McCoy Doherty & Nicolas Ortega
SI 670 - Prof. Schoenebeck

Boston Health Inspection Projection

Introduction

The Boston Department of Innovation and Technology (DOIT) published a large dataset regarding food establishment inspections in the city of Boston. The Health Division of the Department of Inspectional Services (ISD) carries out these inspections, ensuring all establishments meet relevant sanitary codes and standards. Alongside the food establishment inspection dataset, the DOIT has data regarding building permits for buildings across the city of Boston. Our project aims to join these two data sources together in order to build a classifier that can predict whether or not an inspection will result in a pass or a fail. In a time where all of us have become much more aware with the objects we touch and the places we frequent, food establishment hygiene and relevant health-codes are more important than ever.

Related Work

Some Kaggle notebooks exist exploring Chicago health-inspection data, but none seem to exhibit predictive work as we intend to implement with our comparatively-more-extensive features. Chicago also utilized R for predicting food-safety violations and, while cautioning that "specifics do change between cities," a city in Maryland was inspired by the success in Chicago and was later able to achieve success at predicting their food-safety violations by developing a modified predictive approach. We did not find much closely-related work for our problem and type of data (ie, health inspection and building permits).

Methods, Evaluation, and Analysis

Datasets

We used the Boston Food Establishment Health Inspections dataset of 625,000 records detailing the annual health-inspection outcomes of food-service entities in the Boston area from 2006 to present. It includes data about owners, property, location, plus levels and types of existing violations. Some data is absent, including labels, but overall there seems little textual error to be "cleaned". To expand our features, we'll also be using the Approved Building Permits data detailing regulation-required fixes for permit eligibility, square footage, and occupancy coding.

These datasets share a 'property_id' dimension, which we used to merge the data to obtain more features. A main challenge in this project is that there was no identifier for each inspection. Rather, there are repeating values of property_id for each inspection/row. For this reason, our analysis is more appropriate for prediction of the outcome of a given inspection, rather than for a given restaurant/food establishment. Further, since the dataset is updated daily, we have downloaded the data and are using a "frozen" dataset, which we will use for model building and evaluation.

The datasets can be found at:

- Boston Food Establishment Health Inspections
- Boston Approved Building Permits

Data Cleaning

We joined the inspections and permits data based on the property_id field. Because of the duplicated property_id values, our merged dataset became unmanageably large at *(45354642, 71)*, around 8GB CSV file. For this reason, we sampled .001% of our data to speed up the process and see the performance on a small subset of data. Although we would like to use all data, it seems unfeasible to get enough compute for that.

There are four components to our deliverable code:

1) **Initial_clean.ipynb**

   We read in both relevant datasets. Inspections data has shape *(619363, 26)*, and building permits data has shape *(474656, 22)*. We first observed a plethora of text dimensions and other features that may not end up being useful for our purposes, as well as other redundant information. Our pre-processing rationale is as follows:

   - Inspections data: needed to change column dtypes to appropriate types, as well as filter our "result" column to only include PASS/FAIL since the scope is a binary classifier. Shape after pre-processing: *(360232, 15)*
     - Avoid including personal Identifiable Information (PII): drop business name, dbaname, legal owner first and last names, address.

    - Remove text features: drop comments column.
    - Remove temporal data: drop license issued/expired date, violation pass/fail date.
    - One-hot/label encoding for classification: encode license category, one-hot encode zip codes, label encode the result column as binary PASS/FAIL (since it's the target variable).
    - Out of scope: drop violation codes, violation comments, violation level, violation status -- our scope is a binary classifier.
    - Redundant: drop state.

- Permits data: although many features were not very relevant for our purpose, we tried to obtain the most information possible based on "property_id". Shape after pre-processing: *(465787, 9)*
    - Avoid PII: drop applicant, owner, address
    - Remove text features: drop comments, description
    - Remove temporal data: drop issued date and expiration date
    - One-hot/label encoding: one-hot permit status
    - Scaling numerical data: scale declared valuation, total_fees, square footage
    - Redundant: drop state, city, zip, occupancy_type

2) **dask_data_manip.ipynb**

   After merging the two datasets, the resulting 8.8GB CSV file proved difficult to work with locally. This notebook reads the file via parallelized computing package Dask and exports samples that are more manageable on our local machines. Dask samples then convert to conventional Pandas dataframes and dump to CSV. Our samples were, proportionally, 0.001%, 0.0015%, and 0.01% of our data, or 8MB, 41MB, and 82MB.

3) **model_build.ipynb + deep_build.ipynb**

   These notebooks read in the samples generated above, scale the square-footage feature with the robust scaler because of its robustness to outliers, separates target and predictive features, and performs a stratified test-train split to ensure better representation of label frequency during the model training processes to follow. In model_build, dummy

classifiers are utilized to establish a performance baseline for more sophisticated models to compare against, tested on each data sample-size and recorded in Table 1. Following baseline, we tested the performance of classic linear models, logistic regression and linear SVC, as well as popular less-linear models, K-Nearest Neighbors, Random Forest, and XGBoost, including principal-component analysis to improve runtime and performance and grid-search for accuracy optimization. The deep_build notebook also contains our Tensorflow 2 code to experiment with neural-network-based approaches.

Evaluation and Analysis

Starting with our 8MB sample, we ran various dummy classifier strategies to establish a performance baseline, reflected in Table 1, and the best strategies of which were most-frequent and prior, at 0.5113 accuracy. Baseline established, we proceeded to gauge performance of various linear and machine learning models under default parameters, as recorded in Table 2. Observing relatively poor (baseline-comparable) performance in logistic regression and linearSVC, 0.49 and 0.51 accuracy, respectively, we opted to explore less linear models, primarily K-Nearest Neighbors, Random Forests, and XGBoost. These models performed substantially better (0.553, 0.546, and 0.577, respectively) and appeared to be more performance-responsive to the larger data samples, with KNN and Random Forests increasing over a percentage from the ten-fold increase in data, and XGB's response being relatively stagnant and unvaried. Numerous attempts at a neural-network approach seemed to hit a dummy-classifier-reminiscent, disappointingly-low performance ceiling and were not investigated further.

Table 1: Dummy Classifier Performance

Strategy	8mb sample	41mb sample	82mb sample	difference (8mb - 41mb)	difference (41mb - 82mb)	difference (8mb - 82mb)
uniform	0.4963	0.4992	0.50060	0.0029	0.00140	0.0043
stratified	0.5033	0.5034	0.49870	0.0001	-0.00470	-0.0046
most-freq	0.5113	0.5136	0.51350	0.0023	-0.00010	0.0022
prior	0.5113	0.5136	0.51350	0.0023	-0.00010	0.0022

Table 2: Models Performance (Default Parameters, No PCA)

	8mb sample	41mb sample	82mb sample	difference (8mb - 41mb)	difference (41mb - 82mb)	difference (8mb - 82mb)
vanilla logreg	0.4906	0.5009	0.50080	0.0103	-0.00010	**0.0102**
vanilla linSVC	0.5102	0.5122	0.48650	0.002	-0.02570	-0.0237
vanilla KNN	0.553	0.5692	0.56810	0.0162	-0.00110	**0.0151**
vanilla RF	0.5459	0.5552	0.55819	0.0093	0.00299	**0.0123**
vanilla XGB	0.5765	0.5744	0.57907	-0.0021	0.00467	0.0026

The top-three performing classifiers, KNN/RF/XGB, were then tested for accuracy at 8MB of data under one-to-nine principal components, documented in Table 3, and we observed the strongest performances from KNN with six components (0.5855) and Random Forest with three components (0.5997), afterwhich all five methods were tested at three and six components, for each sample size, reported in Table 4 and Table 5. Ultimately, our strongest performance returns appeared to be derived from three components and the KNN and Random Forest algorithms; while XGB and KNN appeared technically stronger with six components, the KNN difference was marginal enough to appear negligible (0.00002) and XGB's six-component performance still trailed the top two models by over a full percentage point.

Table 3: Principal Component Performance of Top Models

Model	PCA=1	PCA=2	**PCA=3**	PCA=4	PCA=5	**PCA=6**	PCA=7	**PCA=8**	PCA=9
KNN	0.568207	0.579032	0.580152	0.581437	0.583753	**0.585543**	0.585181	0.584127	0.579526
RF	0.596698	0.598641	**0.599662**	0.559325	0.560983	0.565012	0.572313	0.572061	0.571973
XGB	0.566802	0.566583	0.569107	0.566034	0.561642	0.572730	0.570205	**0.575694**	0.572730

Table 4: Model Performance -- 3 Principal Components

	8mb sample	41mb sample	82mb sample	difference (8mb - 41mb)	difference (41mb - 82mb)	difference (8mb - 82mb)
PCA=3 logreg	0.52201	0.52662	0.52295	0.00461	-0.00367	0.0009
PCA=3 linSVC	0.48007	0.50146	0.48013	0.02139	-0.02133	0.0001
PCA=3 KNN	**0.57492**	**0.59307**	**0.59733**	0.01815	0.00426	**0.0224**
PCA=3 RF	**0.58436**	**0.59208**	**0.59762**	0.00772	0.00554	**0.0133**
PCA=3 XGB	0.5724	0.57364	0.57699	0.00124	0.00335	0.0046

Table 5: Model Performance -- 6 Principal Components						
	8mb sample	**41mb sample**	**82mb sample**	**difference (8mb - 41mb)**	**difference (41mb - 82mb)**	**difference (8mb - 82mb)**
PCA=6 logreg	0.51729	0.52715	0.52435	0.00986	-0.00280	0.0071
PCA=6 linSVC	0.51542	0.49173	0.47657	-0.02369	-0.01516	-0.0389
PCA=6 KNN	**0.57899**	**0.59101**	**0.59735**	0.01202	0.00634	**0.0184**
PCA=6 RF	0.56219	0.56130	0.56473	-0.00089	0.00343	0.0025
PCA=6 XGB	**0.57273**	**0.57992**	**0.58554**	0.00719	0.00562	**0.0128**

Due to strong performance exhibited, we proceeded to conduct gridsearch tuning for the hyperparameters of the KNN and RF models under three principal components and XGBoost with six components. While KNN and RF seemed to max-out under grid search at 0.576 and 0.587, respectively, our best performance came from our grid-tuned XGBoost, searched for ideal learning_rate, max_depth, min_child_weight, subsample, colsample_bytree, n_estimators, and objective. The gridsearch took over an hour to perform, but returned a model capable of stable 0.60+ accuracy, the best performance witnessed under any set of conditions and a modest improvement of approximately 10% relative to our dummy-classifier baseline.

Discussion and Conclusion

Overall, our results are not a great improvement to our original baseline classifiers. By using XGBClassifier with tuned hyperparameters and PCA, we achieved an accuracy of ~0.60. Although this might be a numerically non-impactful increase in accuracy, and fell short of our expectations, it reflects the importance of a physical presence of an inspector in the process and how impressive the city of Chicago's work in making this type of modeling successful. We believe one major differentiating factor likely could have been how much internally-available data was available to the officially-city-supported modeling work done in Chicago.

It's often said that more data beats better algorithms; in retrospect // given more time, we would try to utilize the location information, like zip codes, to integrate more datasets and ideally identify more strongly-predictive features tied to regionality; we believe this is the strongest improvement we could make moving forward. Given more time and computing power, we also

would have further investigated accuracy scaling under larger proportional samples of the data, as this effect emerged was noticeable between our existing sample sizes.

We also believe that the generation/inclusion of an inspection-based ID would better allow us to compare inspections as a container data-structure containing general inspection data as well as a list of violations occurred. The original data generates one record per no-violation case, and multiple inspection records for violations, one record per violation. This restructuring would have allowed us to more easily reduce structural redundancy within the data, duplicate-handling, and aggregation-by-restaurant inspection.

Last but not least, we believe that setting our random states could have supported better mutual reproducibility of our individually-observed results, and given more success at predicting the binary outcome of businesses as passing/failing inspections, we would have liked to have explored predicting types of violations and, mobilizing temporal data, likelihood of vendors conducting repeat violations.

References

- Data:
    - Boston Food Establishment Health Inspections
    - Boston Approved Building Permits
- Data Manipulation: Dask and Pandas
- Machine Learning: SciKit Learn and Tensorflow 2.0
- Visualization: MatPlotLib