CSC 555 Mining Big Data Final Project Report:
Credit Risk Analysis
Sena Bui 2179040

Introduction & Objectives

In any real-life scenario, there is a risk to everything. This includes our financial institutions and how society handles the flow of financial assets. Corporations and institutions, whether public or private and small or big, must be able to calculate risk. This is heavily seen in sectors such as finance, banking, insurance, as well as other sectors including medical-care and engineering. In this project, we will be analyzing the credit history of customers from a financial institution to perform credit risk analysis. Our objective is to predict possible credit defaulters and to help the financial institutions to take steps accordingly.

Data Source

Curious about how algorithms work within the finance and insurance industry, I chose to go with this dataset. Having worked with various datasets in the past ranging from environmental to diamonds market data, I wanted to take a dive at more finance-related data such as determining credit risk.

https://www.kaggle.com/datasets/ranadeep/credit-risk-dataset/data

The dataset contains 1 csv file and 1 xlsx file:

- Loan.csv (441.77 MB): Contains 74 columns of customer credit history including member id, loan amount, funded amount, interest rate, installments, loan grade definitions, whether or not a customer has home ownership, annual income, verification status, loan status, customer inquiries, late payments, and so on.
- LCDataDictionary.xlsx: Provides a dictionary and explanation of all columns within loan.csv

Data Processing & Transformation Steps

Data Cleaning

I did a good amount of data processing and cleaning just using Pyspark itself before throwing it into Athena to query. I first used Pyspark to identify and remove columns that were missing more than 70% of the data in their perspective columns. This led to dropping a total of 20 columns:

```
['desc', 'mths_since_last_record', 'mths_since_last_major_derog', 'annual_inc_joint',
'dti_joint', 'verification_status_joint', 'open_acc_6m', 'open_il_6m', 'open_il_12m',
'open_il_24m', 'mths_since_rcnt_il', 'total_bal_il', 'il_util', 'open_rv_12m',
'open_rv_24m', 'max_bal_bc', 'all_util', 'inq_fi', 'total_cu_tl', 'inq_last_12m']
```

I continued to remove unnecessary and redundant columns not needed for our data analysis, such as:

```
['url', 'application_type', 'last_pymnt_amnt', 'recoveries', 'policy_code',
'initial_list_status', 'next_pymnt_d', 'zip_code', 'issue_d',
'collections_12_mths_ex_med', 'id', 'addr_state', 'title', 'member_id',
'last_credit_pull_d', 'last_pymnt_d','mths_since_last_delinq']
```

Data Transformation

Looking at my categorical columns of 'grade', 'emp_length', 'home_ownership', verification_status', and 'purpose', I took a deeper look into each of their unique values to see how I could bin or categorize them, and cleaned and dropped any unique categorical column values that I did not need for my analysis. I used one-hot encoding to code for each unique value. For the target value 'loan_status', there were several unique values that did not make sense for our data analysis. To fix this, I combined similar default indicators together and excluded values such as 'Current', 'Issued', and 'In Grace Period' from loan_status so that I can focus on definitive loan outcomes. This will provide for a clearer signal for default prediction. I used binary encoding where 1= Default/High Risk and 0= Non-Default/Low Risk. Further transformations were made after I had queried and loaded up my data into a Spark dataframe including standardizing our numeric features using StandardScaler, converting my vectors from my categorical columns into actual python arrays, and then converting them back to the proper vectors for my ML processing, and checking for class imbalance and then applying weights to level out the class imbalance in binary 'loan_status'.

Data Merging & Aggregation

I then stored my final transformed and pre-processed dataframe df_final to S3 bucket in Parquet format. This data was also kept in an external table in Athena and integrated into my workflow to facilitate effective querying and to load up the data for data analysis and machine learning later on. After querying my final preprocessed and cleaned dataset from Athena, I read my query results directly into a Spark dataframe to proceed with my machine learning model development.

Machine Learning Model Development

For my credit risk analysis, Spark MLlib was used to predict whether a customer would default or not. I decided on Random Forest for the model because it handles non-linear relationships well, is robust with any outliers, does not overfit easily with the proper tuning, and provides feature importance for interpretability. I trained the model and evaluated it based on my final dataset that included all my cleaned and preprocessed feature variables.

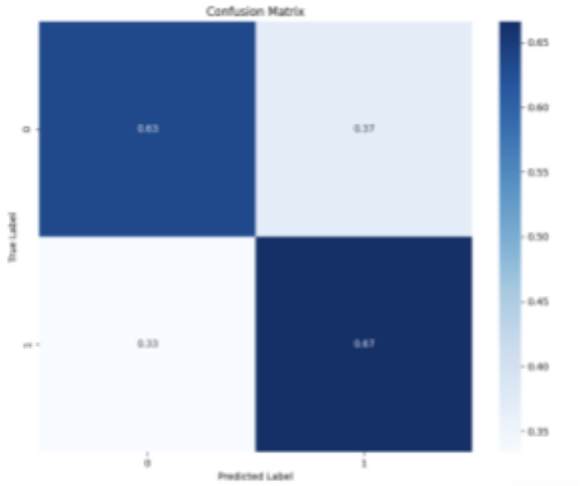
```
# Train a random forest model
# For credit risk analysis, decided on Random Forest because

from pyspark.ml.classification import RandomForestClassifier

# Initialize Random Forest classifier
rf = RandomForestClassifier(
    labelCol="loan_status_binary",
    featuresCol="features",
    weightCol="classWeight",
    numTrees=100,
    maxDepth=10,
    impurity="gini",
    featureSubsetStrategy="sqrt",
    seed=42
)
```

Evaluation & Results

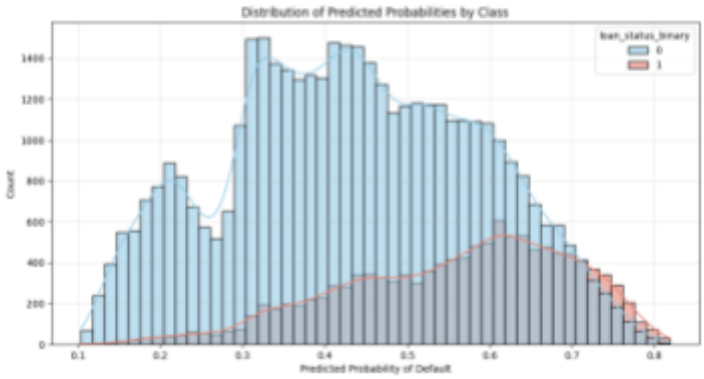
I evaluated my model's performance using ROC AUC, PR AUC, F1 score, and accuracy metrics. For a credit risk model, I believe the overall performance to be moderate. My ROC AUC (0.7040) is acceptable but not excellent, I would prefer it to be above 0.80+ at least. With my PR

AUC at 0.4021, it suggests that my model still struggles with correctly identifying defaulters despite applying class weights to loan_status. Both the F1 score (0.6689) and accuracy (0.64-04) are reasonable but still can be improved.

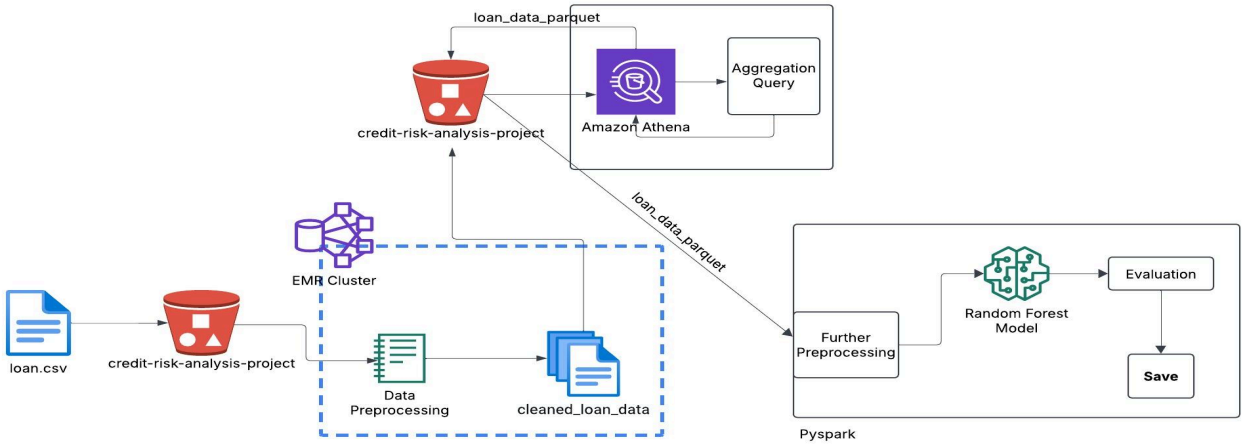
Visualizations & Insights (Including architecture diagrams)

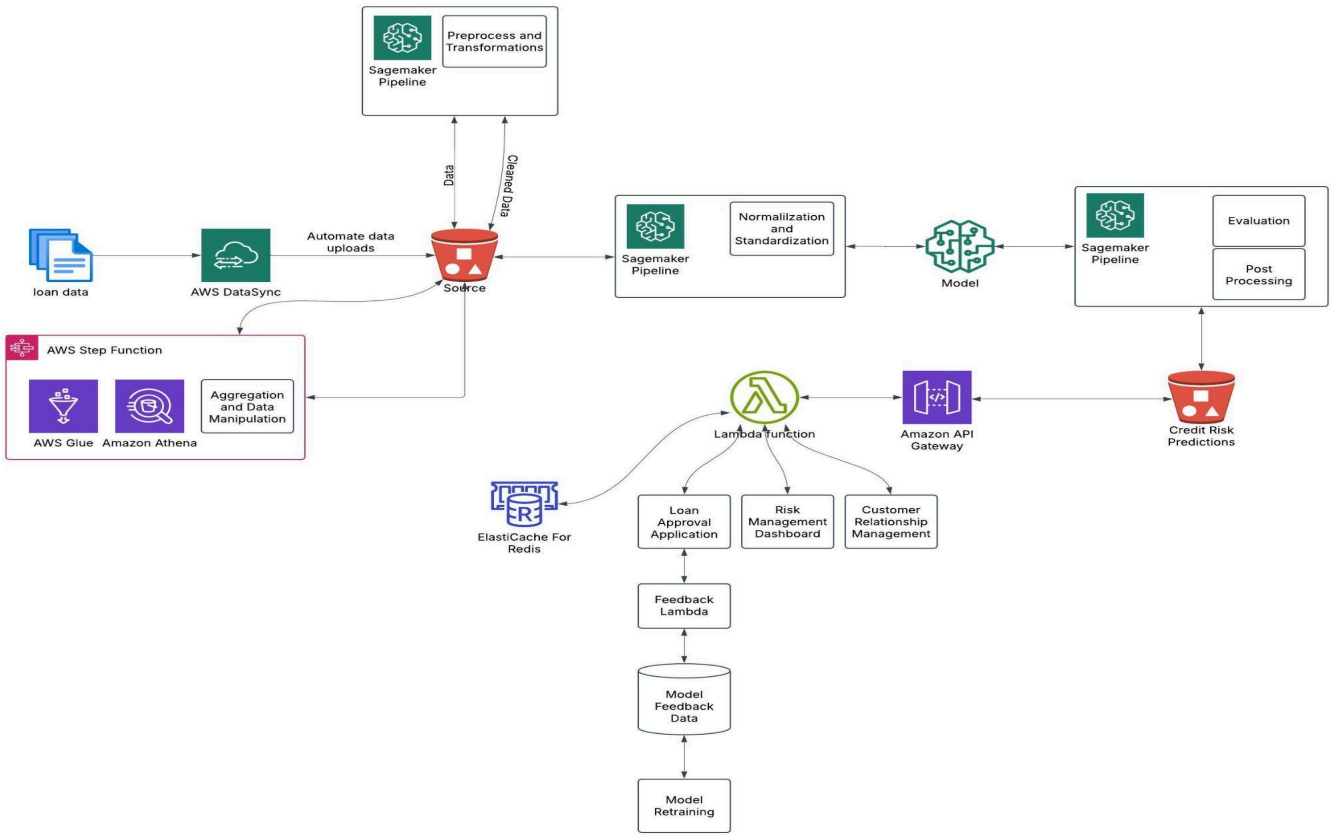


Our model correctly identified 63% of non-defaulters, while misclassifying 37% of them as defaulters. The model also correctly identified 67% of defaulters, while missing 33% of them. The model shows a somewhat balanced error pattern, with a slight tendency to perform better on detecting defaulters than non-defaulters. With my class weighting approach, the model achieved a reasonable balance between the classes, but there is still room for improvement, particularly in reducing false negatives which are costly in credit risk scenarios (the 33% missed defaulters who weren't flagged by our model).



This graph shows how our model assigns probability scores to default predictions across both classes in our dataset. We have blue representing the non-defaulters and red representing the defaulters. The non-defaulters have a bimodal distribution with peaks around 0.3-0.4. Defaulters are more concentrated towards higher probabilities. We observe an overlap of instances between the two classes within the range of 0.4-0.6, indicating uncertainty and difficulty cleanly separating the classes. This may explain our model's moderate performance metrics as a result of difficulty with perfectly separating these classes with our current features and model.





Challenges

Within this project I faced challenges with handling such a large dataset with many columns that required necessary attention to understand which columns to work with for my dataset. Another challenging part was the amount of missing data, the various data types, and making sure they were all cast to their appropriate data types for analysis. Therefore, the whole preprocessing step took a long time for me. It was also challenging navigating AWS and drawing up the architectural diagrams. as it is my first time working with systems such as AWS.

Conclusion & Future Work

Distributed computing with Spark and AWS really helps with scaling the data process and reducing execution time with such a large dataset. This approach really helped to ensure efficiency and scalability across the data pipeline. With this project work, it made me realize that I would like to continue working within cloud platforms like AWS for maximized efficiency, utilizing S3, Spark, and Athena to build up the complete data pipeline system. If I had more time and continued to explore this project, I would try incorporating additional machine learning algorithms such as KNN, gradient boosting, and neural networks or ensemble methods. I would also approach the project again by taking out more features from my dataframe, apply SMOTE to address class imbalance, and create interaction terms between existing features. All-in-all, with the limited amount of time, I am quite happy with where my project results stand.