AV LTFS FINHACK III

2nd Place Solution

Team: *Winners to Bat !!!*

Outline

Problem Statement

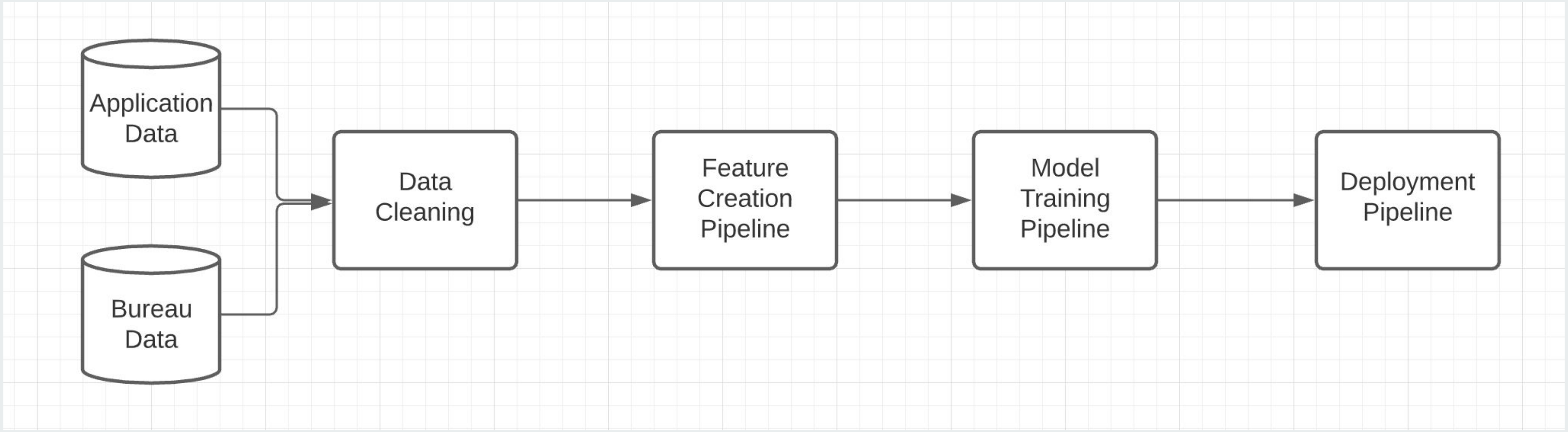
- Objective: To forecast (the time period) when the loan applicant would require a top-up loan
- Data Sources:
  - **Bureau Data**: contains tradelines of the applicants
  - **Application Data**: LTFS loan specific data
- Given portfolio is an **agriculture farm equipment (mostly tractors)**
- Train Sample:
  - Size: 128655
  - Duration: 1st Feb 2010 - 23rd November 2019
- Test Sample :
  - Size: 14745
  - Duration: 2nd July 2011 - 12th April 2019

Distribution of the target label

Top-up Month	**Percentage**
No Top-up Service	82.92%
> 48 Months	6.50%
36-48 Months	2.84%
24-30 Months	2.71%
30-36 Months	2.38%
18-24 Months	1.84%
12-18 Months	0.80%

Approach

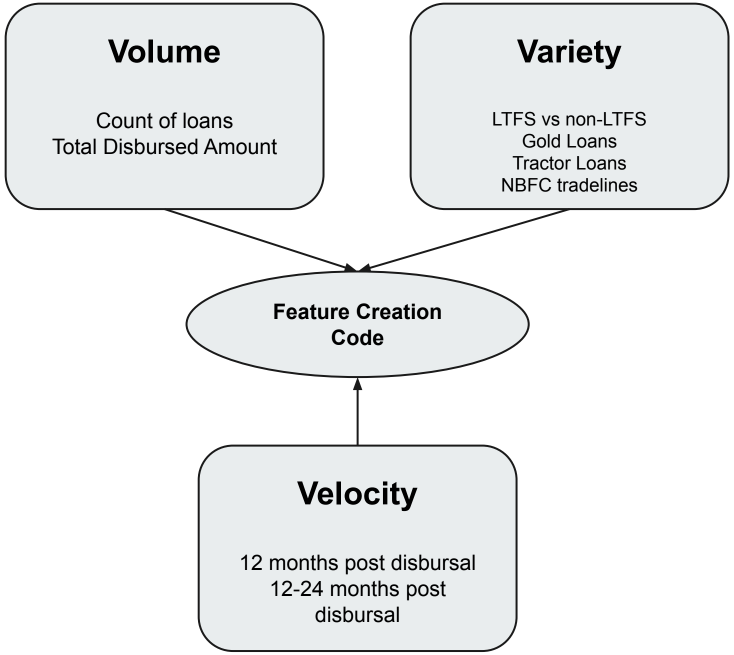
- Our solution pipeline is as follows:



- We will go over each of these steps in detail in the next slides

Feature Engineering Pipeline (1/3)

- Most of feature engineering effort was around *rolling up the bureau information to an application level*
- Our primary goal was to *focus on tradelines post the disbursal of the anchor* loan*
- We created features based on **3V** of the data i.e ***Volume***, ***Variety*** and ***Velocity***
    - **Volume:** focuses on count/amount
    - **Variety:** focuses on type of tradelines
    - **Velocity:** focuses on the time horizons
- Result of this pipeline were features like:
    - Count of loans (*volume*) from LTFS (*variety*) in the 12 months post disbursal (*velocity*)
    - Total disbursed amount from gold loans in the period of 12-24 months post disbursal



*anchor: loan given in the application data

Feature Engineering Pipeline (2/3)

- Insights:
    - Our decision to focus on the bureau information post the disbursal was driven by the fact that, we had to forecast the top-up information of the future. The **tradelines from the future captured the maximum information around top-up loan, it is possible that some of them were top-up loan itself**
    - Features around tractor/gold loan turned out to be most important, especially the time to next tractor/gold tradeline
    - **Importance of Tractor loans from the future**:
        - This kind of tradeline turned out to be very important, we decided to explore this because the portfolio was also tractor loan heavy because of which the top-ups may also be along the same lines (i.e. agri focused)
    - **Importance of Gold loans from the future**:
        - Hypothesis behind this was that an applicant taking a gold loan was an indicator of some sort of distress and hence a scope for top-up loan
        - Same logic could be applied to other tradelines like personal loans/overdraft, but since the gold loans were significant in number it created the most impact
    - **Enquiries data was not provided**, which could have proved beneficial to the model. The number of enquiries made turns out to be a very important signal of need/distress

*anchor: loan given in the application data

Feature Engineering Pipeline (3/3)

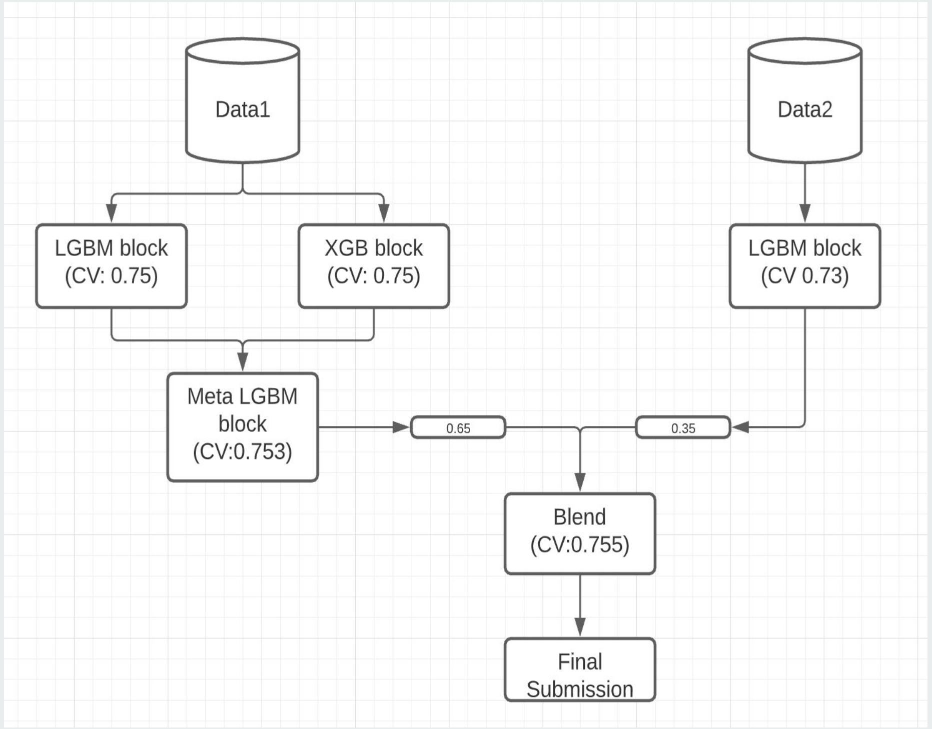
- Things which didn’t work:
    - TF-IDF on the bureau history of the applicant
    - No valuable features from DPD strings, generally these provide valuable signals in underwriting models
    - Historical tradelines don’t seem to be adding much value, most of the signal is present in future tradelines
    - Features around current balance/overdue amount turned out to be not very useful
    - Features only with Application Data does not seem to be much useful compared to bureau data.

*anchor: loan given in the application data

Model Pipeline

- Cross Validation Scheme: Stratified K fold validation (values of K=5&10)
- Models:
  - XGBoost
  - LightGBM
- Modelling Datasets:
  - Data1: 682 features
  - Data2: 250 features
- Final solution was bagged across multiple seeds for reducing the variance of the final solution
- Performance on the leaderboard:
  - Public LB: 0.8495
  - Private LB: 0.835

Model Pipeline Flow



Important Features

Top 5 Application Data Features

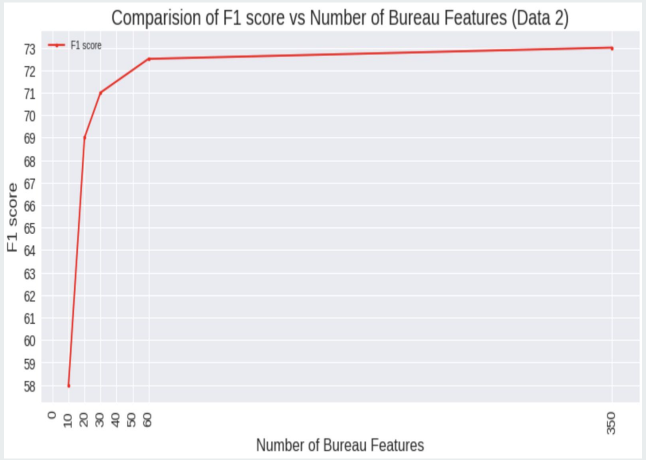
- **MonthlyIncome:** monthly salary of the applicant.
- **Emi_sal_ratio:** Ratio of EMI to monthly income of applicant.
- **DisbursalDate - MaturityDAte:** loan tenor (calculated).
- **Age:** age of the applicant.
- **ZiPCODE:** pincode of the applicant.

Top 5 Bureau Data Features

- ***NonLTFS-Non-Tractor-Loan_DISBURSED_DT_max_days_since_DisbursalDate:*** Number of days between the latest NON-LTFS non tractor loan post the disbursal of anchor loan
- ***NonLTFS_Non Tractor Loan_last_DISBURSED-AMT/HIGH CREDIT:*** Disbursed amount for latest NON-LTFS non Tractor loan post disbursal of anchor loan
- ***Tractor Loan_DISBURSED_DT_next_after_loan:*** Number of days between the anchor loan disbursal date and first Tractor Loan taken post disbursal of anchor loan
- ***DisbursalDT - DISBURSED-DT_next_1:*** Number of days to next loan (post the disbursal of the anchor loan)
- ***gold_2_yr_frm_disbmax_DISBURSED-AMT/HIGH CREDIT:*** maximum disbursed amount of gold loan within 2 years post the disbursal of the gold loan

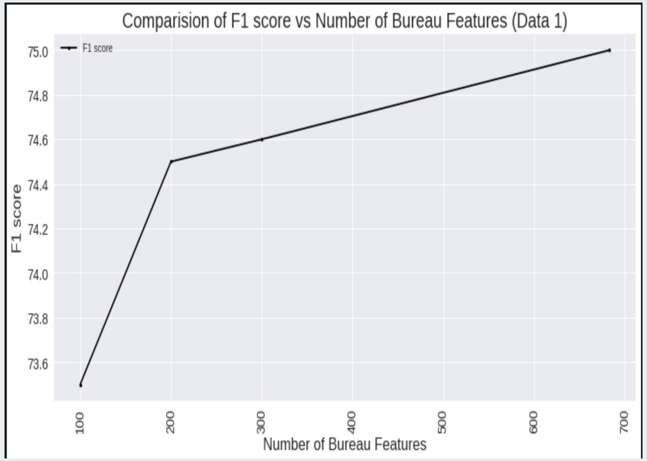
Ablation Study

- We performed multiple experiments to understand the impact of the size of feature space on the model performance.
- We can see that reducing the feature space to merely 100 features we see marginal reduction in performance, this is an important insight from the deployment standpoint



Performance vs Number of Features

Dataset	Number of Features	CV Score
Data1	100	0.735
Data1	200	0.745
Data1	300	0.746
Data1	683	0.75
Data2	252	0.731
Data2	63	0.725

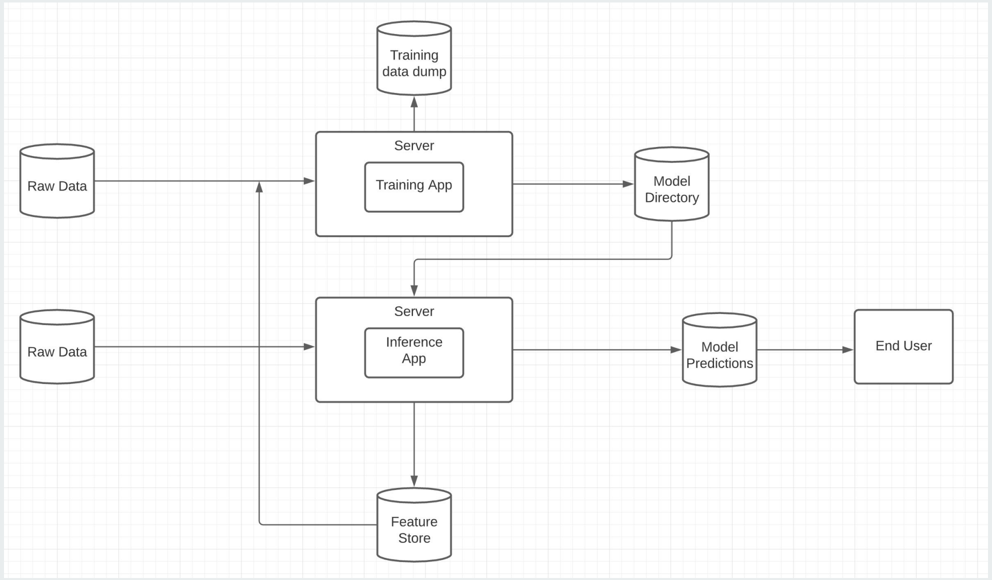


Deployment

- Given the nature of the problem we are solving this can be a standard use case of **Batch Deployment**
- We will have two pipeline one for training and another for inferencing, both will interact in a dynamic manner
- Input to either of the pipeline will be the location where the data is stored (locally/cloud)

Components of the deployment module:

- *Training/Inference app*: Dockerized Flask app which will take folder_path as an input
- *Model Directory*: store all the trained models along with other relevant details like model to column mapping etc.
- *Feature Store*: feature calculated for the model will be stored here, this can be fed to the training pipeline for training and also for audit purposes
- *Training data dump*: stores all the data used to train different versions of the model, used for auditing and debugging purposes
- *Model Prediction*: Stores all the batch scores from the inference pipeline, end users can consume this data for further use



Doubts

- We had number of questions regarding the setup of the problem statement for this hackathon:
  - ***Train/Test Split*** : Why were we given a random split between the train and test set? Given the nature of the problem we are dealing there is bound to be drift in data across time (concept, co-variate etc.). This is also one of the reasons, whenever we benchmark a credit-underwriting model we have an out of time test set to measure performance. This will also to a degree solve for the future information leak problem that we have.
  - ***Size of test data***: The test set was ~10% of the training set and that too was divided into public and private, meaning the results that we saw were effectively only on ~5% of the data. This is too small a dataset to tell which model is better, especially when all the top 3 solutions were very close (this is also compounded by the fact that we have a significant class imbalance, meaning we have very few samples where the top-up was present)
  - ***Presence of future information***: From a problem structuring standpoint, we believe there was future information leakage. For any given loan at some point in time , if we asked to predict top-up why do we have future information from that point on? Had that information been not-present, we would have seen very low scores (of the order of 0.3-0.4*)

*an estimate based on some preliminary analysis