Predicting the Daily Liquidity of Corporate Bonds

Louis Ben Arous
December 14, 2012

Introduction

Let us define the liquidity of a financial asset as its volume, or the number of transactions of that asset during a particular time window. Due to the lack of a major electronic market for fixed income instruments, many liquidity issues may arise when trading corporate bonds, where one security might become highly illiquid and create huge price impacts for investors, as they find themselves unable to close their positions.
This research will set up, test and compare statistical predictors from different fields (time series analysis and machine learning) to accurately forecast the daily volume of US investment grade corporate bonds. The goal is to find a global machine learning algorithm that outperforms univariate time series models out-of-sample.

Data

To implement our study, we need daily data for investment grade US corporate bonds. The question that arises is which bonds should we look at to create an appropriate predictor that could be applied to other bonds. The approach taken in this study is to look at the composition of the Barclays US aggregate index (AGG) , and to pick the top 28 corporate bonds that have the highest weight in this index. The AGG is the most commonly followed index for US fixed income products, which justifies its use to pick the most relevant bonds.
I have grabbed and merged the volume and price data for each of those bonds from Bloomberg into one dataset for a time window from November 15, 2005 to November 15, 2012. Since some of those assets were created well after 2005, special care must be taken to handle gaps and missing values in the data when building our predictors. I normalize my data such that the liquidity values for each bond have mean 0 and variance 1.

Methods

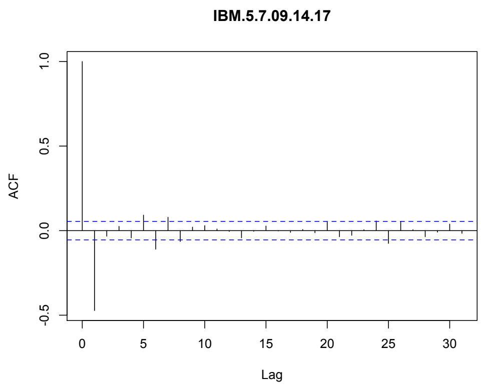
I use as measure of performance the average absolute error (AAE) to compare models. The AAE of a given model is defined as:

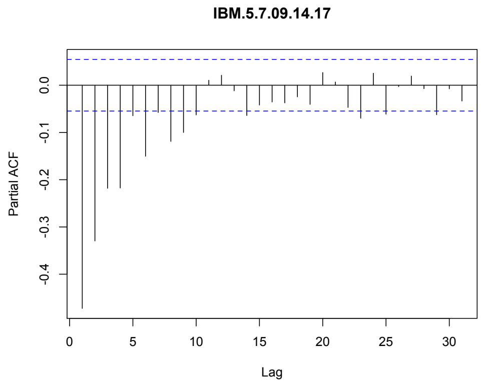
$$AAE = \frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i|$$

where $\hat{y}_i$ is the prediction of the model for the i^{th} observation. The goal of this study is to find a machine learning learning algorithm with a lower AAE than the time series AAE.
As mentioned earlier, the first part of this study is to find the best possible univariate time series model, and to fit this model to each one of the 28 bonds in my dataset. As a result, I will have 28 sets of parameters (one for each bond), and I will consider my general AAE for the time series case as the average of the AAE of each model.
Then I will fit different machine learning models to the complete dataset (all the bonds at once), and see how the error of those models compare to the time series error.

ARMA-EGARCH

Let $l_{t,k}$ be the volume of the kth bond at time t. We have 28 time series l_k for $k = 1, ..., 28$. There is no reason for the volume of a corporate bond to be stationary, since the number of transactions a day of an asset depends highly on the overall market capitalization of the underlying company, which may grow or shrink over time. Since time series model assume stationary inputs, I difference the l_k series to get the r_k series, where $r_{t,k} = l_{t,k} - l_{t-1,k}$.
A quick look at the autocorrelation function (ACF) plots and partial autocorrelation (PACF) function plots for each r_k tells us that the r_k are indeed stationary, and that the ACF cuts off at lag 2, and the PACF cuts off between lag 4 and lag 8. Example:





This hints that we should fit an ARMA(p, q) model, with p between 4 and 8, and q=2. For simplicity, instead of fitting a different ARMA model to each series, I decided to fit an ARMA(5, 2) to all of them. To take into account volatility clustering and asymmetric volatility responses due to movements in r_k, I also add a EGARCH component to the model. I therefore fit by MLE the following ARMA(5,2)-EGARCH(2,2) model to each series by using the matlab garchfit function:

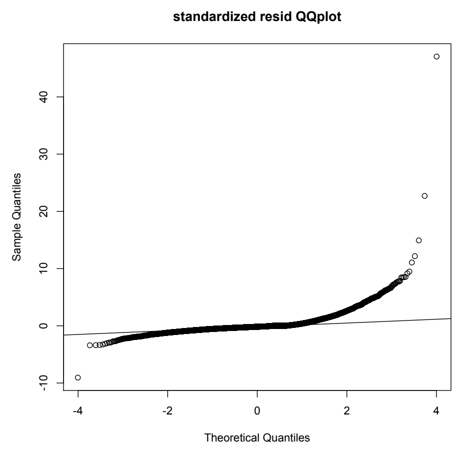
$$r_{t,k} = \phi_{0,k} + \sum_{i=1}^{5} \phi_{i,k} r_{t-i,k} + u_{t,k} + \sum_{j=1}^{2} \psi_{j,k} u_{t-j,k}$$

$$u_{t,k} = \sigma_{t,k}\epsilon_{t,k}$$

$$\log(\sigma_{t,k}^2) = \omega_k + \sum_{i=1}^{2} \beta_{i,k} \log(\sigma_{t-i,k}^2) + \sum_{j=1}^{2} f_i(\epsilon_{t-j,k})$$

$$f_j(\epsilon_{t,k}) = \begin{cases} (\alpha_{j,k}+\gamma_{j,k})\epsilon_{t,k} - \gamma_{j,k}E|\epsilon_{t,k}| & \epsilon_{t,k} \geq 0 \\ (\alpha_{j,k}+\gamma_{j,k})\epsilon_{t,k} - \gamma_{j,k}E|\epsilon_{t,k}| & \epsilon_{t,k} < 0 \end{cases}$$

Looking at qqplots, we see that the normal assumption $\epsilon_t \sim N(0,1)$ is violated. Hence, I assume ϵ_t is a standardized Student-t random variable (mean 0, variance 1).



Once the model is fitted, I test the time series model out-of-sample by fitting the models only on previous data, and then making the forecast. We can easily make new prediction $\hat{r}_{t+1,k}$, and compute $\hat{l}_{t+1,k}$ as

$$\hat{l}_{t+1,k} = l_{0,k} + \sum_{i=0}^{t-1} r_{t-i,k} + \hat{r}_{t+1,k}$$

Assuming we have data from t=0 to t=T and that we want to fit our model on at least 15 values, we compute the AAE of the kth bond as

$$AAE_k = \frac{1}{T}\sum_{t=15}^{T} |\hat{l}_{t+1,k} - l_{t+1,k}|$$

The results are shown at the end of this paper.

OLS and PCA

Now that I have fitted our time series models, I want to fit global learning algorithms to the data to see whether they fare better. Restricting ourselves to the case where we only use previous values as features (as in time series models), the first approach to take is to try to fit an ordinary least square regression to fit the previous daily volume values to the current one. Let h the autocorrelation lag (the number of past values I will look at), and let:

$$l_{t,k} = \sum_{i=1}^{h} \theta_i l_{t-i,k} + \epsilon_t$$

where ϵ_t has mean 0 and variance σ^2. Notice that I seek to get the unique values θ_i for all k. To do that, I set up my data in a panel data format, where I set up a response variable y and a matrix X such that:

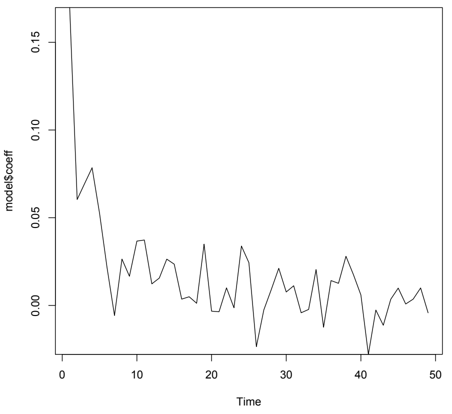
$$y = \begin{pmatrix} l_{t,1} \\ l_{t-1,1} \\ \vdots \\ l_{t,2} \\ l_{t-1,2} \\ \vdots \end{pmatrix}$$

$$X = \begin{pmatrix} l_{t-1,1} & l_{t-2,1} & l_{t-3,1} & \cdots & l_{t-h,1} \\ l_{t-2,1} & l_{t-3,1} & l_{t-4,1} & \cdots & l_{t-h-1,1} \\ \vdots \\ l_{t-1,2} & l_{t-2,2} & l_{t-3,2} & \cdots & l_{t-h,2} \\ l_{t-2,2} & l_{t-3,2} & l_{t-4,2} & \cdots & l_{t-h-1,2} \\ \vdots \end{pmatrix}$$

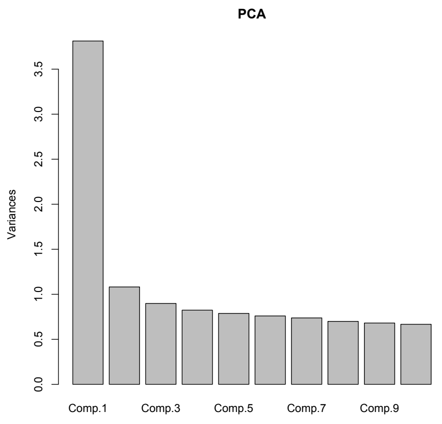
To make my analysis, I remove any row with missing values. I create a training sample and a test sample by randomly selecting 30% of the total number of row indices from X and y and by making the predictor matrix X_{test} and test response y_{test} from those selected rows. The rest of X and y become $X_{training}$ and $y_{training}$. I find the parameter vector $\hat{\theta}$ by

$$\hat{\theta} = (X_{training}^T X_{training})^{-1} X_{training}^T y_{training}$$

For a lag of 50, we see that our parameters θ_i for i=1,..., 50 look as such:



This plot shows that the most relevant variables seem to be the first 5 lagged values in decreasing order (the previous day is the most important, followed by the second one, ...). Given we are looking at a time series, this is fairly intuitive. This, coupled with the fact that only the first 5 lags seem to have a pvalue lower than 0.05, tells us that h=50 is too large, and that we are overfitting. Clearly, our parameters have a large variance. I take two approaches to solve this issue. First, I do a stepwise inclusion of my lagged variables using a bayesian information criterion (BIC), and find that my best model is with h= 15. For h=15, my sample has $\sim$ 18000 observations after removing missing values, with the test set having 5000 observations and the training set having 13000 observations. $X_{training}$ is therefore a 13000 by 15 matrix.
Secondly, I do PCA on the covariance matrix of $X_{training}$, to see whether I can reduce the dimension of this problem. The next figure shows that the first 3 components (and mainly the first one) account for most of the variance.



As such, let u_1, u_2 and u_3 be the top eigenvectors, then I can represent the regression as:

$$y_{training} = \theta_0 + \sum_{i=1}^{3} \theta_i X_{training} u_i + \epsilon_t$$

I can then test the OLS and the OLS with PCA factors models I get on X_{test} and y_set to get the AAE of those two models. The results are listed in the results section.

Kernel Regression

Due to the nature of our kernel, I try to implement kernel regression with an exponential kernel $e^{-(t-t_i)/\tau}$ to see whether it can filter out the noise. Instead of fitting τ by cross validation, a better technique seems to be to do a lasso regression of different kernel fits with different τ. I take 10 values of τ, where $\tau_1 = 1$, $\tau_i = \tau_{i-1} * 1.5$.

$$y_{training,i} = \theta_0 + \sum_{i=1} \theta_i \frac{\sum_{j=1}^{(h)} e^{(-j/\tau)} X_{training,i,j}}{\sum_{j=1}^{(h)} e^{(-(j-1)/\tau)}}$$

where I implement the lasso using the modified least angle regression algorithm from the R lars

package, which find the best fit by k-fold cross-validation. This fit sets to 0 all of the θ except for $\theta_0 = -0.005$, $\theta_1 = 0.185$, $\theta_7 = 0.019$, $\theta_8 = 0.183$, and $\theta_9 = 0.211$.
We can see in the results section that the OLS regression, the OLS regression on PCA factors and the Kernel regression yield similar out-of-sample AAE, which is higher than the average out-of-sample AAE we get from our ARMA-EGARCH models. This is a perfectly understandable result, considering ARMA-EGARCH models do filter and regress the data on its past values, as does kernel regression, but do it one bond at a time, and by modeling the variance as well. However, we can ask ourselves whether there is another way to do better than ARMA-EGARCH using $X_{training}$ as our predictor matrix.

Boosted Trees, SVR

The first step is to use boosted trees using the MART (Multiple Additive Regression Trees) algorithm on our predictor matrix $X_{training}$. MART computes a sum of regression trees of fixed size (in this case, only 6 leaves) which are added to the model via a stagewise forward procedure. This implies that each new regression tree is fitted on the residual of the current model. A regression tree has the form:

$$T(x, \theta) = \sum_{i=1}^{J} \gamma_j I(x \in R_j)$$

where $\theta = \{R_j, \gamma_j\}_1^J$, and J is the number of leaves.
Another technique to try is using Support Vector for Regression (SVR), with a radial basis kernel, which is a modification of Support Vector Machines adapted to regression. I fit MART using the R MART package and SVR using the R e1071 package.
We see in the result section that MART performs better than OLS and Kernel regression, but still does not match the ARMA-EGARCH models. However, the difference between the SVR model and the ARMA-EGARCH models is almost insignificant. As such, we have managed to match the time series model using SVR. However, we do not outperform, and training the SVR model takes a much longer time than training the ARMA-EGARCH Models, even though there are 28 of them. As such, I try two other approaches:

- Using neural networks and seeing whether using additional depth can make us perform better.
- Adding features to the model.

Will Deep Learning help?

I fit a neural network with only 1 hidden layer composed of 100 hidden units on the $X_{training}$, with a learning rate of 0.5, a momentum of 0.2 and a weight decay of 10^{-4}. I find that this model performs better than MART but not as well as SVR and the time series model.
I then fit a 4-hidden layers neural network with 100 hidden units in each layer. I train this network as a Deep Belief net, which means seeing the network (minus the output node) as stacked restricted Boltzmann machines. Those are trained by stochastic gradient descent with Contrastive Divergence of 1 iteration. The resulting parameters are then used as initial values to train the neural network via backpropagation.
I used the DeepLearnToolbox in matlab to do this fit. I find that given the numbers of parameters, I need to increase my weight decay to 10^{-3} for better results. Adding (or subtracting) hidden layers increases the AAE. As such, 4 hidden layers is my best DBN model. We see that this model fares better than the 1-hidden layer network, but not significantly better than SVR of the ARMA-EGARCH models. As such, adding depth did not significantly improve my accuracy in this setting.

Adding Features

The only remaining way to improve our model is to add more features. The first feature I added was the industry label, split between 1- Financials (JP Morgan, Goldman Sachs), 2- Utilities (The Dow Chemical Company, AT&T) 3- Industrials (IBM), Consumers (Walmart). Out of the 28 bonds in my dataset, 12 are Financials, 6 are Utilities, 5 are Industrials and 5 are Consumers.
Additionally, I grabbed the price data for each bond during the same time period (November 15 2005 to November 15 2012), and used returns and volatility, where the returns are computed as

$$R_{t,k} = \frac{Price_{t,k} - Price_{t-1,k}}{Price_{t-1,k}}$$

and the volatility is defined as the standard deviation of the past 20 returns (since a month has around 20 trading days).
For each volume $l_{t,k}$, I then define its predictors to be (as before) the previous 15 volume values for this bond ($l_{t-i,k}$ where i=1,...,15), but also its past 4 returns values and 4 standard deviation values, its industry label, as well as the past 2 volume, returns and standard deviation values of all of the other bonds in the sample. As such, I try to capture whether returns and volatility can influence volume, whether bonds from different industries have different behavior, as well as whether the volume of a bond can be influenced by the volume, price and volatility of other bonds.
There are a few issues with this model. First of all, the number of missing values increases drastically. If I were to remove rows with no missing values, I would lose most of my sample. Moreover, there are now (15+4+4+1+27*6)= 186 features in the model, many of which may be redundant. As such, I need a model that can perform well with missing values, a mix of categorical and numerical values and redundant variables. Neural networks and SVR are of not much help here.
The best candidate in this case is MART, which does enough regularization to remove unneeded variables, and uses surrogate splits to accommodate for missing values. However, I realized that after too many iterations, the MART algorithm is overfitting the data. I therefore restricted the number of iterations to 150, and the number of leaves in each tree to 4 instead of 6.
I find that MART with this new extra set of features outperforms on average the ARIMA-EGARCH models, with a reasonable training time.

Results

Here are the summary statistics for the AAE values for the 28 ARMA(5,2)-EGARCH(2, 2) models.

	Min	Max	Mean	Median	Std
AAE	0.3878	0.6301	0.5138	0.5196	0.0654

And here are the sorted AAE values for the different models:

Model	AAE
OLS + PCA	0.6142
Kernel	0.6028
OLS	0.5974
MART	0.5442
Neural Net, 1 layer	0.5260
SVR	0.5197
ARMA(5,2)-GARCH(2,2) (Mean)	0.5138
Neural Network-4 layers	0.5119
MART + new features	0.4049

We see that ARMA-EGARCH models fare really well against other learning algorithms. The error of ARMA-EGARCH models is on average half the standard deviation of data. We find that using MART with additional features such as an industry label, past returns and volatility get us to an error rate that is 40% of the standard deviation.
We see that this final model gives us for any bond in the sample approximately the minimum AAE we can get with an ARMA-EGARCH model, with the advantage of fitting only one model to the data.

Going forward

I have restricted myself to study the behavior of a limited number of investment grade corporate bonds. This research could be extended to junk bonds or other types of fixed income products. Similarly, we could add other features in the analysis, such as credit rating and maturity. Finally, I have also restricted myself in this study to univariate time series model. It could be valuable to test how multivariate volatility models perform against MART.

References

- Trevor Hastie, Robert Tibshirani, Jerome Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Second Edition. February 2009.

- Tze Leung Lai, Haipeng Xing, Statistical Models and Methods for Financial Markets, 2008.

- Rasmus Berg Palm,Prediction as a candidate for learning deep hierarchical models of data, `http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=6284` 2012.