Predicting Trade Execution Time

Brian Hambre

What if I told you I could predict how long a specific trade would take to execute on the open market?

The problem

Timing is everything when trading securities on the open market and prices can quickly change in seconds. This is why it would be important to be able to predict how fast a trade can be electronically executed through our Trade Order Management System Moxy.

Trade execution time can be defined as the time from when the order is placed into the open market to the time the order is completely filled by a broker. This is all done over the internet using an electronic trading protocol called FIX.

http://www.fixtradingcommunity.org/

Solution

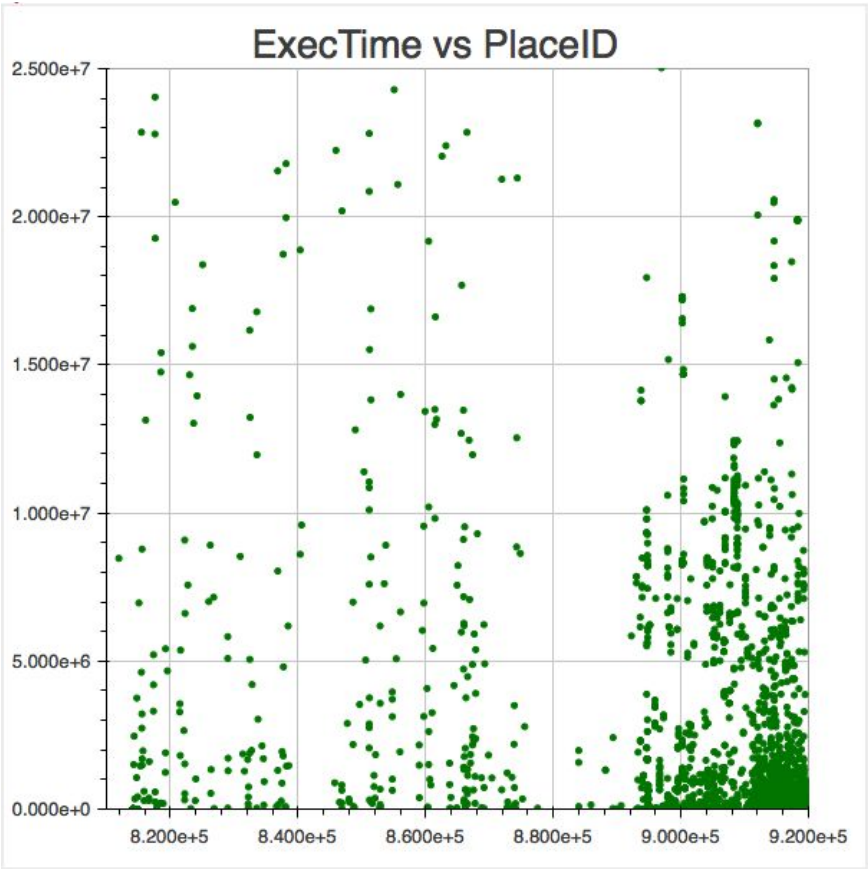
With the use of Data Science, we can use certain features of a trade to predict execution times and determine how specific features of the trade affect execution time.

The Data

A CSV file which contains 6,527 orders with each order having 17 specific trade features

Data Analysis - Execution Time

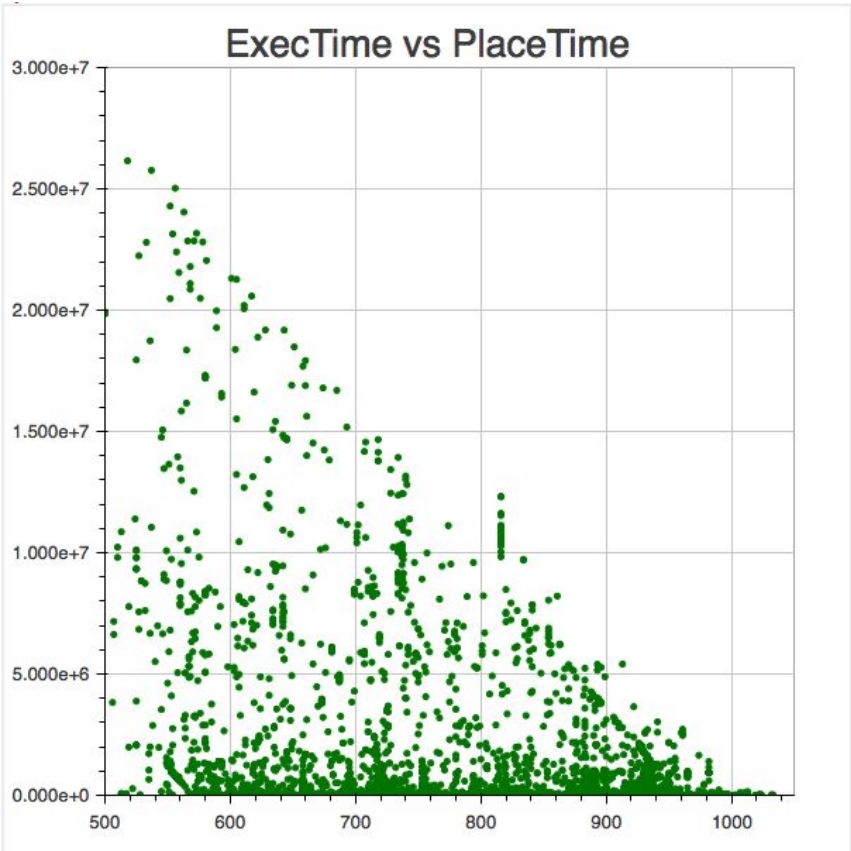
A majority of the execution times fall between 0 - 5,000,000 milliseconds (0 - 83.3 minutes)



Data Analysis - Place Time

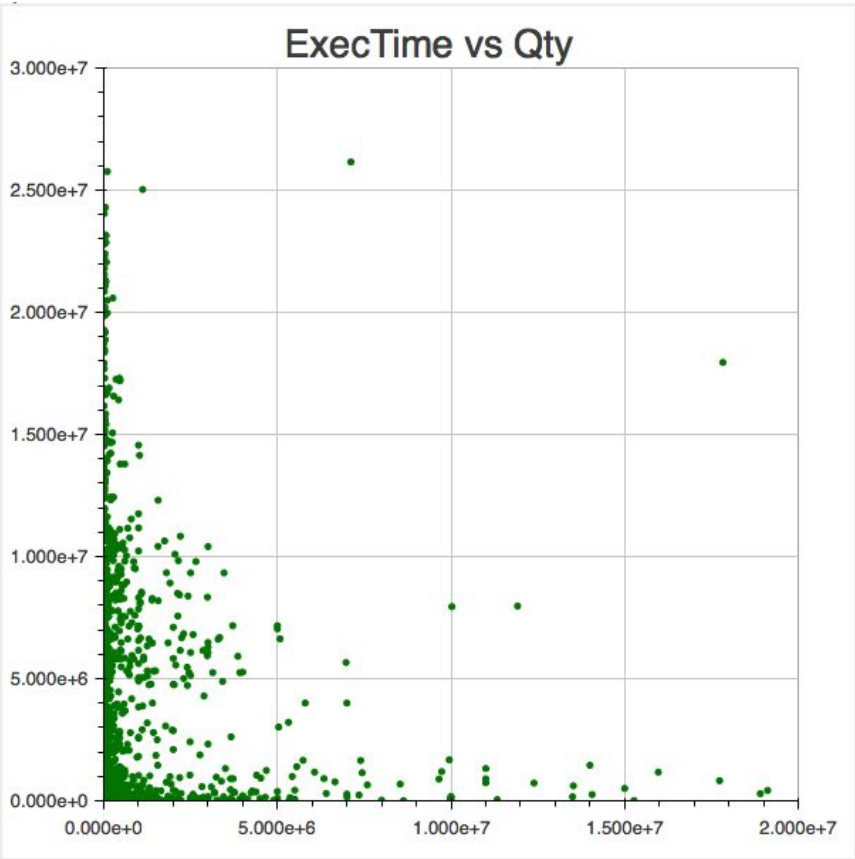
Orders placed earlier in the day are more likely to take longer

500/60 = 8:20AM
1000/60 = 4:40PM



Data Analysis - Share Quantity

A larger share quantity order does not necessarily mean a larger execution time



Data Analysis - Security Type

Avg Execution Time for US Common Stock: 544,500 milliseconds (9.06 minutes)

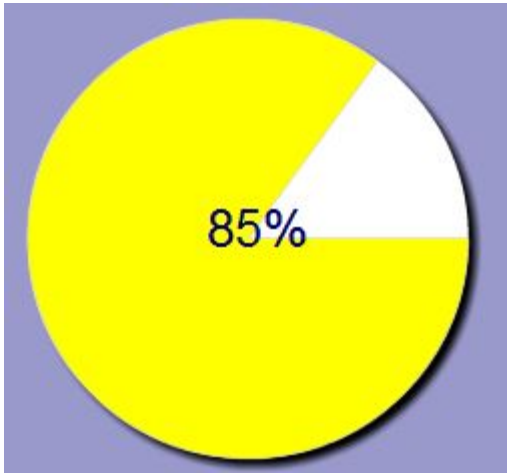
Avg Execution Time for US Corporate Bond: 2,755,214 milliseconds (45.92 minutes)

Avg Execution Time for Japan Foreign Stock: 164,561,356.5 (2,742 minutes or 45.7 hours)

Trade Features

- Place Time
- Transaction Code
- Share Quantity
- Broker
- Security
- Security Type
- Exchange
- Limit Type
- Algorithm Trade

Split the Data to 85% Training Data and 15% Test Data



Training Data: 5,547 Orders will be used to create the model
Test Data: 980 Orders to test and validate the model

Regression Models Tested

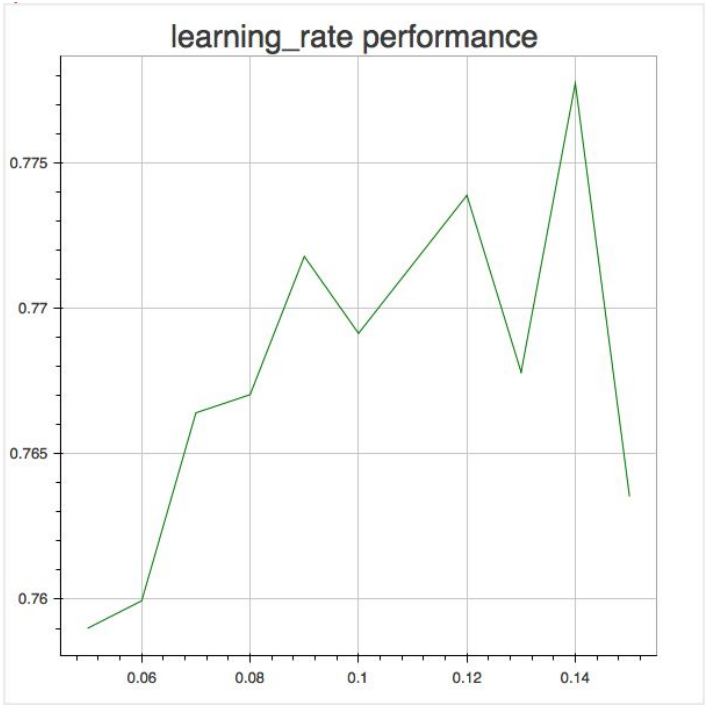
- KNN Regression (K=4)
- Linear Regression
- Ridge Regression (α=1,1st degree polynomial)
- Lasso Regression (α=1, 3rd degree polynomial)
- Support Vector Regression (C=1, epsilon=0.1)
- Gradient Boosting Regression (default)
- Random Forest Regression (default)

Cross Validation Model Scores (CV=6)

- KNN Regression: .5182
- Linear Regression: -3.5e+27
- Ridge Regression: .3968
- Lasso Regression: .4140
- Support Vector Regression: -.0573
- Gradient Boosting Regression: .7650
- Random Forest Regression: .7354

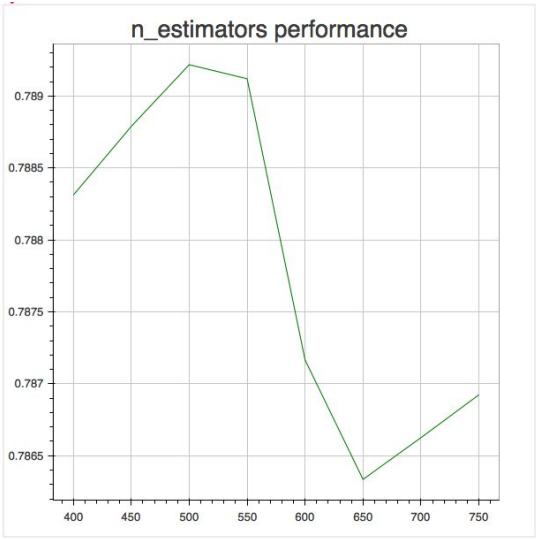
Both Gradient Boosting and Random Forest had the highest score

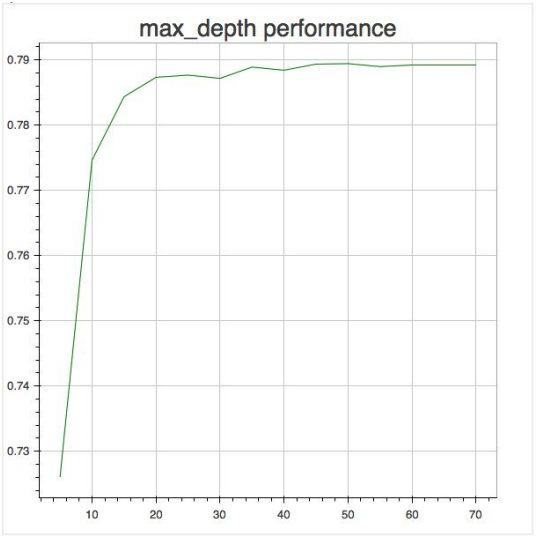
Model Tuning - Gradient Boosting

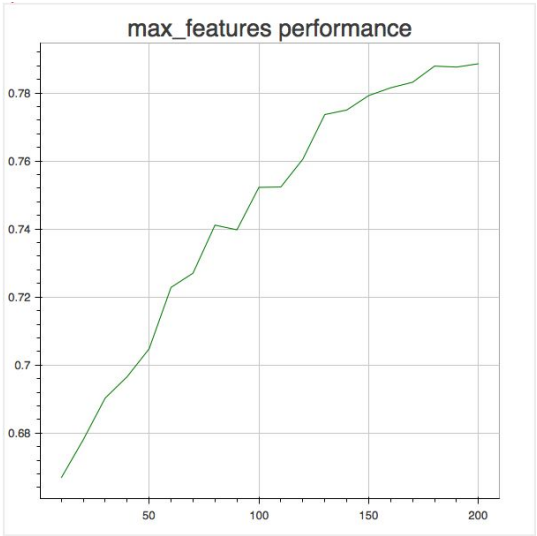


learning_rate = .14

Model Tuning - Random Forest







n_estimators=500, max_depth=50, max_features=210

Tuned Gradient Boosting vs Tuned Random Forest

```
x = X_train.values
y = y_train.values

from sklearn.ensemble import GradientBoostingRegressor
est = GradientBoostingRegressor(learning_rate=0.14,random_state=0)
est.fit(x, y)
print cross_val_score(est,x,y,cv=6).mean()
```

```
0.777729955236
```

```
#Decission Forrest Regression
x = X_train.values
y = y_train.values
# import class, instantiate estimator, fit with training set
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(max_features=210,max_depth=50,n_estimators=500,random_state=0)
rf.fit(x, y)
print cross_val_score(rf,x,y,cv=6).mean()
```

```
0.789516951342
```

The Random Forest Model performs the best with a score of .7895

Applying the Model to the Test Data

Actual Time (miliseconds)	Predicted Time (miliseconds)	Time Difference (miliseconds)
4540	4276.358	263.642
4110	3820.351	289.649
2430	2833.162	403.162
261666	263552.704	1886.704
2486	3210.684	724.684
8093	7401.254	691.746
17990	908107.628	890117.628
8426	8017.125	408.875
2693	2942.438533	249.438533
56736	86931.914	30195.914
717440	1565555.152	848115.152
23746	101441.142	77695.142
36426	36103.69	322.31
2133	6534.86	4401.86
910	2515.312	1605.312

Feature Importance

From our feature importance results, we could see that the Japanese foreign stock ‘fsjp’ plays a very important role when determining the timing of a trade. This corresponds to our initial findings when analyzing the average execution times for specific security types.

```
features_imp = rf.feature_importances_
features_imp = pd.DataFrame(features_imp,index=X_train.columns)
features_imp.sort_values(0)[::-1].head(10)
```

	0
fsjp	0.236094
10	0.217669
10000003	0.162308
placeminutes	0.141131
PlaceQty	0.109528
SecKey	0.030659
TranCode	0.021632
limit	0.010240
fsgb	0.007062
mbus	0.006794

Conclusion

The model I created exceeded my expectations. From my results, the model was able to predict times very close to the true execution time.

There are some predicted execution times which are off from the actual execution time, but as there are a number of outside factors that could affect the trade I would account those differences due to outside variables.

As next steps, I would try to bring in up to date outside variables such as ratings, trading volumes, indexes, and more to see if that could improve the model.