Capstone Project Proposal
Niko Laskaris
September 24, 2018

DEFINITION

DOMAIN BACKGROUND

Price prediction is a category of regression problem with applications in a broad array of industries and academic disciplines. Airlines use complex pricing algorithms to price airline tickets, investors increasingly deploy machine learning-based algorithms to predict stock and asset prices and inform investment decisions. The problem is, at root, a regression problem, but with complex domains comes the need for complex approaches to building predictive models therein.

One such domain is predicting taxi fares. One might extrapolate this problem to consider pricing algorithms used by ride-sharing companies such as Uber and Lyft to price rides. Much of the existing scholarship in this domain considers both fare and travel duration prediction , and builds off of work done in the broader category of price prediction problems [1][2][3].

PROBLEM STATEMENT

The primary goal of this project is to predict the fare price of taxi rides in New York City. This is a task taken from the list of Kaggle open competitions. The loss function for measuring model accuracy will be RMSE (root mean squared error). As the problem of price prediction is a regression problem, many regression techniques can be applied: linear regression, Lasso, Ridge, SVRs, Boosting and other ensemble methods, and more.

DATASETS AND INPUTS

The data for this problem comes from Kaggle, and includes samples of data from 2007 to 2012.

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610	40.712278	1
1	2010-01-05 16:52:16.0000002	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.782004	1
2	2011-08-18 00:35:00.00000049	5.7	2011-08-18 00:35:00 UTC	-73.982738	40.761270	-73.991242	40.750562	2
3	2012-04-21 04:30:42.0000001	7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	-73.991567	40.758092	1
4	2010-03-09 07:51:00.000000135	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	40.783762	1

Each sample contains pickup_datetime, pickup_longitude and latitude, dropoff_longitude and latitude, passenger_count and fare_amount features.

The training dataset contains ~55 million samples, and the testing dataset about 10 thousand. Validation data can and will be stripped from the training dataset. While the feature space is relatively sparse, some clear feature engineering opportunities are to create distance and, perhaps, neighborhood features from the latitude/longitude features, as well as to split the datetime feature into discrete date time fields for time of day, week, month, year, etc.

Background research into the pricing for the NYC metropolitan taxi system will additionally aid in engineering our features, and will likely generate features to/from counties, airports, and other special pricing situations.

SOLUTION STATEMENT

Once the feature space has been established, various regression algorithms will be tested on a subset of the training data to determine which are the best candidates for model selection and tuning. Some such models include Linear Regressors, ElasticNet, Ridge Regressors, Gradient Boosting Regressors, and AdaBoost Regressors. The selected model will undergo hyperparameter tuning and regularization to improve performance.

BENCHMARK MODEL

The official documentation for the Kaggle competition says that a baseline model using just the distance between pickup and dropoff locations will get a RMSE of ~$5-8. Here I use a simple Linear Regression on the cleaned and feature engineered data and achieved a RMSE of ~$5 ($4.98).

EVALUATION METRICS

As our data is labeled we can use a straightforward metric to measure the accuracy of our model. In this case the recommended loss function, and the one I will use, is RMSE.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Where y_i is the target and $\hat{y}_i$ is the prediction. A successful model will produce a RMSE lower than our benchmark model (lower than ~$5).

RMSE has some desirable properties, when compared with MAE (Mean Absolute Error), for example.

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

Since with RMSE, errors are squared before they are averaged, RMSE gives a relatively high weight to large errors compared with MAE. This makes RMSE useful when large errors are particularly undesirable. A knock-on effect of this is that RMSE allows for faster gradient convergence. In a problem like taxi fare regression, we may want to penalize outlier samples severely. As such RMSE is a good choice for our loss function.

PROJECT DESIGN

Project Pipeline:

1. Data exploration and cleaning
2. Data and feature engineering
3. Model selection
4. Model tuning
5. Training and testing

[1] *Data exploration and cleaning*. Training and testing datasets will be explored and cleaned of outliers, missing data, etc.

[2] *Data and feature engineering*. Here a variety of features will be created and explored for relevance to predicting fare. Among these will be: distance between pickup and dropoff, various datetime features, neighborhood and location features relative to specific New York taxi pricing codes, fixed rate taxi rides to airports, and more. These features will be tested and either kept or removed from our data. Once we feel satisfied with our feature exploration, the newly cleaned and engineered training and testing datasets will be saved.

[3] *Model selection*. Various regressors will be trained on a subset of our training data to note which are the best candidates for model selection and tuning. Among these are linear regressors, adaboost regressors, gradient boosting regressors, Ridge, Lasso and ElasticNet regressors. Training time will be considered but is not crucial to this task.

[4] *Model tuning*. Once a model is selected from step [3], that model will undergo hyperparameter tuning to optimize its performance, measured by RMSE. Various performance optimizers will be applied to the selected model to optimize performance on the test data.

[5] *Training and testing*. Tuned model will be trained and then used to predict fare on the test data.

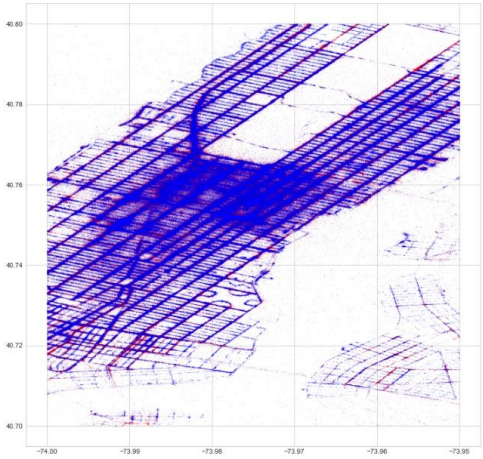
ANALYSIS

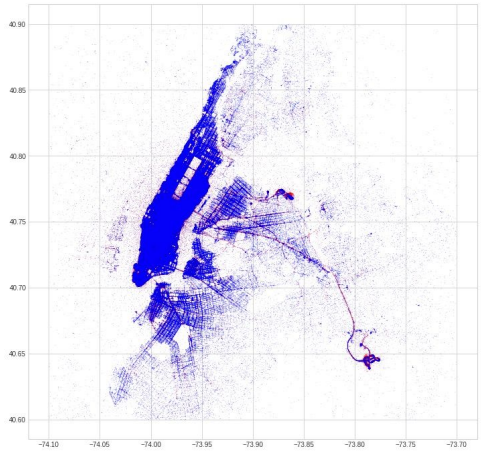
DATA EXPLORATION

Began with 15M samples from the ~55M sample training dataset. Checked for null and zero values and removed them. Used the parameters of the testing dataset to crop the training dataset for outliers. Split the datetime feature into discrete time features for hour, day, month, year, etc. All told data cleaning resulted in a dataset with 14,628,013 samples of 13 features.

The data was fairly clean and easy to work with. I used a subset of the full training simply due to the training set's size. A note for further work on this project would be to open an EC2 instance or otherwise get ahold of more computing power to train with the full dataset (although results were fairly good using a subset).

VISUALIZATIONS





We can tell several things from our visualizations of the data. The cab rides are distributed around New York City but concentrated in Manhattan. A fair number of pick up / drop offs in Brooklyn and Queens, and few into / from the Bronx. There do seem to be a high density of trips that start/end at the three major New York Airports (JFK, LaGuardia, Newark), so we might want to look at that in more detail later on.

ALGORITHMS and TECHNIQUES

To determine which algorithms to try, it's helpful to first determine several heuristics about our data:

1. Regression
2. Large dataset (>50M samples)
3. Should try models that work well with few and many features (as we'll be engineering new features from our data)

Below is a list of the algorithms I've selected to try on this problem, a summary of how they work and why they are sensible choices given this domain.

[1] Lasso
[2] Ridge Regressor
[3] Elastic Net
[4] AdaBoost Regressor
[5] GradientBoost Regressor

Regularization Methods: Ridge, Lasso and Elastic Nets

Lasso, Ridge, and ElasticNet Regression techniques are all *Shrinkage Methods*, or methods by which a subset of the predictors (features) of a given dataset are retained for prediction (and shrunk), while the rest are discarded. Shrinkage methods can have the beneficial quality of producing a model that in interpretable and has lower prediction error than a full model.

All three models work by shrinking parameters to prevent multicollinearity, the condition of a predictor variable being linearly predictable from the others with a substantial degree of accuracy. We care less about the possibility of multicollinearity (although it's worth exploring) and more about the various ways in which each of these algorithms regularize the data. Ridge Regressors use L2 regularization, Lasso Regressors use L1 regularization,

and Elastic Net Regressors use a combination of L1 and L2 regularization. L1 regularization is the minimization of the absolute differences (S) between the target value (y_i) and the estimated values (f(x_i)):

$$S = \sum_{i=1}^{n} |y_i - f(x_i)|.$$

L2 regularization is the minimization the sum of the squares of the differences between the target and estimated values:

$$S = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

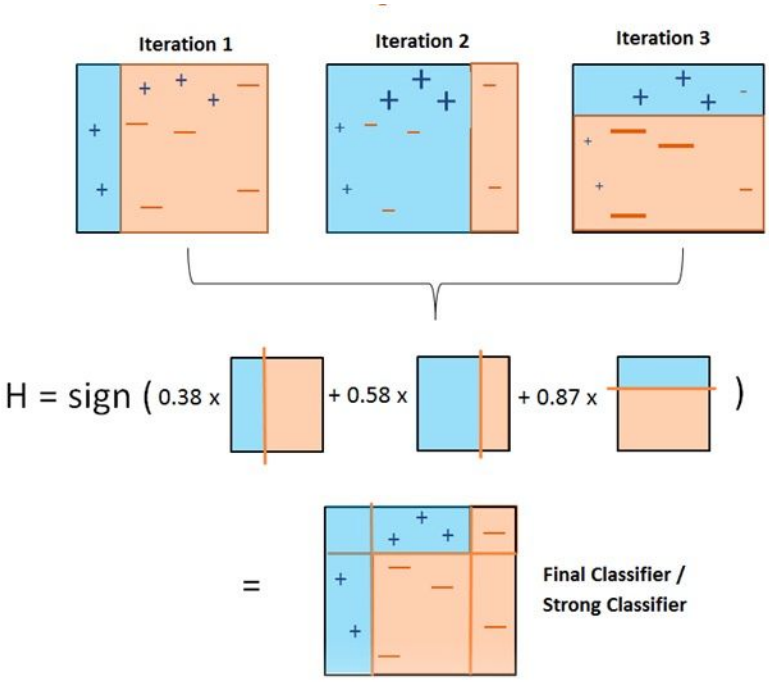
Because L2 regularization squares error, models with L2 regularization are particularly sensitive to large errors than models with L1 regularization.

Ridge regressors shrink the regression coefficients so that variables with minor contribution to the outcome have coefficients close to zero. The shrinkage of coefficients is achieved by penalizing the model with L2 regularization, the sum of squared coefficients. Ridge regressors are highly affected by the scale of the predictors. Another disadvantage is that ridge regression will include all features in the final model. An alternative to this method is **Lasso** regression (least absolute shrinkage and selection operator), which was originally introduced to improve the prediction accuracy and interpretability of regression models by altering the model fitting process to select only a subset of the provided covariates for use in the final model rather than using all of them. Lasso, as mentioned above, uses the L1-norm for the regularization term. Lasso regularization has the effect of having some features with minor contributions to the model prediction to be exactly equal to zero. An advantage of Lasso relative to Ridge is that it produces simpler and more interpretable models. Generally, Lasso can perform better when some predictors have large coefficients and the remaining predictors have small coefficients. If all of the predictors are roughly on the same order of magnitude, Ridge may be a better option.

Elastic Net is, effectively, a combination of Lasso and Ridge methods, using both the L1 and L2 norm, and as such both shrinking the coefficients of features as well as reducing some of them to exactly zero (and thereby effectively removing them from the model).

Ensemble Methods: AdaBoost and GradientBoost Regressors

Ensemble methods use many weak learners (models), in combination, to create a strong learner. The general principle is to set and update weights in a way that forces regressors to concentrate on observations that are difficult to classify correctly, or have been misclassified by previous models. (*Note: this is a description of class of learning algorithms called boosting methods, a class to which both AdaBoost and GradientBoost are both belong. Bagging methods, by contrast, have each model in the ensemble weight equally*.) The beauty of ensemble methods is that, so long as the performance of each weak learner is slightly better than random guessing, the ensemble can deliver predictions that are quite accurate. Disadvantages of boosting methods are that they can be computationally expensive and, in certain cases, can be prone to overfitting (high variance). The 'weak learners' used in AdaBoost and GradientBoost are usually decision trees.



Visualization of boosting classification, not regression, but a good way to intuit what's going on with these boosting ensemble methods.

The difference between AdaBoost and Gradient boost is in **how they create the weak learners** during the iterative process. AdaBoost updates by changing the feature weights at each iteration (increasing the weights of the wrongly predicted instances and decreasing the weights of the correctly predicted instances). GradientBoost doesn't modify the sample distribution: instead the new weak learner trains on the remaining errors of the previous learner alone. Then the performance of the new weak learner compared with the previous ones isn't computed according to performance on the new sample but by using a gradient descent optimization process.

****

These algorithms are all useful for regression problems with need for regularization. To varying degrees, they can handle highly dimensional feature spaces and outliers. In any event for a regression problem such as this, they will be a good starting point.

METHODOLOGY

DATA PREPROCESSING: Feature Engineering and Exploration

Notes on pricing:

From the NYC taxi and limousine commission:
http://www.nyc.gov/html/tlc/html/passenger/taxicab_rate.shtml

The initial charge is $2.50. Plus 50 cents per 1/5 mile or 50 cents per 60 seconds in slow traffic or when the vehicle is stopped. In moving traffic on Manhattan streets, the meter should "click" approximately every four downtown blocks, or one block going cross-town (East-West). There is a 50-cent MTA State Surcharge for all trips that end in New York City or Nassau, Suffolk, Westchester, Rockland, Dutchess, Orange or Putnam Counties. There is a 30-cent Improvement Surcharge. There is a daily 50-cent surcharge from 8pm to 6am.

Since the initial charge for every taxi ride in New York is 2.50 since 2012 (and 2.00 since earlier than our dataset starts recording), let's see how many samples have a fare at or below the $2.00 boundary.From the NYC taxi and limousine commission:
http://www.nyc.gov/html/tlc/html/passenger/taxicab_rate.shtml

The initial charge is $2.50. Plus 50 cents per 1/5 mile or 50 cents per 60 seconds in slow traffic or when the vehicle is stopped. In moving traffic on Manhattan streets, the meter

should “click” approximately every four downtown blocks, or one block going cross-town (East-West). There is a 50-cent MTA State Surcharge for all trips that end in New York City or Nassau, Suffolk, Westchester, Rockland, Dutchess, Orange or Putnam Counties. There is a 30-cent Improvement Surcharge. There is a daily 50-cent surcharge from 8pm to 6am.

Since the initial charge for every taxi ride in New York is 2.50 since 2012 (and 2.00 since earlier than our dataset starts recording), let's see how many samples have a fare at or below the $2.00 boundary.

****

I added Distance feature using Haversine distance. Created dropoff and pickup features for each of the counties mentioned above that have unique pricing. Log-transformed distance. Checked correlation with fare amount and eliminated a few of the newly generated features.

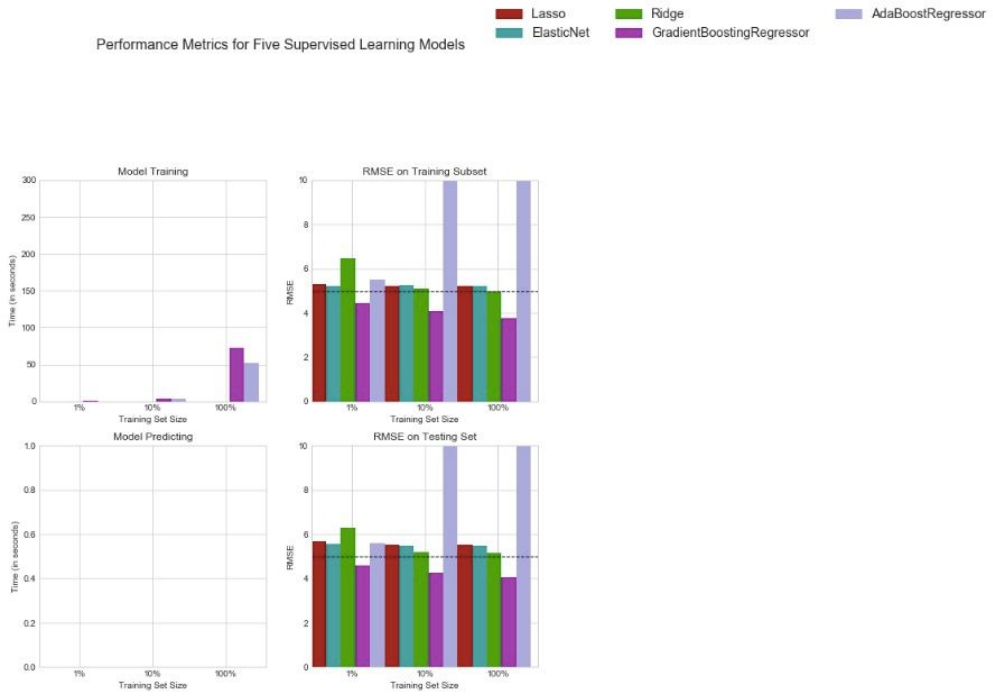
MODEL SELECTION

I used a simple linear regression as a baseline model, and got a RMSE of 4.95 on the testing (validation) data.

I borrowed visuals and some code from the Udacity Finding Donors project from earlier in the program to test several regressors on a subset of the data. Tested:

- Lasso
- ElasticNet
- Ridge
- GradientBoost
- AdaBoost

I tested these models on subsets of the training data (1%, 10%, and 100%), to get a sense for how each model performance improved as it was fed more data. 1, 10, and 100% benchmarks were, ultimately, arbitrary, but borrowed from one of the previous projects and seemed to provide good intuition as the model improvement as a function of more data.



Of these GradientBoost, while the most computationally expensive, performed best.

I decided to go ahead with GradientBoost as my model. On that same subset of the data, I performed a Permutation Importance test to double check that my features were all useful. I tried making an arbitrary cutoff based on permutation importance test scores to eliminate some of my features, but the truncated model actually performed worse on the subset of training data. I decided to keep my original set of features. *Note: This could be another place for future improvements.*

Weight	Feature
0.3477 ± 0.0013	log_distance
0.3240 ± 0.0011	distance
0.0810 ± 0.0006	dropoff_longitude
0.0421 ± 0.0017	pickup_year
0.0234 ± 0.0009	dropoff_latitude
0.0072 ± 0.0011	night_hour
0.0046 ± 0.0002	dropoff_distance_to_jfk
0.0035 ± 0.0002	pickup_distance_to_ewr
0.0034 ± 0.0002	dropoff_distance_to_Nassau
0.0028 ± 0.0001	pickup_distance_to_jfk
0.0026 ± 0.0003	pickup_distance_to_Nassau
0.0023 ± 0.0003	pickup_distance_to_Rockland
0.0017 ± 0.0001	pickup_longitude
0.0012 ± 0.0001	dropoff_distance_to_Westchester
0.0012 ± 0.0000	pickup_distance_to_center
0.0011 ± 0.0001	dropoff_distance_to_Suffolk
0.0010 ± 0.0001	dropoff_distance_to_Orange
0.0006 ± 0.0001	pickup_distance_to_Suffolk
0.0006 ± 0.0001	pickup_distance_to_Orange
0.0006 ± 0.0000	pickup_distance_to_Westchester
… 12 more …	

Once I had determined my model (GradientBoostRegressor) and had decided to keep my original feature space after conducting a permutation importance test, I performed a Grid analysis to tune the hyperparameters of my model (using 1M samples). For this model the grid search was performed to tune the hyperparameters n_estimators (the number of trees — weak learners) to be modeled, max_depth (how many levels our decision trees are allowed to reach), learning_rate (how quickly the

weights are updated at each iteration), and min_samples_split (the minimum number of datapoints a node must have before it splits into new nodes. After gridsearch the optimal values for each of these hyperparameters were:

- N_estimators: 100
- Max_depth: 5
- Learning_rate: 0.1
- Min_samples_split: 3

RESULTS

MODEL EVALUATION and VALIDATION

I introduced a similar version of GradientBoost, Light Gradient Boost (LGBM). LGBM was delivering slightly lower RMSE scores than the original model, so I proceeded using LGBM. I trained first 3M and then 15M samples with the LGBM model (hyperparameter tuned according to my Grid Search). My best RMSE score once submitted to Kaggle was ~3.066, good for the top 25% of the competition.

Is the final model reasonable and aligning with solution expectations? Are the final parameters of the model appropriate?

The benchmark model we trained — a simple Linear Regressor — produced a RMSE of ~$5. As our LGBM, tuned model has produced a RMSE of ~$3, the results of the model are reasonable and in line with solution expectations. The final hyperparameters of our model — 100 n_estimators, max_depth of 5, learning_rate of 0.1 and min_samples_split of 3 — seem appropriate.

Cross validation: In order to confirm the robustness of my model to new data, I performed a k-fold cross validation analysis:

Fold #	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
RMSE	3.7149	3.7129	3.7291	3.7178	3.7162

The results of a 5-fold cross-validation test suggest that the model is fairly robust to knew, unseen data / new cross-sections of data.

JUSTIFICATION

The LightGBM model trained on ~15M samples returned an RMSE of ~3.06. Our benchmark model returned an RMSE of ~5, so our trained model has significantly improved upon the benchmark. The k-fold cross-validation testing confirms the relative robustness of this model, as RMSE scores across 5 folds deviated little. In the

Is the final solution significant enough to have solved the problem?

This problem has not been 'solved', per se, but a better solution has been arrived at a far lower RMSE than our benchmark model, good enough for placing in the top ~20-25% of the Kaggle competition.

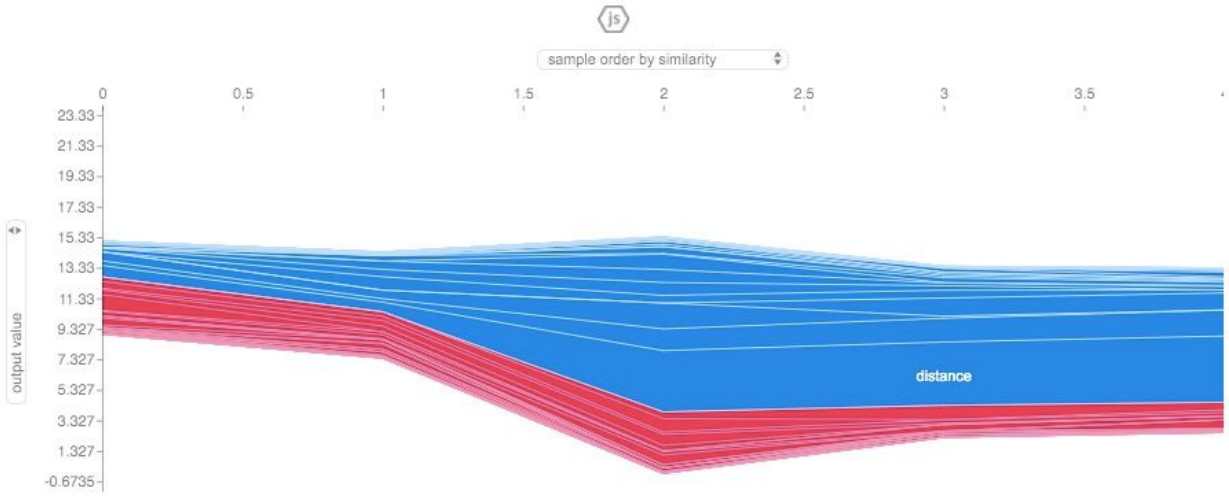
CONCLUSION

FREE-FORM VISUALIZATION

The visualization above of the results of a Permutation Importance test is interesting and shows the relative predictive power of each of the features in the dataset, determined by randomly rearranging each feature value set and determining how significantly such a rearrangement would impact the predictions.

Another visualization that's worth exploring is a representation of Shap (SHapley Additive exPlanations) values break down a prediction to show how each feature impacted the returned value. In the displays below, the features are broken up based on how much they contribute to pushing the predicted value of our model either up or down, based on distance from the 'baseline' value, which in this case is the expected value, or mean, of the target variable in the test set.

Above: Shap value graphic for one sample.



Above: Shap value graphic for five samples.

This visualization provides similar intuitions to the permutation importance visual shown above. The additional information of directionality (up or down) is also useful. We see in these graphics the strong impact of features distance, night_hour, dropoff_latitude, and pickup_distance_to_Dutchess on the prediction. The last of these, pickup_distance_to_Dutchess, does not show up in our permutation importance graphic. This makes sense. Our first Shap value graphic concerns one discrete data point, whereas the permutation importance test averages across the entire dataset, so likely the importance of pickup_distance_to_Dutchess is unique to the one datapoint analysed here.

REFLECTION

I found this a fun, at times challenging, and rewarding project. The project report and summary does a comprehensive job of explaining my process. There were, admittedly, many explorations into different features, correlations, etc., that proved not to be fruitful

and thus were not included. (I went deep into datetime features, for example. None of them proved to matter too much)

Some enjoyable / new takeaways from this project:

- Felt like a crash course in feature engineering, when given a dataset with features that contain a lot of information that need to nevertheless be engineered to create features that reflect that data. In this case, distance and datetime features and specific county/airport features tied to taxi pricing in New York.
- Learning techniques for handling large datasets. Lost probably 1-2 full days of time waiting for code to run, only for errors to throw after ~30 minutes of sitting. Got more precise at ensuring code was airtight before running, taking subsets of data and running code on smaller datasets before using the full one.
- Fun exploring visualizations. Learned about permutation importance and SHap values. Had also simultaneously signed up for a Kaggle 4-day challenge that included these two topics, which both came in handy.

Some challenges:

- All of the large dataset learnings mentioned in the section above were incredibly frustrating. Lots of wasted time.

All in all my final model and solution line up rather well with what I expected I'd be able to accomplish. With more time I believe I'd be able to get an even lower RMSE score.

IMPROVEMENT

Some thoughts on avenues to pursue for future work on this project / model improvements:

- Continue to tweak feature space. We wound up with a lot of features for each county drop off or pick up that had unique pricing. Perhaps principal component analysis could reduce the featurespace based on explained variance.
- Could train with more data. I trained with 15M samples, but the dataset was ~55M.
- Continue to tweak hyperparameters. Even though I performed several Grid searches to optimize certain hyperparameters, could do with more.

- Explore even more models. Dip into neural nets, perhaps.

References

[1] Antoniades, Christophoros, Fadavi Delara, Foba Amon Jr., Antoine. Fare and Duration Prediction: A Study of New York City Taxi Rides (2016).

[2] Hegazy, Osman & Soliman, Omar S. & Abdul Salam, Mustafa. (2013). A Machine Learning Model for Stock Market Prediction. International Journal of Computer Science and Telecommunications. 4. 17-23.

[3] Vanajakshi, L., S. C. Subramanian, and R. Sivanandan. "Travel time prediction under heterogeneous traffic conditions using global positioning system data from buses." IET intelligent transport systems 3.1 (2009): 1-9.