Section 1: Week 2: Evaluate Data Mining Techniques

Nate Bachmeier

TIM-8130: Data Mining

February 15th, 2020

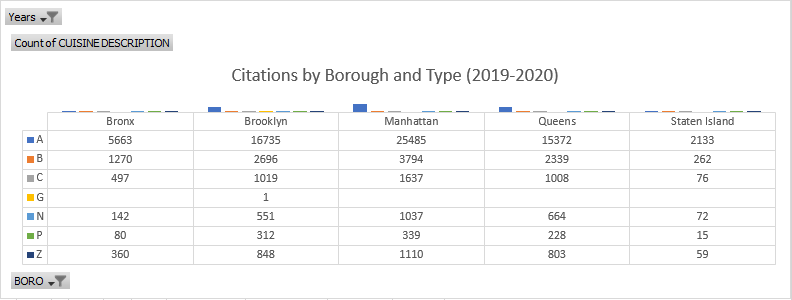
North Central University

Evaluate Data Mining Techniques

# Section I: Using Data Mining

The city of New York offers multiple data sets for businesses and researchers to explore and make discoveries. The Department of Health and Mental Hygiene publishes a daily feed of NYC Restaurant Inspection Results that enumerates all citations since July 2015 (DOHMH, 2020). Each establishment receives a grade of A, B, C, P(ending), N(ot available), or Z based on the severity of citations. According to NYC Health (2020), “a business must maintain a grade of A or B, or close until passing a future inspection.” Figure 1 displays the distribution of citations across the different boroughs for January 2019 through Feb 2020, with approximately 87.4% of locations maintain a high-quality rating, with only 5% receiving a grade of C. There are several outliers, such as Dunkan Donuts is consistently the most infractions, followed by McDonalds and Star Bucks, in part due to the number of locations. Specific ethnic categories, such as Chinese in Queens and Indian in the Bronx, have high infractions due to their concentration. Niche categories, such as Cajun-Creole, have an alarming rate of 22% non-compliance due to their only being 58 total businesses in the category.

Figure Citations by Borough



The feed also details the violation code for the citation along with a numeric score that typically has a mean value of 20.4 and a standard deviation of 14.8 and a critical flag, with lower scores representing more minor infractions. A goal of the research is to predict the score that an organization would receive. First, the features Cuisine Description, Critical Flag, Violation Code, Inspection Type were One Hot Encoded into a 586 by 200730 matrix and fed into Scikit Learn’s Logistic Regression algorithm using a 50% split for testing/training data. The test data accuracy was 10.8%, making this strategy unusable. Next, the Borough, Critical Flag, Grade, Violation Code, and Inspection Type was One Hot Encoded into a 75 x 200730 matrix for the Decision Tree Regressor algorithm. This solution has an accuracy of 67.8% and was took significantly less time to train. Next, a Principal Components Analysis (PCA) over this same data frame was able to reduce the matrix dimensionality to 25 x 200730. This result did not change the accuracy results of either algorithm though it did speed up the training by 2-3x. Then the score was partitioned into value ranges -5 to 15, 16 to 25, 26 to 50, and 50 to 200. Using the Decision Tree Classifier was able to predict the bucket with a 93.8% accuracy. Additional analysis found that the Grade feature has the strongest influence, and its removal decreases the Decision Tree Classifier’s accuracy to only 51.7%. Removing both the Grade and PCA preprocessing increases the One Hot Encoded matrix from 10 x 190803 to 68 x 190803 with a total accuracy of 0.03%. This test suggests that there is insufficient data for this strategy, and the boost from using PCA correlations is critical for the sparse data set. Finally, MLPRegressor, a neural network-based algorithm was tested to have a 64.8% accuracy when both the Grade and PCA are available. The Appendix section contains an example Python script that calculates these values within the latest Docker container of jupyter/scipy-notebook.

Table : Machine Learning Algorithms Used

|  |  |  |
| --- | --- | --- |
| Algorithm | Strategy Description | Comments |
| Logistic Regression | Maps numeric features to a numeric range. | The label-centric data set was not very applicable |
| Decision Tree Regression | Maps vectorized labels to a numeric range. | Effective with many guesses in general ballpark |
| PCA | Uses the correlation between features and combine columns | Improves runtime and accuracy of sparse data |
| Decision Tree Classifier | Maps vectorized labels to finite categories (e.g., value-ranges) | More accurate than DTR as general guesses are sufficient |
| MLPRegressor | Uses backpropagation to build a fitting expression | Suffers from the same challenges as LR |

# Section II: Analyze the Process

Four distinct phases need to occur before an organization can operationalize predictive capabilities. First, relevant data that needs to be collected and curated into a useable format. Understanding the health violations provides one aspect to restaurant recommendation, however other sources such as Yelp Reviews can augment this feed. Those secondary assets need some mechanism for extract features and schematize the unstructured data. Second, using exploratory programs, such as Microsoft Excel or Jupyter Notebooks, pivot tables, and data visualizations, convey the shape and complexity associated with building a model. Third, the data engineer needs to evaluate the accuracy of different machine learning algorithms. Fourth, the bundling of the model into an Application Programming Interface (API) needs to occur. It is not sufficient to perform this process linearly once, as a continuous feedback loop needs to provide further insight into the solution.

IoT, Cloud, Mobile, and Big Data (ICBM) increase the complexity to perform these steps as the volume, variety, velocity, and veracity add unique challenges. For instance, production datasets are commonly several factors of magnitude larger than the local resources of a single computer. Users need to either apply a sampling strategy or rely on distributed data stores, such as Azure Data Lake, Amazon DynamoDB, or Apache Hadoop. The time required to train a model can also introduce challenges, as it impacts the user’s ability to iterate. Consider the DOHMH dataset, where applying Logistic Regression to 586 by 200000 matrix took several minutes to converge. Jassy (2019), CEO of AWS, recently stated that “after creating these models, substantial more processing power is typically necessary to run the inferences.” Google is addressing these challenges with custom hardware that runs at the network edge. By running machine learning algorithms at the edge, it removes the need to move across the network these massive workloads and also decreases latency. Minimizing network latency is critical for many machine learning solutions, such as health and safety monitoring, that need to detect anomalies in real-time.

# Appendix: Python Code

# Load the CSV file into Pandas

# This creates a DataFrame that allows efficient exploration

import pandas as pd

csv = pd.read\_csv('DOHMH\_New\_York\_City\_Restaurant\_Inspection\_Results.csv')

# This snippet creates the one-hot encoded data frame

# Garreta, R (2013). Learning sckit-learn machine learning in python

from sklearn import feature\_extraction

def one\_hot\_dataframe(data, cols, replace=False):

    vec = feature\_extraction.DictVectorizer()

    mkdict = lambda row: dict((col, row[col]) for col in cols)

    vecData = pd.DataFrame(vec.fit\_transform(

    data[cols].apply(mkdict, axis=1)).toarray())

    vecData.columns = vec.get\_feature\_names()

    vecData.index = data.index

    if replace:

        data = data.drop(cols, axis=1)

        data = data.join(vecData)

    return (data, vecData)

# Split the test and training data

train = csv[:200000][["BORO","CRITICAL FLAG","GRADE","VIOLATION CODE","INSPECTION TYPE","SCORE"]].dropna()

test = csv[200000:][["BORO","CRITICAL FLAG","GRADE","VIOLATION CODE","INSPECTION TYPE","SCORE"]].dropna()

# Build the train set

X\_train = train.drop(columns=['SCORE'])

Y\_train = train[["SCORE"]]

# Build the test set

X\_test = test.drop(columns=['SCORE'])

Y\_test = test[["SCORE"]]

# One Hot encode

x\_train\_hot = one\_hot\_dataframe(X\_train, ["BORO","CRITICAL FLAG","GRADE","VIOLATION CODE"], replace=True)

x\_test\_hot = one\_hot\_dataframe(X\_test, ["BORO","CRITICAL FLAG","GRADE","VIOLATION CODE"], replace=True)

# Drop these columns as they do not exist in both halfs

x\_train\_data = x\_train\_hot[1].drop(columns=["VIOLATION CODE=06H"])

x\_test\_data = x\_test\_hot[1].drop(columns=['VIOLATION CODE=02E','VIOLATION CODE=04I'])

from sklearn import tree

clf = tree.DecisionTreeRegressor()

clf.fit(x\_train\_data, Y\_train['SCORE'])

clf.score(x\_test\_data, Y\_test['SCORE'])

# 67%

from sklearn.decomposition import PCA

pca = PCA(n\_components=10)

pca = pca.fit(x\_train\_data, Y\_train['SCORE'])

x\_train\_pca = pca.transform(x\_train\_data)

x\_test\_pca = pca.transform(x\_test\_data)

from sklearn import tree

clf = tree.DecisionTreeRegressor()

clf.fit(x\_train\_pca, Y\_train['SCORE'])

clf.score(x\_test\_pca, Y\_test['SCORE'])

# 67%

bins = [-5, 15, 25,50,200]

Y\_train\_bins = pd.cut(Y\_train['SCORE'], bins=bins, labels=bins[0:-1])

Y\_test\_bins =  pd.cut(Y\_test['SCORE'], bins=bins, labels=bins[0:-1])

from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier()

clf.fit(x\_train\_pca, Y\_train\_bins)

clf.score(x\_test\_pca, Y\_test\_bins)

# 93.8%

import numpy as np

Y\_rand\_bins = pd.cut((np.random.random(Y\_test\_bins.shape)\*100), bins=bins, labels=bins[0:-1])

clf.score(x\_test\_pca, Y\_rand\_bins)

# 10.7%