Predicting Bank License Revocation

An Interactive Qualifying Project Report
Submitted to the Faculty
of the
Worcester Polytechnic Institute

in Partial Fulfillment of the Requirements for the
Degree of Bachelor of Science
by:

Jacob Bortell Michael Giancola

Everett Harding Parmenion Patias

Date: 13 October 2016

Report Submitted to:
Professor Thomas J. Balistrieri, Co-Advisor
Professor Oleg V. Pavlov, Co-Advisor

In Collaboration with:

Deloitte.

Abstract

The Central Bank of Russia has revoked 20% of all bank licenses since 2012. This project sought to determine a computational method for forecasting the probability of license revocation. To do this, we compared two classification algorithms, logistic regression and random forest. Using a Random Forest classifier produced higher F1 scores than logistic regression. We recommend that Deloitte continues the development of a random forest model by investigating clustering and additional features to improve the model's performance.

Acknowledgements

We would like to thank our sponsors at Deloitte for their assistance in completing this project: Dr. Jaroslav Bologov, our main liaison, Dr. Alexey Kozionov, and Dr. Alexey Minin, of the Data Analytics Institute at Deloitte.

We would also like to express our gratitude to our advisors from WPI, Dr. Oleg Pavlov and Dr. Thomas Balistrieri, for their assistance in developing this report and providing us with an enjoyable and intellectually stimulating IQP experience.

Finally, we would like to thank our hosts at the Financial University for their sincere hospitality and warm welcome to Moscow.

Table of Contents

Authorship

The table below indicates the primary contributors to each section of the paper. The contributors are denoted by their initials (JB=Jacob Bortell, MG=Michael Giancola, EH=Everett Harding, PP=Parmenion Patias)

Section Title	Primary Author(s)	Primary Editor(s)
Abstract	JB, MG	MG
Acknowledgements	MG	-
Authorship	PP	EH, MG
Executive Summary	PP	MG, EH
1 Introduction	PP	EH
2 Background	-	-
2.1 Macroeconomic Conditions	PP	EH
2.2 Banking Regulations	EH	PP
2.3 Previous Predictive Models	EH	
3 Methodology	-	-
3.1 Build Basic Dataset	JB	EH
3.2 Develop Basic Models	MG	EH
3.3 Determine Significant Features	MG	EH
3.4 Evaluating & Optimizing Model Performance	MG	EH
4 Results and Findings	EH	-
4.1 Feature Set	EH, PP	MG
4.2 Performance of Models	EH, PP	MG
5 Recommendations	PP	EH, MG

6 Conclusion	PP	EH
Appendix A: Regression Models	EH	-
Appendix B: Random Forest Models	EH	-
Appendix C: Performance Metrics	MG	EH
Appendix D: Model Code	MG	-
Appendix E: Parser Code	JB	-
Appendix F: Glossary	PP	JB
Appendix G: Significant Features	EH	-

Executive Summary

In this project our team compared two models that were created to forecast when the Central Bank of Russia (CBR) may revoke a bank's license. The models our team used were multinomial logistic regression and random forest. The two models were created and run using the same data in order to compare how they operate and determine which has the best performance.

The data used to develop the models was collected from two databases: Banki.ru and CBR.ru. The data included information found from the financial documents that are published monthly by banks for the CBR and are available to the public. The data included numerical standards, ratios required by the CBR to decide whether a bank is in an unstable state. Examples of these standards are Current Liquidity and Capital Adequacy Ratio. We created a parsing program to download and update the data collected in order to both save time in this project, but also to provide Deloitte with a tool that can be used in the future.

This data was formatted to form our base dataset. The monthly statistics were sorted into observations. Each observation contained all the statistics reported about a single bank from that month, providing a "snapshot" of the bank's financial situation. Each observation was also assigned a label that reflected how many months before the bank lost its license the "snapshot" was taken. These labels were: 1 month to 24 months, "Greater than 2 Years", and "Still Active". The last of these was used to label observations of banks that still retained their license at the time of our research.

To compare the models, we used three different metrics: precision, recall, and F1. Precision is concerned with the accuracy of the model's predictions while recall measures the completeness of the predictions. F1 is the harmonic mean of precision and recall.

To optimize the models, the team came up with three alterations that could help identify the best model, as well as differences between the models. The first modification was to preprocess the dataset before passing it to the model. Since the order of magnitude of our data varies greatly from one feature to another, scaling the data can benefit some types of models. Second, we tested the model's performance on monthly predictions versus quarterly predictions. Finally, we experimented with removing the top 20 and bottom 20 banks by net assets from our dataset. We hypothesized that by removing the banks with extreme values from the dataset, trends in the banks in the middle would become more apparent, improving the performance of both models.

The results of our experiments indicate that the random forest model is better suited for this problem than the regression model. We generated several sets of results to analyze the effects of the optimizations we implemented. Preprocessing improved the regression model slightly, but had negative results on the random forest model. This was understandable, as the benefit of preprocessing is dependent on the way that a particular model processes the data it is given. The

method by which regression makes predictions is much more sensitive to the magnitude of features than random forest is. Next, predicting on a quarterly basis produced much higher F1 scores than monthly. The reason is intuitive: the models had more data points per category to base their predictions on. Finally, removing the "extreme" banks had mixed results. It slightly worsened the results of regression, while slightly improving the random forest model. However, the improvements seen in random forest weren't large enough to state that the change was responsible for the improvement.

Our team recommends Deloitte to use the random forest model instead of the regression model, and continue with its development. They should check for revocations quarterly, as the predictions are much more accurate than monthly while giving more precise predictions than a yearly analysis. If they need the granularity of a monthly analysis, we suggest they input data which is recorded weekly, as monthly data proved to be insufficient to produce accurate results at that scale. We recommend they do additional research into methods of clustering banks into groups and running the model separately on different clusters.

1 Introduction

Between 2012 and 2015, the licenses of more than 200 out of 900 banks in the Russian Federation were revoked by the Central Bank of Russia (CBR). (The Moscow Times, 2015) Mr. German Gref, the CEO of one of the largest Russian banks, Sberbank, predicts that another 10% of banks in Russia will lose their licenses by the end of 2016. Bank license revocation affects not only investors, but also commercial and retail clientele who have life savings deposited in these banks. (In Russia, economic recovery remains elusive.2015)

It is difficult to predict when the CBR will revoke a bank's license due to the wide array of circumstances that can lead to a license revocation. Failure of banks to meet the required standards set in the banking legislation is a sure condition for the CBR to revoke a bank's license. Additionally, the CBR can revoke a bank's license if they have substantial proof that fraud has taken place, if the bank has conducted operations not covered by the bank's license, or based on 20 other cases included in Federal Law 395-8, Article 20. (On Banks and Banking Activities, 1990)

Predictive models have been developed to forecast the probability of a bank going bankrupt. Researchers have used different methods, including regression models and neural networks to achieve accuracy level upward of 80%. (Boyacioglu, Kara, & Baykan, 2009) Some papers have provided analysis of the factors included in the model. For example, Mayes and Stremmel (2012) found that "the non-risk-weighted capital measure explains bank distress and failures best." (Khalafalla Ahmed Mohamed Arabi, 2013) Due to the instability of the economy in past years, an accurate model to forecast the closure of banks can provide consumers and investors with a way to confirm that their money will be safe with a particular bank.

The goal of this project was to generate a model capable of accurately forecasting when a bank's license will be revoked by the CBR. To expand upon previous work, we developed and compared two multinomial classification models which would forecast up to two years in the future when a bank may lose its license. The models' performances were verified using several methods of performance analysis. Finally, one of the models was selected as a recommendation for continued research.

2 Background

2.1 Macroeconomic Conditions

Banks in Russia are in the midst of several challenges, including economic sanctions that prevent foreign investments and the decline of oil prices starting in 2014. After the drop of the price of oil in 2008, it became clear that Russia's economy was based on two staples which are heavily connected: oil and hedge fund investments. Every time one of the two undergoes a crisis the second follows, due to the high investments of hedge funds in oil (The Economist, Issue 8985, 2016). Currently, increasing the oil price is a matter of great importance for the Russians. This concern was reflected in the first day of the 2016 G20 Summit in China, when the President of Russia (and the Crown Prince of Saudi Arabia) commented on the need to discuss oil prices (Workman, 2016).

Given the probability of an oil crisis the Kremlin has promoted a diversification of investing, focusing on exporting non-petroleum products (such as metal). At the same time the CBR has increased Russia's international reserves from 140Bn USD to 500Bn USD, during the 2009-2013 period, while allowing the Russian Ruble to float (The Economist, Issue 8985, 2016). Instead of spending all the reserves on keeping the value of the Russian Ruble on par with the dollar, they concentrated on helping institutions affected by the economic sanctions imposed by the European Union and the United States. This shift of focus from defense of the Russian Ruble to support of domestic institutions resulted in a 10% decrease in the actual salary of Russians (inflation fluctuation). However, it bears mention that the average Russian's salary remains triple the size it was before Mr. Putin became President, which shows a general upward trend since the 1998 crisis.

From 2014 to 2016 Russia's currency depreciated from 35 Russian Rubles to 1 USD to 64 to 1. The Kremlin put its trust in Mr. Elvira Nabiullina as the new head of the Central Bank of Russia (CBR) in 2013 (Reuters.com, 2016). Since her instatement at the head of the CBR, the organization has rapidly revoked the licenses of more than 200 financial institutions (Weaver, 2015). The motive behind the revocations has been debated. Ms. Nabiullina "received *carte blanche* from the president to go after those banks that were earlier untouchable," says Mr. Oleg Vyugin, chairman of MDM Bank and a former deputy governor of the Central Bank. Ms. Nabiullina stated "This [revoking of licenses] is not a cleansing effort. This is an effort to make the banking sector viable and get rid of the weak players".

Russian Banks are usually divided in three categories by specialists. The top tier, which can be thought of as "too strong to fail," can be considered almost certain to keep their license. The second tier, the top 100 banks, is comprised of those institutions expected to survive the worst of crises but with less certainty. The third tier contains the remaining banks, institutions of varying stability which are trying to survive and maintain their licenses. However, a good ranking does not guarantee a bank's survival. Despite its position as one of the "Top 100" banks, Master Bank's license was revoked in 2013.

The CBR revoked the Master Bank's license based on money-laundering violations and suspicious transactions. The bank had a 2 Bn USD hole in its balance sheet, owed almost 1 Bn USD's to its depositors, and disrupted the operations of numerous other banks (Makhonin, 2013). Russia's banking system has been hit many times by these kinds of illegal activities and the stock market is still seen by many as a money laundering paradise. The problematic state of the banks can be seen again through some extreme examples of banks' illegal activities. According to the CBR, Admiralteisky Bank's license was revoked due to "inability to follow anti-laundering rules and because it practically stopped serving its clients." Moreover, when CBR officials opened the bank's vaults, they found iron bars painted gold, evidence of a bank falsifying its assets (Amos, 2015).

Between the years 2008 and 2013 the CBR revoked, on average, 30 licenses per year. This number was almost tripled in 2014 when more than 80 licenses were revoked (Weaver, 2015). The uptick in revocations shows that Russia has decided it needs to rely on more than its size and its oil in order to keep its position as one of the strongest countries. Strengthening the economy is one important step. To be certain of this strength Russia needs to make sure that no bank will unexpectedly reach a point where its license will need to be revoked, possibly leading to events similar to the events of 2008 in the United States. To reach such a state of certainty it needs strong, law-abiding banks.

2.2 Banking Regulation

Banking regulation refers to the laws that banks must follow to legally operate in the Russian Federation. The regulations are set by the Central Bank of Russia (CBR). The regulation outlines the criteria and process for evaluating the performance of each bank. A key component of these regulations are the banking normatives, a set of numerical standards that regulate the levels to which a bank can take risks. These standards are defined in Federal Law No. 87 FZ, Article 62. Those below are outlined in Bank of Russia Instruction No. 139-I:

1) Capital Adequacy (N1) The minimum allowable value for this ratio is 10%.
   a) Capital Adequacy ratio (N1.0) Capital adequacy ratios make sure that banks have enough capital to absorb losses without going bankrupt. Each ratio governs different components of a bank's capital. N1.0 regulates the entirety of a bank's capital. The minimum allowable value for this ratio is 8%.
   b) Common Capital Adequacy ratio (N1.1) This is a ratio of a bank's equity capital combined with its disclosed reserves to its risk-weighted assets. The minimum allowable value for this ratio is 4.5%.

   c) Tier 1 Capital Adequacy ratio (N1.2) This expands on the Common Capital adequacy ratio, including some forms of preferred and common stock with the equity capital. The minimum allowable value for this ratio is 6%

2) Liquidity

   a) Quick bank liquidity ratio (N2) - Regulates possible liquidation losses in the next business day. Calculated minimum ratio of high-liquidity assets to on-call obligations must be at least 15%

   b) Current bank liquidity ratio (N3) - Regulates the risk of liquidation losses in the 30 days after its calculation. It is the ratio of the bank's liquid assets to its liabilities due within the regulation time period. The minimum allowable value for this ratio is 50%.

   c) Long-term bank liquidity ratio (N4) - Regulates the risk of banks losing liquidity to long-term investments. It is the ratio of a bank's credit claims maturing after the next 365 days from the date of calculation to the sum of bank's equity and liabilities that will mature in the same timeframe. This is also adjusted for liabilities of the bank maturing within the next 365 days. This ratio can be at most 120%. In other words, the long-term fiscal stability of the bank cannot be wholly dependent on money the bank does not yet possess.

3) Maximum Risk per borrower or group of borrowers (N6) - Regulates the credit risk of a bank in relation to one borrower or group of borrowers. It is calculated as the ratio of the total credit claims the bank has against a borrower to the bank's equity. There are some cases for which N6 is not calculated, such as when the borrower is the Russian Federation, federal executive bodies, or the CBR. It is also not calculated for borrowers that are also stakeholders of the firm (these are covered by N9 and N10). The maximum value for this ratio is 25%.

4) Maximum value of major credit risk (N7) - Regulates the maximum total value of the major credit risks of an institution. It is calculated as the ratio of the major credit risks to the bank's equity. The maximum value for this ratio is 800%. This large value allows the major credit risks to exceed the capital of the bank up to 8 times in a year.

5) Maximum value of loans, guarantees, and sureties issued by the bank to shareholders (N9.1) - Regulates credit risk on a bank in relation to its shareholders. Calculated as the ratio of the total loans, sureties, and guarantees issued by the bank to its shareholders to the bank's equity. The maximum value for this ratio is set at 50%.

6) Total risk on bank insiders (N10.1) - Regulates the amount of risk the bank can have on anyone able to influence the decision to issue a loan to (place risk on) someone. This ratio is calculated as the total credit risks and financial derivatives that the bank

has with everyone able to influence the decision to go ahead with those kind of transactions to the bank's equity. This ratios maximum value is set at 3%.

7) Use of equity to purchase shares of other legal entities (N12) - Regulates the amount of a bank's equity that can be used to purchase shares of other legal entities. The maximum value of this ratio is set at 25%.

2.3 Previous Prediction Models

A paper examining bankruptcy in Taiwan of examined the effectiveness and predictive accuracy of multiple discriminant analysis, logit regression, probit regression and Artificial Neural Networks (ANN) techniques (Lin, 2009). This study used univariate logistic regression to select which financial ratios to use as variables in the development of each of the four models above. The selection narrowed 44 ratios considered to 20 ratios used in the study. By narrowing the focus of the model, the study was able to eliminate factors that had no effect on the dependent variable. If included in the model, these factors would only add noise to the data, detracting from the accuracy of the model.

A model developed in 2006 compared the accuracy of logit regression and trait recognition models (models that look for approximate patterns in a dataset and can make predictions on data points that are not exact matches to the trend) (Lanine & Vennet, 2006). The paper predicted bank failures in the Russian sector quarterly, in periods of 3 months, 6 months, 9 months, and 12 months. In this application, the models showed that liquidity of bank assets to be an important determinant of bank license revocation predictions. It also cites asset quality and capital adequacy as significant determinants. Other factors used in the models developed in the paper are return on assets (ratio of net income: total assets), government debts securities (ratio of government debts: total assets), overdue loans (ratio of overdue loans + overdue promissory notes: total loans), ratio of loans to total assets, and a value for the size of the bank calculated as Log(total assets) (Lanine & Vennet, 2006).

Another model, aimed specifically at predicting defaults in the Russian banking sector in the timeframe 1997-2003, explored the benefits of clustering banks into groups of comparable banks, then developing individual models for each cluster. The study separated banks in four ways: by the ratio of their investments in government bonds to their total assets, the size of the bank's assets relative to the total market assets, the ratio of banks credits-to-non-financial firms to total assets, and the ratio of the bank's equity to its total assets. This paper clustered each bank into three categories based on "small", "medium", and "large" values of each of these four parameters. After clustering, the paper found the liquidity, equity and government investments to be significant indicators of a bank's eventual default/avoidance of default (Peresetsky, Karminsky, & Golovan, 2011).

The findings of these papers indicate that the significant factors in predicting defaults (license revocations) of banks can be seen in the capital adequacy or equity ratio (defined as N1 by the CBR) and liquidity ratios (defined as N2 through N4 by the CBR). Given the significance of these factors determined by the CBR and third parties, our model development will begin with the inclusion of the normatives N1, N2, and N3.

3 Methodology

The goal of this project was to forecast when a bank is likely to have its license revoked by the CBR. Previous papers have framed the problem of forecasting bank license revocations as a binary situation (revoked/not revoked). Our approach attempts to forecast the time until a bank will lose its license. We created a pair of models with this in mind. Provisions are made in the development of both models for the case where a bank does not lose its license.

The first of the models we used was a multinomial logistic regression model. This model is a statistical method that takes the logistic regression algorithm used as a baseline in previous works and extends it to forecast months rather than whether or not the license was revoked. For simplicity, we refer to this model as our "regression" model. This regression model produces a set of equations that estimate the number of months until a bank loses its license. For further details on the implementation of regression models, see Appendix A.

The second of the models we used was a random forest model. This is a machine learning method that has not to our knowledge been used in this application. Like the regression model, the random forest attempts to forecast the date a bank's license is revoked. Instead of an equation, a random forest produces a set of decision trees to forecast the number of months until a bank loses its license. For more information on random forest models see Appendix B.

The objectives and corresponding methods for the development of these models are summarized in Table 3.

Table 3: Objectives and Methods	
Objectives	Methods
1. Collect financial data into base dataset	● Create a script to draw data from online databases ● Organize data into a manageable form
2. Determine the significant financial factors which indicate a propensity for license revocation	● Evaluate the relevance of each available factor ● Graph factors over revocation status to find trends
3. Develop the models	● Discuss with Deloitte liaisons the best practices for creating effective models using financial data ● Use the Python library *scikit-learn*
4. Analyze and compare performance of models	● Apply Precision, Recall & F1 metrics ● Generate optimal models for comparison with each other

3.1 First Objective: Build Basic Dataset

We downloaded our data from two online databases: banki.ru and the Central Bank of Russia's (CBR) public records. The first database, banki.ru, contains 89 monthly statistics on each financial institution in the Russian Federation dating back to March 2008. It also contains a list of license revocation dates for banks that are no longer operational. The CBR database contains bank statistics reported monthly in compliance with the standards described in Chapter 2.2 of our report. After downloading data from both sources, we used it to generate our dataset for the development of the models.

It should be noted that each observation in the dataset is independent of a specific license number. Before being put into the model, the license numbers are removed, so each observation becomes a different picture of the same nameless bank. The problem is not to find trends that lead banks to fail over time, but to find sets of values that correspond to banks that are (x) months or quarters from failure.

This combination was done in two steps. First, we combined the data from banki.ru with the data from CBR to create a set of monthly ***observations***. Each of these observations provides a snapshot of the financial situation of a bank at the beginning of the month. This observation of a bank is a collection of the monthly statistics about that bank. Each of these statistics is called a ***feature*** of the observation. Initially we used only three statistics, N1, N2, and N3, as features of the observations. More banking statistics were added as features later.

Second, we classified each of the observations from the first step by the number of months between the date of that observation and the date the bank eventually lost its license. If this number was greater than 24, we assigned the observation a special value of 1000 to signify that it represented the bank's financial situation more than two years before the bank's license was revoked. If an observation is of a bank that has not yet lost its license, it is assigned a value of 9000, signifying that there is no revocation date for that bank. The final combination of our observations and their classifications formed our dataset. For more details on the downloading and combination of our data, see the Python code in Appendix E.

We split our dataset into two similar parts for model development. Each part had the same proportion of observations from each class. Creating the sets in this way is called ***stratifying*** the two sets. The first set, called the ***training set***, was used to build the models. The second set, called the ***testing set***, was used to evaluate the performance of the models.

Finally, we added one more component to our observations before inserting them to the models. For various reasons, the value of some statistics may be missing for a bank in a particular month. However, the model cannot run on incomplete data. To handle not-reported data, we set the value of the missing feature to 0, and add a new feature called ***X_M?***, where ***X*** is the name of the feature with the missing value. This feature is given the value 0 when the corresponding feature has no value, and 1 when the feature has a value.

3.2 Second Objective: Develop Basic Models

In completion of this objective we built simple versions of each model (Logistic Regression and Random Forest) which were later improved and compared. We developed the models using the *scikit-learn* library for Python. *Scikit-learn* is a machine learning library designed to aid developers in data analysis. It has functions to create classification models, implement clustering algorithms, and a variety of related operations. We used the classification functions to build our model because they best fit the way we structured the dataset.

Both models were built with the same training set. Each model is created by an algorithm that looks for trends in the observations that correlate to the classes (number of months until license revocation) assigned to the observations. This process of building the model based on the training data is called ***fitting*** the model. After the first fitting of the models, we repeated the process in our first objective to include more statistics as features in the observations.

3.3 Third Objective: Determine Significant Features

We determined which of the available banking statistics were significant as features of our observations for the two methods: one for the regression model and one for the random forest model. The random forest determines internally which features of the observations are significant (Fawagreh, Gaber, & Elyan, 2014). To determine which features were significant in the regression model we used a method of combining the emphasis placed on that feature with its standard deviation in the dataset (Peresetsky, Karminsky, & Golovan, 2011). This method is described further in Appendix A.

We compared the two determinations and selected the features of greatest significance as the features to include in the next iteration of the model. Having selected our significant features, we repeated the processes of generating our dataset, splitting it into the two stratified sets (training and testing), and fitting the model to the training set.

3.4 Fourth Objective: Evaluating & Optimizing Model Performance

3.4.1 Evaluating Model Performance

After fitting our models and selecting the significant features, we used the testing set from our first objective to determine how well the models could forecast the date a bank would lose its license. We input the observations from the testing set into the models, which output the estimated classification of each observation. We compared these estimated classifications with the actual classifications using three different evaluation metrics.

The first of these metrics is called ***precision***. This is a measure of how correct each model's estimations were. It answers the question "how many observations did the model classify correctly?" The second of these metrics is called ***recall***. This is a measure of how complete each model's estimations were. It answers the question "how many observations of this classification did the model identify?" The third metric is a combination of the first two called ***F1***. It provides an answer to the general question "how well did the model perform?" Details on the calculation of each metric can be found in Appendix C.

3.4.2 Optimizing Model Performance

The goal of optimizing model performance is to achieve the highest scores possible in each of our evaluation metrics. To maximize our model's scores, we made three edits to the original models.

First, we added a step to the dataset creation called ***preprocessing***. Preprocessing is a method of regulating the orders of magnitude in a dataset. For example, the feature Total Assets has values that range from thousands in some observations to billions in other observations. Preprocessing transforms all the values of this feature into values within a much more manageable range. This is intended to minimize the impact of the discrepancy between magnitudes of the values.

Second, we changed the way we classified our observations. Instead of classifying each observation by how many months it was from the date of the bank's license revocation, we classified each observation by how many business quarters it was from the date that bank's license was revoked. By changing the classifications we hoped to increase the resolution of our data, as there would be multiple observations of each bank in each classification.

Third, we removed the extreme observations from the dataset. We elected to remove all observations of the 20 largest and 20 smallest banks as reported by net assets. By excluding the extremes we hoped to remove anomalies from the data that would skew the results of the model.

4 Results & Findings

Before analyzing our results, we must discuss the composition of our dataset, as it has a significant effect on the results of our models. Despite the increase in bank license revocations, there are many more banks that are still active than there are banks that have lost their license. Because each of these banks are still active, they continue to produce new statistics each month, while the banks whose license has been revoked no longer produce new statistics. Therefore, as shown in Figure 1, we have much more data about the banks that are still active than we have about the banks that have lost their license. To further compound the bias, all observations of the "still active" banks (76% of the dataset) are in one class, while the data about banks that have lost their license (24% of the dataset) was further divided into 9 classes. More than half of the data that reflects banks whose licenses have been revoked (14% of the total dataset) is contained in the "Greater than 2 Years Left," leaving only 10% of the dataset to be divided among the remaining eight classes.

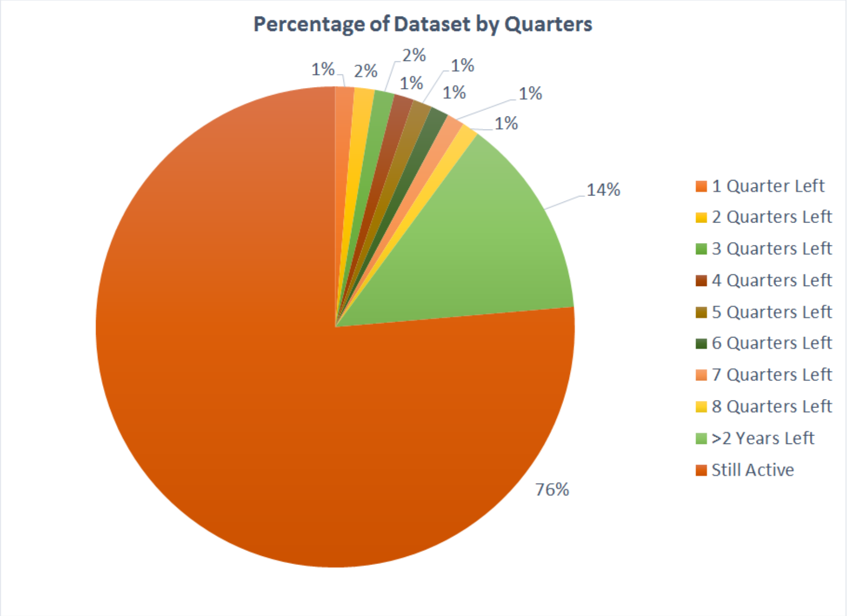


Figure 1: Percentage of data in each class,

With such a significant bias in the dataset, the models are provided with much higher amount of information in the “Still Active” class than the “> 2 years” and quarterly classes, which allows a better estimate of what constitutes a “Still Active” bank than the estimates of the lower classes.

4.1 Performance of Models

Since F1 is determined by both the precision and recall score, it is an appropriate metric to compare models. However, for additional insight into the performance of the model, we will consider the specific precision and recall scores individually.

4.1.1 Multinomial Logistic Regression

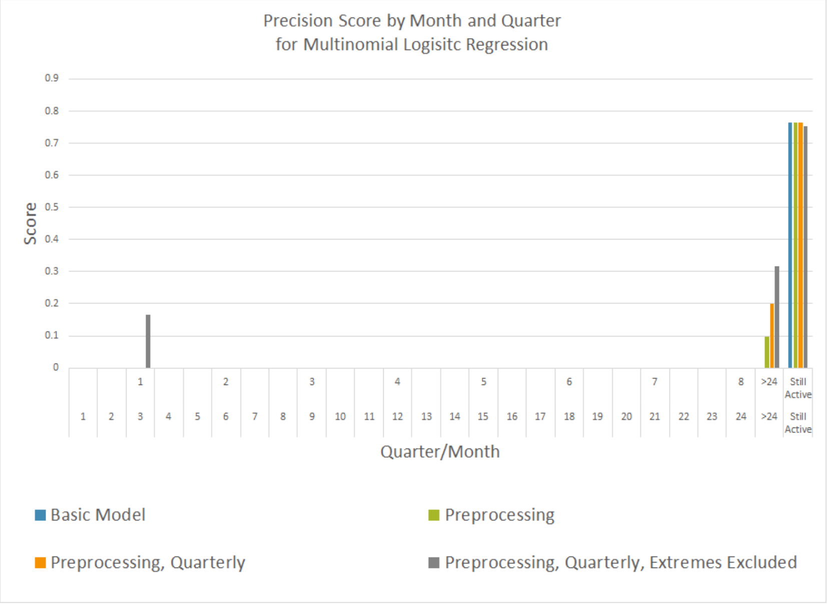


Figure 2: Precision scores for each iteration of our regression model.

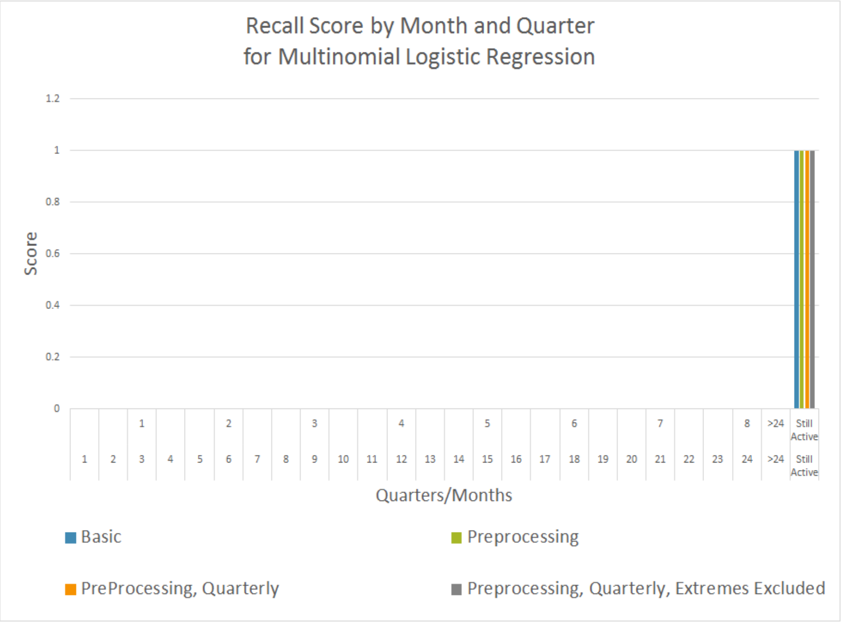


Figure 3: Recall scores for each iteration of our regression model

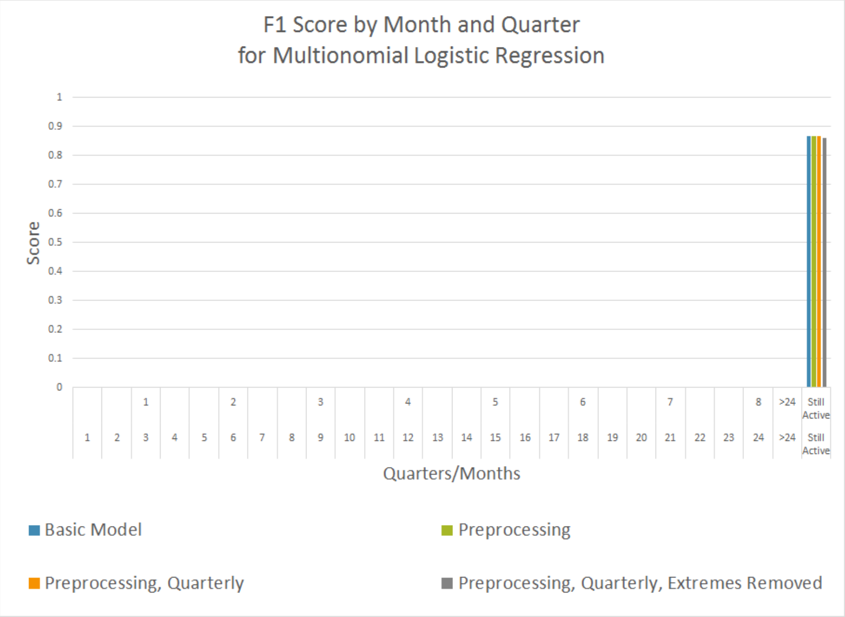


Figure 4: F1 scores for our regression model.

From Figures 2, 3, and 4 we can see that our regression model produces results with next to zero scores in most classes for all three metrics. The precision scores show the most success, with ~15% in the one quarter to revocation (for its best version) and between 10% and 30% in the > 2 years until revocation class for all versions but the basic version. All metrics show very high performance in the “Still Active” class. The recall score in this class is 100%, but the precision in this class is only ~75%, which indicates the model defaults to guessing that an observation belongs in the “Still Active” class, likely due to the bias in the dataset described above.

4.1.2 Random Forest Model

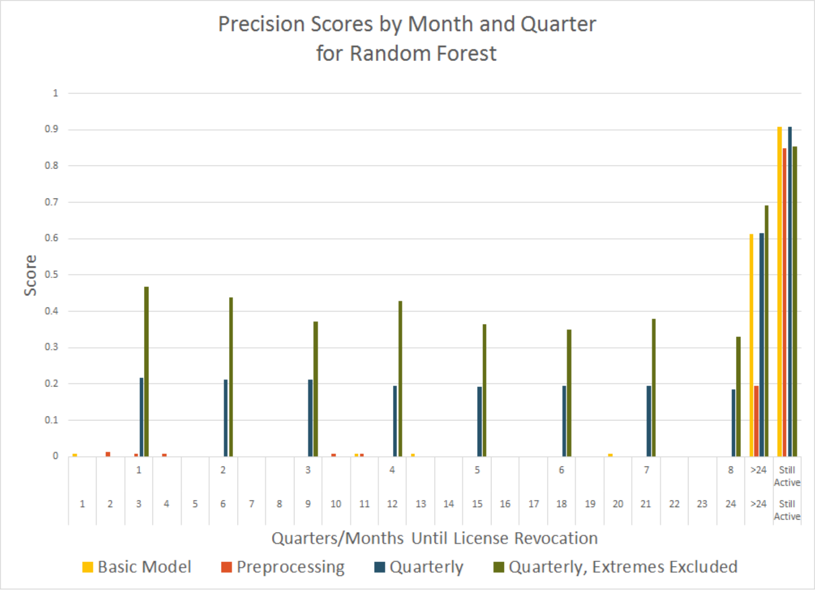


Figure 5: Precision scores for each iteration of our random forest model

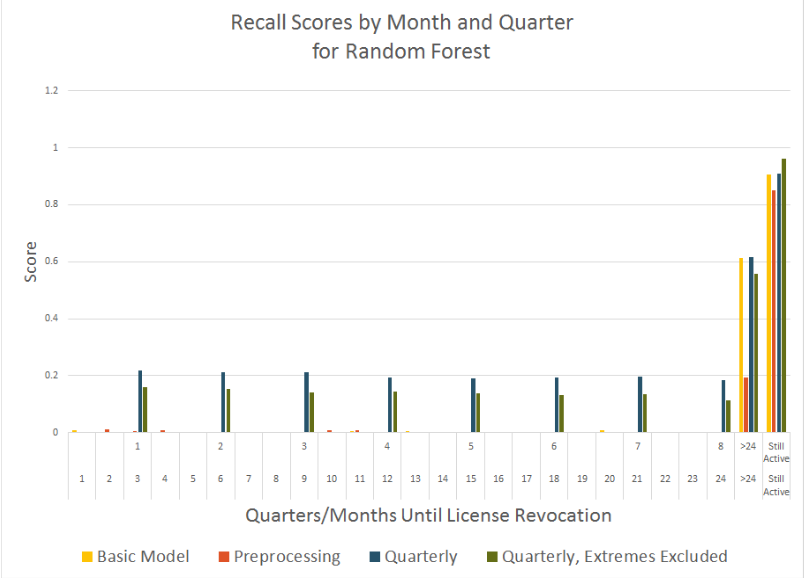


Figure 6: Recall scores for each iteration of our random forest model

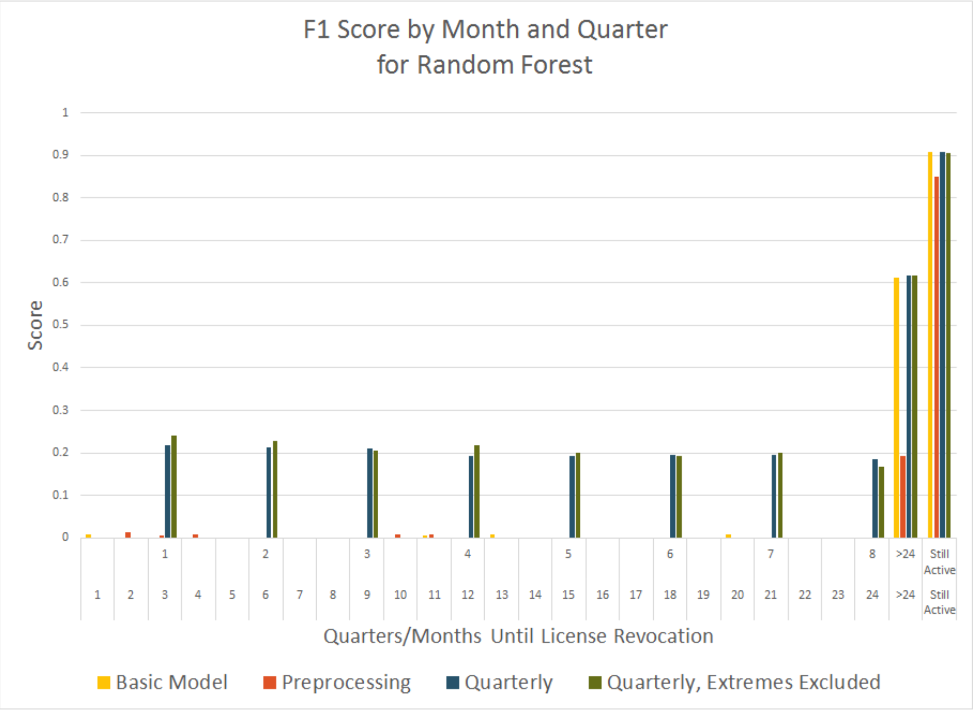


Figure 7: F1 scores for each iteration of our random forest model.

As we can see from Figures 5, 6, and 7 the random forest model was not improved by adding preprocessing, although it performs much better when run quarterly (as opposed to monthly). Generally, the quarterly model with extreme banks excluded (the top and bottom 20 banks by net assets) scores highest by F1, however it is too close to be statistically significant.

Achieving an over 20% average score between Q1-Q8 is a positive result. The model's precision and recall were similar for all iterations except the "Quarterly, Extremes Excluded". As illustrated in Figure 8, the precision score for this iteration was much higher (between 35%-45%) than the recall (around 10%).

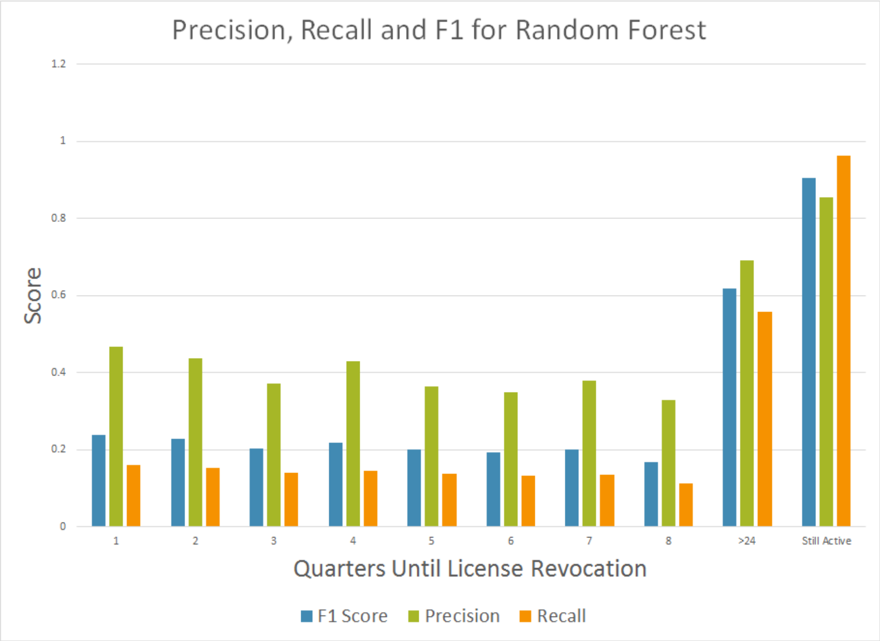


Figure 8: All metrics for best performing random forest model, calculate quarterly with the top 20 and bottom 20 (by net assets) banks excluded

We compared the "Quarterly, Extremes Excluded" model to the results of randomly guessing which observation from our testing set belonged to which category. The distribution our guesses matched the distribution of the dataset, meaning 76% of our guesses were that an observation belonged to the "Still Active" class, 14% in the "> 2 years" class, etc. When applying our performance metrics to our random guesses, we found that the random forest model outperformed our distributed random guessing, as can be seen in Figure 8.

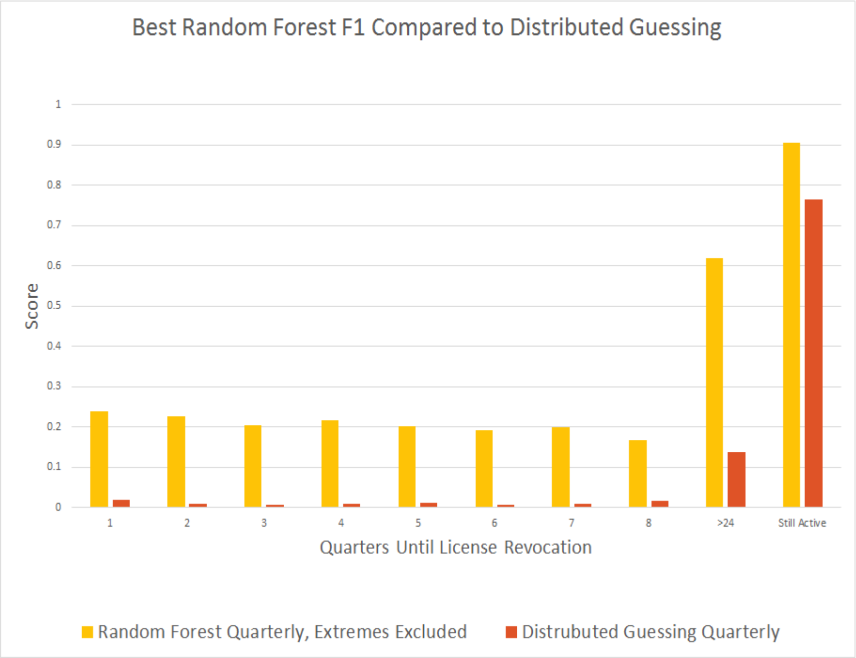


Figure 8: F1 scores for random forest model graphed next to F1 scores for random guessing with the same distribution as the dataset.

4.1.3 Model Comparison

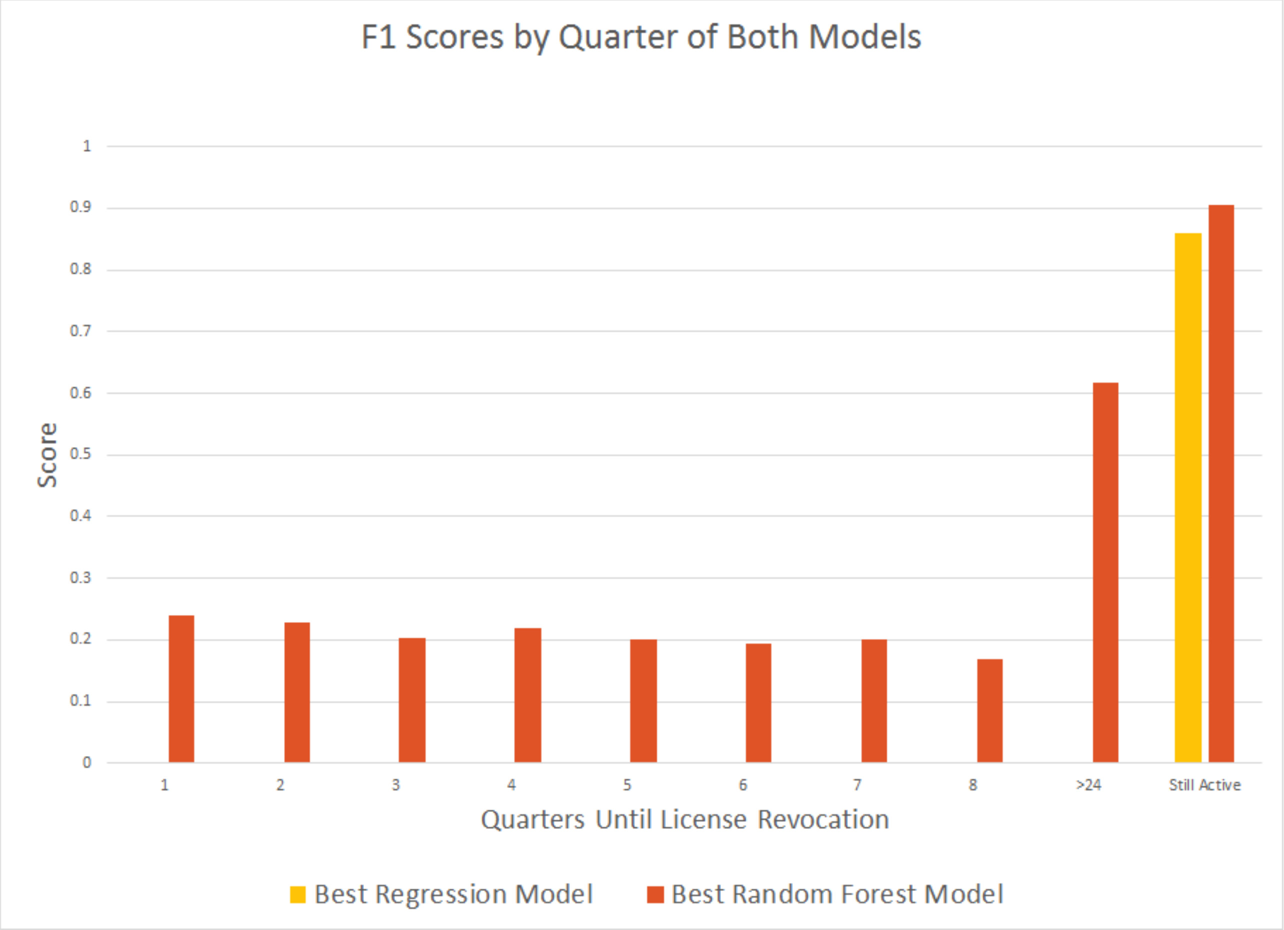


Figure 9: Comparison of best regression and best random forest model

Figure 9 shows the comparison of the optimal regression and random forest models. As discussed previously, regression only performs better than randomly guessing in one class, so clearly the random forest model is our best model. Our random forest model scores in the 15-25% range for the first 8 classes, scoring over 60% in the 9th and over 90% in the 10th class.

4.2 Significant Feature Set

The feature set used to generate our results is shown in Appendix G. Feature significance was determined for the random forest model using the Gini Importance (G.I.), which is the mean of the Gini Impurity Index[1] of each feature determined by all the decision trees in the random forest. For simplicity's sake it can be thought of as a "Percent Weight", or a percentage of the model's

[1] See Appendix B for details on Gini Index

prediction determined by that feature. The percent weights are shown in Figure 10 for each of the top 16 most-significant features. These features comprise 94% of the random forest's consideration. The other 31 features make up only 6% of the consideration of the model.

Figure 10 shows that the most significant factor is the total of all loans extended to institutions and businesses to be repaid in a three year period. Loans are directly involved with risk, thus we can speculate that the CBR pays attention to the investing of different banks and tries to keep them safe in and stable levels. By revoking licenses of banks that take too much risk, the CBR strengthens the economy by lowering the probability of these credit institutions going bankrupt and snowballing others with them.

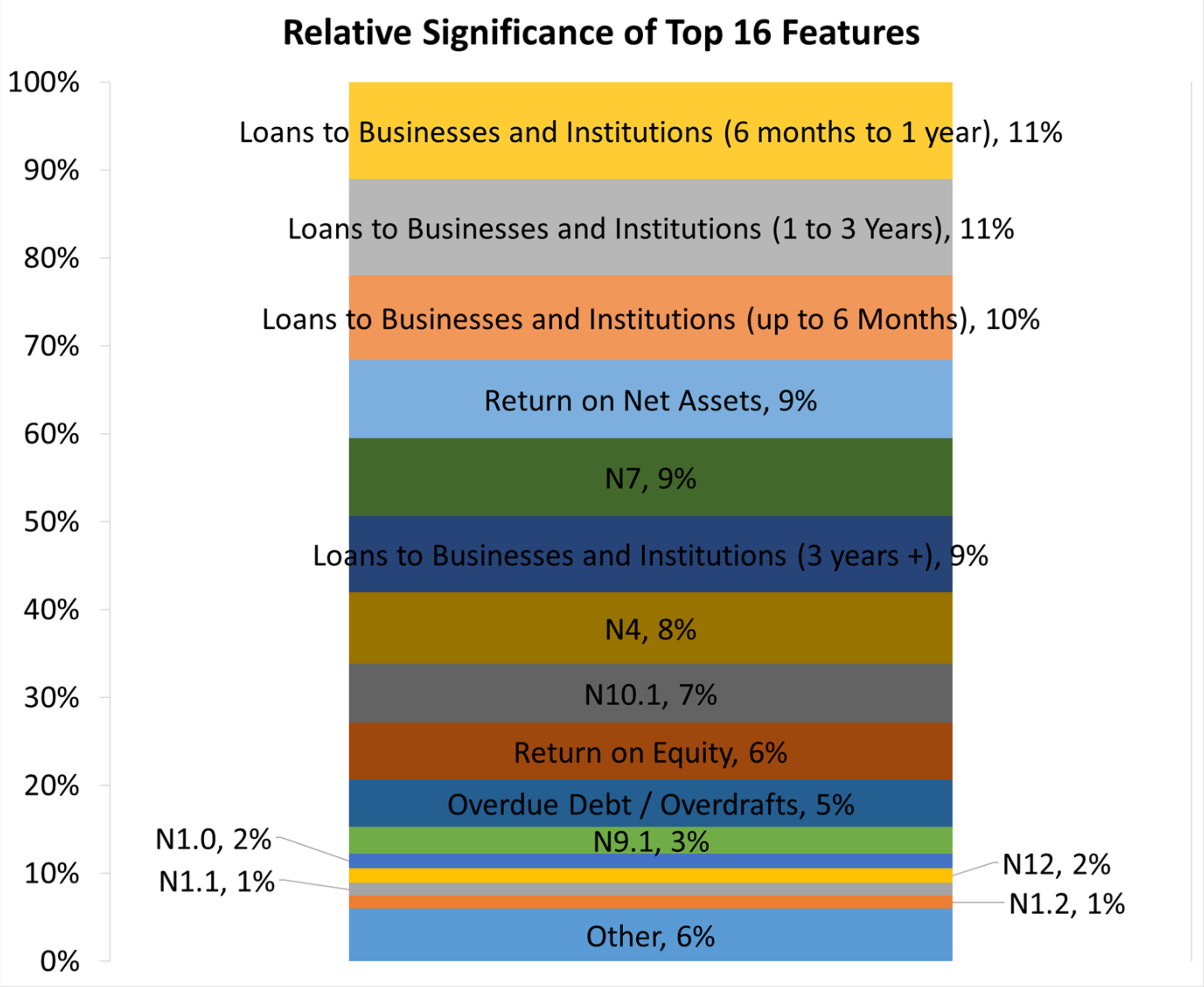


Figure 10 also shows that the features indicating whether or not a value was reported are entirely contained in the "other" features, as none of them appear in the "Top 16". The nature of the approach of the random forest model makes this type of feature redundant. However, for consistency in the comparison of the models, they were included for the random forest model because they were necessary in the logistic regression.

We expected N2 and N3 to be two of the most significant features but the random forest model calculated their significance as 25th and 21st respectively. Despite their use by the CBR, these ratios were shown to be insignificant in the random forest model. They were indicated as the two values with the greatest significance in the logistic regression model, but as that model did not yield a high performance, we cannot confirm the validity of this significance. The standards considered significant (greater than 1% of the consideration) were N9.1, N1.0, N12, N1.1, and N1.2.

5 Recommendations

5.1 Use Random Forest

Our results have shown that, for this application, the random forest algorithm scores higher on precision, recall, and F1 than the logistic regression algorithm. Specifically, it scores higher in all classes of the target variable. Therefore, we recommend that Deloitte continues to pursue the random forest approach rather than the multinomial logistic regression.

5.2 Predict on Broader Timeframes

We recommend Deloitte build models that predict on a quarterly basis, as both random forest and regression performed better under fewer target variable values. One likely cause of this is that our data is recorded every month. Predicting into quarters instead of months gives each class of target variable more observations, and a higher likelihood the model will find trends. Another option to improve model performance is to predict over even broader timeframes. For example, we ran our model on 4 classes, revocation in 1 year, 2 years, more than 2 years, or never, and got F1 scores of 43%, 41%, 63%, and 91% respectively. Those scores are generally higher than their respective counterparts in the quarterly model, although the predictions provided by this model are less precise.

5.3 Identify More Significant Features

We also recommend that Deloitte use more of the available data. One statistic in particular that we recommend Deloitte investigate is the ratio of precious metals over net assets. The precious metals data can be found from CBR’s form 110. The past years there has been a global shift from USD reserves to Gold reserves, thus the amount of gold reserves in a bank could indicate said bank’s importance for the Russian economy. Deloitte should also test additional data related to loans and risk evaluation. Based on results from both models, the features with most significance were the ones related to risk or loans. This is probably because the CBR tries to stop any over-risky investment, due to the probability bankruptcy.

5.4 Consider Effect of Preprocessing Data

Preprocessing the data before running the regression and the random forest models produced different results in each model. Preprocessing improved performance in the metrics for our regression model, but worsened results for Random Forest. This was expected, as the benefit of preprocessing is dependent on the way the models interpret the training data.

Regression develops an equation that combines the numerical values of each feature in the observations. Therefore, it is possible that the importance of some features might be skewed if they have a large variance in their values (some values between 0 and 1 and others in millions).

Random Forests internally divide all values of each feature into ranges during their development. Preprocessing reduces the resolution of those ranges, so we see a slight drop in performance values. Therefore we recommend Deloitte avoid using preprocessing if they continue to develop the random forest model.

5.5 Improvements Using Geopolitical Features

In addition to economic considerations, there are geopolitical and sociological factors that can lead to bank license revocation. We recommend, to further improve our model, that Deloitte add additional algorithms that analyze the current geopolitical and sociological environment and study how it influences banks. An example for such an implication would be to search how often a bank in referred to in the news, in a positive or negative way, and how this affects the bank.

5.6 Removing Extreme Banks

We ran both models with and without data from the top 20 and bottom 20 banks by net assets. We found that these banks, due to their size, behave differently than most banks. We hypothesized that removing them may make trends in most banks more clear, leading to improved results. We saw slight improvements in the random forest model using this method, however the improvements weren't significant enough to fully recommend this practice.

However, we would recommend that Deloitte investigate the effect of removing banks from the dataset which had their licenses revoked due to non-economic reasons (eg. fraud). Banks that lost their licenses due to illicit activities likely don't follow the same patterns as banks which lost their license due to economic failure, so removing these banks should make trends in the other banks more clear.

5.7 Use Clustering

Finally, we recommend Deloitte cluster the banks in the dataset. By clustering, we mean to group the banks based on a specific characteristic and generate a separate model each cluster. For example, instead of removing the extreme banks by net assets, Deloitte could cluster the banks into the top 10%, middle 80%, and bottom 10% by net assets, then run the model separately on each cluster. The idea is that banks within a cluster follow trends that are unique to banks in that cluster, but are obfuscated by observations outside the cluster in the general dataset. Therefore, clustering makes these individual trends clearer, resulting in improved performance in each cluster as opposed to running the model once on all banks.

6 Conclusion

Our research shows that a model using random forest scores higher on precision, recall, and F1 than one using multinomial logistic regression when used to forecast revocation of Russian bank licenses. We found that both models perform better when predicting quarters of the year instead of months. Also, our regression model is improved by preprocessing, while the performance of our random forest model worsens. The effect of excluding the top 20 and bottom 20 banks by net assets is negligible, although it improved the random forest model slightly.

To continue our work, Deloitte could research and add additional features for the Random Forest model. We found that adding features generally improves model performance. Specifically, the model would be improved by data that is recorded more frequently (eg. per week) than the data currently available, which is only updated every month. This made predicting failure on a monthly basis nearly impossible, but was sufficient for a quarterly analysis. Finally, Deloitte could investigate clustering the banks based on certain characteristics (eg. net assets, loans to businesses) and running the model on individual clusters separately. There may be trends in the data which exist only in certain clusters; separating banks in this way would help the model recognize these trends and improve model performance.

References

Amos, H. (2015). Russia's biggest banking crashes of the last 2 years. Retrieved from https://themoscowtimes.com/articles/russias-biggest-banking-crashes-of-the-last-2-years-49505

Boyacioglu, M. A., Kara, Y., & Baykan, Ö K. (2009). Predicting bank financial failures using neural networks, support vector machines and multivariate statistical methods: A comparative analysis in the sample of savings deposit insurance fund (SDIF) transferred banks in turkey. *Expert Systems with Applications, 36*(2), 3355-3366. doi:10.1016/j.eswa.2008.01.003

Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5-32. Doi:1010933404324

Capstone. (2003). *The capstone encyclopaedia of business* (1st ed.). GB: Capstone Publishing Ltd. Retrieved from http://replace-me/ebraryid=10788037

Fabozzi, F. J., Focardi, S. M., & Rachev, S. T. (2014). *Basics of financial econometrics*. Somerset, US: Wiley. Retrieved from http://site.ebrary.com/lib/alltitles/docDetail.action?docID=10845560

Fawagreh, K., Gaber, M. M., & Elyan, E. (2014). Random forests: From early developments to recent advancements. *Systems Science & Control Engineering, 2*(1), 602-609. doi:10.1080/21642583.2014.956265

Greene, W. H. (2003). *Econometric analysis* (5. ed.). Upper Saddle River, NJ: Prentice Hall.

In Russia, economic recovery remains elusive. (2015). *Stratfor,* Retrieved from https://www.stratfor.com/analysis/russia-economic-recovery-remains-elusive

Karminsky, A., & Kostrov, A. (2014). Comparison of bank financial stability factors in CIS countries. *Procedia Computer Science, 31*, 766-772. doi:10.1016/j.procs.2014.05.326

Kerr, A., Ngondi, G. E., & Butterfield, A. (2016). *A dictionary of computer science* Oxford University Press. Retrieved from http://www.vlebooks.com/vleweb/product/openreader?id=none&isbn=9780191002885&uid=none

Khalafalla Ahmed Mohamed Arabi. (2013). Predicting banks' failure: The case of banking sector in Sudan for the period (2002-2009). *Journal of Business Studies Quarterly, 4*(3), 160. Retrieved from http://search.proquest.com/docview/1476893509

Lanine, G., & Vennet, R. V. (2006). Failure prediction in the Russian bank sector with logit and trait recognition models. *Expert Systems with Applications, 30*(3), 463-478. doi:10.1016/j.eswa.2005.10.014

Liaw, A., & Wiener, M. (2002). Classification and regression from random forest. *R News, 2/3*, 18-27. Retrieved from ftp://131.252.97.79/Transfer/Treg/WFRE_Articles/Liaw_02_Classification%20and%20regression%20by%20randomForest.pdf

Lin, T. (2009). A cross model study of corporate financial distress prediction in Taiwan: Multiple discriminant analysis, logit, probit and neural networks models. *Neurocomputing, 72*(16–18), 3507-3516. doi:10.1016/j.neucom.2009.02.018

Makhonin, A. (2013). $1Bln owed to depositors as master bank has license revoked. Retrieved from https://themoscowtimes.com/articles/1bln-owed-to-depositors-as-master-bank-has-license-revoked-29777

Mitchell, T. M. (1997). Chapter 3: Decision tree learning. *Machine learning* (pp. 52-77) McGraw-Hill.

On banks and banking activities, Federal LawU.S.C. (1990).

On statutory ratios for banks, InstructionU.S.C. (2012).

Peresetsky, A., Karminsky, A., & Golovan, S. (2011). Probability of default models of Russian banks. *Economic Change and Restructuring, 44*(4), 297-334. doi:10.1007/s10644-011-9103-2

Putin's right-hand woman; Russia's central-bank governor. (2016). *The Economist, 419*(8985), 60.

The Moscow Times. (2015). 500 Russian banks to close within 5 years, VTB head says. Retrieved from https://themoscowtimes.com/articles/500-russian-banks-to-close-within-5-years-vtb-head-says-47965

Weaver, C. (2015). Russia's anatoly motylyov: Rise, fall, repeat - FT.com. Retrieved from http://www.ft.com/cms/s/0/4eecb666-52eb-11e5-b029-b9d50a74fd14.html

Workman, D. (2016). Crude oil exports by country. Retrieved from
http://www.worldstopexports.com/worlds-top-oil-exports-country/

Appendix A: Regression Models

Running regression analysis on a set of data yields an equation that estimates the outcome of an unknown statistical event. It finds application in a variety of financial analysis techniques and models in various forms. The most common of forms of regression are linear, probit, and logit regression. Regression can be categorized by the number of independent variables considered. These categories are univariate (one independent variable) and multivariate (multiple independent variables). The regression models used for forecasting bank failures are generally the latter (Fabozzi, Focardi, & Rachev, 2014; Greene, 2003).

The most basic regression technique, linear regression, describes the trend in a data set with the assumption that any trends will be linear in nature. In univariate regression, this will yield a "line of best fit" for the data. In multiple regression, the output will be a series of coefficients that describe a hyperplane. This hyperplane is analogous to the "line of best fit" seen in univariate analyses, as it provides an estimation of the general trend of the dataset. The general form of the linear regression model is:

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k + \epsilon \qquad (1)$$

where y is the dependent variable and x_i is a component of vector **x**. The error term is included to account for the inherent inaccuracy in the approximation of the model (Greene, 2003).

The second technique, probit regression, provides a non-linear analysis of a set of data. It gives as an output the probability of a binary event, with predictive values between 0 and 1. This method takes the general form:

$$P(y|X_1, X_2, \ldots, X_n) = N(\beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k) \qquad (2)$$

where N means the error term takes the cumulative normal standard distribution function. In essence this means the variables are inserted, transformed by the coefficients, and then the probability of failure is calculated based on an assumed probability density curve (Fabozzi, Focardi, & Rachev, 2014).

The logit regression model is similar but it assumes that the error term takes a different probability density function than probit regression. For the logit model, it is assumed the data takes a logistic probability form, so its general form becomes:

$$P(y|X_1, X_2, \ldots X_n) = F(\beta_0 + \beta_1 x_1 + \cdots \beta_k x_k) \qquad (3)$$

where F represents the error term taking a logistic distribution. With a little algebra, this becomes:

$$P(y|X_1, X_2, \ldots, X_n) = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\cdots+\beta_k x_k)}} \qquad (4)$$

(Fabozzi et al., 2014) Logit regression has been found to work best in some cases of estimating the probabilities of defaults in banks (Karminsky & Kostrov, 2014).

Another regression model often used in the literature is binary logistic regression, which takes the general form:

$$\log \frac{x_{avg}}{1-x_{avg}} = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k \qquad (5)$$

where $\mathbf{x}_{\mathbf{avg}}$ is the mean value of $\mathbf{x}$**.** The left side is the log of the odds function of $\mathbf{x}_{\mathbf{avg}}$. The log of the odds function is also called the logit function, expressed as logit($\mathbf{x}_{\mathbf{avg}}$). The right side is the linear combination of $\mathbf{x}$ as a vector of k regressors. When used in binary logistic regression, the value of the left side of the equation can be described as the logit(π) (Greene, 2003).

Binary logistic regression models show how a binary dependent variable (in this case the revocation/continuation of a bank license) depends on a set of independent variables $x_1...x_k$ (in this case the collection of normatives and indicators selected). The application of the regression allows the calculation of the weight of factors involved in the probability of default (PD) of a bank (Karminsky & Kostrov, 2014). In the application of the probability of default, these factors are selected ratios that compare capitalization and liquidity of banks. When the regression is run, it returns values of the coefficients of the ratios used in the model. The values of these coefficients indicate the effect that each ratio will have on whether or not the bank experiences a default.

Regression models are rarely, if ever, computed by hand. Instead, scripts are written in code. The scripts take in large sets of empirical data, and use those data points to estimate values for each coefficient in the model (Greene, 2003).

Appendix B: Random Forests

Random forests are built as a collection of decision trees. Decision trees are a method of predicting a target variable that takes discrete values. As shown in Figure 2.1, they are comprised of a series of decision nodes, where the decision at each node is based on the value of an independent variables. At the end of each branch of the tree is a leaf node that names the predicted class for that path through the tree. All independent variables are assigned to a node at some level of the tree. In some cases, if a variable is determined to be insignificant, it is omitted from the tree. When selecting which variable to test in a node, the algorithm evaluates which of the available and untested features is most telling about the value of the independent variable (Mitchell, 1997). The full ID3 algorithm for developing decision trees is as follows:

```
1. Create root node
2. If all examples are same class, return single node tree, labeled 'class'
3. If attributes[] is empty, return one node tree label = most common target value in targetSet
4. Otherwise begin
    a. A <- attribute that best describes examples
    b. The decision attribute for root <- A
    c. For each possible value $v_i$ of A
        i.   Add a branch for $v_i$
        ii.  $\text{Examples}_{vi}$ <- subset of Examples that have $v_i$ for A
        iii. IF Examples is empty then
            1. Add leaf node under this node with label = most common target value
               in targetSet
            2. ELSE- below this new branch add subtree: ID3($\text{Examples}_{vi}$, targetSet,
               Attributes - {A})
5. End
6. Return root
```

ID3 a basic recursive algorithm for decision trees defined for a set of discrete-valued features and targets. Our application uses an extension of this algorithm to divide real-valued data into discrete ranges. Also in the random forest application is a function to randomly select a subset of the available attributes at each node to encourage diversity among the many decision trees created. The attribute chosen at each node is not included in the available attributes of the next node to ensure that all significant features are evaluated at some point in the tree (Mitchell, 1997).

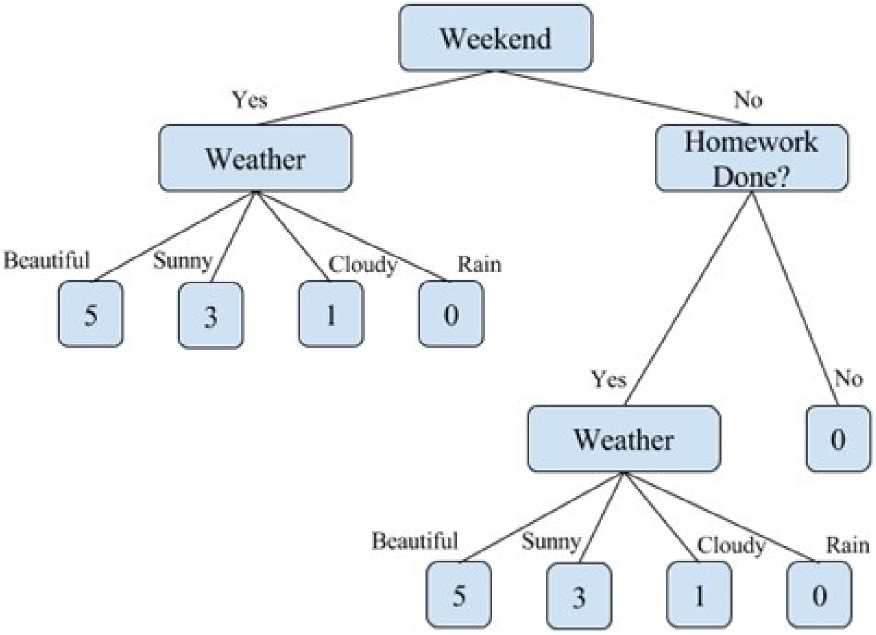


Figure 2.1: Decision Tree for how long a child may play outside. Note "Homework Done?" is omitted when weekend = yes. Based on Mitchell, 1997

There are a variety of methods for evaluating which variable is the most informative. The algorithm shown above uses the information theory concepts of entropy and information-gain to evaluate how "telling" a variable is. Another widely applied method, used by the trees in the random forest algorithm, is the Gini index. Introduced in 1912 by Carrado Gini, the Gini index is a measure of the impurity in a set of data (Fawagreh, Gaber, & Elyan, 2014). Both methods provide an indication of the level of information gained by examining that variable. Further explanation of decision trees can be found in (Mitchell, 1997).

Random forests are an example of a machine learning technique known as ensemble classification. Ensemble classification is a method of combining multiple classifiers to attain a better accuracy than any of the individual classifiers in the ensemble. The classifiers used should be accurate and diverse. Accurate classifiers have lower error rate than random guessing. Trees are considered diverse if the produced errors are different when considering new data points (Fawagreh et al., 2014). In the ensemble paradigm, each classifier develops its own conclusion of how to classify a given unlabeled example. The models then "vote" for the class label that most fits the example. There are many different voting schemes, the simplest of which is majority voting. In majority voting, each classifier votes once for the class label it produced. The class with the most votes out of all the classifiers is selected as the overall prediction of the ensemble.

There are three widely used methods of constructing the classifiers used in the ensemble, identified as boosting, bagging and stacking (Fawagreh et al., 2014). Boosting is a technique where each classifier in the ensemble is built from the misclassified data of the previous classifier. Bagging (more formally Bootstrap Aggregating) creates each classifier from a randomly selected subset (called a bootstrap sample) of the training data. Stacking combines the predictions of each classifier as inputs to a final combination algorithm, often a logistic regression model.

Random forests use bagging to build the classifiers. In this case, the classifiers are built as decision trees, each from its own bootstrap sample. In the construction of the trees, an extra element of randomness is added at each node. Rather than selecting the best-classifying decision attribute at each node, the decision attribute is selected from a randomly selected sample of the available attributes (Liaw & Wiener, 2002). The introduction of added randomness to the creation of each tree promotes diversity among them. It has been shown that increased diversity among the trees helps to reduce the overall error of the forest and improves performance.

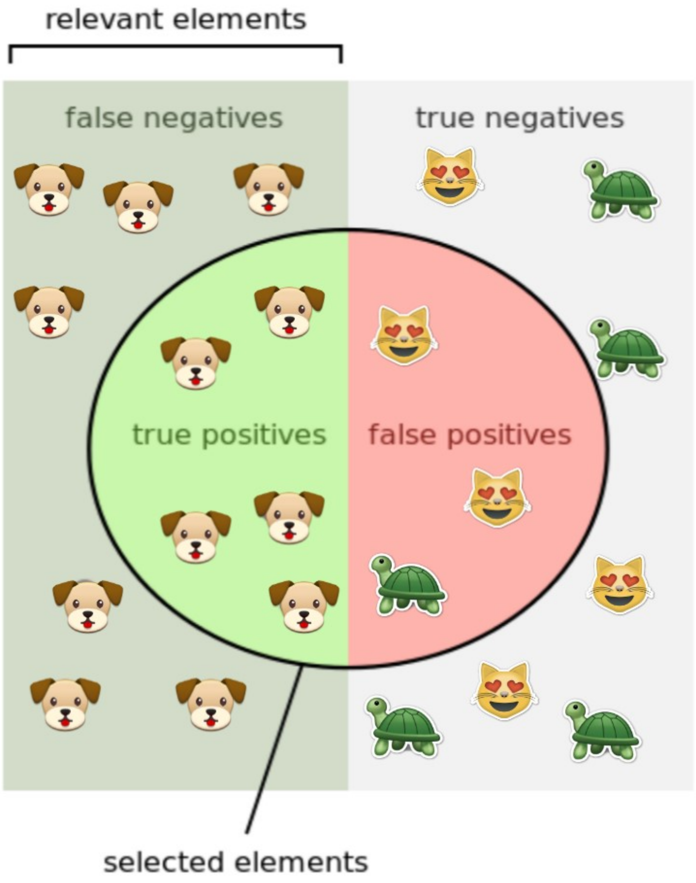
In the original paper on random forests, the advantages of random forests are outlined as follows: (Breiman, 2001)

- Comparable performance to boosting
- Robust to outliers and noise
- Faster than normal bagging or boosting
- Makes internal estimates of error, strength, correlation and variable importance
- Simple and easily parallelized (executed in parallel)

In the application of predicting bank failure, an algorithm robust to outliers and noise is a useful alternative to clustering to handle the variety in sizes of banks. Random forest provides additional advantages in its use of decision trees, which are robust to missing or incorrect attributes in the dataset. Missing or incorrect data can occur from human error in a bank’s monthly report or quarterly statements. This is not to say a random forest can detect fraud, but is theoretically resilient to accounting errors. To our knowledge this is the first application of random forests to forecasting license revocations.

Appendix C: Performance Metrics

We analyzed our models using three performance metrics called precision, recall, and F1. To explain them, consider the following example. Imagine there is a picture which contained dogs, cats, and turtles. The objective is to identify all of the dogs in the picture.



We can split the animals into four categories:

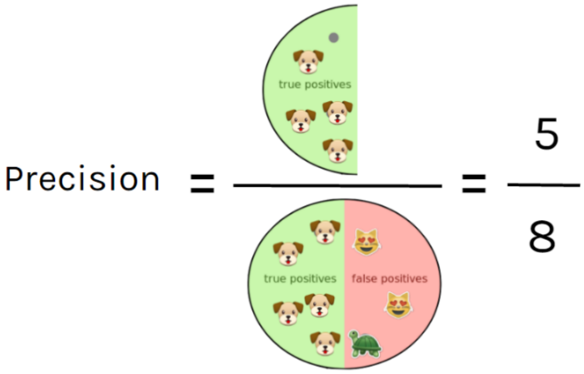
- *True positives* are dogs which were *correctly* identified as dogs
- *False positives* are cats and turtles which were *incorrectly* identified as dogs
- *False negatives* are dogs which were *failed* to identify as dogs
- *True negatives* are cats and turtles which were *correctly* identified as not dogs

There are two additional distinctions in the diagram above:

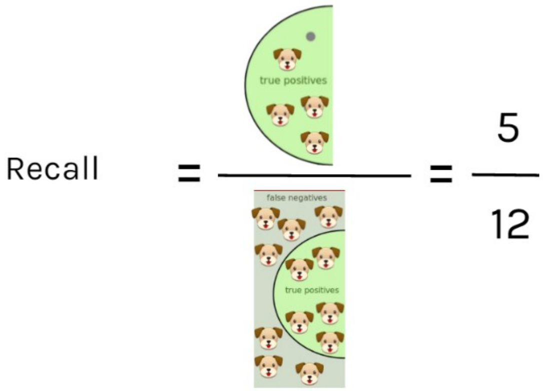
- *Selected elements* are all animals which were identified as dogs
- *Relevant elements* are all the dogs in the picture

With these definitions, we can define our first two metrics, precision and recall. They are both ratios of some of the categories defined above.

Precision is the ratio of true positives over selected elements. In terms of our example, out of all of the animals which we said were dogs, what percentage are actually dogs?



Recall is the ratio of true positives over all relevant animals. In terms of our example, out of all of the dogs in the picture, what percentage did we correctly identify?



These two ratios have identical meanings in the context of our project. Precision is a ratio between the number of correct predictions in a specific class and the total number of predictions made for that class. For example, if the model guesses that 8 banks will retain their licenses indefinitely and only 3 of those banks will actually retain their licenses, then the precision of the model is ⅜.

Recall is the ratio between the number of correct predictions in a specific class and the total number of relevant observations for that class. Continuing the previous example, if there were another 2 banks that the model did not predict would retain their licenses, but actually did, then the model's recall is ⅗ (3 correct over 3+2=5 total banks in that category).

F1 is the harmonic mean of the precision (*p*) and recall (*r*) values for each class.(scikit-learn.org, 2014) It is calculated for the *i*-th class as:

$$F1_i = \frac{p_i r_i}{p_i+r_i} * 2 \qquad (5)$$

The advantage of the harmonic mean of the first two metrics is its stability when handling outliers. The aim of calculating F1 is to find a measure of the model’s accuracy influenced equally by the precision and recall. The harmonic mean of the two ensures that if the model achieves a high precision and very low recall, the F1 score will remain low. An F1 value of 1 indicates every classification made by the model was correct (a precision value of 1) and the model classified all possible observations (a recall value of 1).

Appendix D: Model Code

multinomial_logistic_regression.py

This file contains the bulk of the model. This is the file that is run to generate the model and output results.

```python
#!/usr/bin/python

################################### PREAMBLE ###################################

import csv, sys, argparse, subprocess
import numpy as np
from timeit import default_timer
from sklearn.preprocessing import scale
from sklearn import linear_model, ensemble
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score

# Class I made for storing details (used for exporting results to txt file)
from ModelResults import ModelResults
# A file I wrote to export results
from export_test import *

np.set_printoptions(threshold=np.inf)# Turns off truncation (forces numpy to print large arrays)
np.set_printoptions(precision=3) # Sets number of printed digits

# These two arrays contain the top and bottom 20 banks in terms of net assets as ranked on Banki.ru on 27/09/16
top_banks = [1481,1000,354,2209,1623,3349,1326,3466,1978,3251,1,2562,3292,2272,2748,328,2888,436,963,2289]
bottom_banks = [3430,3420,3309,3353,2688,3318,3514,384,2605,2435,3343,3502,3509,3427,3511,3447,3483,3324,149,3332]

################################## MODEL CODE ##################################

start_time = default_timer() # To measure program execution time

print("WPI/Deloitte Model for Predicting License Revocation of Russian Banks\n")

# Argument Parsing
parser = argparse.ArgumentParser()
desc = parser.add_mutually_exclusive_group()
desc.add_argument("-d", "--description", help="Give text to describe model run type. Will be stored in description.txt")
seed_par = parser.add_mutually_exclusive_group()
seed_par.add_argument("-s", "--seed", help="Pass seed for train_test_split.")
c_par = parser.add_mutually_exclusive_group()
c_par.add_argument("-c", "--pass_c", help="Pass in value for C for model to use. Only used when running LogisticRegression")
c_test = parser.add_mutually_exclusive_group()
```

```
c_test.add_argument("-ct", "--c_test", help="Used by c_test.py to output several runs to one file; give this option the path to the file where all results should be stored. Only used when running LogisticRegression")
args = parser.parse_args()

# Write description to txt file if given
if args.description != None:
    generate_path() # Generate folders for path if they don't exist

    descFile = path + "description.txt"
    with open(descFile, "w") as fp:
        fp.write(args.description)
        fp.close()

call_parse = ["../parser/parser.py", "-s"]

normatives = ["N1", "N1_0", "N1_1", "N1_2", "N2", "N3", "N4", "N7", "N9_1", "N10_1", "N12"]
loans_to_businesses_and_institutions = ["for_a_term_of_up_to_6_months", "for_a_term_of_6_months_to_1_year", "for_a_term_of_1_year_to_3_years", "for_a_term_over_3_years"]
ratios = ["return_on_net_assets", "return_on_equity", "reserve_to_loans", "mortgaged_property_to_loans", "foreign_currency_operations_to_net_assets"]
custom_ratios = ['"(/ interbank_credit_in_cbr_turnover interbank_credit_in_cbr)"', '"(/ overdue_debt_1 overdrafts)"', '"(/ attracted_interbank_loans_from_cbr_turnover attracted_interbank_loans_from_cbr)"']
# NOTE: custom_ratios must be enclosed in double quotes for the parser to read them correctly

# Created feature set this way because of the length of the strings
# To add features, append feature strings to *this* array
features = normatives + loans_to_businesses_and_institutions + ratios + custom_ratios
call_parse += features

# Run parser to generate custom model_data.csv file
print("Generating datafile...")
try:
    subprocess.call(call_parse, cwd="../parser") # cwd = Current Working Directory
except WindowsError: # subprocess.call doesn't work on Windows
    print("\nWARNING: Auto-generation of model data not supported in Windows. Please run 'python parser.py' before running model to update model_data.csv file\n\n")

print("Importing data...")
with open('../csv/model_data.csv', 'rb') as csvfile:
    my_reader = csv.reader(csvfile)
    firstRow = True # So that loop runs different code on first row
    i = 0           # Keep track of loop number

    for row in my_reader:   # Iterate over all rows in csv
        if firstRow == False:

            target = float(row[2])  # Get target value from file
            lic_num = float(row[0]) # Get license number from file

            # Ignore negative targets (ie. values from after revocation)
            # Uncomment rest of line to exclude extreme banks
            if target > 0: # and lic_num not in top_banks and lic_num not in bottom_banks:

                # Generate array of features in this row
                new_feat = []
                for j in range(3, numFeatures+3): # Iterate over features, add to array
                    curr_feat = row[j]

                    # No data provided for this feature
                    if curr_feat == "":
                        new_feat.append(float(0))   # Meaningless value
                        new_feat.append(float(0))   # 0 = Value missing

                    else:
                        # Boolean values provided (for reporting in bounds / out of bounds)
                        if curr_feat == "True":
                            new_feat.append(float(1))   # 1 = In Bounds

                        elif curr_feat == "False":
                            new_feat.append(float(0))   # 0 = Out of Bounds

                        # Numeric value provided (normal -- value of feature)
                        else:
                            new_feat.append(float(curr_feat)) # Use value of feature

                        new_feat.append(float(1)) # 1 = Value present

                X = np.concatenate(( X, np.array([new_feat]) )) # Add new feature set to array

                # Uncomment to run on all 26 classes, and comment out the QUARTERLY block
                # Y = np.append(Y, target)

                # QUARTERLY
                if target <= 3: Y = np.append(Y, 1)
                elif target <= 6: Y = np.append(Y, 2)
                elif target <= 9: Y = np.append(Y, 3)
                elif target <= 12: Y = np.append(Y, 4)
                elif target <= 15: Y = np.append(Y, 5)
                elif target <= 18: Y = np.append(Y, 6)
                elif target <= 21: Y = np.append(Y, 7)
                elif target <= 24: Y = np.append(Y, 8)
                else: Y = np.append(Y, target)


        else:
            firstRow = False
            numFeatures = len(row)-3 # Everything past first three columns are features

            # Create array of feature labels
            feature_labels = []
            for j in range(3, numFeatures+3):
                feature_labels.append(row[j])       # Add feature name
                feature_labels.append("%s_M?" % row[j]) # For "missing" column (1 if feature present, 0 if not)

            # Create the feature and target datasets respectively
            X = np.empty((0,numFeatures*2), float) # Create empty array with shape attributes (so it can be used in concatenate)
            Y = np.array([])

        # Print dots to indicate progress
        if i % 350 == 0:
            sys.stdout.write('.')
            sys.stdout.flush()
        i += 1

feature_labels = np.array(feature_labels) # Convert feature_labels to a numpy ndarray

# Split data into testing & training, with 66% training, 33% testing
# If seed passed, use it
# Otherwise, seed used to keep split consistent
if args.seed != None:
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=int(args.seed), stratify=Y)
else:
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=42, stratify=Y)

# Data preprocessing (uncomment below to add preprocessing)
# scale(X_train, copy=False)
# scale(X_test, copy=False)

print("\nFitting model...")
# Store results in ModelResults object
results = ModelResults(X_train, X_test, Y_train, Y_test, feature_labels)

# If passed, use passed in C value
# c_val is only relevant to LogisticRegression, but for compatibility with
# ModelResults objects, a value is included either way
if args.pass_c != None: c_val = float(args.pass_c)
else: c_val = 0.01

# Create the model & fit to training data
# Uncomment line to run LogisticRegression
# model = linear_model.LogisticRegression(penalty='l1', multi_class='ovr').fit(X_train, Y_train)
model = ensemble.RandomForestClassifier().fit(X_train, Y_train)

print("Generating predictions...")
predict_arr = model.predict(X_test)   # Run a prediction for test dataset (ie. compare this array to Y_test)
prob_arr = model.predict_proba(X_test) # Runs prediction, outputs probability vectors

print("Evaluating performance...")
precision = precision_score(Y_test, predict_arr, average=None) # Calculate the precision
recall    = recall_score(Y_test, predict_arr, average=None) # Calculate the recall
f1        = f1_score(Y_test, predict_arr, average=None) # Calculate f1
```

```

exec_time = default_timer() - start_time # Calculate execution time

# Add results to "results" object
# Uncomment this line when running LogisticRegression
# results.addResults(c_val, model.coef_, predict_arr, prob_arr, precision, recall, f1, exec_time)
results.addResults(c_val, model.feature_importances_, predict_arr, prob_arr, precision, recall, f1, exec_time)

# If running with c_test option, export results
# Uncomment for LogisticRegression
#if args.c_test != None:
#   export_c_test(results, args.c_test)
#   exit() # Quit early so full results aren't exported

print("\nTotal Execution Time: %f seconds\n" % exec_time)

# Uncomment for LogisticRegression
#print("\nC: %s\n" % c_val)
print("Precision: %s\n" % precision)
print("Recall: %s\n" % recall)
print("f1: %s\n" % f1)

print("Exporting results...")
export_test(results)
print("Data sets and results exported to %s" % path)

################################################################################
```

ModelResults.py

This file contains the definition of the ModelResults class, which is used to store the results of model execution and pass them to export_test() for the creation of results files.

```
import numpy as np
from math import log10, floor

class ModelResults:

############################### CLASS FUNCTIONS ################################

 def __init__(self, X_train, X_test, Y_train, Y_test, feature_labels):
     # Round values in X array before storing
     self.round_2d_arr(X_train, 3)
     self.round_2d_arr(X_test, 3)

     self.X_train = X_train
     self.X_test  = X_test
     self.Y_train = Y_train
     self.Y_test  = Y_test
     self.feature_labels = feature_labels

 def addResults(self, C, coef, predict_arr, prob_arr, precision, recall, f1, exec_time):

     self.C       = C
     self.coef    = coef
     self.predict_arr = np.array(predict_arr)
     self.prob_arr    = np.array(prob_arr)
     self.precision = np.array(precision)
     self.recall    = np.array(recall)
     self.f1        = np.array(f1)
     self.exec_time = exec_time

############################### HELPER FUNCTIONS ###############################

 # Rounds x to n significant figures
 def n_sig_figs(self, x, n):
     # x = 10 ^ (log10(x)), the floor of which is the number of places the first significant digit is away from the decimal point (in one direction)
     # And the round() function rounds n places after the decimal place (or before, if n is negative)
     # So without adding in -(n-1), it rounds to one significant figure
     if x == 0: return 0
     return round(x, -(int(floor(log10(abs(x))))-(n-1)))

 # Rounds all the elements in an 2d numpy array to n significant figures
 def round_2d_arr(self, arr, n):
     bounds = arr.shape

     for i in range(bounds[0]):      # Iterate over vectors in array
         for j in range(bounds[1]):  # Iterate over elements in vector
             arr[i][j] = self.n_sig_figs(arr[i][j], n)

################################################################################
```

export_test.py

This file contains functions for writing results of model execution to files. The model calls the export_test() function to do most of the exporting; some other functions are used in other parts of the project.

```
################################### PREAMBLE ###################################

import os
import datetime
import numpy as np

# Class I made for storing details (& exporting to txt file)
from ModelResults import ModelResults

############################# CREATE PATH VARIABLE #############################

date = str(datetime.date.today())
time = str(datetime.datetime.now().time())[:8]
```

```python
time = time.replace(":", "-") # Replace colons - Windows paths cannot have them

path = "../out/%s/%s/" % (date, time)

############################### HELPER FUNCTIONS ###############################

# Generates the folders necessary for the path variable (where the output goes)
def generate_path():
    # Create folders if they don't exist
    if not os.path.exists(path):
        os.makedirs(path)

# Saves arr to filename.csv
def output_arr(arr, filename):
    var_path = path + filename      # Create full path to file
    np.savetxt(var_path, arr, delimiter=',')# Save array as csv

# Adds header before exporting to csv
def output_arr_w_header(arr, filename, header):
    # Create header string
    head_str = str(header[0])
    for i in range(1, header.size):
        head_str = head_str + "," + str(header[i])
    head_str = head_str + "\n"

    # Output file with header
    var_path = path + filename
    with open(var_path, 'w') as outfile:
        outfile.write(head_str)
        np.savetxt(outfile, arr, delimiter=',')
        outfile.close()

################################ FILE WRITERS ##################################

# Creates four files, one for each data set (X_train, X_test, Y_train, Y_test)
# Also export value of C
def export_data_sets(r):
    output_arr_w_header(r.X_train, "X_train.csv", r.feature_labels)
    output_arr_w_header(r.X_test, "X_test.csv", r.feature_labels)
    output_arr(r.Y_train, "Y_train.csv")
    output_arr(r.Y_test, "Y_test.csv")

    # Uncomment for LogisticRegression
#   c_path = path + "c.txt"
#   with open(c_path, 'w') as outfile:
#       outfile.write(str(r.C))
#       outfile.close()

# Creates files for all results
def export_results(r):
    time_path = path + "execution_time.txt"
    with open(time_path, 'w') as outfile:
        mins = 0
        while r.exec_time > 60:
            mins += 1
            r.exec_time -= 60
        outfile.write(str(mins) + " minutes and " + str(r.exec_time) + " seconds")
        outfile.close()

    output_arr_w_header(r.coef, "coef.csv", r.feature_labels)
    output_arr(r.predict_arr, "predict_array.csv")

    # Create header with target values for some results files
    header = np.arange(1,9)
    header = np.append(header, [1000, 9000])

    output_arr_w_header(r.prob_arr, "probability_vectors.csv", header)
    output_arr(r.precision, "precision.csv")
    output_arr(r.recall, "recall.csv")
    output_arr(r.f1, "f1.csv")

# Writes run reports to txt files, returns paths to files
# r = a ModelResults object
def export_test(r):
    generate_path()

    export_data_sets(r)
    export_results(r)

# Used by c_test.py to compare several C values on the same model
# Generally don't want to use this manually
def export_c_test(r, filePath):
    with open(filePath, "a") as myfile:
        # Print header with list of features (will print each iteration, not worth trying to fix)
        head_str = "Used features: " + str(r.feature_labels[0])
        for i in range(1, r.feature_labels.size):
            head_str += ", " + str(r.feature_labels[i])
        head_str += "\n\n"
        myfile.write(head_str)

        print("C: %f" % r.C)
        myfile.write("C: %f\n" % r.C)
        myfile.write("Precision:\n%s\n\n" % str(r.precision))
        myfile.write("Recall:\n%s\n\n" % str(r.recall))
        myfile.write("F1:\n%s\n\n" % str(r.f1))
        myfile.close()

################################################################################
```

rand_test.py

This file contains a script for generating random model results. It requires a Y_test file (containing sample data). The script generates an array of random predictions and exports the precision, recall, and F1 scores. We used this for comparing our model's performance vs. random guessing.

```
import numpy as np
import argparse
from random import randrange
from export_test import * # For outputting results to txt
from sklearn.metrics import precision_score, recall_score, f1_score

# Argument Parsing
parser = argparse.ArgumentParser()
runType = parser.add_mutually_exclusive_group()
runType.add_argument("-t", "--runType", help="Pass 'month' or 'quarter'")
yArr = parser.add_mutually_exclusive_group()
yArr.add_argument("-y", "--y_test", help="Pass filename file containing Y_test from model run")
args = parser.parse_args()

if args.y_test == None:
 print("ERROR: Provide Y_test file with the '-y' option.")
 exit()

# Load Y_test (expected values)
Y_test = np.loadtxt(args.y_test)

# Create random array
rand_arr = np.array([])

# Generate a random array the same size as Y_test
for i in range(0, Y_test.size):
 # Generate new random value (in range depending on type)
 if args.runType == "month":
     val = randrange(1, 27)
     if val == 25: val = 1000
     elif val == 26: val = 9000
 elif args.runType == "quarter":
     val = randrange(1, 11)
     if val == 9: val = 1000
     elif val == 10: val = 9000
 else:
     print("ERROR: Input '-t' and either 'month' or 'quarter'")
     exit()

 # Add random value to array
 rand_arr = np.append(rand_arr, val)

precision = precision_score(Y_test, rand_arr, average=None)  # Calculate the precision
recall    = recall_score(Y_test, rand_arr, average=None) # Calculate the recall
f1        = f1_score(Y_test, rand_arr, average=None)     # Calculate f1

print("Precision: %s\n" % precision)
print("Recall: %s\n" % recall)
print("f1: %s\n" % f1)

generate_path() # Generate results filepath
output_arr(precision, "precision.csv")
output_arr(recall, "recall.csv")
output_arr(f1, "f1.csv")
```

c_test.py

This file contains a script for testing the model on different values of C. C is a parameter for the LogisticRegression model, so this script is only run on the regression model.

```
################################### PREAMBLE ###################################

import subprocess
import numpy as np

from export_test import *

# USAGE: This script runs model.py on the same seed with different C values to
#        compare them. The vals array below contains the C values that the script
#        will iterate over. Change it to include whichever values you want.
#        Make sure model.py is properly uncommented to run LogisticRegression (default is RandomForest)

#################################### SCRIPT ####################################

# Array of C values to run model on
vals = np.array([ 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 0.75, 0.9, 0.95, 1.0 ])

fileName = path + "c_test.txt" # Create header of results file

generate_path() # Create folders if they don't exist

with open(fileName, "w") as myfile:
 myfile.write("Results of C-value testing\n")
 myfile.close()

# Iterate over test values for C, run model on same seed to get comparative results
for C in vals:
 subprocess.call(["./model.py", "-ct", fileName, "-c", str(C), "-s", "42"])
print("\nResults stored in %s" % fileName)
```

Appendix E: Parser Code

load_banki.py

Downloads and merges indicator and license revocation data from banki.ru. Months are calculated by finding the difference between the observation date from indicator dataset with revocation date from revocation dataset. Accepts update or redownload requests.

```python
import pandas as pd
import numpy as np
import datetime as dt
from sys import stdout
import os
import pdb
import signal

ERASE_LINE = '\x1b[2K'
RETURN = '\r' + ERASE_LINE

# Global 90 second timeout for read_csv from banki. See handler below.
TIMEOUT = 90

execfile("dictionaries.py")

def timeout_handler(signum, frame):
    raise Exception('\n' + str(TIMEOUT) + ' second timeout reached. Continuing...')

# Updates or redownloads banki indicator and revocation datasets.
# Calculates months. Merges with CBR.
# returns complete DataFrame.
def load_banki(update = False, redownload = False):

    if update:
        print 'Updating...'
        banki    = update_banki()
        revoked  = load_banki_revoked(update=True)
        complete = complete_banki(banki, revoked)
    elif redownload:
        print 'Redownloading...'
        banki    = redownload_banki()
        revoked  = load_banki_revoked(redownload=True)
        complete = complete_banki(banki, revoked)
    else:
        complete = pd.read_csv('../csv/banki_complete.csv', index_col=False)
        complete['period'] = pd.to_datetime(complete['period'])

    return complete

# Download a set of indicator observations for a given month for all banks.
# Clean, and add metadata to remember indicator and observation dates.
# param ind Indicator to download, the banki-defined ID.
# param year Year to download.
# param month Month to download.
# returns DataFrame
def download_ind_from_date(ind, year, month):

    now = dt.datetime.now()

    # Skip downloads for future dates.
    if year == now.year and month > now.month:
        return None

    # We're iterating downwards in time, where month and year
    # are ending dates, not starting. If the ending date is January, then
    # starting date is December last year.
    if month == 1:
        start_month = 12
        start_year = year - 1
    else:
        start_month = month - 1
        start_year = year

    # Convert to strings:
    month       = str(month)
    year        = str(year)
    start_month = str(start_month)
    start_year  = str(start_year)

    # Create date strings for banki request.
    time_end = year + "-" + month + "-01"
    time_start = start_year + "-" + start_month + "-01"

    url = "http://www.banki.ru/banks/ratings/export.php?LANG=en&" + \
        "PROPERTY_ID=" + str(ind) + \
        "&search[type]=name&sort_param=rating&sort_order=ASC&REGION_ID=0&" + \
        "date1=" + time_end + \
        "&date2=" + time_start + \
        "&IS_SHOW_GROUP=0&IS_SHOW_LIABILITIES=0"

    stdout.write(RETURN)
    stdout.write( '\rDownloading ' + get_ind(ind) + ', ' + time_end[0:-3])
    stdout.flush()

    # Download banki table.
    signal.signal(signal.SIGALRM, timeout_handler)
    signal.alarm(TIMEOUT)
    try:
        df = pd.read_csv(url, delimiter=";", skiprows=3, error_bad_lines=False,
warn_bad_lines=True)
    except Exception, e:
        df = pd.DataFrame()
        print e

    signal.alarm(0) # Cancel signal alarm

    # Cleanup data if it exists.
    if not df.empty:

        nine_cols = ['rating','rating_change','bank_name','lic_num',
            'region','ind_val','ind_start','change','perc_change']

        eight_cols = ['rating_change','bank_name','lic_num','region',
            'ind_val','ind_start','perc_change','empty']

        # Rename and drop columns for 9 or 8 tables.
        if len(df.columns) == 9:
            df.columns = nine_cols
            df.drop(['rating', 'change'], axis=1, inplace=True)
        else:
            df.columns = eight_cols
            df.drop('empty', axis=1, inplace=True)

        # Drop shared columns
        df.drop(['bank_name', 'region','rating_change', 'ind_start','perc_change'],
            axis = 1, inplace = True)

        # Convert indicator value to a number by first removing the
        # white spaces and replacing the comma decimal with the point decimal.
        df['ind_val'] = df['ind_val'].str.replace(' ', '').str.replace(',', '.')
        df['ind_val'] = pd.to_numeric(df['ind_val'])

        # Remember important details about this dataset, such as
        # the indicator and date.
        df['ind']     = pd.Series(get_ind(ind), index=df.index)
        df['period'] = pd.Series(time_end, index=df.index)
        df['period'] = pd.to_datetime(df['period'])
    else:
        df = None

    return df

# Make banki wide or banki tall using pd.pivot_table
# param df Generally, new DataFrame on export, banki.csv on import.
# param wide_or_tall
    # -> 'wide' to spread indicators over columns
    # -> 'tall' to gather indicators into rows
def pivot_banki(df, wide_or_tall=None):

    if wide_or_tall == 'wide':
        df = pd.pivot_table(df, index=['lic_num', 'period'],
            columns='ind', values='ind_val')
        df.reset_index(inplace=True)

    if wide_or_tall == 'tall':
        df = pd.melt(df,id_vars=['lic_num','period'],
            value_vars = list(df.columns[2:]),
            var_name = 'ind', value_name = 'ind_val')
        df['period'] = pd.to_datetime(df['period'])

    return df

# Redownload banki from scratch, using the indicators
# defined in dictionaries.py. Writes to banki.csv.
# returns DataFrame
def redownload_banki():

    now = dt.datetime.now()
    ind_codes = ind_dict_banki_ru()['ind_num'] # Defined in dictionaries.py
    final = pd.DataFrame()

    for ind in ind_codes:
        for year in range(now.year, 2007, -1):
            for month in range(12, 0, -1):
                tmp = download_ind_from_date(ind, year, month)
                # If tmp is None, then skip.
                if type(tmp) == pd.DataFrame:
                    final = final.append(tmp)

    final = export_banki(final)

    return final

# Update banki using local banki.csv. Writes to banki.csv.
# returns DataFrame
def update_banki():

    ind_codes = ind_dict_banki_ru()['ind_num'] #  Defined in dictionaries.py
    banki = pd.read_csv('../csv/banki.csv', index_col=False)

    # Revert wide banki to tall, so we can append and compare it
    # with the temporary tables we download from banki.ru
    banki = pivot_banki(banki, wide_or_tall='tall')

    # Here, the year/month for loops are refactored to the function,
    # help_iterdate_update() so we can break from downlading
    # an indicator once we see duplicates.
    for ind in ind_codes:
        final = help_iterdate_update(ind, banki)

    final = export_banki(final)

    return final

# Helps update_banki. Starts at the current date, moving backwards.
# Only downloads up to one set of duplicates, then moves on
# to the next indicator.
# param ind Indicator to download
# param banki Local banki.csv.
# returns DataFrame
def help_iterdate_update(ind, banki):

    now = dt.datetime.now()

    for year in range(now.year, 2007, -1):
        for month in range(12, 0, -1):
            tmp = download_ind_from_date(ind, year, month)
            if type(tmp) == pd.DataFrame:
                banki = banki.append(tmp)
                dups = banki.duplicated(['lic_num','period','ind'], keep=False)
                if any(dups):
                    banki.drop_duplicates(['lic_num','period','ind'],inplace=True)
                    return banki

# Takes final banki download, pivots, sorts, writes to file.
# param final Final banki download.
# returns DataFrame
def export_banki(final):

    final = pivot_banki(final, wide_or_tall='wide')
    final.sort_values(['lic_num', 'period'],
        ascending=[True, False], inplace=True)

    print 'Writing banki.csv...'
    final.to_csv('../csv/banki.csv', index=False)

    return final

# banki.ru records the dates of license revocations in webpage-separated
# html tables. One webpage has 50 records. When the page index becomes
# greater than the actual number of pages, instead of showing an empty
# table, it returns to Page 1, but the index keeps incrementing.
# Highly inconvenient.
#
# Instead, in order to detect when we've reached the end of banki's
# dataset, we'll keep downloading until we find the first duplicate.
# If a csv already exists, then we'll check it first, then check
# banki's most recent records.

# param update If banki_revoked.csv exists then it will
#              fetch updates from banki.ru
# param redownload Download banki.ru's full dataset again.
# returns Pandas Dataframe of banki.ru's license revocations. Writes to csv.
def load_banki_revoked(update=False, redownload=False):

    print "Loading banki.ru's license revocations dataset."

    br_file = "../csv/banki_revoked.csv"
    # If the user wants to redownload then redownload.

    if redownload:
        banki_revoked = pd.DataFrame()
    # If the user doesn't want to update it,
    #   then read the csv and return it as a pandas DataFrame.
    # If they want to update it, then set banki_revoked to local dataset,
    #   and read banki.ru, stopping as soon as there are duplicates.
    elif update:
        print "Updating..."
        banki_revoked = pd.read_csv(br_file, index_col=False)
        banki_revoked['revoc_date'] = pd.to_datetime(banki_revoked['revoc_date'])
    else:
        return pd.read_csv(br_file)

    i = 1
    while True:
        sys.stdout.flush()
        sys.stdout.write("\rReading Pages... " + str(i))
        url = 'http://www.banki.ru/banks/memory/?PAGEN_1=' + str(i)

        # The content we want is in [2] of the returned web thingy.
        signal.signal(signal.SIGALRM, timeout_handler)
        signal.alarm(TIMEOUT)
        try:
            tmp = pd.read_html(url)[2]
        except Exception, e:
            print e
            tmp = pd.DataFrame()
        signal.alarm(0) # Cancel signal alarm

        # Rename columns
        tmp.columns = ['idx', 'bank', 'lic_num', 'cause', 'revoc_date', 'region']
        tmp.drop(['idx', 'bank', 'region', 'cause'], axis=1, inplace=True)
        tmp['revoc_date'] = pd.to_datetime(tmp['revoc_date'])

        # Remove rows whose license numbers have "-" + character.
        if tmp['lic_num'].dtype == 'object':
            tmp = tmp[~tmp.lic_num.str.contains("-")]
        tmp['lic_num'] = tmp['lic_num'].astype(int)
        banki_revoked = banki_revoked.append(tmp)

        # As soon as we find a duplicate, break.
        d = banki_revoked.duplicated(['lic_num', 'revoc_date'])
        if any(d): break
        i += 1

    print 'Cleaning...'
    # Actually remove duplicates.
    banki_revoked.drop_duplicates(['lic_num', 'revoc_date'],
                                  keep = False, inplace = True)
    banki_revoked.reset_index(inplace=True, drop=True)
    banki_revoked.sort_values(['revoc_date', 'lic_num'],
                              ascending=[False,True], inplace=True)

    print "Writing to file..."
    banki_revoked.to_csv("../csv/banki_revoked.csv", index=False)

    print "Returning dataset."
    return banki_revoked

# Takes banki dataset and banki's revoked dataset and merges them.
# param banki Indicator dataset
# param revoked License revocation dataset
# Writes to banki_complete.csv
# returns DataFrame
def complete_banki(banki, revoked):

    print 'Completing banki...'

    merged = banki.merge(revoked, how='left', on='lic_num')

    merged['period'] = pd.to_datetime(merged['period'])
    merged['revoc_date'] = pd.to_datetime(merged['revoc_date'])

    print '    Calculating months...'
    for row in merged.itertuples():
        if pd.notnull(row[-1]):
            merged = merged.set_value(row[0], 'months',
                12 * (row[-1].year - row[2].year) +
                (row[-1].month - row[2].month))
        else:
            merged = merged.set_value(row[0], 'months', 9000)

    for row in merged.itertuples():
        if row[-1] > 24 and row[-1] < 9000:
            merged = merged.set_value(row[0], 'months', 1000)


    merged.drop('revoc_date', axis=1, inplace=True)

    # Moves 'months' from last column to the third column.
    cols = merged.columns.tolist()
    cols = cols[0:2] + cols[-1:] + cols[2:-1]

    complete = merged[cols]

    print '    Merging with local CBR file...'

    cbr = pd.read_csv('../csv/cbr_standards_complete.csv', index_col=False)
    cbr['period'] = pd.to_datetime(cbr['period'])
    cbr.drop(['N1','N2','N3'], axis = 1, inplace=True)

    complete = complete.merge(cbr, how='outer', on=['lic_num', 'period', 'months'])
    complete.to_csv('../csv/banki_complete.csv', index=False)

    print '    Returning...'

    return complete
```

parser.py

Combines all the functionalities of the previous script into a command-line program. The user can update or redownload the indicator and revocation datasets from banki.ru. The user can also specify the inclusion of special indicator calculations, such as testing if an indicator is within its CBR-required/analyst-recommended range, or performing custom calculations between indicators, such as dividing net assets by number of loans. The result of running parser.py is model_data.csv, which is fed into the script which runs the model.

```
#!/usr/bin/python

import pandas as pd
import numpy as np
import datetime as dt
import argparse
import string

pd.options.mode.chained_assignment = None
pd.set_option('mode.use_inf_as_null', True)

###############################
## Set Command Line Options
###############################

# Initialize Argument Parser object.
parser = argparse.ArgumentParser()

# Banki indicator dataset argument group.
# One can only update or redownload, but not both.
group_banki = parser.add_mutually_exclusive_group()

group_banki.add_argument(
    '-u',
    '--update',
    action='store_true',
    help='Update banki.ru indicator and revocation datasets.'
    )
group_banki.add_argument(
    '-r',
    '--redownload',
    action='store_true',
    help='Redownload banki.ru indicator and revocation dataset.'
    )

# Print indicators one can use for the model and exit.
```

```
parser.add_argument(
    '-i',
    '--indicators',
    action='store_true',
    help='Prints list of available indicators from banki.ru and CBR and exits.'
    )

# User can pass column names to the script
parser.add_argument(
    '-s',
    '--select',
    metavar='COLUMNS',
    nargs='*',
    help='Select columns from indicator dataset to use in model. \
    To receive binary indicators (ie. is indicator in acceptable range) \
    add a bang after the indicator (ie. "N1!"). To calculate values between \
    indicators, use Lisp notation: "(/ (+ net_assets net_profit) loans)" Make sure
to use double-quotes!'
    )

args = parser.parse_args()

##################################
## Load Local Scripts, File Paths,
## and Variables
##################################

execfile("dictionaries.py")
execfile("load_banki.py")

# Local datasets:
banki          = "../csv/banki.csv"           # Banki indicators, no months.
banki_revoked  = "../csv/banki_revoked.csv"   # Banki revocation dates.
banki_complete = "../csv/banki_complete.csv"  # Banki inds. and months.

# Banki indicator names and cbr hacked in there.
banki_ind_names = ind_dict_banki_ru()['ind_name'] + cbr_standards()

if args.indicators:
    print 'Selectable indicators to run for the model:'
    for i in banki_ind_names:
        print i
    raise SystemExit(0)




##################################
## Function Definitions
##################################

# Take the column select argumens (from -s, --select), and put them
# into bins for further processing.
```

```python
# param select The argument list from args.select
def parse_select(select):

    singular_cols = []   # Keep these basic columns.
    range_cols    = []   # Columns are true/false if the Indicator is in range.
    equation_cols = []   # These are equations between columns.

    # Iterate through each selected item from the command-line.
    for i in select:
        # If i is immediately found in the list of indicators,
        # then is a singular, basic column.
        if i in banki_ind_names:
            singular_cols.append(i)
        # Now, look for ! operator
        else:
            bangs = i.find('!')
            paren = i.find('(')
            if bangs != -1:
                ind_name = string.split(i, '!')[0]
                range_cols.append(ind_name)
            elif paren != -1:
                equation_cols.append(i)
            else:
                print i, 'is ivalid. Run `python parser.py -i` for full list of indicators.'
                raise SystemExit(0)

    return [singular_cols, range_cols, equation_cols]

# Parses equation string into recursive list for eval_eq.
# Example:
#   in: '(/ (+ net_assets net_profit) loans)'
#  out: ['(+ net_assets net_profit)', '(/ (+ net_assets net_profit) loans)']
def eq_helper(eq):

    parens = [] # Keeps list of index location of open parentheses.
    parts  = [] # Collects parts of equation string.

    # Iterate through equation string, remembering positions of open parens,
    # and collecting substrings between open and closed parens.
    for c in range(0, len(eq)):
        # Remember position of open positions.
        if eq[c] == '(' : parens.append(c)
        # Create substring from open to closing paren.
        if eq[c] == ')' : parts.append(eq[parens.pop() : c+1])

    return parts

# Takes output from eq_helper to calculate operations on columns.
# param eq_list The list returned from eq_helper.
def eval_eq(eq_list):

    # banki_complete.csv has all the indicators, so we can use it
    # to calculate new values for our current dataframe.
    banki = pd.read_csv('../csv/banki_complete.csv', index_col=False)

    # Create temporary columns for extended operations.
    tmp_col = 0

    # By the final calculation, eq_list will become the value
    # of the Series we want.
    while type(eq_list) != pd.Series:

        this = eq_list[0]          # Set working equation part to first element.
        operator = this[1]            # Set operator.
        operands = this[3:-1].split() # Set operands.

        # Check names for validity.
        bad_names = []
        for op in operands:
            if not op in list(banki.columns):
                bad_names.append(op)
        if len(bad_names) > 0:
            print 'There were invalid indicator names found in equation',this
            print 'Invalid:', bad_names
            print 'Run `python parser.py -i` for full list of indicators.'
            raise SystemExit(0)

        # The result begins with the value of the first operand.
        tmp_result = banki[operands[0]]

        # For each operand.
        for op in range(1, len(operands)):
            # Set the column with which to operate.
            op_value = banki[operands[op]]

            # Math.
            if operator == '/': tmp_result = tmp_result / op_value
            if operator == '*': tmp_result = tmp_result * op_value
            if operator == '+': tmp_result = tmp_result + op_value
            if operator == '-': tmp_result = tmp_result - op_value
            if operator == '^': tmp_result = tmp_result ** op_value

        # If len > 1, there are more parts to calculate.
        if len(eq_list) > 1:
            # Bump up the temporary column.
            tmp_col += 1
            # Set the temporary column to our temporary result.
            banki[str(tmp_col)] = tmp_result
            # Replace the part of the equation we just calculted with the
            # name of temporary column. It will be called in next calculations.
            eq_list[-1] = eq_list[-1].replace(this, str(tmp_col))
            # Remove first element, now that we've finished.
            eq_list = eq_list[1:]
            # If there are no more elements, then tmp_result is final result.
        else:
            eq_list = tmp_result

    return eq_list


# Puts all the equation helper functions together to add string equations passed
# from the command line to the model_data DataFrame.
# param df The DataFrame to which to add the results of the equations.
# param eq_list The string equations passed from the command line.
def add_eqs(df, eq_list):

    # For each equation passed from the command line...
    for eq in eq_list:
        # Get the recursive form of the equation.
        recursive_eq_str = eq_helper(eq)
        # Evaluate the column.
        col = eval_eq(recursive_eq_str)
        # Create a syntatically appropriate column name.
        eq_str_col_name = eq.replace(' ','_')
        # Add it to the DataFrame
        df.insert(len(df.columns), eq_str_col_name, col)

    #df.replace(np.inf, np.nan)

    return df


# Parse args.select
if args.select == None: args.select = []
parsed_args = parse_select(args.select)

singular_cols  = parsed_args[0]
range_cols     = parsed_args[1]
equation_cols  = parsed_args[2]

print 'Loading banki.ru...'
banki_complete = load_banki(update = args.update, redownload=args.redownload)

### Add singular columns first. ###
model_data = banki_complete[['lic_num','period','months'] + singular_cols]

### Add ! columns. ###
if len(range_cols) > 0: print 'Calculating ranges...'
for col in range_cols:

    r = get_ratio(col) # Ratios defined in dictionaries.py

    model_data[col + '!'] = banki_complete[col].apply(
        lambda x: r[0] <= x <= r[1] if pd.notnull(x) else None
        )

### Add equation columns. ###
if len(equation_cols) > 0: print 'Evaluating expressions...'
```

```
model_data = add_eqs(model_data, equation_cols)

### Finish ###

print 'Writing to csv...'
model_data.to_csv('../csv/model_data.csv', index=False)
```

Appendix F: Glossary

The following definitions were taken from “The Capstone Encyclopaedia of Business” (Capstone, 2003; Kerr, Ngondi, & Butterfield, 2016) and from “a dictionary of computer science” (Kerr et al., 2016)

Application Programming Interface (API)	A set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service
Asset	A resource with economic value.
Bank	Licensed Credit Financial Institution.
Bank Failure	A bank has lost its license to operate.
Bank License	License a credit financial institution needs in order to operate.
Capital	Wealth in the form of money or other assets owned by a person or organization or available or contributed for a particular purpose such as starting a company or investing.
Capital Adequacy	Minimum of reserves that a bank needs to have available.
Cash	Money in coins or notes, as distinct from checks, money orders, or credit.
CBR	Central Bank of Russia. The CBR has established numerical standards which every bank must follow or else their license will be revoked.
Common Capital Ratio	Common capital ratio is a measurement of a bank's core equity capital compared with its total risk-weighted assets that signifies a bank's financial strength.
Common Stocks	Ordinary shares
Credit Claims	Pecuniary claims arising out of an agreement whereby a credit institution grants credit in the form of a loan.
Credit Risk	It is the risk of default on a debt that may arise from a borrower failing to make required payments

Current Assets	Assets that can be quickly liquidated (eg. Gold, oil, stocks)
Current Liability	Company's obligations that are due within the next 12 months
Depreciation	Decrease of a tangible asset's price over time due to wear
Equity	The value of the shares issued by a company.
F1	F1 a performance metric given by the following formula: 2(Precision*Recall)(Precision+Recall)
Forensic Accounting	The specialty practice area of accounting that describes engagements that result from actual or anticipated disputes or litigation.
Guarantees	A formal pledge to pay another person's debt or to perform another person's obligation in the case of default.
Insiders	A person within a group or organization, especially someone privy to information unavailable to others.
Intangible Assets	An asset without a physical form (eg. stock, I.P. rights)
Liability	A company's legal debt or obligation.
Liquidate	To convert an asset into cash.
Liquidity	Ability of a financial institution to meet its short-term commitments.
Logit Regression Model	Method of mathematically determining the probability of a binary dependent variable from a collection of one or more continuous independent variables with logistic distribution.
Machine Learning	A field study that allows computers to learn without being explicitly programmed.
On Call Obligations	Amount of money that needs to be daily available for customer transactions
Parser	A program that analyses a string or text into logical syntactic components.

Precision	The number of relevant items over the total of selected items
Preferred Stocks	A share which entitles the holder to a fixed dividend, whose payment takes priority over that of ordinary share dividends.
Probit Regression Model	Method of mathematically determining the probability of a binary dependent variable from a collection of one or more continuous independent variables with normal distribution
Python	Dynamic programming language.
Random Forest	Ensemble learning method for classification, regression, etc.
Recall	The number of the selected relevant items over the total of relevant items
Regression Model	Statistical model that estimates the relationship among variables
Risk Weighted Assets	RWA is a bank's assets or off-balance-sheet exposures, weighted according to risk
Script	a program or sequence of instructions that is interpreted or carried out by another program rather than by the computer processor
Solvency	Ability of a financial institution to meet its long-term commitments (In financial analysis: Solvency vs. liquidity ratios.)
Sureties	A person who takes responsibility for another's performance of an undertaking, for example their appearing in court or the payment of a debt.
Tangible Assets	An asset with a physical form (eg. land, machinery)
Tier 1 Capital	It includes the value of its common stock, retained earnings, accumulated other comprehensive income (AOCI), noncumulative perpetual preferred stock and any adjustments to those accounts.

Appendix G: Significant Features

This table lists the relative significance of each feature for the Random Forest model. Features are listed in descending order of their importance in the Random Forest model. As discussed in section 3.1 of this report, features ending in "M?" indicate a "Missing" feature; it is a Boolean value, which is False if the observation has a value for that feature, and True if there is no value provided. This allows the model to distinguish if a bank failed to report a statistic in one of the observations.

Feature	**G.I.**
Loans to institutions and businesses: For a term of 6 months to 1 year	**1.10E-01**
Loans to institutions and businesses: For a term of 1 year to 3 years	**1.09E-01**
Loans to institutions and businesses: For a term of up to 6 months	**9.66E-02**
Return on Net Assets (RoA)	**8.88E-02**
N7	**8.83E-02**
Loans to institutions and businesses: For a term over 3 years	**8.68E-02**
N4	**8.15E-02**
N10.1	**6.69E-02**
Return on Equity (RoE)	**6.46E-02**
Overdue Debt/ Overdrafts	**5.38E-02**
N9.1	**3.03E-02**
N1.0	**1.70E-02**
N12	**1.63E-02**
N1.1	**1.50E-02**
N1.2	**1.44E-02**
(Overdue Debt/ Overdrafts) M?	6.43E-03
Return on Equity M? (RoE M?)	4.49E-03
N1	2.40E-03

Reserve to Loans	4.20E-03
N1.0 M?	6.75E-03
N3	3.16E-03
Foreign currency operations to net Assets	3.51E-03
Mortgaged property to loans	2.64E-03
Reserve to loans M?	6.14E-03
N2	2.84E-03
Return on net assets M?	2.34E-03
N1.1 M?	1.00E-04
Foreign currency operations to net Assets M?	1.92E-03
Mortgaged property to loans M?	1.71E-03
N12 M?	8.63E-04
N4 M?	1.14E-03
N7 M?	7.45E-04
Attracted interbank loans from CBR Turnover/ Attracted interbank loans from CBR	1.08E-03
N2 M?	2.05E-03
Loans to institutions and businesses: For a term of up to 6 months M?	6.17E-04
N9.1 M?	3.60E-04
Interbank credit in CBR turnover/ Interbank credit in CBR	1.48E-03
N3 M?	3.82E-05
Loans to institutions and businesses: For a term over 3 years M?	1.29E-03
Loans to institutions and businesses: For a term of 6 months to 1 year M?	6.27E-04
(Attracted interbank loans from CBR Turnover/ Attracted interbank loans from CBR) M?	2.73E-04
N10.1 M?	4.01E-04

N1 M?	8.43E-05
N1.2 M?	6.66E-05
Interbank credit in CBR turnover/ Interbank credit in CBR M?	1.66E-04
Loans to institutions and businesses: For a term of 1 year to 3 years M?	2.82E-06