Explainable Credit Default Prediction using Amex Data

Yatharth Bansal
UNI:yb2540
Department of Electrical Engineering
Columbia University
yb2540@columbia.edu

Aishwarya Patange
UNI:aap2239
Department of Electrical Engineering
Columbia University
aap2239@columbia.edu

Miheer Prakash
UNI:mp3939
Department of APAM
Columbia University
mp3939@columbia.edu

***Abstract*—Credit card default prediction is an imperative component of the credit card industry to manage and optimize lending decisions. With the rapid increase of the credit card industry, it has become essential to deal with the increasing delinquency rates to prevent fraud and financial loss to the industry. In this work, we aim to solve this crucial problem by exploring the realms of big data and machine learning to predict the default behavior of a customer by developing an explainable credit card default prediction model. We build a cloud model utilizing Apache Spark engines to achieve efficient and optimized performance to run our model on the high-volume AMEX dataset. To serve the industry standards, this work incorporates the issue of explainability for such models by applying game theory to our prediction model. We incorporate various machine learning algorithms like Logistic Regression, Linear SVM, Decision Tree, Random Forest, and Gradient Boosted Trees to compare the performance of each algorithm using the F1 score evaluation metric. We showcase the significant performance of our model and explainability of our model with the help of Logistic Regression and game theory techniques like shapley.**

***Index Terms*—Machine Learning, Big Data, Credit Default, Model Explainability, PySpark.**

I. Introduction

Credit card default prediction has become a crucial key component of the financial industry. The industry has become heavily reliant on the default prediction mechanism in order to function and manage customers and profits in an optimized manner. In the past few years, this issue of credit card default has had an adverse impact on the economy and financial institutions, ultimately affecting consumers. This issue arises when an individual fails to comply and pay back the dues in a given required period of time which leads to losses to the financial institutions.

So, in order to deal with this problem the industry has evolved and leveraged consumer data to apply statistical methods and predict the default behavior of the consumers in order to prevent sanctioning of further loans and credit to such consumers and setting up policies for the same. But these advancements are based on the incorporation of huge data and predictions based on statistical methods that are highly dependent on users' interpretation and judgment which is prone to human error and bias. These methods have been found to be inefficient and not to be optimized in a manner to be deployed on a large scale and be heavily relied upon for dealing with this issue. Moreover, these traditional statistical methods are unable to leverage the possible potential of big data and fail to provide significant accuracy and efficiency for making decisions.

Recently, with the advancement in the field of big data and machine learning, the industry has been able to identify the need to integrate big data techniques to achieve the required solution to these issues and have a significantly better model than previous solutions for credit card default prediction. With the use of machine learning techniques, credit card default can be predicted using customers' past data associated with the financial institution based on different factors and variables that affect or lead to such defaults. Big data and machine learning algorithms are able to provide logical and optimized solutions with scope for improvement and continuous growth to achieve significant outcomes and profitability in the financial industry.

In this work, we utilize the real potential of big data by leveraging the large-scale AMEX dataset which helps us define a realistic model matching industrial standards and thus, making our work more relevant and realistic to be adopted by the industry. We build an end-to-end cloud model based on the Google Cloud Platform and use Apache spark engines for each and every step of our model. The pipeline of the model includes data sourcing from cloud storage, data preprocessing, feature engineering, exploratory data analysis, model training, and finally, deployment of the model on the front end. This work specifically makes use of spark engines to get better and more significant efficiency in dealing with such large volumes of data in the best possible optimized way to make sure that the least memory and time is utilized and can be deployed easily on a front-end platform. We experiment with various machine learning techniques like logistic regression, linear SVM, decision tree, random forest, and GBTclassifier to find the best optimal solution for a defined problem. Finally, we showcase the significant performance of our model using Logistic regression and using the F1 score as our main evaluation metric along with other evaluation

metrics like weighted precision, weighted recall, and accuracy.

The work also focuses to provide a solution for explainability by employing game theory techniques like shapley [1]. Incorporating explainability makes our model more understandable and easy to interpret, thus making the model more adaptable and usable by the industry as it is easier to reason out the behavior of the model.

The paper has been divided into 5 sections. In section II, we describe and explore the related and relevant work in this field of work, section III explains the dataset and data preprocessing techniques, and section IV showcase data visualization to get a better understanding and insight into our data. Finally, section V explains the system model, followed by section VI and VII which depicts the observations and conclusions of this work.

II. Related work

To deal with the problem of default prediction of customers, we investigate some of the previous works and literature that have been used in this field. In the previous literature, various data mining techniques [2] have been showcased to predict the likelihood that a customer would default on the payment. Recently, the increase in the amount of data being generated in the industry has led to solving problems and finding solutions using the realms of big data and machine learning. Few of the recent works [3], [4] showcase using different types of machine learning techniques like logistic regression, simple decision tree classifiers, and random forest to build credit card default prediction models. Most of these models fail to leverage and incorporate the large data that can be used to train the model and build a robust architecture that is efficient and explainable. The current techniques being used in the literature make use of basic algorithms like decision trees, random forests, and logistic regression. The Bayesian approach has also been taken in to evaluate the credit score by utilizing graph representation models which calculate the probability of interconnection of variables.

III. Dataset description

In our work, as previously explained we aim to provide a relevant and practical model suitable for the financial industry, so for these purposes, we approach building the model incorporating the Amex dataset. As we know that American Express (or Amex) is a multinational corporation specializing in payment card services and in 2016, credit cards using the Amex network accounted for 22.9% of the total dollar volume of credit card transactions in the United States. We observed that recently Amex came up with a default prediction competition on Kaggle [5] and we took benefit of this opportunity to build a robust model based on such high volumes of industry data. So, the dataset being used in this work is the one given by Amex for that competition.

The motivation for choosing this dataset was that Amex provided us with an enormous dataset (which helped us to incorporate the volume property of big data) which helps us to leverage the machine learning algorithms for training models.
The aim of this dataset is to predict the probability that a customer will pay the amount back in the future based on the profile that is analyzed on a monthly bases. The dataset includes time series behavioral data and anonymized customer profile information in order to maintain customer privacy. In the given dataset, a customer is considered to be a defaulter if the customer is not able to pay for the due amount in the given time period, i.e., 120 days after their latest card statement. And in addition to this, an eighteen-month performance was analyzed after the customer's last statement to define the target variable, i.e., customer default behavior.

Initially, the raw dataset provides 190 predictor variables(or features) which can be mainly classified into 5 general categories as the following:

- D_* = Delinquency Variables
- S_* = Spend Variables
- P_* = Payment Variables
- B_* = Balance Variables
- R_* = Risk Variables

We further analyze the given classification of the variables provided to us in the dataset to gain a profound insight into the data we are utilizing to predict customer default behavior and gain inference to apply better feature engineering to build an accurate model. So, delinquency in the context of the financial industry means that a customer is behind on his credit card payments and these delinquency variables determine if the customer has ever been delinquent or not. Apart from this other delinquency variables may include elements like how many times has the person been delinquent in a particular time frame, the time period since the last delinquency, etc [6]. The spend variables, as the name suggests, give us a time series relation of the customers' spending habits in given particular time frames. The payment variables similarly refer to the payment done by the customers in those time frames for the due amounts which may be classified as overpaying or underpaying their credit card bills. If the customer underpays, he might become a delinquent as well. The balances in the customers' accounts are kept track of in the balance variables. The risk variables in the dataset refer to the risk-associated factors defined by the financial institution.

In addition to the above-discussed variables, we also have the statement date variable stated as S_2 and the $customer\_ID$ which refers to the anonymized customer name

in the dataset. So, providing an overview of our data, the dataset consists o 178 floating point variables, 11 categorical variables, and 1 DateTime variable (S_2). The above-mentioned 190 variables are encoded by Amex itself so that they do not give out any personal information about the customers and thus, maintain the privacy and security of the customers. Furthermore, to ensure anonymity, all the float-type columns have a random uniform noise of $[0, 0.01]$ added to each column. We analyze that ideally these variables should be monotonically increasing or decreasing.
In the next section, we discuss the approaches and techniques we have used for building the data preprocessing pipeline for finally feeding the data to the machine learning algorithms.

IV. Data Visualisation

We make use of data visualization to gain insights and a better understanding of our data. The visualizations of the data have been performed using the Plotly, Seaborn, and Matplotlib libraries for the best visualization output. Among the various visualizations done as a part of this work for gaining a better perspective and understanding to work on this project, we showcase a few major graphs in this paper. In fig 1., we showcase the target variable distribution of the labeled dataset provided to us which is basically the corresponding binary values of customers showcasing whether a customer is going to default or not. The target variable distribution gives insight into detecting whether that dataset is biased towards a particular result. From this figure, we can observe that we have an imbalanced class distribution for this dataset with 74% being non-defaulters and only 26% being defaulters. We need to take this into account while evaluating the model.

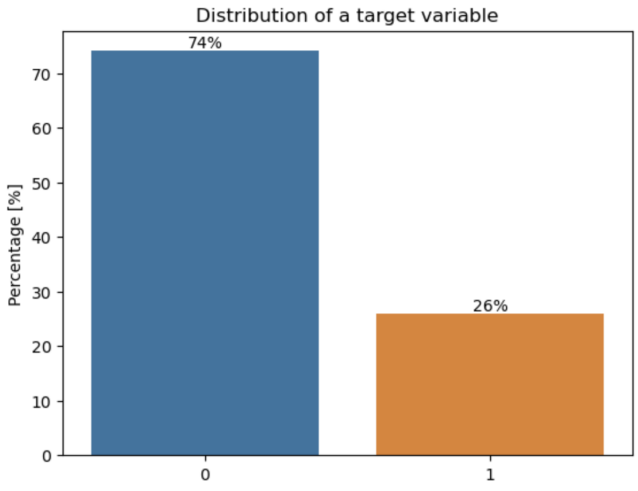

Fig. 1. Target Variable distribution

In fig 2., we look at the distribution of credit statements for the customers. We see that 80% of the customers have 13 credit statements per month. This means that there are usually 13 instances for each customer id. However, there is only 1 target variable associated with each customer id. Due to this, we group by customer id and aggregate values for all the features so that we end up with 1 instance for each customer.

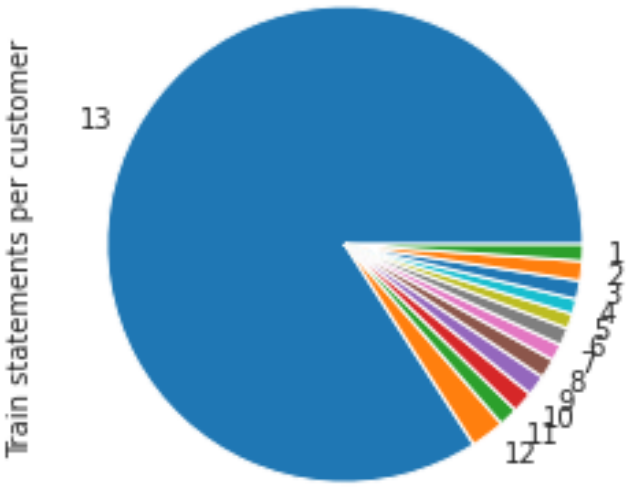

Fig. 2. Target Variable distribution

In fig 3, the categorical features distribution and correlation with respect to the target variable have been depicted. We observe the distribution of the categorical variables with respect to each target value (i.e. 0 and 1) to see if there is a significant difference between the 2 distributions, and to determine which variables have an effect on the target. The same has been implemented for continuous features. We also gain the insight that every feature consists of a maximum of eight categories which also includes nan categories and it would be a feasible approach to use one hot encoding.

Fig. 3. Categorical features

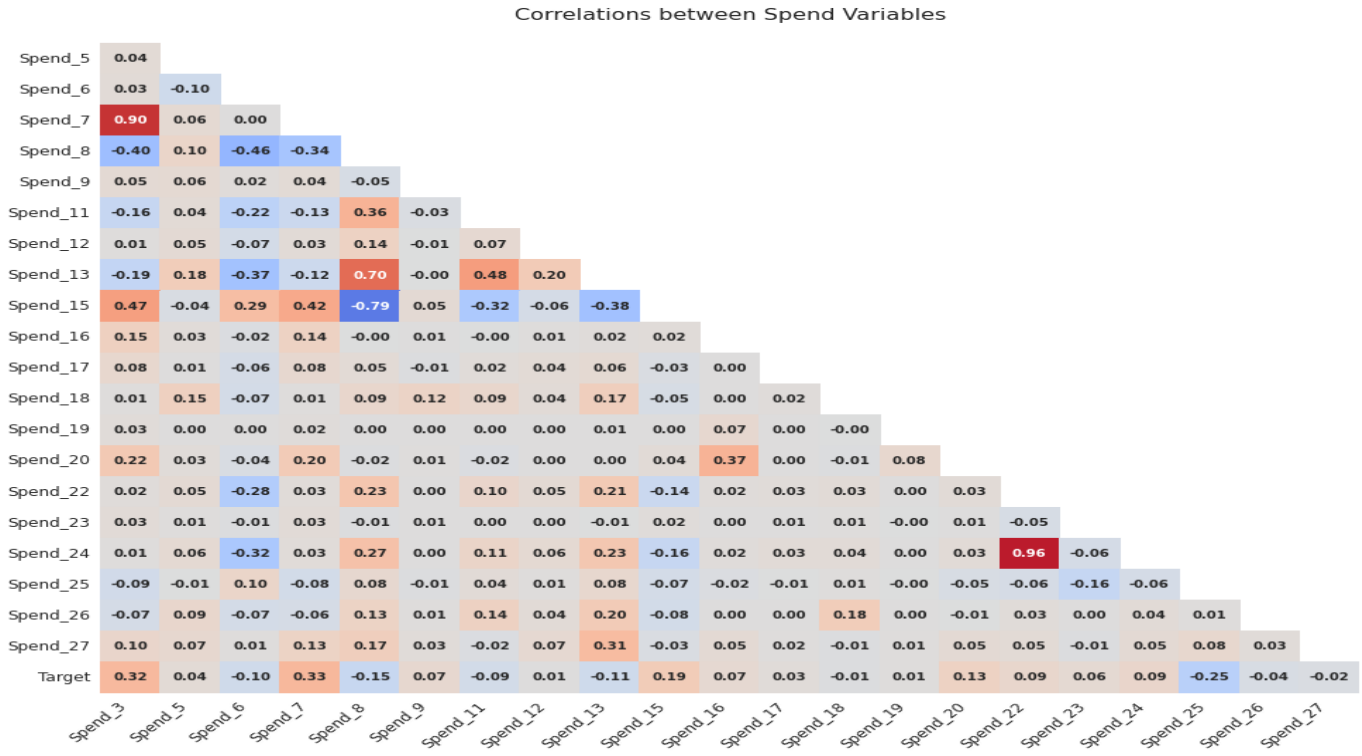

Fig. 4. Spend variables correlation plot

In fig 4 we see a heatmap that shows the correlation between all the Spend Variables and Target as well. The information given by this plot helps us during feature selection. 2 Variables that are highly correlated with each other usually influence our prediction power in a similar way so we may drop one of these variables as no added information is being given to the model by having both these variables present. Furthermore, variables with extremely correlation with the target variable add negligible value to the predictive power of our model so these can be dropped as well.

V. Methodology

A. Data Preprocessing pipeline

1) Dropping Columns with Null Values: After loading the the `*.csv` dataset as a PySpark dataframe, the first thing we do is drop columns having null values with more than a certain threshold percentage. For this project, we choose a threshold percentage of 5%, i.e. columns having more than 5% of it's values null / missing will be removed from the dataset. Not only does this help in simplifying our data, but also it ensures that the amount of processing power required to impute the missing values with the remaining columns is reduced. Removing columns with missing values also helps us reduce the size of our dataset. This is especially useful for us as we are working with a large dataset. Morever, since there are missing values for various features for many customers, we group by customer_ID and impute the missing values. This does not affect the output of credit default predictions [7]

2) Encoding Categorical Features as Numerical Features: The PySpark module has the a transformer called as the `StringIndexer`. This classi s used to encode categorical variables as numerical values. This is often necessary when building machine learning models, as most algorithms can only work with numerical data. The `fit` method fits the indexer on the data, learning the mapping from categorical values to numerical values. The `transform` method then applies this mapping to the data, creating a new column with the encoded values. After encoding the data, the resulting DataFrame will have a new column with numerical values representing the categorical data. You can then use this column as input to a machine learning model or for other purposes.
We also use the `setHandleInvalid()` function and set it to *keep*. This ensures the null values in the categorical columns are indexed as a separate category and in a way handles the imputation.

3) One Hot Encoding Categorical Variables: We now convert the categorical columns which have been String Indexed and have their missing values imputed into OneHotEncoded columns. This is because we are not sure if the categorical variables have any hierarchy among their values. If there is no hierarchy, it makes more sense to separate the categorical variables into their dummy columns. We also set the dropLast parameter to True as if we do not drop the last category, we will only make an extra linearly dependent vector. This extra linearly dependent vector will not store any extra information and hence dropping it will reduce the size of the dataset and

make the pipeline more efficient. Using this transform, we convert a single valued column to a column containing vectors of length equal to the total number of unique categories.

4) Missing Values Imputation for Numerical Columns: The `StringIndexer` class takes care of handling the missing values for the categorical variable. For the numerical variable, however, we need to impute the missing values separately. In this project, we opted to go with the median strategy for imputing the missing values. This strategy has several benefits. First, median imputation is resistant to the influence of outliers. Because the median is not affected by extreme values, it is generally less sensitive to the presence of outliers in the data. This can be useful if you have a few extreme values in your dataset that might otherwise skew the results if you used a different imputation method (such as mean imputation). Furthermore, median imputation is simple and easy to implement. Calculating the median of a dataset is straightforward and can be done quickly, even for large datasets. This makes median imputation a convenient and efficient method for handling missing values.

5) Aggregation of Data: We know that the customer_IDs get repeated in the training dataset and the label file contains one target value for each customer_ID. Now to join these two files to create one dataframe, we group the rows in the training dataset and then perform aggregation operations like sum, min, max stddev, and mean and then combine the rows having the same customer_ID into one row and then append the target column to it. We only use the n numerical columns here to compute the aggregations. All other categorical columns are kept as it is. This reduces our 5.5 million row dataset to a 450 thousand row dataset. This makes it easier to process and on top of that converting it into a parquet file ensures that the data is faster to access.

6) Assembling Columns into a Vector: As we are using PySpark's ML library, we first convert the predictor features into vectors using VectorAssembler. The prime motivation here is to assemble everything into a vector data type. This is reasonable since, after one-hot encoding and other preprocessing techniques, we end up with a mishmash of integers, floats, sparse vectors, and maybe dense vectors. And what we do next is bundle them all together and call the collection features. If we do not initialize the column names for the dataframe, PySpark, by default takes in `features` as X and `labels` as y [8].

B. Machine learning algorithms

We experiment with various machine-learning algorithms to compare the performance of each algorithm on our preprocessed data to achieve the best algorithm for the final system model. In this work, we experiment with LogisticRegression, DecisionTreeClassifier, RandomForestClassifier, Linear SVM, and GBTClassifier to find the best optimal algorithm for our model. These models have been implemented using PySpark's roster of models where all these models can be easily found and implemented according to our requirements. Further, in the next section, we describe the evaluation metric to analyze and evaluate the performance of our experiments and state the results of the significantly performing machine learning algorithm which we further implement in our final system design.

VI. Performance Analysis

A. Evaluation Metric

In this section, we showcase the different evaluation metrics which we use for analyzing the performance of our machine-learning algorithms. As seen before in the data visualization section, we observe the dataset has an imbalanced class distribution for our dataset. So, in order to ensure that the model is not biased, we use the F1 score, weighted precision, and weighted recall as our main evaluation metrics which have been stated to be the optimal choice of metrics to be used for such cases [9]. Further, we describe the evaluation metrics being used in this work.

Recall can be thought of as a model's ability to find all the data points of the class of interest from our dataset.

The formula for the recall is:-

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{1}$$

Precision is the measure of how many positive observations our model correctly predicted over the amount of correct and incorrect positive predictions. Precision tells us a models ability to correctly find data points in the relevant class that were indeed relevant.

The formula for precision is :-

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{2}$$

Precision and Recall have a trade-off. If the model has high precision, it means that it is good at identifying positive cases, but it is not necessarily good at identifying all of the negative cases. On the other hand, if the model has high recall, it means that it is good at identifying all of the positive cases, but it may not be as precise in its predictions.

Weighted precision/Weighted recall, the formula is:-

$$\frac{1}{\Sigma_{l \in L} |\hat{y}_l|} \Sigma_{l \in L} |\hat{y}_l| \phi(y_l, \hat{y}_l) \tag{3}$$

where,

- L is the set of labels
- $\hat{y}$ is the true label
- y is the predicted label
- $\hat{y}_l$ is all the true labels that have the label l
- $\phi(y_l, \hat{y}_l)$ computes the precision or recall for the true and predicted labels that have the label l.

F1 score is a metric that combines precision and recall. It is often used in the evaluation of a classification model, and it can be calculated using the following formula:

TABLE I
MACHINE LEARNING ALGORITHMS PERFORMANCE

Model Name	F1 Score	Weighted Precision	Weighted Recall	Accuracy
Logistic Regression	0.888979	0.888435	0.889861	0.889861
Decision Tree	0.873663	0.873008	0.874661	0.874611
Random Forest	0.873663	0.873008	0.874661	0.874611
Linear SVM	0.887130	0.886851	0.88791	0.887991
GBTClassifier	0.876992	0.876990	0.876994	0.876994

$$F1 = 2 * \frac{(precision * recall)}{(precision + recall)} \tag{4}$$

The F1 score is useful because it balances precision and recall, giving equal weight to both. A model with a high F1 score is considered to be a good classifier, as it is able to both accurately predict positive examples and also cover a high proportion of the actual positive examples in the data.

The F1 score can range from 0 to 1, with a higher score indicating better performance. A perfect score of 1 is achieved when the precision and recall are both 1, meaning that the model is able to correctly predict all positive examples and does not make any false positive predictions.

B. Results

We depict the results of the different machine learning algorithms that we have experimented with using the evaluation metrics mentioned above. We showcase the results for all metrics that we mention in the previous section but we focus our results mainly on the F1 score. We can observe in Table I that the best algorithm from our experiments is the Logistic Regression classifier which achieves the highest F1 score of 0.889 outperforming the other algorithms. Further, we also like to add that it also showcases a higher weighted precision of 0.888, weighted recall of 0.889, and accuracy of 0.889. Thus, proving to be the best optimal machine learning algorithm for our work to predict customer default behavior.

VII. MODEL EXPLAINABILITY

In this section, as discussed before this paper also aims to provide an explainable model to serve the industry standards, and thus, we showcase the explainability of our model. So, we first explain what Model Explainability means and also define Global and Local explainability. We then proceed to give a brief overview of Shapley values and then explain how we have used them to make our model explainable.

Model explainability refers to the concept of being able to understand the machine learning model and explain the behavior in human terms. Once we understand a model, we can detect if there is any bias present. With complex models, you cannot fully understand how and why the inner mechanics impact the prediction. However, through model agnostic methods like SHapley Additive exPlanations (SHAP) dependence plots, you can discover the meaning between input data attributions and model outputs, which enables you to explain the nature and behavior of the AI/ML model. Model Explainability becomes important while debugging a model during the development phase. It also helps to determine if the models are suitable to be deployed in real life. There are two ways to explain the model – Global Explainability and Local Explainability.

1. *Global Explainability*

Global explainability refers to the ability to understand and explain the behavior and decisions of a machine learning model on a global or overall level. This means that the model's behavior and decisions can be understood and explained across all or a significant portion of the input data that the model was trained on, rather than just for individual input data points.

There are various techniques and approaches that can be used to achieve global explainability in machine learning models. Some examples include feature importance analysis, which helps identify the most important features that contribute to the model's predictions, and model-agnostic explainability methods, which can be applied to a wide range of different types of models to provide global explanations of their behavior.

One key challenge in achieving global explainability is that many machine learning models, can be very complex and difficult to understand. This can make it difficult to identify the underlying factors that contribute to the model's decisions, and to explain them in a way that is meaningful and understandable to humans. Overall, the goal of global explainability is to improve the transparency and accountability of machine learning models.

2. *Local Explainability*

This helps in understanding how the model makes decisions for a single instance. Local explanations help in understanding the behavior of the model in the local neighborhood. There are several ways to locally interpret an ML model:

1. Feature importance: This method identifies the most important features in the model by calculating how much each feature contributes to the model's predictions.
2. Partial dependence plots: This method visualizes the relationship between a feature and the model's output, while holding all other features constant.
3. Shapley values: This method decomposes the prediction of a model into the contribution of each feature, taking into account the interaction between features.
4. LIME (Local Interpretable Model-Agnostic Explanations):

This method creates a simple, interpretable model that explains the predictions of a complex model in a local region around the prediction.

5. Counterfactual explanations: This method generates explanations for model predictions by showing what would need to change in the input data for the model to make a different prediction.

In our case, we have chosen to use Shapley values for model explainability purposes. We now explain what Shapley values mean.

Shapley values

Shapley values are used in the field of game theory to determine the contribution of each player in a game. In this analogy, each feature is a "player" and the "game" is reproducing the outcome of the model. We use them in the context of machine learning to help explain the comparative importance of each feature in a decision made by a model.

Essentially, Shapley values provide a way to quantify the contribution of each feature to the overall prediction or decision made by the model, which can help to identify which features are most important and why. SHAP shows the impact of each feature by interpreting the impact of a certain value compared to a baseline value. The baseline used for prediction is the average of all the predictions. SHAP values allow us to determine any prediction as a sum of the effects of each feature value. This can be useful for improving the model's performance and understanding how the model is making its decisions, which is important for building trust and confidence in the system. The Shapley values can be combined together and used to perform global interpretations as well. The only disadvantage with SHAP is that the computing time is high.

We now proceed to look at Shapley Values from the perspective of our model- the first step is observing the SHAP feature importance plot for our model. The idea behind the importance of SHAP features is straightforward: Features with high absolute Shapley values are important. We average the absolute Shapley values per feature across the data, (using equation 5) because our concern is global explainability.

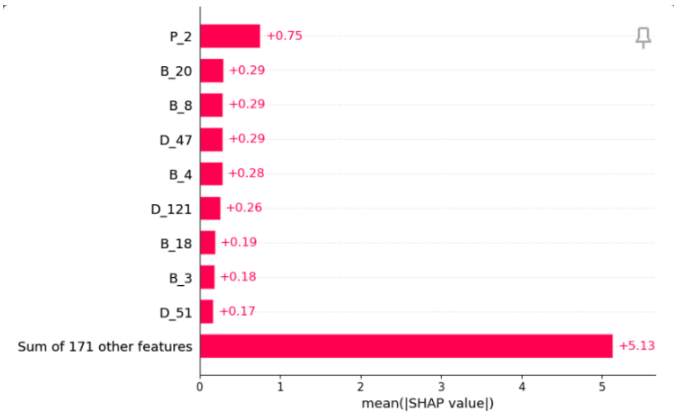

Fig. 5. SHAP Feature Importance

$$I_j = \frac{1}{n}\sum_{i=1}^{n} |\phi_i^j| \tag{5}$$

here "i" refers to an observation from the dataset and "j" is the feature we are referring to.

From fig 5 we can see that after averaging Shapley values over all observations, the 5 most important features are P2, B20, B8, D47 and B4. The value associated with P2 is 0.75. This means that on average, knowing the value for the P2 feature for an observation increases the predicted target variable value by 0.75 points.

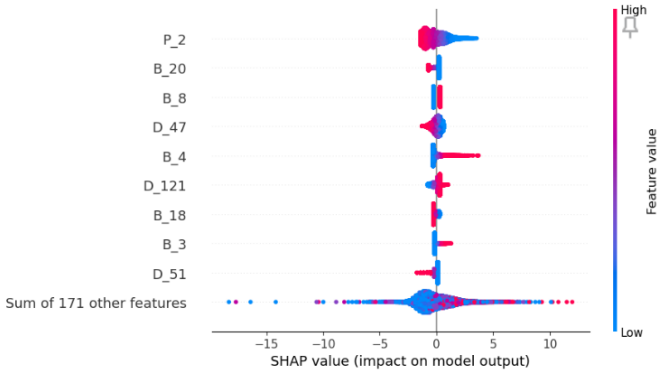

Fig. 6. SHAP beeswarm plot

The feature importance plot is useful, but gives a general outlook on feature importance, without looking at each instance of the data for all the features. For a more informative plot, we will next look at the beeswarm plot. The beeswarm plot in fig 6., combines feature importance with feature effects. In this plot, we see the relationship between the value of a feature and the impact on the prediction.

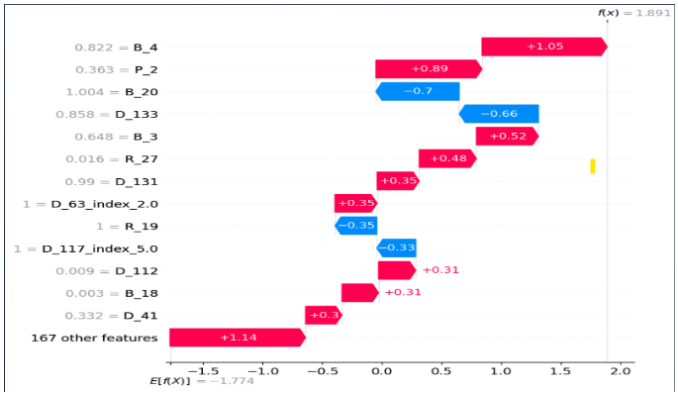

Fig. 7. SHAP Local Explainability

Here we can see that low P2 values have very high Shap values which implies that when P2 has a low value, they influence the target variable to move towards 1 by a great amount. On the other hand, when P2 has high values, the shap values are negative which means that the target variable is pushed towards 0. In contrast to this, for B4, high values for

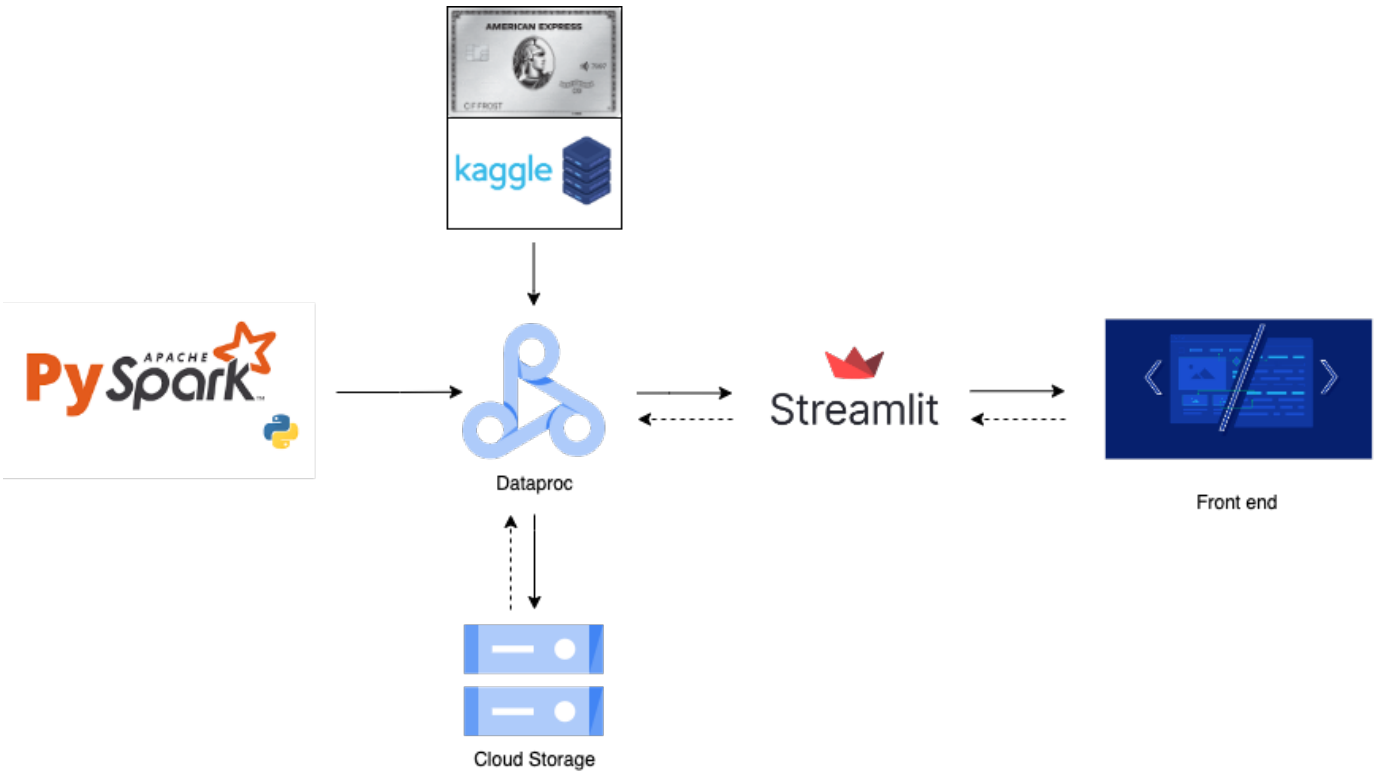

Fig. 8. The system model

this feature have positive shap values which push the target variable towards 1. On the other hand, when B4 has low values, the shap values are negative which means that the target variable is pushed towards 0.

Next, we look at the local interpretations for a test case in fig 7. For this test case we can see feature importance for each feature sorted in a decreasing order of importance. While P2 is the most important feature from a global explainability perspective, for this particular case we see that B4 has the highest importance which is 0.822. This means B4 is responsible for pushing the target variable up by 0.822 for this test case.

VIII. System Model

In this work, we build an end-to-end system utilizing a cloud platform and all computation is executed on Spark engines to achieve the highest computational performance on such big data sets and utilize the paralleled computation provided by spark for faster data processing and modeling.

In fig 8, we define the system model as a collection of a few steps and processes that ensure the smooth and sound running of our system model and depict the stages occurring in our model for final execution on the front end. The main key components of the system model are dataproc, cloud storage, and streamlit platform. The system model initiates with the execution of dataset ingestion. We source the data from the Kaggle servers into our cloud storage running Kaggle's API in the mater node of our dataproc in GCP. As our training dataset is static, we do not keep ingesting data at regular intervals. This dataset is then fetched from the GCS buckets and fed first to our data format conversion pipeline and then eventually to our PySpark data preprocessing pipeline. Once we have the preprocessed dataset, it is ready for the modelling stage. The modelling stage invokes the required PySpark classifier objects and fits the dataset onto it.

The model object and the feature transforms objects are converted to a pipeline object model and then stored as a pickle file in another GCS bucket. This pipeline object along with a sampled input dataset is sent to the shap module. The calculated shapley values are then stored in the GCS bucket again.

We now create a frontend for our web app. This webapp is created on the same dataproc cluster we used to train the model. This is because we want to run the trained PySpark models everytime we get a prediction request from the user. Everytime we instantiate the webapp, we fetch the pipeline object and the shapley values from the backend (GCP Cloud Storage) and store them in our memory. The moment an instance of the dataset is sent through the frontend, we run it through the pipeline object and get a prediction.

This instance from the user is also sent along with a sampled dataset stored in the GCS bucket to calculate new shapley values and get the local explanation for that instance. All of these predictions and shapley plots are displayed on the frontend to the user.

IX. Conclusions

This work has aimed to provide an end-to-end cloud-based explainable model for predicting customer default behavior in the financial industry. This work leverages the power of big data and machine learning to provide solutions for issues of the customer default behavior faced by the financial industry and thus, preventing further losses. In this work, we have showcased the power of using Apache spark engines to deal with and process enormous high-volume Amex data along with utilizing machine learning algorithms like Logistic regression to achieve a significant and efficient performance. In addition to showcasing the significant performance of our algorithm, i.e, Logistic Regression, we also analyze and brief the explainability of our model using Shapley values.
In the future, we would like to experiment with different enormous high-volume datasets and also aim to dive deeper into applying various other game theory techniques to increase the explainability of such models.

References

[1] S. M. Lundberg and S. Lee, "A unified approach to interpreting model predictions," *CoRR*, vol. abs/1705.07874, 2017. [Online]. Available: http://arxiv.org/abs/1705.07874

[2] I.-C. Yeh and C. hui Lien, "The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients," *Expert Systems with Applications*, vol. 36, no. 2, Part 1, pp. 2473–2480, 2009.

[3] Y. Yu, "The application of machine learning algorithms in credit card default prediction," in *2020 International Conference on Computing and Data Science (CDS)*, 2020, pp. 212–218.

[4] Y. Sayjadah, I. A. T. Hashem, F. Alotaibi, and K. A. Kasmiran, "Credit card default prediction using machine learning techniques," in *2018 Fourth International Conference on Advances in Computing, Communication Automation (ICACCA)*, 2018.

[5] D. X. H.-k. H. V. i. N. S. D. Addison Howard, AritraAmex, "American express - default prediction," 2022. [Online]. Available: https://kaggle.com/competitions/amex-default-prediction

[6] C. Chen, K. Lin, C. Rudin, Y. Shaposhnik, S. Wang, and T. Wang, "An interpretable model with globally consistent explanations for credit risk," vol. abs/1811.12615, 2018. [Online]. Available: http://arxiv.org/abs/1811.12615

[7] A. Al-qerem, G. Al-Naymat, and M. Alhasan, "Loan default prediction model improvement through comprehensive preprocessing and features selection," in *2019 International Arab Conference on Information Technology (ACIT)*, 2019, pp. 235–240.

[8] I. T. Jolliffe and J. Cadima, "Principal component analysis: a review and recent developments," vol. 374.

[9] L. Jeni, J. Cohn, and F. De la Torre, "Facing imbalanced data - recommendations for the use of performance metrics," vol. 2013, 09 2013.