Detecting fraud from customer transactions using machine learning

One Semester Individual Project Final Report

Author: Stanislav Kataev

Supervisor: Dr. Yuhua Li

School of Computer Science & Informatics Cardiff University

May 2020

Abstract

Fraud detection is a complicated and usually inefficient process, done by checking against specifically generated rules. This process is slow and can be unrealisable because of missing values in transaction datasets. This project explores different strategies of implementing machine learning and feature engineering to improve both the speed and accuracy of the process.

Acknowledgments

I would like to thank my supervisor Dr Yuhua Li for supporting me during this project and introducing to the fascination field of machine learning.

Introduction

Problem and motivation

Fraud prevention has been a big problem in the ecommerce sector since the start of the ecommerce in 1994. When the Internet was only at its early stages of popularity, fraud prevention relied on flagging fraudulent transactions based on predetermined conditions: simple if-statements checking the validity of a transaction. With the rapid development of the sector and growth of companies like Amazon and the possibility of sending goods around the world, fraudulent behaviours have evolved and became harder to identify. Fraud can be defined as an unauthorized transaction, fraudulent transaction due to identity theft or fraudulent requests for returns. Due to the various ways fraud can be committed, a rule-based fraud-detection system may struggle to correctly label transactions.

This led to an increase use of machine learning algorithms for fraud detection. The advantage of using machine learning (ML) compared to rule-based model is the ability of an ML algorithm to identify patterns within transactions and label them based on the previous data that the algorithm learned from. Thus, there is no need to manually make rules to identify legitimate transactions in order to match all types of fraud. This project will look at different ML algorithms suitable for implementation in ecommerce and evaluate on algorithms' performances. ML algorithms used in e-commerce should be fast because it will be used in online mode, efficient in reading a dataset and evaluating specific patterns to identify fraud. In order to make the algorithm efficient it needs a lot of data to learn from. For this project, a dataset from Kaggle.com "IEEE-CIS Fraud Detection" competition was used. The dataset is provided by Vesta Corporation, the world's leading payment service company, and comes from real e-commerce transactions and contains a wide range of features. This project can be divided into following steps:

- Feature engineering and feature extraction in the dataset
- Testing of supervised ML algorithms
- Optimization of supervised ML algorithms
- Testing of unsupervised ML algorithms with supervised algorithms
- Evaluation of results and final verdict for each approach

Aims of the project

The aim is to develop a machine learning algorithm that can accurately predict and successfully identify fraudulent transactions.

Background

This chapter is a brief overview of essential concepts needed to understand the outcomes of the paper.

Machine learning

Machine learning is a subgroup of artificial intelligence, where an algorithm learns from the data, makes predictions or classifies data without much of human interaction. Machine learning does not require as much code as artificial intelligence algorithms because a ML algorithm can learn from the data and rather than follow heuristics only specific to the problem it is solving. Therefore, ML algorithm are more versatile, and the same algorithms can be used to solve different problems. Machine learning is split into two categories: supervised and unsupervised learning.

Supervised learning

Supervised learning algorithms learn from labelled data. This means that a dataset has already labelled values for the target an algorithm tries to predict. To train a supervised algorithm a training and test set are required. The target values are usually called Y and the rest of the dataset is called X. When an algorithm learns the training set data, it can identify which values in X cause a specific value of Y and based on that knowledge it can predict which values of Y should be in the test set. There are two types of functions a supervised algorithm can do: Classification and Regression. As the name suggests, classification assigns the output a class, regression predicts a specific value. Identification of fraudulent transactions is a classification problem, where the instance belongs to either fraud (1) or non-fraud (0) classes.

Boosted Decision Trees

The ML algorithms used for the project are ensemble of Decision Trees algorithms with gradient boosting [1]. Decision Trees are versatile ML algorithms that can perform both classification and regression tasks, and even multioutput tasks []. A tree ensemble trains a group of Decision Trees, where to get a final prediction, each individual trees' predictions is obtained, and the class with the most votes is chosen as the final result. Boosting refers to any Ensemble methods that can combine several weak learners into one strong learner [ ]. The idea is to try train predictors in a sequence and correct their predecessors. Gradient boosting uses gradients to try fit predictors to the *residual errors* of previous predictors. The most popular gradient boosting algorithms are XGBoost, LightGBM and CatBoost. These algorithms are used in this project because of their speed, scalability and accuracy of predictions.

[1] *Greedy Function Approximation: A Gradient Boosting Machine*, by Friedman

XGBoost stands for Extreme Gradient Boosting and is an open-sourced framework for many programming languages. LightGBM is a gradient boosting framework developed by Microsoft. CatBoost is an open-sourced gradient boosting library developed by a Russian search engine company Yandex. It is used in search engines, recommendation systems, self-driving cars and etc. because of its high accuracy and performance even with default parameters. It is widely used by Yandex itself, as well companies like CERN, Cloudflare and more. Hyperparameters tested for this algorithm are:

Unsupervised learning

Unsupervised learning is a type of machine learning where a model is trained on unlabelled data. The most popular type of unsupervised learning is clustering, where an algorithm identifies instances with similar features and groups them into clusters. Another uses of unsupervised learning are density estimation, dimensionality reduction and anomaly detection, where the latter is crucial in identifying fraudulent transactions. Unsupervised learning is useful in identifying groups in data, as well as outliers which do not belong to any other major groups. In the used dataset, there are 95% of legitimate and 5% of fraudulent transactions. Due to the majority being legitimate transactions, they are likely to form a cluster, whereas fraudulent transactions will be outside of that cluster, hence they can be called anomalies.

K-Nearest Neighbours

This is a simple non-parametric machine learning algorithm used for classification and regression. In classification, it plots instances on a single plane and each instance is classified based on the majority vote of its k nearest neighbours. The only changeable parameter is k and it determines the number of neighbours votes. In k=1 classification, an instance is assigned to the class of its neighbour. Changing k can will change the algorithms accuracy and its performance depends on how data is structured. Making k too big or too big may result in incorrect classification, especially if the instances are closely grouped together.

DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN) algorithm was proposed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu in 1996 [2]. It clusters regions of high density by looking at each instance's and how many instances are located within a small distance of ε from it, which is called instance's *ε-neighbourhood* [3]. If there are at least *min-samples* of instances within the neighbourhood, the instance is called a *core-instance*. There can be multiple core-instances that can join into one big neighbourhood. Those instances which are not in any of the neighbourhoods are identified as anomalies. Although

[2] Ester, Martin; Kriegel, Hans-Peter; Sander, Jörg; Xu, Xiaowei (1996). Simoudis, Evangelos; Han, Jiawei; Fayyad, Usama M. (eds.). *A density-based algorithm for discovering clusters in large spatial databases with noise*.

[3] Hands-on machine learning with Skikit-Learn, Keras and TensorFlow

DBSCAN does not have a predict method, it is a powerful clustering algorithm and can be used with *predict()* methods of KNN. It only has two hyperparameters (eps and min_sample) and its computational complexity is O(mlogm).

PCA

Machine learning algorithms sometimes are used for analysing datasets with thousands and even millions of features, and those features can make the training time extremely slow. A lot of features can also make the model overfit and reach a bad solution. The problem of having too many features is often referred to as the curse of dimensionality. In order to solve this problem, a dimensionality reduction algorithm such as PCA is used. Principal component analysis (PCA) is a technique used to emphasize variation and bring out strong patterns in a dataset [[4]]. PCA identifies an axis with the largest amount of variance. It also finds the second axis which accounts for the remaining amount of variance orthogonal to the first axis. This axis is the ith principal components of the data and the number of principal components is equal to number of dimensions. To find principal components a matrix factorization technique called Singular Vector Decomposition (SVD)[[5]] is used. After all principal components are identified, the dimensionality of a dataset is reduced to d number of dimensions by projecting it onto hyperplane defined by the first d principal components [[6]]. The selected hyperplane ensures that projection preserves as much variance as possible.

Dataset

The datasets used in the project are taken from a Kaggle competition "IEEE-CIS Fraud Detection". The provided datasets are "Transactions" and "Identity". The Transactions dataset contains Identity dataset consists of only categorial features and describes information related to the identity of a purchaser, such as device type, device information, network connection information and digital signatures. Transactions dataset contains both numerical and categorical features, and contains transactional information such as transaction amount, transaction time delta. product code, card information, address, distance, email domains of purchaser and recipient, and special features with hidden meaning engineered by Vesta. The datasets are joined by TranscationID column, however not all transactions have corresponding identity information. For a better visualization of features and training models on one dataset, both datasets are merged on the TranscationID column. The shape of the merged dataset is 590540 rows and 434 columns.

Evaluation techniques

The evaluation techniques for the test are Confusion Matrix, Precision, Recall, ROC AUC score. Confusion matrix evaluates performance by comparing predictions to actual values of targets. Each row represent actual classes, while columns represent predicted classes. The first cell in the first row represents true negatives, second cell is false positive, in the second row the

[4] https://setosa.io/ev/principal-component-analysis/

[5] Hands-on machine learning with Skikit-Learn, Keras and TensorFlow

[6] Hands-on machine learning with Skikit-Learn, Keras and TensorFlow

first cell is false negatives and last cell is true positive. Confusion matrix helps to show how well does the model distinguishes between classes and visualizes in a simple way. Precision is the number of accuracy of positive predictions. Recall or true positive rate (TPR) is the ratio of positive instances correctly identified by a classifier. It is also referred to as sensitivity. Precision and recall are usually measured together and they follow a trade-off, which needs to be taken into account when constructing a model. The receiver operating characteristic (ROC) curve plots true positive rate against false positive rate (FPR), which is the ratio of negative instances incorrectly identified as positives. FPR is equal to 1 – true negative rate, which is the ratio of negative instances correctly identified as negative. FPR is referred to as specificity. Therefore ROC is sensitivity versus 1-specificity [7]. A good classifier has the area under the curve (AUC) score equal to 1, whereas a random classifier has a score of 0.5.

Bayesian Optimization

Bayesian optimization, also called Sequential Model-Based Optimization (SMBO), is an efficient method for tuning hyperparameters of a machine learning model. The method builds a probabilistic model of an objective function that maps input values to a probability of the lowest output value, also called the loss. The objective function indicates how much each variable contributes to the value to be optimized in the problem [8]. The probability model is much easier to optimize than the objective function, therefore the method selects a criteria such as Expected Improvement to select next values to evaluate. The method uses Bayesian Reasoning, meaning it re-calculates the surrogate function while incorporating previous information. This makes this method more efficient than other optimization algorithms such as Random Search and Grid Search. The method with which the surrogate function is constructed for the project is Tree Parzen Estimator (TPE) [9].

Software

Python

In this project, all of the code is written in Python 3.7.6. It is a high-level programming language with a lot of built-in libraries for Machine Learning and Visualization. Python is an efficient and simple programming language for writing machine learning algorithms and its list manipulation functionality was essential for feature engineering.

Numpy

Numpy is a Python library, which adds support for large multi-dimensional arrays and matrices, along with high-level mathematical functions.

[7] Hands-on machine learning with Skikit-Learn, Keras and TensorFlow
[8] https://www.courses.psu.edu/for/for466w_mem14/Ch11/HTML/Sec1/ch11sec1_ObjFn.html
[9] https://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf

Pandas

Pandas is a Python library built on top of Numpy, which provides fast, efficient and flexible data structures for data analysis. Pandas DataFrame is used for displaying and manipulation of provided csv datasets.

Ski-Kit learn (Sklearn)

This is a free machine learning library for Python. It provides a big selection of common machine learning algorithms such as Support Vector Machines, K-Nearest Neighbours and Decision Trees for both classification and regression tasks. It also provides modules for pre-processing, evaluation and visualization of data.

The modules used in the project are: StandardScaler, K_Fold, Train_Test_Split, PCA, Roc_Auc_Score, Precision_Score, Recall_Score and Confusion_Matrix. This modules were used for k-fold validation, standardizing data to perform PCA, split data into subsets and evaluation of models performances.

Optuna

Optuna is a hyperparameter optimization framework, which allows for a fast and efficient way of tuning hyperparameters for all of the gradient boosting algorithms. The optimization can be parallelized and provides detailed visualization options.

Specification & Design

System Overview

In this project, the objective is find an efficient implementation of machine learning to identify fraudulent transaction. In order to that, the project is divided into stages to efficiently organise the workflow. The original steps were proving that machine learning can be applied to fraud detection, after which different models and optimization techniques were used. Using the workflow allowed for fast analysis of models and data, as well as ways to modify them. The project is focused on exploring performances of specific algorithms and therefore eliminates any unnecessary calculations, resulting in a clear work pipeline.

The project is broken down into three stages:

- **Data Pre-Processing**
- **Supervised/Unsupervised model selection**
- **Optimization**

Project Structure

The development process started with researching past works related to application of machine learning for classification using high-dimensional datasets. The found material clearly outlined the effectiveness of using machine learning and the focus of the development process was quickly turned to searching the right models for the objective. The stricture of the development process is outlined below:

1. **Research**: Research of most common supervised algorithms and their application for fraud identification. This later focused on researching Decision Trees and Boosted Trees. Research of feature engineering and feature extraction techniques.

2. **Data Pre-processing**: Downloading data, encoding categorial values, filling missing values and creating new features.

3. **Training models:** Training default models of the chosen supervised algorithms and analysing their performances.

4. **Semi-supervised learning:** Experimentations with PCA for dimensionality reduction and DBSCAN for identifying clusters. Both PCA reduced data and clusters from DBSCAN are tested with the chosen supervised algorithms.

5. **Optimization:** Hyperparameter optimization for best model combinations.

6. **Results evaluation:** Final evaluation of all models performances and identification of the best model.

Data Pre-Processing

The original dataset had a lot of missing values and it was difficult to see patterns due to most of features having abstract and hidden meanings. A lot of pre-processing had to be done to prepare data for training models. Boosted trees have a similar concept to normal decision trees and can overfit if there are too many features in the dataset. The merged dataset consisted of 434 features, where some features had a majority of values missing or with high cardinality. When a decision tree looks at a dataset, it splits it in a similar to how a human would do by grouping data based on feature values. When there are many features which either mostly consist of missing values or feature values have high cardinality (over 90%), these features add “noise” to a model and cause it to overfit.

To solve this problem, a lot of feature engineering had to be done. The transactions in the dataset are sequential and happen on a specific time in a year. In Vesta competition data description, it describes that the data was taken in 2017 and spans over 6 months. The strategy for feature engineering was to turn time deltas provided in TransactionDT column into time representation, grouping transactions by month, day and hour. To reduce the noise generated by abstract features, a possible solution was to aggregate features, so decision

trees can better split the dataset. Therefore, It was also important to make columns representing mean and standard deviation of transactions per a period of time. Another strategy was to make features which represent customers id. For that card information and customer addresses were combined. Mean and standard deviation of transaction amounts per customer id were also created to better identify similar customers and behaviors of customers committing fraud. There were also categorical features which had to be turned into numerical before training models.

Supervised/ Unsupervised model selection

They were chosen due to their speed and efficiency handling large databases. This project is focused on selecting of the best model out of gradient boosted trees algorithms XGBoost, LightGBM and Catboost. The problem the project tries to solve is a binary classification problem. Out of the three algorithms, only CatBoost can handle categorial values, thus dataset preprocessing was for XGBoost and LightGBM was different in to comparison to CatBoost. Firstly, each algorithms is tested with default hyperparameters on the pre-processed dataset. It is evaluated against performance metrics, such as precision, recall and training time.

Unsupervised learning algorithms are used in a mixture with supervised. PCA is a dimensionality reduction technique and is used to determine the minimum number of dimensions needed without losing much information. PCA does not guarantee an increase in performance but it proven to decrease training time. PCA is also used for visualization and in combination with DBSCAN can be used to clearly distinguish between classes in the dataset. This stage tests if the addition of unsupervised algorithms increases performance of supervised algorithms. One of the weaknesses of supervised learning is a bad ability to identify the minority class if the size of the class is really small. The algorithm may not have enough data to train how to distinguish between fraud and non-fraud classes. The advantage of using DBSCAN is its ability to identify outliers (fraud) and its fast speed of computation. Fraud prevention requires to be a fast process, therefore it is essential to test if semi-supervised learning is not slower in computation than supervised and increases performance before choosing the models to optimize.

Optimization

Most of the hyperparameters in the boosted tree models deal with the bias-variance tradeoff. It is important to combinations of different hyperparameters to find the most efficient model. In order to do so, Bayesian optimization is used because of its superiority over other hyperparameter techniques such as Grid Search and Random Search. The size of the dataset is large, thus training a model with all combinations of hyperparameters is too time consuming and inefficient. The use of Bayesian Optimization significantly reduced development time and was essential in showcasing models` performances.

Implementation

For the project workflow being more efficient, two versions of the dataset were preprocessed and saved as .cvs datatype. Models were then trained without the need to modify the original dataset for each individual training. All models were trained on Mac OS system using CPUs, meaning models were not parallelized using CUDA GPUs, due to Mac OS being compatible with Nvidia video cards.

Preprocessing

Loading data

The provided datasets are provided as .csv files and in order to make them readable by machine learning algorithms, they need to be loaded using pandas function *read_csv*(). The datasets are loaded as pandas DataFrame objects, which clearly visualized the datasets instances and features. DataFrames have a lot of useful functions such as *info*(), which shows what data types are in which features, and *describe*(), which computes standard deviation, mean, medium for each feature, as well as maximum and minimum values. This functions helped with understanding distribution of values in different features and what features may represent.

The original Identity and Transaction datasets are merged together on the TransactionID column and is called *train*. The labels are put into a separate DataFrame named y_train and copied from "isFraud" column, which is later deleted from train set. In order to decrease the size of the original dataset, a memory reduction function is created. It goes through each column identifying if the value in a column is either an integer or a float, and assigns it a data type based on its size. The function reduced the dataset by 66.8% to 650.48 megabytes.

Splitting data

To test machine learning models the train dataset is split into training and test subsets. This process is beneficial for testing models because it takes less time to train on smaller data sets. There is also no test dataset provided by Kaggle and models can only be tested when submitted on the website. Submitting code for each time would be too time consuming, thus the train dataset is split into two subsets using Slearn train_test_split(). The module splits the dataset 80/20 between a new training set (X_train) and test set (X_test) by firstly shuffling and then stratifying the data to preserve the same ratio of classes as in the original set. The function also creates y_train and y_test labels referring to the same TranscationID as in X_train and X_test. Train_test_split() takes the original set, labels set, test size and random state for reproducible output as parameters.

Feature engineering

As mentioned previously, the dataset has a lot of features but most of them have obscure meaning because of personal data protection. Follow the strategy described in "System and Design" chapter, a function addUids() was created to add three new features: uid1, uid2, uid3. All of feature engineering functions are applied to both X_train and X_test. These features represent different combinations of personal information which may be useful to identify individual card holders. The uid1 feature is a combination of card1 and addr1 features, which supposed to represent card owners card details and location. The uid2 feature is a combination of uid1 with card3 and card5 features. The card4 feature was not used in any of the uids because it represents card issue (Visa, Mastercard, American Express,etc) and is a good feature on its own. The uid3 feature is the combination of uid2 with card2 and addr2 features. This is the final combination and supposed to represent a complete card owners identity. The new features are created by adding values of specific columns as strings and separated by "_". This is to prevent some numerical values to add and create more "noise", and instead be a combination of values, which can form patterns.

To better form an identity of each card holder, new features representing the mean and standard deviation of transaction amount made by each *uid* group are created. Firstly, *X_train* and *X_test "TranscationAmt*" columns are concatenated together, then each *uid* is grouped with the aggregated type of transaction amount. The index is reset and properly assigned to values in a temporary pandas DataFrame object *temp_df*. The values from *temp_df* are mapped to *X_train* and *X_test* new columns. There are some cases where values of standard deviation ended up being infinite, so they were filled with median values of each column. Another step of checking identity is to see if recipient`s (*R_emaildomain*) and purchaser`s (*P_emaildomain*) emails match, since it is more suspicious if recipient's email does not match that of the purchaser. For that a new feature email_match was created, which checks if emails match (1) or do not (0).

The dataset provides time deltas that can be reverted to normal visualization of time. To do that, datetime module was used. Time deltas are converted to seconds and added to the start date of 30/11/2017. From that new features representing month (DT_M), week (DT_W), day of the yeat (DT_D), hour (DT_hour), day of the week (DT_day_week) and day of the month (DT_day) were created to better group periods when transactions took place. DT_M_total, DT_W_total and DT_D_total were also created to show on which dates there the most transactions. Finally, for uid3 as the complete representation of a card holder, three more features were created representing the frequency of transactions done by a card holder per specific month, week and day of the year.

Dealing with categorical features only had to be done for XGBoost and LightGBM, thus there are two versions of the final dataset. In the first dataset categorical features are converted into numerical using frequency encoding. This method of encoding was chosen because it assigns each value its frequency within its column and does not assign a number that can be repeated across other columns, which would be the case if label encoding was used. EncodeFE() function is used to convert 48 categorical features in X_train and X_test.

The last step of the feature engineering process was to delete redundant features and fill NaN values. The function get_useless_columns() returns a list of columns that have either to many

missing values or values with very high cardinality (over 90%). The function returns 12 columns with mostly missing values and 66 columns with high cardinality values. These columns are dropped from both X_train and X_test. Decision tree algorithm are able to deal with missing values, however they do not take them into account. Hence, they are filled with -999 because it is the lowest out of all values in the dataset and models will consider them when training. The last features to drop are TransactionDT and DT because they were used to create other features and strongly correlated to them, adding unnecessary noise to the data. The second dataset undergoes the same transformation except turning categorical features into numerical and is used to train CatBoost.

Supervised models

Each supervised algorithms is trained using default parameters on X_train with y_train, and then predicts labels of the X_test. To calculate the AUC ROC score, boosted trees algorithms need to used predict_proba() method to return an array of rows per instance and columns for every class, each containing the probability of belonging to either class. To calculate precision and recall predict() method is used, which returns an array with a class value in each row. The chosen supervised models have an eval metric parameters that calculates AUC ROC score for each iteration and is set to “auc”.

Models are evaluated using evaluationMetrics() function which trains each model and predicts probabilities for X_test. The functions takes assigns the predicted class to *predictions* and uses it to calculate performance metrics. These sklearn methods are used for evaluations:

- **Recall**: recall_score()
- **Precision**: precision_score()
- **Confusion Matrix**: confusion_matrix()
- **ROC_AUC**: roc_auc_score()

Hyperparameters for the baseline LighGBM are automatically set to default and only eval_metric parameter is used. The baseline model is trained on X_train and tested on X_test without cross-validation. The time to train is measured in second using timer() method.

```
import time
start_time = time.time()

lgb_model = lgb.LGBMClassifier()
lgb_model.fit(X_train,y_train,eval_metric='auc')

default_lgbm_time = time.time() - start_time

evaluationMetrics(lgb_model,X_test,y_test,'LightGBM')

print("Baseline LightGBM train time: ", default_lgbm_time)
```

Baseline XGBoost takes does not take any parameters but eval_metric=”auc” and early_stopping_rounds=10 were used to reduce training time and stop training if performance does not increase after a certain number of rounds. The baseline model was also measured in seconds using timer() method.

```
start_time = time.time()
xgb_clf = xgb.XGBClassifier(random_state=42)
xgb_clf.fit(X_train, y_train,
                early_stopping_rounds=10, eval_metric = 'auc')
default_xgb_time = time.time() - start_time

evaluationMetrics(xgb_clf,X_test,y_test,'XGBoost')

print("Baseline XGBoost train time: ", default_xgb_time)
```

Default CatBoost takes iterations=500, early_stoping_rounds=20, eval_metric=”auc” and categorical_features parameters, which includes categorical column indices. The model also has a plot parameter that was set to True so it can show the ROC curve after training. The baseline was timed in seconds using timer() method.

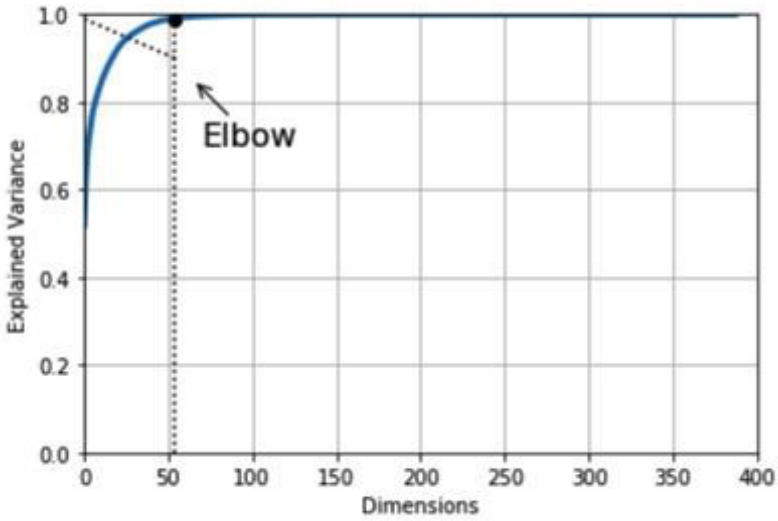
```
start_time = time.time()

cat_boost=CatBoostClassifier(iterations=500, early_stopping_rounds=20, eval_metric="AUC")
cat_boost.fit(X_train, y_train,cat_features=indeces,plot=True)
default_cat_time = time.time() - start_time

evaluationMetrics(cat_boost,X_test,y_test,"catboost")
print("Baseline CatBoost train time: ", default_cat_time)
```

Unsupervised

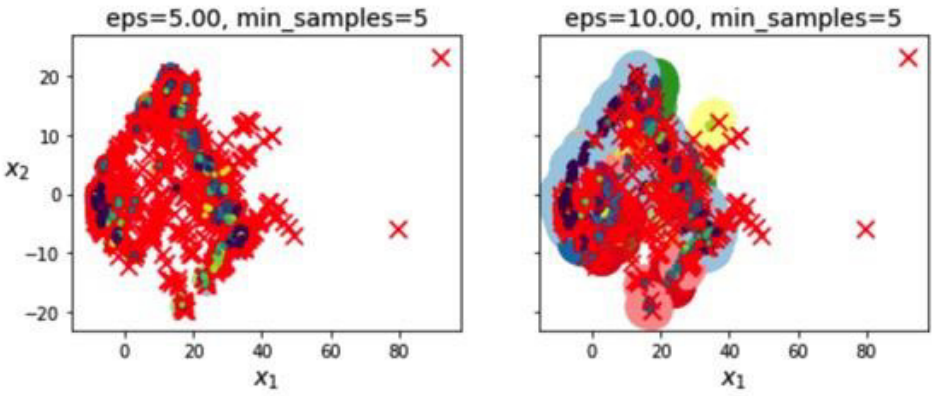
The X_*train* dataset is high-dimensional, therefore to use DBSCAN it is better to perform dimensionality reduction. Since DBSCAN is density-based algorithm, it takes a lot of time to cluster data. To use PCA, X_train has to be scaled. To do so, a sklearn StandardScaler() method is used, which standardizes values by the subtracting mean value and dividing by the standard deviation. After standardization, the number of dimensions to reduce down to has to be chosen first. This is done by choosing the number of dimensions that add up to a sufficiently large portion of the variance, 99% in this example. The following code returns the number of dimensions without transforming X_train dataset.

```
pca = PCA()
pca.fit(X_red)
cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.99) + 1
d
```

```
54
```



The graph shows that shows that variance ratio stops growing at 54 dimensions. Using this number of dimensions, a copy of X_train is created and transformed. X_reduced is then used to fit DBSCAN, which produces indices of the core instances in the core_sample_indeces istance variable and core instances in components_ instance variable. The graph shows how DBSCAN clustered instances after fitting X_reduced. Finally, to predict values of X_test, K-Nearest Neighbours algorithm is used. Two versions of DBSCAN are used with one having eps=10 and eps=5 in another. The min_sample value was left at 5 for both tests. KNN parameter was set to 1 because the aim is to group legitimate and fraudulent transactions into completely separate clusters. To test if dimensionality reduction increase performance of the supervised models, classifier are trained on X_reduced and tested against X_test.



Optimization

Bayesian optimization is applied to LightGBM, XGBoost and CatBoost. Three different classes are created to optimize the classifiers separately. These hyperparameters are chosen to be tuned:

CatBoost:

- **cat_features**: this feature tests CatBoost`s ability to handle categorical data and essential for testing the full potential of the algorithm.
- **max_death**: This is the depth of the base tree. It has high impact on training time.
- **subsample**: Sample rate of rows.
- **colsample_bylevel**: Sample rate of columns.
- **scale_pos_weight:** Defines the weight of a class in binary classification.
- **random_strength:** The amount of randomness to use for scoring splits when the tree structure is selected. Used to avoid overfitting.
- **one_hot_max_size**: Uses one-hot encoding for all categorical features with a number of different values less than or equal to the given parameter value.
- **reg_lamda**: L2 regularization coefficient
- **bagging_temperature**: defines the settings of the Bayesian bootstrap.

XGboost:

- **max_depth**: maximun depth of a tree.
- **reg_lambda**: L2 regularization term on weights.
- **reg_alph**a: L1 regularization term on weight.
- **eta**: equal to learning rate in GBM. Corresponds to how quickly error is corrected from each tree to the next.
- **gamma**: minimum loss reduction required to split a node.
- **tree_method**: “hist” method is chosen to speed up computation with CPUs.
- **objective**: “binary:logistic” means logistic regression for binary classification.

LightGBM:

- **learning rate**: equal to eta in XGBoost
- **max_depth**: maximum depth of a tree.
- **num_leaves**: controls the complexity of a tree.
- **min_data_in_leaf**: minimum data in a one leaf node. Used to deal with overfitting.
- **feature_fraction**: determines the fraction of parameters the model will select in each iteration when building a tree.
- **subsample**: Sample rate of rows.

Optimized models were pickled using joblib.dump() and cross-validated on the *X_train* dataset using sklearn K-Fold validation where the number of folds k=5.

Results

The sections reviews the outcomes of experiments described in Implementation section. The hardware used for these experiments is 1.4 GHz Quad-Core Intel Core i5 CPU and 8 GB 2133 MHz LPDDR3 RAM. GPU was not utilized due to MACOS incompatibility with NVIDIA cards.

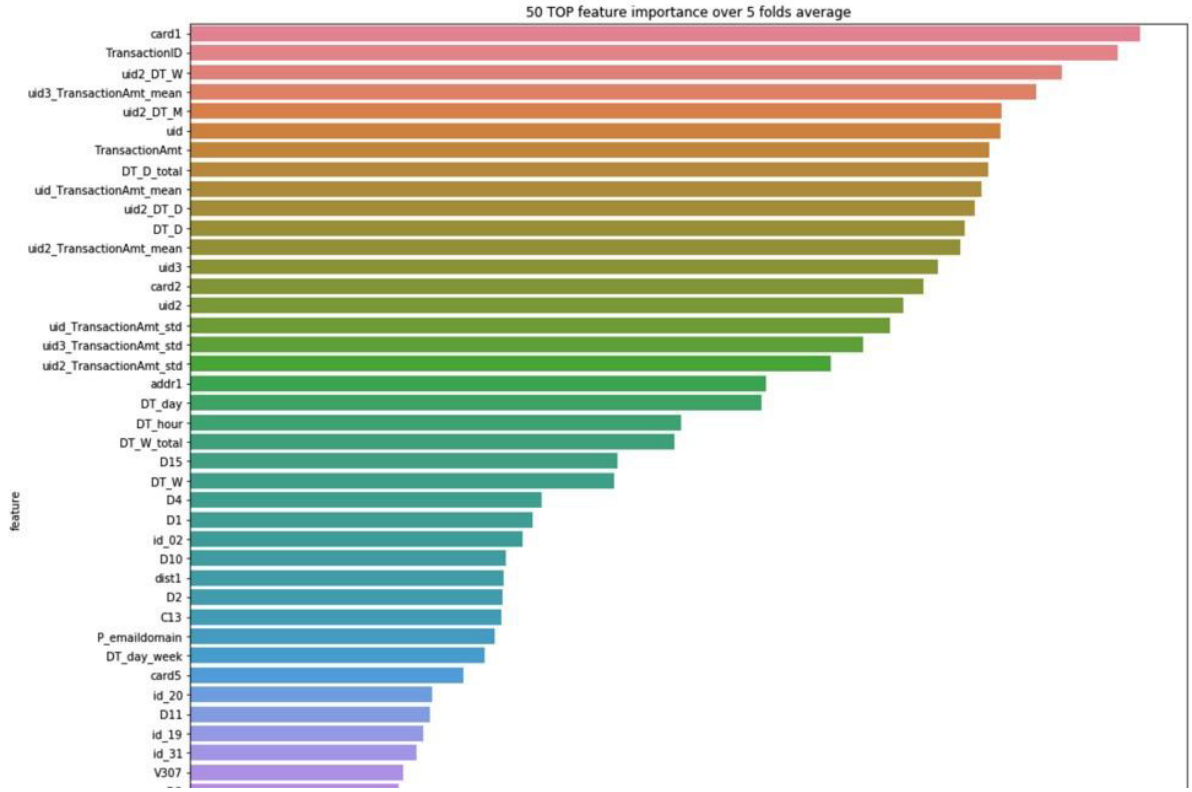
Default LightGBM

```
LightGBM
AUC 0.9336096374579206
Recall 0.44130127298444133
Precission 0.9030390738060782
Confusion Matrix
[[113665    201]
 [  2370   1872]]
Baseline LightGBM train time:  26.975600957870483
```

Optimized LightGBM

```
Early stopping, best iteration is:
[1281]  training's auc: 1       valid_1's auc: 0.969766
Fold 3 | AUC: 0.969765994957916
Training until validation scores don't improve for 500 rounds
[200]   training's auc: 0.997802        valid_1's auc: 0.960495
[400]   training's auc: 0.999979        valid_1's auc: 0.963383
[600]   training's auc: 1       valid_1's auc: 0.9645
[800]   training's auc: 1       valid_1's auc: 0.964863
[1000]  training's auc: 1       valid_1's auc: 0.964711
[1200]  training's auc: 1       valid_1's auc: 0.964773
[1400]  training's auc: 1       valid_1's auc: 0.965027
[1600]  training's auc: 1       valid_1's auc: 0.964727
[1800]  training's auc: 1       valid_1's auc: 0.964351
Early stopping, best iteration is:
[1449]  training's auc: 1       valid_1's auc: 0.965176
Fold 4 | AUC: 0.9651757454226192
Training until validation scores don't improve for 500 rounds
[200]   training's auc: 0.998042        valid_1's auc: 0.96303
[400]   training's auc: 0.999983        valid_1's auc: 0.966609
[600]   training's auc: 1       valid_1's auc: 0.96692
[800]   training's auc: 1       valid_1's auc: 0.9674
[1000]  training's auc: 1       valid_1's auc: 0.96698
[1200]  training's auc: 1       valid_1's auc: 0.967062
Early stopping, best iteration is:
[751]   training's auc: 1       valid_1's auc: 0.967513
Fold 5 | AUC: 0.9675129945904906

Mean AUC = 0.9686640196923737
Out of folds AUC = 0.9656333618964867
```

LightGBM is also able to clearly visualizes the importance of top features with its best_feature() module.



Default XGBoost

```
[98]    validation_0-auc:0.890915
[99]    validation_0-auc:0.891203
XGBoost
AUC 0.8912030877291242
Recall 0.29207920792079206
Precission 0.893294881038212
Confusion Matrix
[[113718    148]
 [  3003   1239]]
```

Optimized XGBoost

```
[0]     validation_0-auc:0.840505
Will train until validation_0-auc hasn't improved in 200 rounds.
[99]    validation_0-auc:0.950042
[0]     validation_0-auc:0.810599
Will train until validation_0-auc hasn't improved in 200 rounds.
[99]    validation_0-auc:0.95345
[0]     validation_0-auc:0.817011
Will train until validation_0-auc hasn't improved in 200 rounds.
[99]    validation_0-auc:0.950207
[0]     validation_0-auc:0.807011
Will train until validation_0-auc hasn't improved in 200 rounds.
[99]    validation_0-auc:0.948529
[0]     validation_0-auc:0.841213
Will train until validation_0-auc hasn't improved in 200 rounds.
[99]    validation_0-auc:0.950008
####################
Out of folds AUC= 0.9505259444672531
```

Default CatBoost

```
catboost
AUC 0.9729065347273339
Recall 0.7248939179632249
Precission 0.9449907805777504
Confusion Matrix
[[113687    179]
 [  1167   3075]]
Baseline CatBoost train time:  634.1967852115631
```

Optimized CatBoost

```
Shrink model to first 565 iterations.
0:      learn: 0.5653663        test: 0.5675777 best: 0.5675777 (0)     total: 151ms    remaining: 2m 30s
100:    learn: 0.2095883        test: 0.2077236 best: 0.2077236 (100)   total: 15.5s    remaining: 2m 17s
200:    learn: 0.1702384        test: 0.1878232 best: 0.1878232 (200)   total: 31.1s    remaining: 2m 3s
300:    learn: 0.1437714        test: 0.1788318 best: 0.1788318 (300)   total: 46.2s    remaining: 1m 47s
400:    learn: 0.1263564        test: 0.1765377 best: 0.1762575 (391)   total: 1m 1s    remaining: 1m 31s
500:    learn: 0.1129381        test: 0.1745743 best: 0.1745743 (500)   total: 1m 16s   remaining: 1m 15s
600:    learn: 0.1015091        test: 0.1746672 best: 0.1744625 (511)   total: 1m 29s   remaining: 59.7s
700:    learn: 0.0926775        test: 0.1752717 best: 0.1744625 (511)   total: 1m 45s   remaining: 44.9s
Stopped by overfitting detector  (200 iterations wait)

bestTest = 0.1744624625
bestIteration = 511

Shrink model to first 512 iterations.
####################
Out of folds AUC= 0.9727395781019154
```

KNN for eps = 5

```
from sklearn.neighbors import KNeighborsClassifier
dbscan = dbscan2
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(dbscan.components_, dbscan.labels_[dbscan.core_sample_indices_])
preds=knn.predict_proba(X_val)[:,1]
roc_auc_score(y_valid,preds)
```

```
0.3389767117640948
```

KNN for eps = 10

```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(dbscan.components_, dbscan.labels_[dbscan.core_sample_indices_])
preds=knn.predict_proba(X_val)[:,1]
roc_auc_score(y_valid,preds)
```

```
0.46686059566139726
```

Summary of results

Out of all of the supervised models, CatBoost performed the best. Even with default parameters its AUC ROC score is higher than optimized LightGBM and XGBoost have. This proves that its ability to process categorical values does increase performance. The only downside was its speed, where LightGBM showed its superiority. Both KNN algorithms performed very badly and calculation of dbscan2 with higher eps value took long time, which means that supervised learning is the definite winner in solving this problem.

Conclusion

The projects aim was to develop efficient machine learning models to identify fraudulent transactions and research different strategies to increase performance of those models. The project researched both supervised and unsupervised approaches to solving the given problem, as well as methods of optimization. This was a challenging project mostly because of the large size of the dataset and its obscurity. Results show, that using grouping strategies for feature engineering proved beneficial and algorithms give those features high rankings. It was challenging to come with a way to sort data, however feature engineering payed off and made the data more structured. It is important to mentioned individual performances of each algorithm. CatBoost is a clear winner in terms of AUC score, however it took the most amount of time to optimize.

The problem with not being able to parallelize the training and optimization process using GPU really affected the workflow. LightGBM and XGBoost also showed great performances, with LightGBM being the fastest of all three algorithms. The use of Bayesian optimization helped to quickly find performance boosting combinations of parameters, although it is possible that better combinations exist. Bayesian optimization may have started exploiting error minimizing values before exploring other possibilities. This could have been done using GridSearch, however it would be inefficient to build using a CPU. In comparison to supervised models, unsupervised models achieved very low results. DBSCAN was proven useless with kNN, even after using PCA. There is a possibility that DBSCAN could work better if correct values of eps and min_sample were found, however only increasing eps value proved to increase performance, and large eps started to require O(m2) memory. Unfortunately, it was not possible to try to use neural networks and work with Keras or Tensoflow libraries due to projects time constraints.

To summarize, this was an interesting and demanding project but an overall a great learning experience, which showed how different strategies work when trying to solve the same

problem. It also opens up possibilities for future work and further improvement to the models, which are not perfect but do an already great job in identifying fraud.

Future work

As it was mentioned before, one of the main challenges of working with a large dataset is a slow speed of calculation and fitting of models. The use of CUDA and NVIDIA would drastically decrease computation times and allowed for a faster workflow. CUDA is well integrated with Python through many libraries and it would not be a hard task to implement GPU parallelism to solve the given task. Perhaps the use of GPU could find better hyperparameter combinations using either Bayesian Optimization or other methods of parameter tuning. Even Grid Search would be a viable option if the code was parallelized.

This would also allow for a faster computation of each model and possibility of using them as an ensemble. Using multiple models in an ensemble where each model could vote on the value of a specific instance may result in increase in recoil, since it was the metric which showed the lowest values. This would not be feasibly to do using CPU due to the long speed of calculation when the goal is to make a model that can be deployed fast. The gained speed would allow the model to be used online and operate on an updated data.

Grouping strategy and frequency encoding showed good results during feature engineering, however the strategy does not solve the problem of most of the values missing. The V-columns seem to be closely correlated, thus it would be beneficial to perform a correlation analysis of these columns, leave those with the most unique values and discard those that are closely correlated. Another strategy would be to perform time-complexity analysis for all columns and discard those which do not increase AUC ROC score. It would be interesting to see how discarding features would affect performance of each model.

Using GPU parallelism would also allow to search for optimal values of eps and min_sample for DBSCAN. Although unsupervised algorithm did not perform well, completely discarding the idea of using them would seriously narrow down future research. There are many unsupervised algorithm that can be tried for both clustering and dimensionality reduction. An example of a possible improvement would be the use of HDBSCAN, which is a recent algorithms developed by creators of DBSCAN. The new version only has one parameter min_cluster_size and completely eliminates eps parameter. Due to this being an anomaly detection problem, it would be interesting to research more about anomaly detecting algorithms such as Isolation Forests. Although PCA was already explored in this project, its function fit_inverse() is a popular choice for anomaly detection, as it identifies outliers when tries to revert compressed data back to its original state.

Reflection

Being introduced to the world of machine learning has been a great experience both from learning and fun perspective. When working on the project I have managed to research a lot

of different techniques how to build efficient machine learning models and how they are deployed. Although I had no prior knowledge of machine learning, I enjoyed researching how to feature engineering works and its application in managing data. I am inspired to learn more about machine learning and start learning how to apply neural networks. This challenging but yet rewarding project taught me a lot of new information and helped me to improve my knowledge of programming.

References

1. *Greedy Function Approximation: A Gradient Boosting Machine*, by Friedman
2. Ester, Martin; Kriegel, Hans-Peter; Sander, Jörg; Xu, Xiaowei (1996). Simoudis, Evangelos; Han, Jiawei; Fayyad, Usama M. (eds.). *A density-based algorithm for discovering clusters in large spatial databases with noise*.
3. Hands-on machine learning with Skikit-Learn, Keras and TensorFlow
4. https://setosa.io/ev/principal-component-analysis/
5. Hands-on machine learning with Skikit-Learn, Keras and TensorFlow
6. Hands-on machine learning with Skikit-Learn, Keras and TensorFlow
7. ands-on machine learning with Skikit-Learn, Keras and TensorFlow
8. https://www.courses.psu.edu/for/for466w_mem14/Ch11/HTML/Sec1/ch11sec1_ObjFn.html
9. https://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf