This paper and PySpark code was presented on 12 July 2018 and published by Springer in 2023. For citation, please use the following:
Ashofteh, A. (2023) 'Big Data for Credit Risk Analysis: Efficient Machine Learning Models Using PySpark', in Pilz, J., Melas, V. B., and Bathke, A. (eds) Statistical Modeling and Simulation for Experimental Design and Machine Learning Applications. Cham: Springer International Publishing, pp. 245–265. doi: 10.1007/978-3-031-40055-1_14.

Big Data for Credit Risk Analysis: Efficient Machine Learning Models Using PySpark

Afshin Ashofteh

Abstract Recently, Big Data has become an increasingly important source to support traditional credit scoring. Personal credit evaluation based on machine learning approaches focuses on the application data of clients in open banking and new banking platforms with challenges about Big Data quality and model risk. This paper represents a PySpark code for computationally efficient use of statistical learning and machine learning algorithms for the application scenario of personal credit evaluation with a performance comparison of models including logistic regression, decision tree, random forest, neural network, and support vector machine. The findings of this study reveal that the logistic regression methodology represents a more reasonable coefficient of determination and a lower false-negative rate than other models. Additionally, it is computationally less expensive and more comprehensible. Finally, the paper highlights the steps, perils, and benefits of using Big Data and machine learning algorithms in credit scoring.

Keywords: *Credit score; Big Data; Machine Learning; Risk Management; Finance.*

1.1 INTRODUCTION

Risk management with the ability to incorporate new and Big Data sources and benefit from emerging technologies such as cloud and parallel computing platforms is critically important for financial service providers, supervisory authorities, and regulators if they are to remain competitive and relevant [1].

Financial institutions' growing interest in non-traditional data may be seen as a hypothetical occurrence, a reaction to the most recent financial crisis.

Afshin Ashofteh
NOVA Information Management School (NOVA IMS), Universidade Nova de Lisboa, Campus de Campolide, 1070-312 Lisboa, Portugal. e-mail: aashofteh@novaims.unl.pt

However, the financial crisis not only prompted several statutory and supervisory initiatives that require significant disclosure of data but also provided a positive atmosphere to get the advantages of new data sources such as non-traditional data sets [2, 3].

There are sources of supply and demand for this increased acceptance of non-traditional data. On the supply side, technology advancements like mobile phones [4] that have expanded storage space and computing power while cutting costs have fueled rises in the new data sources. In addition, mobile data and social data have recently been used to monitor different risks [5]. On the demand side, loan providers are becoming more interested in learning how data analysis may improve credit scoring and lower the risk of default [6].

Some of the largest and most established financial institutions such as banks, insurance companies, payday lenders, peer-to-peer lending platforms, microfinance providers, leasing companies, and payment by installment companies are now taking a fresh look at their customers' transactional data to enhance the early detection of fraud. They use innovative machine learning models that exploit novel data sources like Big Data, social data, and mobile data. Credit risk management may benefit in the long run if these advancements result in better credit choices. However, there are shorter-term hazards if early users of non-traditional data credit scoring mostly disregard the model risk and technical aspects of new methods that might affect credit scoring [7]. For instance, one crucial issue in credit evaluation is the class imbalance resulting from distress situations for loan providers [8]. These distress situations are relatively infrequent events that make the imbalance data very common in credit scoring. In addition, the limited information for distinguishing dynamic fraud from a genuine customer in a highly sparse and imbalanced environment makes default forecasting more challenging.

Even though banks and loan providers should follow different regulations to reduce or eliminate credit risk, regulatory changes can potentially change the microfinance environment that generates the distribution of non-traditional data. It could be the source of changes in the probability distribution function of credit scores over time. In this case, the reliability of the models based on historical data will decrease dramatically. This time dependency of the training process needs new approaches adopted to deal with these situations and to avoid interruptions of ML approaches for Big Data over time [9]. These issues in credit evaluation show the importance of comparing the machine learning techniques for evaluating the model risk for credit scoring. Figure 1.1 summarizes the application of Big Data and small data in credit risk analytics.

This paper presents greater insight into how non-traditional data in credit scoring challenges the model risk and addresses the need to develop new credit scoring models. The rest of the paper is structured as follows. Section 1.2 describes the PySpark code for data processing as the first step of a personal credit evaluation. Section 1.3 represents the model building and model eval-

uation methods. The results are shown in section 1.4 for a complete personal credit evaluation. Finally, section 1.5 contains some concluding remarks.

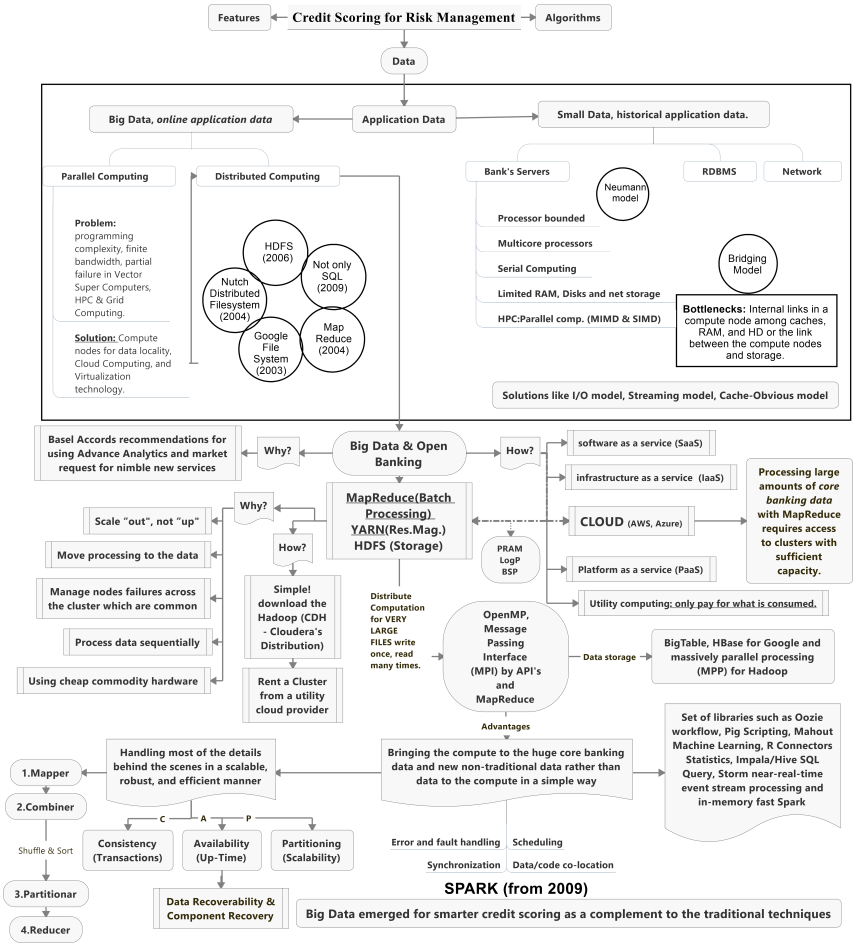

Fig. 1.1 Graphical summary of Big Data analytics on credit risk evaluation

1.2 Data Processing

In this paper, a public dataset is used, which is provided by Lending Club Company, a peer-to-peer lending company based in the US, to compare the

performance of the proposed algorithms. Lending clubs provide a sort of bridge between investors and borrowers.

The author adopted a consumer loan dataset with 2,260,668 observations by combining two versions of the lending club loan dataset; one contains loans issued from 2007 to 2015 and another from 2012 to 2018. As a result, the funded loans of these two datasets were combined, and the duplicates were removed to obtain a dataset from 2007 to 2018 with 1,048,575 customers and 145 attributes. The new dataset includes 108,623 default loans (True or 1) and 939,952, good loans (False or 0).

Table 1.1 Loan status in combined datasets without duplicates includes the personal credit history of customers 2007-2018.

Loan Status	Default loans (1/TRUE)	Good loans (0/FALSE)
Current	-	603,273
Fully Paid	-	331,528
In Grace Period	-	5,151
Charged Off	94,285	-
Late (31-120 days)	12,154	-
Late (16-30 days)	2,162	-
Default	22	-
Total	108,623	939,952

Redundant columns were removed by applying the ExtraTreesClassifier approach for variable importance levels, the correlation of data features was optimized, and the 145-dimensional data was reduced to 35-dimensional data. For instance, the correlation coefficient between two attributes *funded_amnt* as the total amount committed to that loan at that point in time and *funded_amnt_inv* as the total amount committed by investors for that loan is One, which shows the complete similarity. Based on the dimensionality reduction idea of preventing overfitting, *funded_amnt_inv* can be eliminated. Almost the same situation is for two variables, *loan_amnt,* and *installment*, with a Pearson correlation over 0.94. The dataset includes a row number column and current loan status feature with seven types of debit and credit states (Current, Fully paid, Charged off, Late (31–120 days), In grace period, Late (16–30 days), Default) as target variable (see Table 1.1 and Table 1.2).

Each row includes information provided by the applicant, loan status (current, fully paid, charged off, late (31–120 days), in grace period, late (16–30 days), and default), and information on the payments to the Lending Club Company. The author will use Python and Spark to predict the probability of default and identify the credit risk of each customer: Zero/FALSE for Non-Default and One/TRUE for Default (see Table 1.2). Spark is a tool for parallel computation with large datasets and integrates well with Python.

Table 1.2 Analytical base table for lending club loan dataset.

Row	Attribute	Description	Scale
1	Id	A unique LC assigned ID for the loan listing.	index
3	annual_inc	The self-reported annual income provided by the borrower during registration.	continuous
4	delinq_2yrs	The number of 30+ days past-due incidences of delinquency in the borrower's credit file for the past two years.	continuous
5	dti	A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower's self-reported monthly income.	continuous
7	emp_length	Employment length in years. Possible values are between 0 and 10, where 0 means less than one year and 10 means ten or more years.	continuous
9	funded_amnt	The total amount committed to that loan at that point in time.	continuous
11	grade	LC assigned loan grade	categorical
12	home_ownership	The homeownership status provided by the borrower during registration. Values: RENT, OWN, MORTGAGE, and OTHER.	categorical
31	il_util	The ratio of total user balance to high credit/credit limit	continuous
13	inq_last_6mths	The number of inquiries in the past six months (excluding auto and mortgage inquiries).	continuous
14	installment	The monthly payment owed by the borrower if the loan originates (Monthly arrears).	continuous
15	int_rate	Interest Rate on the loan.	continuous
16	issue_d	The month in which the loan was funded.	continuous
17	loan_amnt	The listed amount of the loan is applied for by the borrower. If the credit department reduces the loan amount, it will be reflected in this value.	continuous
18	loan_status	Current status of the loan.	categorical
32	mo_sin_old_il_acct	Number of months since the earliest bank account was opened	continuous
33	mo_sin_old_rev_tl_op	Number of months since the earliest revolving account was opened	continuous
19	mths_since_rcnt_il	Number of months after the installment account was opened	continuous
30	mths_since_rcnt_il	Number of months after the installment account was opened	continuous
34	mths_since_recent_bc	Number of months since last online payment	continuous
35	mths_since_recent_inq	Number of months of last loan inquiry	continuous
36	num_rev_tl_bal_gt_0	Number of revolving accounts	continuous
20	open_acc	The number of open credit lines in the borrower's credit file.	continuous
21	out_prncp	The remaining outstanding principal for the total amount funded.	continuous
38	percent_bc_gt_75	Recent income-expenditure ratio	continuous
22	pub_rec	Number of derogatory public records.	continuous
24	revol_bal	Total credit revolving balance.	continuous
25	revol_util	Revolving line utilization rate, or the amount of credit the borrower uses relative to all available revolving credit.	continuous
26	term	The number of payments (installments) on loan. Values are in months and can be either 36 or 60.	categorical
39	tot_hi_cred_lim	Credit card credit limit	continuous
27	total_acc	The total number of credit lines currently in the borrower's credit file.	continuous
40	total_bc_limit	Amount limit of online banking	continuous
41	total_il_high_credit_limit	Limit of overdue repayment amount	continuous
28	total_pymnt	Payments received to date for the total amount funded.	continuous
29	verification_status	Indicates if income was verified by LC, not verified, or if the income source was verified.	categorical

1.2.1 Data Treatment

We must connect to a cluster to use PySpark[1], R, SQL, or Scala for Big Data. Our cluster was hosted on a machine in Databricks Community Edition, connected to all other nodes. Typically, we have one computer as the master that manages splitting up the data and the computations. The master connects to the rest of the computers in the cluster, called workers. The master sends data and calculations to the workers to run. The workers also send their results back to the master. When we are just getting started with Spark on a local computer, we might just run a cluster locally. If we are using only one local node for Big Data computations in a simulated cluster, we should monitor this node (i.e., computer) if, for instance, the fan is working very hard or one process is wasting a long time to preserve the node when dealing with Big Data. However, in Databricks and cloud platforms, we safely use online resources.

For this use case in credit scoring, this study uploaded the dataset into the data repository of Databricks. When we create a notebook in our cloud platform, we build a SparkContext as our connection to the cluster. This SparkContext will recognize the specified clusters after running the first command in the notebook. This paper uses Spark MLlib, the Apache Spark library of algorithms and transformers included in a distributed Spark, to be used for ETL work and to build our models.

1.2.2 Data storage and distribution

The dataset is in CSV format, and Spark can convert it into a Universal Disk Format (UDF). Additionally, it is exportable to MLeap, a standard serialization format and execution engine for machine learning pipelines. It supports Spark, Scikit-learn, and TensorFlow for training pipelines and exporting them to an MLeap Bundle. As a result, if we continue leveraging Spark contents, it would be easy to import a batch in the batch stream, a collection of data points grouped within a specific time interval. Another term often used for this process is a data window (see Appendix 1 – Lines 1-7).

This study started with the CSV file format. However, it is not an optimal format for Spark. The Parquet format is a columnar data store, allowing Spark to only process the data necessary to complete the operations versus reading the entire dataset. Parquet gives Spark more flexibility in accessing the data and improves performance on large datasets (see Appendix 1 – Line 9).

Before starting to work with Spark, it could be recommended first to check the Spark Context and the Spark version to check the version and

[1] See PySpark program and dataset here: https://github.com/AfshinAshofteh/creditscore_pyspark.git

compatibility of our packages and program (see Appendix 1 – Lines 10-11). Second, check the dataset available on our cluster with the catalog to check the metadata or the data describing the structure of the data (see Appendix 1 – Line 12-13).

1.2.3 Munge Data

According to the data schema, the types of attributes are String. Before continuing with Spark, it is essential to covert the data types to numeric types because Spark only handles numeric data. That means all the data frame columns must be either integers or decimals (called "Doubles" in Spark). Therefore, the .cast() method is used to convert all the numeric columns from our loan data frame (for instance, see Appendix 1 – Lines 14-17). The other attributes are treated similarly according to Table 1.3.

Additionally, the *emp_length* column was converted into numeric type (see Appendix 1 – Lines 18-20) and map multiple levels of the *verification_status* attribute into a one-factor level (see Appendix 1 – Line 21). Finally, the target vector *default_loan* was created from the *loan_status* feature by classifying the data into two values: users with poor credit (default) including Default, Charged Off, Late (31-120 days), Late (16-30 days), and users with good credit (not default) including Fully Paid, Current, and In Grace Period (see Table 1.4 and Appendix 1 – Line 22).

1.2.4 Creating New Measures

Three new measures were created to increase the model's accuracy and decrease the data dimension by removing less critical attributes according to the ExtraTreesClassifier approach. For this purpose, we must know that the Spark data frame is immutable. It means it cannot be changed, and columns cannot be updated in place. If we want to add a new column in a data frame, we must make a new one. To overwrite the original data frame, we must reassign the returned data frame using the .withColumn command.

The first measure refers to the length of credit in years to know how much each person returned to the bank from the total loan amount. Therefore, we need to make a new column by subtracting the "loan payment" from the "total loan amount" (see Appendix 1 – Line 23).

The second measure is the total amount of money earned or lost per loan to show how much of the total amount of the loan should be repaid to the bank by each person (see Appendix 1 – Line 24).

Finally, the third measure is the total loan. Customers in this database could have multiple loans, and it is necessary to aggregate the loan amounts

Table 1.3 Attributes data format modification and their description.

Characteristic	Data format		Description
	Original	**Current**	
annual_inc	string	integer	
credit_length_in_years	string	integer	(issue_year - earliest_year)
delinq_2yrs	string	integer	
dti	string	integer	
funded_amnt	string	float	
earliest_year	string	double	substring(loan_df.earliest_cr_line,5, 4)
emp_length	string	float	Delete characters: " () [] * + [a-z A-Z] . , * | (n/a) Delete values lower than 1 Replace 10+ with 10
funded_amnt	string	float	
il_util	string	float	
inq_last_6mths	string	integer	
instalment	string	integer	[loan_amnt + sum(remain(t) × int_rate(t); t from 1 to term)]/term
int_rate	string	float	In percentage (%)
issue_year	string	double	substring(loan_df.**issue_d**, 5, 4)
loan_amnt	string	integer	
mo_sin_old_il_acct	string	float	
mo_sin_old_rev_tl_op	string	float	
mths_since_rcnt_il	string	float	
mths_since_rcnt_il	string	float	
mths_since_recent_bc	string	float	
mths_since_recent_inq	string	float	
num_rev_tl_bal_gt_0	string	float	
open_acc	string	integer	
out_prncp	string	Integer	
percent_bc_gt_75	string	float	
pub_rec	string	Integer	
remain	string	integer	(loan_amnt - total_pymnt)
revol_bal	string	Integer	
revol_util	string	integer	In percentage (%)
tot_hi_cred_lim	string	float	
total_acc	string	integer	
total_bc_limit	string	float	
total_il_high_credit_limit	string	float	
total_pymnt	string	float	

Table 1.4 Lending status as a binary target label vector.

	Loan_status	***Default_loan***
Poor credit (default)	Default, Charged Off, Late (31-120 days), Late (16-30 days)	true/1
Good credit (not default)	Fully Paid, Current, In Grace Period	false/0

based on the member IDs of the customers. Then according to the Basel accords and routine of the banking risk management, the maximum and minimum amounts could be reported and reviewed by risk managers to be checked for concentration risk in the risk appetite statement of the financial institutions (see Appendix 1 – Lines 25-29).

1.2.5 Missing Values Imputation and Outliers Treatment

The primary purpose of this section is to make a decision for the imputation of the missing values and deal with outliers.

For this large-scale dataset, it is reasonable to have NULL values, and handling the missing values in Spark is possible with three options: keep, replace, and remove. With missing data in our dataset, we would not be able to use the data for modeling in Spark if we have empty or N/A in the dataset. It would cause training errors. Therefore, we must impute or remove the missing data, and we could not keep them for the modeling step. This PySpark code uses the *fillna()* command to replace the missing values with an average for continuous variables, the median for discrete ordinal ones, and mode (the highest number of occurrences) for nominal features. Additionally, the variables with more than half of the data in a sample as null were discarded (see example in Appendix 1 – Lines 30-34).

The processing of outliers in this paper follows the following principles:

1. We need to consider the reasonable data range in each attribute and delete the sample data with outliers. This paper uses a simple subsetting for indexing the rows with outliers, removes the outliers with an index equal to TRUE, and checks again if the outliers are removed according to the criteria (see example in Appendix 1 – Lines 35-36).
2. Then, this paper uses cross tables to find possible errors. Cross tables for paired attributes with min and max functions as aggregate functions (*aggfunc = "min"* and *aggfunc = "max"*) could show the possible errors which exceed the minimum or maximum of attributes (see example in Appendix 1 – Lines 37).
3. Finally, box plots with the interquartile rule are used to measure the spread and variability in our dataset. According to this rule, data points below Q1-1.5*IQR or above Q3+1.5*IQR are viewed as being too far from the central values.

1.2.6 One-Hot Code and Dummy Variables

This paper discretizes the continuous variables by the chi-square box-dividing method and standardizes the discrete variables by transforming them into dummy variables. In machine learning, the standard encoding method is one-hot encoding. For encoding class variables, we have to import the one-hot encoder class from the machine learning library of Spark and create an instance of the one-hot encoder to apply to the discrete features of the dataset (see example in Appendix 1 – Lines 38-42).

Additionally, this paper considers the predictive power of discrete variables by looking at the Information Value (IV) to understand the possible transformations in categorical variables and to create multiple categories with similar IVs. IV for each class of categorical variable is the Weight of Evidence (WoE) times the difference between the proportion of all good loans in the class and the proportion of all bad loans in the class. Generally speaking, WoE is the logarithm of the proportion of good loans to the bad loans in a class. The calculation formula of WOE and IV are shown in equations (1.1), (1.2), and (1.3).

$$WoE_i = \ln\left(g_i/g_T\right) \; - \; \ln\left(b_i/b_T\right) \tag{1.1}$$

$$IV_i = \; WoE \; \times [(g_i/g_T) - (b_i/b_T)] \tag{1.2}$$

where g_i/b_i represents the number of good/bad loans in the grouping, and g_T/b_T denotes the total number of good/bad loans in all data. Formula 3 represents the IV value of the whole variable, which is an aggregation of the corresponding IV values of each group:

$$IV = \sum IV_i \tag{1.3}$$

Table 1.5 shows the criteria for excluding some variables or grouping certain variables.

Table 1.5 Criteria for IV values.

Prediction Power	**Useless**	**Weak**	**Medium**	**Strong**	**Suspicious**
IV	< 0.02	0.02 to 0.1	0.1 to 0.3	0.3 to 0.5	> 0.5

For example, the variable "*issue_d*" shows similar IVs for its first four categories (Aug-18, Dec-18, Oct-18, and Nov-18) and the following five categories (Sep-18, Dec-15, Jun-18, Jul-18, and May-18). Therefore, they could be grouped into two new categories. Furthermore, the results from IV show strong prediction power for most variables (e.g., term, grade, *home_ownership*, *verification_status*, etc.), and none were transformed.

1.2.7 Final Dataset

When the data treatment is completed, there are 1,043,423 customers in rows and 35 features in the dataset, including four categorical and 31 numeric attributes in addition to one binary target variable with two values default and not default. After finalizing the data treatment, the Spark Cache is used to optimize the final dataset for iterative and interactive Spark applications and improve Jobs' performance (see Appendix 1 – Line 43). The dataset contains 331,528 fully paid loans. (see Table 1.1 and Appendix 1 – Lines 44-45). Considering the imbalance of default and non-default loans in the dataset, good customers are much fewer than bad customers, which may cause prediction deviation in some modeling approaches such as logistic regression. For these models, the paper applies a paired sample technique to the training base by randomly selecting bad customers as the same number of total good clients. This undersampling method or equivalent approach, such as the synthetic minority oversampling technique (SMOTE), is essential to increase the efficiency of the models, which suffer from imbalanced datasets [10].

Table 1.6 shows that the data set was randomly divided into two groups, 65% for model training (678,548 observations) and the other 35% for the test set (364,875 observations) to apply different algorithms.

Table 1.6 Loan status in training and test datasets.

Loan Status	Training set	Test set
Default	607,981	326,820
Not default	70,567	38,055
Total	678,548	364,875

For dividing the dataset into test and train sets, the function *.randomSplit()* was used in PySpark, equivalent to *test_train_split()* in python (see Appendix 1 – Line 46). The training dataset for developing the model would normally have twelve months dedicated to the training to have a full annual cycle to recover the seasonality of each month. Additionally, some recent months could be considered just for testing the optimal model with an unseen dataset.

1.3 Method and Models

Financial institutions must predict the customers' credit risk over time with minimum model risk. Recently, machine learning models have been applied to Big Data to determine if a person is eligible for receiving a loan. However, the pre-processing for the data quality and finding the best hyper parameters

in model development are necessary to overcome the overfitting problems and instability of model accuracy over time.

1.3.1 Method

According to the dataset, we have a credit history of the customers, and this study tries to predict the loan status of the customers by applying statistical learning and machine learning algorithms. It helps the loan providers to guess the probability of default to determine whether or not a loan should be granted. For this purpose, this paper makes a preliminary statistical analysis of the credit dataset. Then, different models were developed to predict the probability of default. The models include Logistic regression, Decision tree, Random Forest, Neural network, and Support vector machine.

Finally, the results (predictive power of models) were evaluated by evaluation metrics such as the Area Under the ROC Curve (AUC) and the Mean F1-Score. Receiver operating curves (ROC) show the statistical performance of the models. In the ROC chart, the horizontal axis represents the specificity, and the vertical axis shows the sensitivity. The greater the area between the curve and the baseline, the better the feature performance in default prediction. After investigating the characteristics of the new credit score model, the research employs the area ratio of ROC curves to compare the classification accuracy and evaluates how well this credit scoring model performs. The F1 score, is commonly used in information retrieval, measures the model'saccuracy using precision (p) and recall (r). Precision is the ratio of true positives (tp) to all predicted positives $tp + fp$. A recall is the ratio of true positives to all actual positives $tp + fn$. The $F1$ score where $p = \frac{tp}{tp+fp}$ and $r = \frac{tp}{tp+fn}$ is given by:

$$F1 = 2\frac{p.r}{p+r}$$

The F1 metric weights recall and precision equally, and a good retrieval algorithm will simultaneously maximize precision and recall. Thus, moderately good performance on both will be favored over excellent performance on one and poor performance on the other.

For creating a ROC plot in PySpark, we need a library that is not installed by default in Databricks Runtime for Machine Learning. First, we have to install plotnine and its dependencies based on the ggplot2 package (see Appendix 1 – Lines 47-48). Second, PyPI mlflow package could be installed into the cluster to track the model development and packaging code into reproducible runs.

1.3.2 Model Building

This section builds and evaluates supervised models in PySpark for personal credit rating evaluation. This paper applies the obtained dataset to Logistic regression, Decision tree, Random Forest, Neural network, and Support vector machine.

The model-building phase started with three statistical learning methods and penalized linear regression models: Lasso, Ridge, and ElasticNet. They eliminate variables that contribute to overfitting without compromising out-of-sample accuracy. They have L1 and L2 penalties during training and some hyperparameters (*maxIter, elasticNetParam*, and *regParam*), which should be set to assign how much weight is given to each of the L1 and L2 penalties and which model should be fitted. The *elasticNetParam* For Ridge regression is 0, for LASSO is 0.99, and for ElasticNet regression is 0.5 (see Appendix 1 – Lines 64-66). The results from this notebook in Databricks were tracked for storing the results and comparing the accuracy of different models. (see Appendix 1 – Lines 67-80). Then the logistic regression model was built (see Appendix 1 – Line 81). A pipeline was defined, which includes standardizing the data, imputing missing values, and encoding for categorical columns (see Appendix 1 – Lines 82-83). Setting the mlflow of model tracking and reproducibility of the input parameters is useful to log the model and review later (see Appendix 1 – Lines 84-88). Finally, the accuracy measures were calculated by Logging the ROC Curve (see Appendix 1 – Lines 89-93), setting Max F1 Threshold for predicting the loan default with a balance between true-positives and false-positives (see Appendix 1 – Lines 94-99), scoring the customers (see Appendix 1 – Lines 100-114), and logging the results (see Appendix 1 – Lines 115-116).

The leave-one-out cross-validation method examines the between-sample variation of default prediction. This paper divides the available data into ten disjoint subsets to train the models on nine subsets and evaluate the model selection criterion on the tenth subset. This procedure is then repeated for all combinations of subsets by the Python API of Apache Spark (see Appendix 1 – Lines 117-118). Finally, this paper uses the MLflow UI built-in as part of the Community Edition of Databricks to compare the models and choose the ultimate best model. The best model might be selected with an AUC greater than a threshold (see Appendix 1 – Lines 124-127) or maximum AUC (see Appendix 1 – Lines 126-130). The details of the best model with maximum AUC could be checked (see Appendix 1 – Lines 131-132), and the model's score with the test data (see Appendix 1 – Lines 133-134). This final model could predict the amount of money earned or lost per loan (remain=loan payments - total loan amount) and the outstanding loan balance (see Appendix 1 – Line 135).

As a result, the Ridge method represents a better performance than Lasso. The Logistic regression in this paper is based on the Ridge penalty with elastic net regularization zero and regparam 0.3 as the best hyperparameters.

In addition to the Logistic regression classifier as an industry standard for building credit scoring models, this paper uses other binary classifiers such as random forests and linear support vector machines for the empirical analysis. Although they are more complex and powerful than Logistic regression in the application, the outputs explainability of these models could not be guaranteed.

The codes are almost the same for the other models, such as Random Forest and Linear Support Vector Machine, with the possibility to use Scala for more complicated models and to use some features that are not available in PySpark.

1.4 Results and Credit Score Card Conversion

The results show that A-grade loans have the lowest interest rate because of the minimum evaluated risk for these customers. A significant amount of loans is allocated to grade A and B customers with the minimum interest rate and minimum risk of default. We have a descending trend for the grades D, E, F, and G because banks typically have some sort of criteria to reject high-risk applications. The optimal cut-off for logistic regression is considered 0.167.

Table 1.7 Evaluation results of the algorithms on personal credit data.

Algorithm	**Confusion matrix** **Predict Label [[11,10] [01,00]]**	**AUC value**	**F1 score**
Logistic regression	[[9%,2%] [2%,87%]]	0.909	0.815
Decision tree	[[7%,0.5%] [3.5%,89%]]	0.901	0.766
Random Forest	[[10%,13%] [2%,75%]]	0.884	0.577
Neural network	[[2%,2.5%] [9%,86.5%]]	0.580	0.258
Support vector machine	[[1%,0%] [10%,89%]]	0.530	0.113

This study discovers a high level of False Negative Rate in any approach. This rate represents an unexpected loss for the bank. A False Negative rate also shows a loss in the bank's balance sheet since it does not let the new business increase. These two rates are summarized in the F1 score, indicating a trade-off between False-positive and False Negative. As a trade-off between model sensitivity and specificity, AUC in Table 1.7 shows almost the same performance among the logistic regression, decision tree, and random forest. However, the logistic regression model obtained a higher F1 score (i.e., 0.815) than the decision tree and random forest, with F1 scores of 0.766 and 0.577, respectively (see Appendix 2). Overall, the logistic regression performs the best, and the support vector machine performs the worst with three times more training time compared to other algorithms.

1.5 Conclusion

This study described machine learning approaches to assess credit candidate applicants' profiles and continued credit scoring based on the non-traditional dataset of Lending Club Company. Regarding the classification accuracy, the results showed that the logistic regression is more accurate, informative, and conservative for personal credit evaluation. Furthermore, the model predictions could be used to score new and old clients in an accurate scorecard complementary to traditional credit evaluation methods.

For further study, the following items could be suggested for more investigation:

For data treatment, one might consider the following new attributes for a possible increase in the model's accuracy:

1. Effort rate by dividing the *installment* by the *annual income.*
2. The ratio between the number of open accounts and the total number of accounts.
3. Percentage of the loan that is still left to be paid. It would be similar to the "remain" variable but divided by the total amount.
4. A continuous variable to represent the duration of the client's credit line. It could be built based on "*earliest_cr_line*" by subtracting the *earliest_cr_line*'s values from the current time.
5. Decomposing *issue_d* into months and years to lead to better insight into any eventual seasonal events.

For model development, one might consider other machine learning approaches as the author did a preliminary study on Gradient Boosted Trees, and the AUC was increased to 0.966 with a longer run time compared with other models in this paper. For the hyperparameter tuning stage and finding the best hyper-parameters that enable a higher F1-Score, one could use the GridSearchCV. Instead of the undersampling approach in this paper for the imbalanced dataset, one might use StratifiedKFold in the Cross-Validation stage.

Acknowledgment

The author of this paper would like to thank José L. CERVERA-FERRI (CEO of DevStat) for his invitation to CARMA 2018 (International Conference on Advanced Research Methods and Analytics) at the Polytechnic University of Valencia, which motivated this research.

Data and Code Availability

Data and code used to support the findings of this paper are available from the author upon request, his GitHub or Kaggle page.

Appendix 1

```
1. file_location = "/FileStore/tables/loan.CSV"
2. file_location = "/FileStore/tables/loan_complete.CSV"
3. file_type = "CSV"
4. infer_schema = "false"
5. first_row_is_header = "true"
6. delimiter = ","
7. loan_df = Spark.read.format(file_type).option("inferSchema", infer_schema).option("header",
   first_row_is_header).option("sep", delimiter).load(file_location)
8. print(" >>>>>>> " + str(loan_df.count())+ " loans opened in this
   data_set!")
9. loan_df.write.parquet("AA_DFW_ALL.parquet", mode="overwrite")
10. print(sc)
11. print(sc.version)
12. Spark.catalog.listTables()
13. display(loan_df)
14. loan_df = loan_df.withColumn("loan_amnt", loan_df.loan_amnt.cast("integer"))\
15. .withColumn("int_rate", regexp_replace("int_rate", "%", "").cast("float"))\
16. .withColumn("revol_util", regexp_replace("revol_util", "%", "").cast("float"))\
17. .withColumn("issue_year", substring(loan_df.issue_d, 5, 4).cast("double"))
18. loan_df = loan_df.withColumn("emp_length", trim(regexp_replace(loan_df.emp_length,
    "([ ]*+[a-zA-Z].*)|(n/a)", "") ))
19. loan_df = loan_df.withColumn("emp_length", trim(regexp_replace(loan_df.emp_length,
    "< 1", "0") ))
20. loan_df = loan_df.withColumn("emp_length", trim(regexp_replace(loan_df.emp_length,
    "10\\+", "10") ).cast("float"))
21. loan_df = loan_df.withColumn("verification_status", trim(regexp_replace(loan_df.verification_status,
    "Source Verified", "Verified")))
22. loan_df = loan_df.filter(loan_df.loan_status.isin(["Default", "Charged Off",
    "Late (31-120 days)", "Late (16-30 days)", "Fully Paid","Current"] )).with-
    Column( "default_loan", (~(loan_df.loan_status.isin(["Fully Paid", "In Grace
    Period" , "Current"]))).cast("string"))
23. loan_df = loan_df.withColumn("credit_length_in_years", (loan_df.issue_year
    - loan_df.earliest_year))
24. loan_df = loan_df.withColumn("remain", round( loan_df.loan_amnt - loan_df.total_pymnt,
    2))
25. customer_df = loan_df.groupBy("member_id").agg(f.sum("loan_amnt").alias("sumLoan"))
26. loan_max_df = customer_df.agg({"sumLoan": "max"}).collect()[0]
27. customer_max_loan = loan_max_df["max(sumLoan)"]
28. print(customer_df.agg({"sumLoan": "max"}).collect()[0],customer_df.agg({"sumLoan":
    "min"}).collect()[0])
29. print(customer_df.filter("sumLoan = " +str(customer_max_loan)).collect())
30. pandas_df = loan_intrate_income.toPandas()
31. null_columns = pandas_df.columns[pandas_df.isnull().any()]
32. pandas_df[null_columns].isnull().sum()
```

```
pandas_df.int_rate.fillna(pandas_df.int_rate.median())
loan_intrate_income=loan_intrate_income.dropna()
indices = pandas_df[padas_df["income"] >= 1500000].index
pandas_df.drop(indices, inplace=TRUE)
pd.crosstab(pandas_df.grade, pandas_df.default_loan, values=pandas_df.annual_inc, aggfunc="min").roundGrindEQ$_2$
from PySpark.ml.feature import OneHotEncoderEstimator
onehot = OneHotEncoderEstimator(inputCols=["grade"], outputCols=["grade_dummy"])
model_df3 = model_df.select("int_rate","annual_inc","loan_amnt", "label","grade")
onehot = onehot.fit(model_df3)
str_to_dummy_df_onehot = onehot.transform(model_df3)
loan_df.cache()
loan_df.filter(col("loan_status") == "Default").count()
loan_df.filter(col("loan_status") == "NOTDefault").count()
train, valid = dataLogReg.randomSplit([.65, .35])
%sh
/databricks/python/bin/pip install plotnine matplotlib==2.2.2
import sklearn.metrics as metrics
import pandas as pd
from plotnine import *
from plotnine.data import meat
from mizani.breaks import date_breaks
from mizani.formatters import date_format
from PySpark.ml import Pipeline
from PySpark.ml.feature import StandardScaler, StringIndexer, OneHotEncoder, Imputer, VectorAssembler
from PySpark.ml.classification import LogisticRegression
from PySpark.ml.evaluation import BinaryClassificationEvaluator
from PySpark.ml.tuning import CrossValidator, ParamGridBuilder
import mlflow
import mlflow.Spark
from PySpark.mllib.evaluation import BinaryClassificationMetrics
from PySpark.ml.linalg import Vectors
maxIter = 10
elasticNetParam = 0
regParam = 0.3
with mlflow.start_run():
labelCol = "default_loan"
indexers = list(map(lambda c: StringIndexer(inputCol=c, outputCol=c+"_idx", handleInvalid = "keep"), categoricals))
ohes = list(map(lambda c: OneHotEncoder(inputCol=c + "_idx", outputCol=c+"_class"), categoricals))
imputers = Imputer(inputCols = numerics, outputCols = numerics)
featureCols = list(map(lambda c: c+"_class", categoricals)) + numerics
model_matrix_stages = indexers + ohes + \
```

```
74. [imputers] + \
75. [VectorAssembler( inputCols=featureCols, outputCol="features" ), \
76. StringIndexer( inputCol= labelCol, outputCol="label" )]
77. scaler = StandardScaler(inputCol="features",
78. outputCol="scaledFeatures",
79. withStd=True,
80. withMean=True)
81. lr = LogisticRegression(maxIter=maxIter, elasticNetParam=elasticNetParam,
    regParam=regParam, featuresCol = "scaledFeatures")
82. pipeline = Pipeline(stages=model_matrix_stages+[scaler]+[lr])
83. glm_model = pipeline.fit(train)
84. mlflow.log_param("algorithm","SparkML_GLM_regression") #put a name
    for the algorithm.
85. mlflow.log_param("regParam", regParam)
86. mlflow.log_param("maxIter", maxIter)
87. mlflow.log_param("elasticNetParam", elasticNetParam)
88. mlflow.Spark.log_model(glm_model, "glm_model") #log the model.
89. lr_summary = glm_model.stages[len(glm_model.stages)-1].summary
90. roc_pd = lr_summary.roc.toPandas()
91. fpr = roc_pd["FPR"]
92. tpr = roc_pd["TPR"]
93. roc_auc = metrics.auc(roc_pd["FPR"], roc_pd["TPR"])
94. fMeasure = lr_summary.fMeasureByThreshold
95. maxFMeasure = fMeasure.groupBy().max("F-Measure").select("max(F-
    Measure)").head()
96. madFMeasure = maxFMeasure["max(F-Measure)"]
97. fMeasure = fMeasure.toPandas()
98. bestThreshold = float ( fMeasure[ fMeasure["F-Measure"] == maxFMea-
    sure] ["threshold"])
99. lr.setThreshold(bestThreshold)
100. def extract(row):
101. return (row.remain,) + tuple(row.probability.toArray().tolist()) + (row.label,)
     + (row.prediction,)
102. def score(model,data):
103. pred = model.transform(data).select("remain", "probability", "label", "pre-
     diction")
104. pred = pred.rdd.map(extract).toDF(["remain", "p0", "p1", "label", "pre-
     diction"])
105. return pred
106. def auc(pred):
107. metric = BinaryClassificationMetrics(pred.select("p1", "label").rdd)
108. return metric.areaUnderROC
109. glm_train = score(glm_model, train)
110. glm_valid = score(glm_model, valid)
111. glm_train.registerTempTable("glm_train")
```

```
glm_valid.registerTempTable("glm_valid")
print( "GLM Training AUC :" + str( auc(glm_train)))
print( "GLM Validation AUC :" + str(auc(glm_valid)))
mlflow.log_metric("train_auc", auc(glm_train))
mlflow.log_metric("valid_auc", auc(glm_valid))
cv = CrossValidator(estimator=pipeline_rf, estimatorParamMaps=params,
evaluator=BinaryClassificationEvaluator(), numFolds=5)
rf_model = cv.fit(train)
import mlflow
import mlflow.Spark
from mlflow.tracking import MlflowClient
from PySpark.sql.functions import *
from PySpark.ml import PipelineModel
client = MlflowClient()
runs = client.search_runs(experiment_ids =["#number#"], filter_string =
"metrics.valid_auc >= .65")
run_id1 = runs[0].info.run_uuid
client.get_run(run_id1).data.metrics
runs = client.search_runs(experiment_ids=["#number#"], order_by=
["metrics.valid_auc DESC"], max_results=1)
run_id = runs[0].info.run_uuid
client.get_run(run_id).data.metrics
runs = mlflow.search_runs(experiment_ids=["1636634778227294"],
order_by=["metrics.valid_auc DESC"], max_results=1)
runs.loc[0]
score_df = Spark.table("final_scoring_table")
predictions = model1.transform("score_df")
display(predictions.groupBy("default_loan","prediction").agg((sum(col("remain"))).alias("sum_net")))
```

Appendix 2

The first branch of the decision tree shows that if the value of out_prncp is more extensive than 0.01, we will automatically receive that probability default value of 0.03%. When an individual does not meet demanded value, the proposal should be checked for total payment with a limit of 5,000. Finally, the branches show how the application should go through the confirmation process.

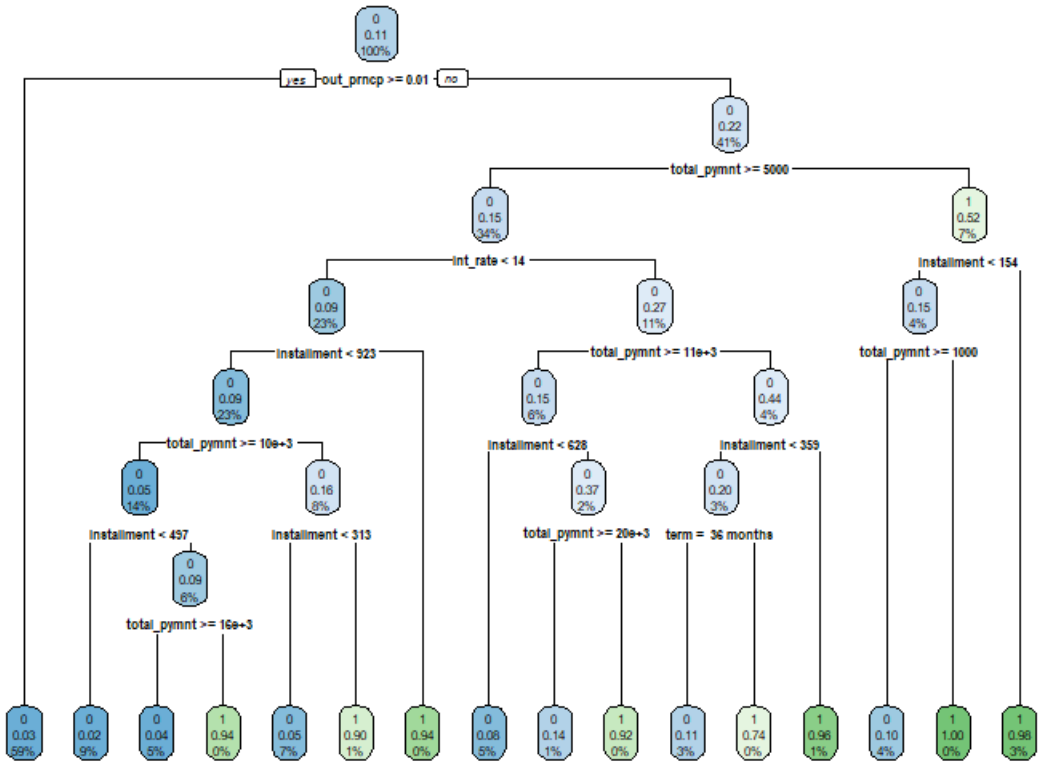

Fig. 1.2 A decision-tree sample with selected attributes.

References

1. C. Onay and E. Öztürk*A review of credit scoring research in the age of Big Data* J. Financ. Regul. Compliance, vol. 26, no. 3, pp. 382–405, Jul. 2018.
2. A. Ashofteh*Mining Big Data in statistical systems of the monetary financial institutions (MFIs)* in International Conference on Advanced Research Methods and Analytics (CARMA), 2018, p. doi: 10.4995/carma2018.2018.8570.
3. M. Óskarsdóttir, C. Bravo, C. Sarraute, J. Vanthienen, and B. Baesens*The value of big data for credit scoring: Enhancing financial inclusion using mobile phone data and social network analytics* Appl. Soft Comput. J., vol. 74, pp. 26–39, Jan. 2019.
4. J. S. Pedro, D. Proserpio, and N. Oliver*Mobiscore: Towards universal credit scoring from mobile phone data* in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2015, vol. 9146, pp. 195–207.
5. D. Björkegren and D. Grissen*Behavior revealed in mobile phone usage predicts credit repayment* arXiv. arXiv, 09-Dec-2017.

6. B. M. Henrique, V. A. Sobreiro, and H. Kimura*Literature review: Machine learning techniques applied to financial market prediction* Expert Syst. Appl., vol. 124, pp. 226–251, Jun. 2019.
7. G. dos Reis, M. Pfeuffer, and G. Smith*Capturing Model Risk and Rating Momentum in the Estimation of Probabilities of Default and Credit Rating Migrations* Quant. Financ., vol. 20, no. 7, pp. 1069–1083, Sep. 2018.
8. H. Zhang and Q. Liu*Online learning method for drift and imbalance problem in client credit assessment* Symmetry (Basel)., vol. 11, no. 7, Jul. 2019.
9. A. Ashofteh and J. M. Bravo*A non-parametric-based computationally efficient approach for credit scoring* in Atas da Conferencia da Associacao Portuguesa de Sistemas de Informacao, 2019.
10. A. Gicić and A. Subasi*Credit scoring for a microcredit data set using the synthetic minority oversampling technique and ensemble classifiers* Expert Syst., vol. 36, no. 2, p. e12363, Apr. 2019.

- For citation:
Ashofteh, A. (2023) 'Big Data for Credit Risk Analysis: Efficient Machine Learning Models Using PySpark', in Pilz, J., Melas, V. B., and Bathke, A. (eds) Statistical Modeling and Simulation for Experimental Design and Machine Learning Applications. Cham: Springer International Publishing, pp. 245–265. doi: 10.1007/978-3-031-40055-1_14.

- A research that was developed based on this research is also available full-text here:
https://doi.org/10.1016/j.eswa.2021.114835