Problem Definition, Given Resources and Assumptions

The objective of the challenge is to *identify home loan default* based on a given dataset for training using an appropriate Machine Leaning model. The model is required to run through on a fresh, unseen test dataset and predict whether the individual data lines belong to a default on a loan or not. The classes are provided in the training set based on previous observations while they need to be predicted in the test set.

Based on the above information, this is a supervised ***binary classification*** problem.

General Assumptions

- Both Training & Testing data have been procured by similar processes/data sources and there is no bias in sampling
- A complete row of observation is uniquely composed of all its features
- A satisfactory ***Training*** accuracy (**above 90%)** will provide a well generalized model

About the given dataset

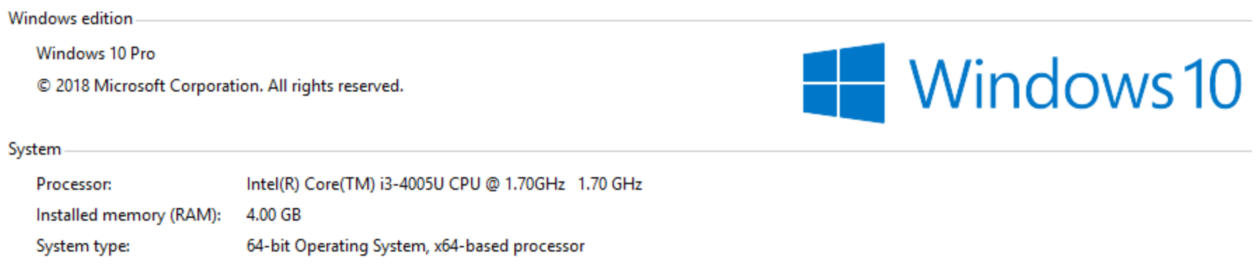
The given dataset shows the following properties:

1. It is moderate in size ~ 100K+ records. This is a ***good training sample size***
2. It has 20+ features and *many features(m1-m12)* are sparsely populated, i.e. significant missing values
3. The Target *class is imbalanced*, i.e. too few observations of one class versus the other
4. It has ***some date fields*** which should ideally be cleansed to unix epcohs for better scalability for a machine learning algorithm

Solution Methodology

Technical Specifications & Environment:

My Laptop specifications→ RAM, processor and Operating System


Windows edition

Windows 10 Pro

© 2018 Microsoft Corporation. All rights reserved.

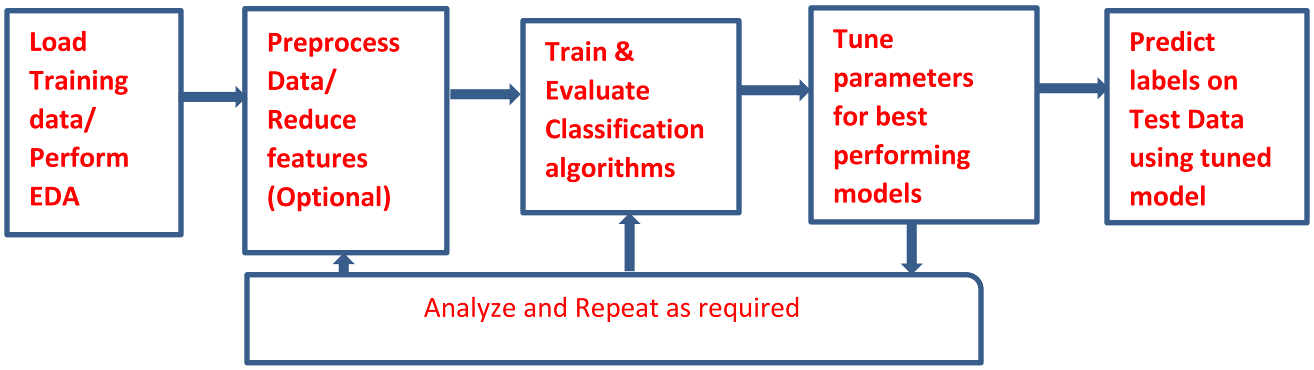
System

Processor:	Intel(R) Core(TM) i3-4005U CPU @ 1.70GHz 1.70 GHz
Installed memory (RAM):	4.00 GB
System type:	64-bit Operating System, x64-based processor

Programming environment/libraries

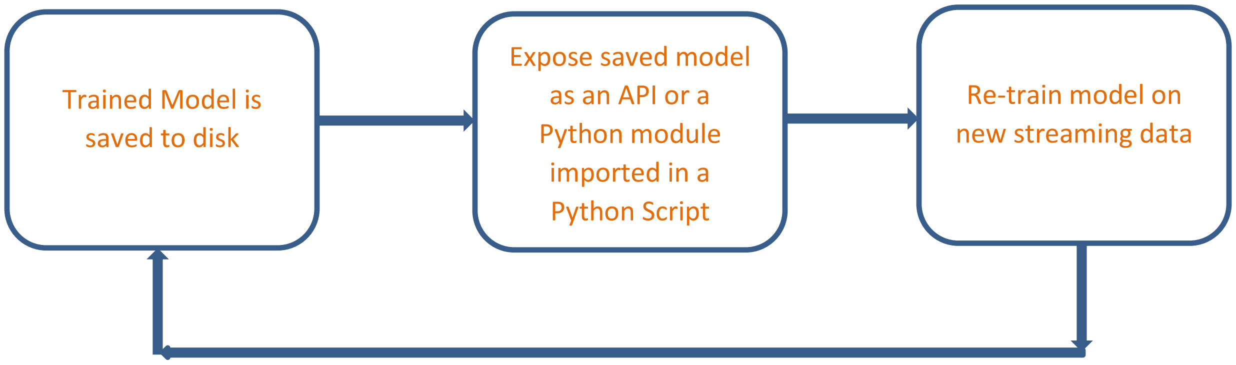
Solution was created on Python3 kernel on *Anaconda Jupyter Notebook* environment. Scikit Learn libraries were used for Machine Learning and Pandas for data manipulation. Other supporting modules for time, plotting etc. were also imported a required. I have created separate class for data processing called, Preprocess for modularity

High Level Process Architecture Diagram



Load Training data/ Perform EDA
Preprocess Data/ Reduce features (Optional)
Train & Evaluate Classification algorithms
Tune parameters for best performing models
Predict labels on Test Data using tuned model
Analyze and Repeat as required

Training a first cut model on training data



Trained Model is saved to disk
Expose saved model as an API or a Python module imported in a Python Script
Re-train model on new streaming data

Continually enrich the created model on new/streaming data

Data Science is an experimentative/stochastic process and a classical binary classification model follows the steps below:

1. Pre-processing

Load Training Data → Deduplicate records→ Check missing data and impute(most frequent value) → Encode categorical data(label encoder, mapping categories to numbers) → Scale continuous variables to be in range (0, 1) → Resample (up sample minority class and down sample majority class) based on class distribution of target variable → *Perform feature engineering to remove redundant predictors* (optional) → Segregate predictors versus target variables (X, y)

All steps above are standard and apply to any machine learning project. However, choosing the methods in each step is a matter of debate and in this case, they were chosen based on both experience and the data properties in question.

I have chosen not to use the ***Pipeline*** *class from scikit learn* as I wanted to customize the individual pre-processing steps and view the intermediate outputs if necessary. Production models should use Pipelines as it helps automate the entire process

Feature engineering→Feature selection

As per Wikipedia, *feature selection* is a specific discipline of feature engineering which aims to find the most relevant features from a given set based on certain criteria. It is suggested that it be *applied when number of features/predictors is greater than 10.* In this case, we have 29 and hence it fits the case. Less features can also help reduce training time in most cases. It is still not an exact science and requires human interpretation. However, automated feature engineering using *evolutionary algorithms* is possible but doesn't guarantee the most optimal result always.

In my implementation, I have used **Tree Based Feature Importance using ExtraTreesClassifier along with correlogram of features**

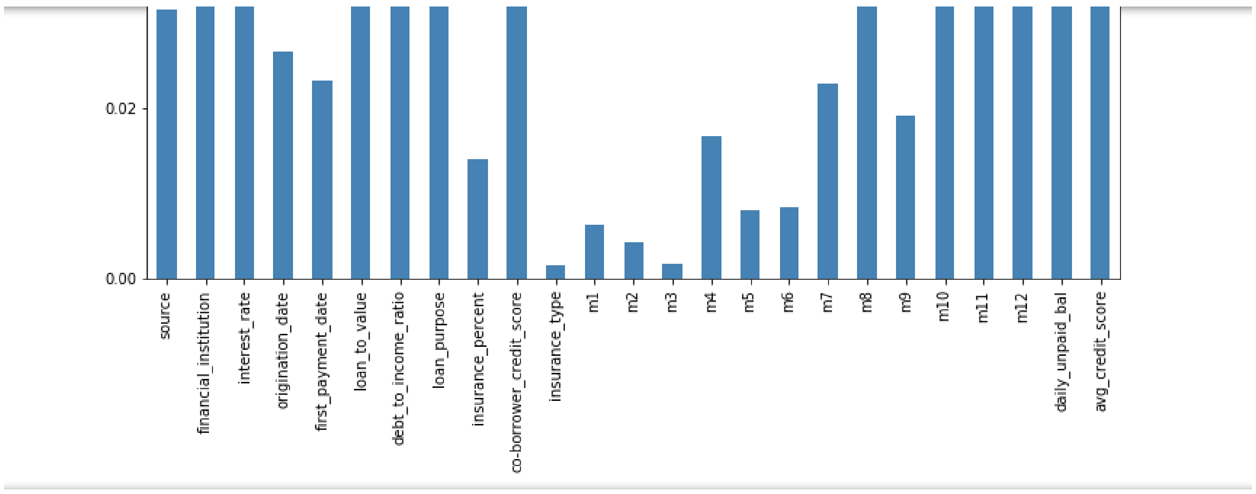
Four step feature engineering/dimensionality reduction

Step 1: Remove any unique key or non-random variables ***or noise variables***

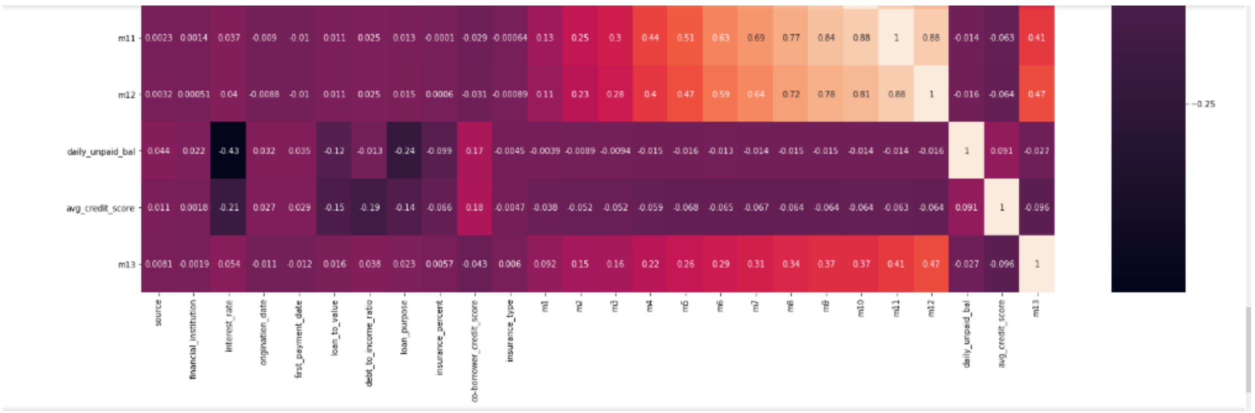
Step 2: Create new features out of existing ones; remove the older features

Example: 'daily_unpaid_bal' = 'unpaid_principal_bal'/ 'loan_term'

Step 3: Look at a feature importance graph generated by a Tree based algorithm and remove least significant ones



Example: From the graph below we can easily spot **insurance_type** as a candidate for removal as it has 0 importance



Step 4: look at a correlation graph and remove features which are a) very lowly correlated with the TARGET or b) are mutually strongly correlated

Example: **Insurance_rate** is strongly correlated to **DAILY_UNPAID_BALANCE** but the latter has higher correlation with TARGET. Hence, we can remove **Insurance_rate**

2. Training classifiers

Train→ check score → keep model with best accuracy → perform a 10-fold F1 score Validated score for each model and pick the best

In order to evaluate training accuracy a ***10-fold cross validated AUC score*** is calculated for each of the 3 different types of classifiers and the best is retained. The best model is then tuned for best parameters.

3. Predict target class

Finally, the best model is used to predict the classes and a file a per specifications is produced for evaluation

Obvious challenges and possible shortcomings of the solution

Training and tuning models is a ***memory and time intensive*** process. Some challenges and shortcomings as below can significantly reduce model performance and reliability of results

- Lack of domain expertise: Though data science is a general science, having knowledge about the domain of data helps to make knowledgeable decisions like the kind of imputation required, the expected number of features to be retained and the accuracy of prediction (precision versus recall) needed for the problem
- Making the right judgements: Most data science coding is ad-hoc at this stage and even though we have automatons like creating a pipeline process or using Grid Search or Random Search of the hyperparameter space, we still can't decide on the best approach without consulting the web. There is surely no 'perfect solution' till we try everything!!
- Overfitting: In our effort to get the best training accuracy, we might fit our model too close to the training dataset and thus increase variance. This is the reason testing accuracy post evaluation is always lower than training accuracy obtained.
- Avoiding memory Intensive algorithms: In the given problem, I couldn't run some more memory intensive algorithms like KNN Classifier for benchmarking. ***Training Neural Networks was costly as well.***
- Execution Speed: ***Speed*** is also an issue as the total runtime of the notebook is around ***40-50 minutes***
- Preprocessing mismatch: As data properties/distribution might vary between test and train the same set of preprocessing steps might not be ideal for both. For example, some new, unseen categories in the test data might not be recognized by the trained model and it might produce an incorrect classification

Questions posed by challenges

➔ Is training accuracy a good indicator of comparable testing accuracy? This can **prevent overfitting** before it's too late

➔ Can we **modularize pre-processing** for a domain-based dataset (like a bank, insurance etc.)? This can help standardize process for a problem type