The Butterfly: Relating NYC taxi trips with Dow Jones Industrial Average (DJIA) Progress Report

Julian Gao (julianyg)

November 17, 2016

1 Framework

This is a command-line run python project, with functionalities separated into two main parts: the learning phase and the prediction phase. The learning phase focuses on learning data. The learner will split the dataset into training and validation parts to evaluate models and features. The prediction phase focuses on generating labels for provided data, which is completely different from the notion of testing, since testing is already included in the learning phase as validation. Prediction phase only takes well-trained models.

An important optimization on data processing is that the framework allows reuse of extracted features. This optimization greatly reduces the extra time spent on data preprocessing and feature extracting, especially when using the same set of features with a different model. It also allows inputs of multiple data files, which are then combined into one single RDD for further processing. This enables the combination of small monthly datasets into a giant data pool, which can provide more samples for model training.

The selection of feature extractors and use of models can all be specified on command-line inputs. The trained model weights are stored in the file system, so that it can be loaded directly in the future without training again.

2 Data Pre-Processing

The NYC taxi data is provided as gigantic CSV files. Each CSV file contains data of a single month, with a size of 1.8 - 1.9 GB. The DJIA data is downloaded from Yahoo! Finance, which contains each day's open and close DJIA index. To achieve a reasonable processing time on those huge files, I choose to use Apache Pyspark for the project. The text files are read in and stored in RDD's (Resilient Distributed Dataset), and all operations on those RDD's are performed by MapReduce jobs, which greatly accelerates the process.

For the taxi data, pre-processing is focused on removing corrupted rows and entries. One important observation is that on certain dates, the average trip distances are anomalously higher than general dates (59.8 mi comparing to 16 mi), some are even negative numbers. Those are induced by the bad reading from odometer systems. Similar issues also occur on pick-up/drop-off GPS coordinates, where some entries are 0 or incorrect (different than NYC geography locations). By filtering out these bad entries, I get a clean processed piece of data.

For the DJIA data, the main task is to generate labels. This is easily done by comparing the open and close index value within a single day, and mark that results on the previous date (prediction is for the future). Another consideration is the magnitude of index change. For the effectiveness of training, I only select dates with index raise/drop more than 0.1%, such that the change is noticeable and worth using for training.

The pre-processed taxi data are fed into feature extractors for further processing. The extracted feature RDD is then joined with pre-processed DJIA data to generate labels data points, which can be then used for the learning process.

3 Feature Extraction

Feature extraction is the most crucial part of this project. The DJIA and taxi data seem to be two uncorrelated events, and many people have discouraged me on this project idea. However I deeply believe that the taxi dataset is a comprehensive record of people activity: if some great event influences the DJIA, that same event could have impacts on human activity to some extent as well. To utilize that nuance and construct a relatively strong correlation is a harsh task, and feature extraction is the key to build up this correlation.

There are two main types of information to select from: one is monetary, another one is geographic. For the former part, values such as fare amount, MTA tax, toll amount, etc. are useful; for the latter part, the traffic flow indicates the move of population within the city, and is a great indicator for events taking place. Currently I have implemented four types of light feature extractors, which only takes account of monetary information.

One major concern is that, since the prediction is based on one single day's data, all records within that single day must be combined in certain way that can effectively represent that day's data, and avoids any loss of useful information. There are multiple ways to do this, the two obvious ones are summation and averaging; however those may not be representative, and for different types of features there exist different methods. Below are a list of feature extractors I am planning to try on, some of them (1 through 4) already implemented and generated results (results will be shown in section 6). Some are designed for SVM/logistic regression classifiers, and some can be more generally applied to other type of classifiers.

1. Simple aggregating feature extractor.

   This feature extractor only extracts four features: trip distance, passenger count, trip duration, and total fare amount. Those values are summed up for each single day to generate one data point. This is not a useful feature extractor by apparent reason.

2. Simple averaging feature extractor.

   This feature extractor extracts the same four features as above, however it divides those numbers by the total number of rides within each single day. This feature extractor performs slightly better than the one above.

3. General averaging feature extractor.

   This feature extractor extracts all columns except GPS coordinates and indicators, and average over each single day. Currently this is the best performance feature extractor.

4. Baseline averaging feature extractor.

   The baseline feature extractor. Simply sums up every column for each day, and takes average (only for non-indicator values; this excludes the payment type column). This feature extractor performs relatively bad, due to the fact that the GPS coordinate entries, if simply averaged, are detrimental to the learning process.

5. Simple grid feature extractor

   The simple grid feature extractor only makes use of GPS coordinates and passenger count data, to see if geographic data alone is correlated with the DJIA. The idea is to gridify NYC, and compute top k busiest pick-up/drop-off grids. It is likely to perform slightly better than the averaging extractors above, but still, this depends on the correlation between DJIA and population flow in NYC.

6. Comprehensive grid feature extractor

   The comprehensive grid feature extractor will gridify the NYC and compute pick-up/drop-off rates for every single grid. This will generate a huge matrix, similar to an “image". Those features can be fed into CNN's for training, or even RCNN since the data is a time series.

7. High-dimension grid feature extractor

   This will the most comprehensive feature extractor ever written for this project. Just as the one described above, it will generates a grid map comparable to an image, but instead of RGB-d 4 layers images, the blob generated by this feature extractor will have $24n$ layers that includes n features for every hour within that single day. This is likely to be effective CNN training, but also greatly suffers from overfitting since training dataset is too small. Probably requires dimensionality reduction.

4 Model Selection

The project is defined as a simple classification problem. Currently I have tried an SVM trainer with SGD, for the linear features extractors. There are many other models to select from:

1. Logistic regression.

   The most common model to try on for classification problems. Just a raw estimation for the entire algorithm and model, because the data is most likely linearly separable, and this model will perform badly.

2. SVM.

   Support vector machine is an intuitive choice for such binary classification problems. The linear feature vectors are very useful for SVM training; however such vectors have close feature values, and the average of millions of record within a single day may lose too much information. Requires implementation of kernelized SVM.

3. Random forests.

   Another great model for classification. Combines many decision trees to reduce risk of overfitting. It does not require feature scaling, and is able to capture non-linearities.

4. Convolutional Neural Network

   CNN is the state-of-art method for many machine learning tasks. The idea that the geographic information can be extracted and encoded in forms similar to that images is the main motivation for selecting this model. It can detect some high dimensional patterns from the geographic image, however only simple networks with less parameters work here, since there are not enough data to populate the entire parameter space. Note that Recurrence CNN also worths try, because the taxi data is a time series.

However currently I am still concentrated on feature selection, using SVM. It is important to fix on one model first to select the best features, and then try out different models for the selected feature.

5 Preliminary Results

I have run the feature extraction on data from Oct, 2012 to June, 2016. The data that fits training requirements (fluctuating around 0.1%) are 164 points, and are split into 70% and 30% for training and validation. This is a small size dataset, just to confirm the feasibility of training, and it already takes over 90 minutes to generate. The running on linear SVM have generated following results on non-geographic feature vectors, with parameter tuned for the best performance, correspondingly:

Features	Training Error	Validation Error
Simple Aggregation	0.5321	0.6182
Simple Averaging	0.3872	0.4219
General Averaging	0.4833	0.3182
Baseline	0.4821	0.3402

Note that this is a tentative try with little amount of non-geographic data on linear classifiers, so these results are not as bad as they seem. This table suggests that the features from general averaging feature extractor is the best to combine with geographic features later. The less than 50% errors also suggest that NYC taxi data is indeed correlated with DJIA.

6 Next Steps

I am planning to use the general averaging feature extractor to extract features from Jan 2009 - June 2016, in order to further confirm the correlation with more data points. At the mean while I will implement the grid-related feature extractors to see the influence of geographic features on prediction. I will also use the Spark tool to generate high-resolution, multi-layer data and use some RCNN framework to see prediction performance.