0] / float(false_negatives.shape[0] +
 true_positives.shape[0]))
```

```
Our Tuned Model's Sensitivity is 0.946915614131
The False Negative Rate is 0.0282708744247
```

This is a pretty low false negative rate, which is rather reassuring.

Multinomial Logistic Regression

In [8]:
```
FI.Results.unique()
```

Out[8]:
```
array(['Pass', 'No Entry', 'Pass w/ Conditions', 'Fail', 'Out of Business',
       'Not Ready', 'Business Not Located'], dtype=object)
```

In [127]:
```
fi, multi_y = clean_and_split(FI, multiclass=True)
```

In [133]:
```
multilogreg = linear_model.LogisticRegression(penalty='l2', random_state=777, max_iter=10000, tol=10,
 solver='newton-cg', multi_class='multinomial')

multilogreg.fit(fi.as_matrix(), multi_y)
```

Out[133]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=10000, multi_class='multinomial',
          n_jobs=1, penalty='l2', random_state=777, solver='newton-cg',
          tol=10, verbose=0, warm_start=False)
```

In [ ]:
```
```