Project Report

NAME: Ali Rafeeq

Contents

METHODOLOGY

1.1. Data Preparation

1.1.1. Analyzing the Data

The first step we did was to analyze the data to know the percentage of nulls in each column where we found that the null values in several columns exceeded 50%.

```
owner_2_score                          Missing: 1596   (88.2%)
RATE_owner_2                           Missing: 1596   (88.2%)
CAP_AMOUNT_owner_2                     Missing: 1608   (88.8%)
PERCENT_OWN_owner_2                    Missing: 1551   (85.7%)
owner_3_score                          Missing: 1800   (99.4%)
RATE_owner_3                           Missing: 1800   (99.4%)
CAP_AMOUNT_owner_3                     Missing: 1800   (99.4%)
PERCENT_OWN_owner_3                    Missing: 1768   (97.7%)
years_in_business                      Missing: 16     (0.9%)
RATE_ID_FOR_years_in_business          Missing: 16     (0.9%)
fsr                                    Missing: 495    (27.3%)
RATE_ID_FOR_fsr                        Missing: 1791   (99.0%)
location                               Missing: 11     (0.6%)
RATE_ID_FOR_location                   Missing: 74     (4.1%)
funded_last_30                         Missing: 0      (0.0%)
RATE_ID_FOR_funded_last_30             Missing: 1804   (99.7%)
judgement_lien_percent                 Missing: 0      (0.0%)
RATE_ID_FOR_judgement_lien_percent     Missing: 0      (0.0%)
INPUT_VALUE_ID_FOR_judgement_lien_amount  Missing: 0   (0.0%)
RATE_ID_FOR_judgement_lien_amount      Missing: 0      (0.0%)
INPUT_VALUE_ID_FOR_judgement_lien_time Missing: 1810   (100.0%)
RATE_ID_FOR_judgement_lien_time        Missing: 1810   (100.0%)
```

1.1.2. Data Preprocessing

We created a function to preprocess the data. The first step we did was drop the columns with null values greater than 65%.

The other columns with null values we dealt with using different approach:

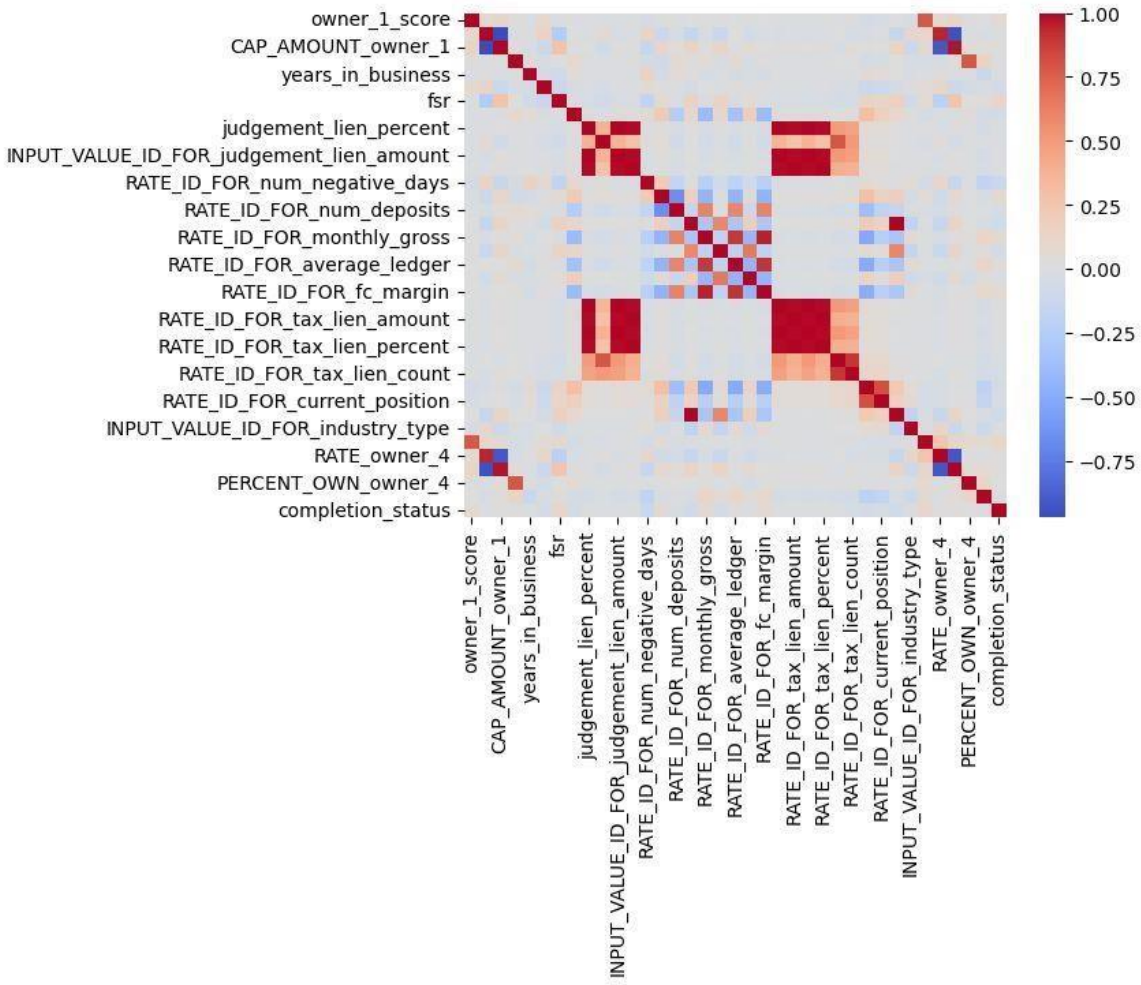
1. The columns where the null percentage was less than 1.5% and were float or int data type we used KNNImputer from scikit.impute library which is a method used to impute missing values in a dataset using k-nearest neighbors algorithm.
2. The columns where the null percentage was greater than 1.5%, we filled the nulls with the mean value of its column.
3. The columns where the null percentage was less than 10% and had object data type were filled with mode of its column.

1.1.3. Data Encoding

For the encoding we used OrdinalEncoder which performs ordinal encoding on categorical values.

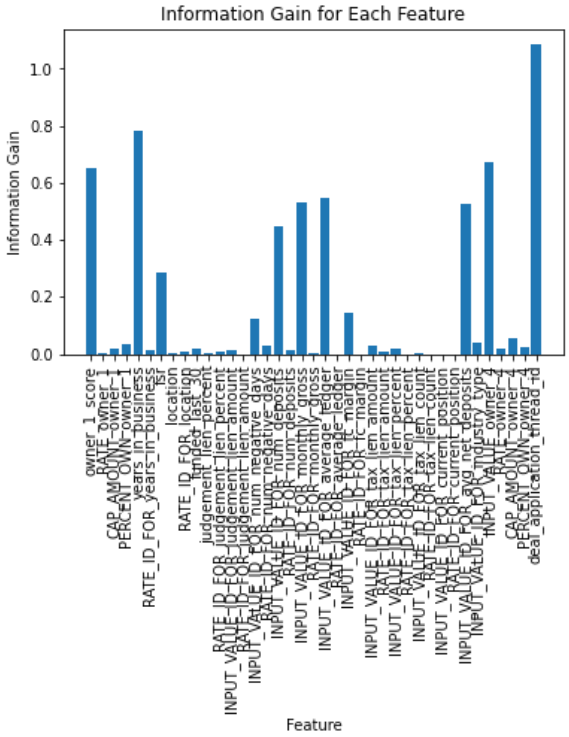
1.2. Data Visualization and Feature Selection

We used correlation to drop the columns irrelevant to our data. As we can see in the figure below:



We can tell from the correlation graph that most of the correlations with the "completion_status" column is greater than 0.1 therefore we dropped all the columns less than 0.1.

We also visualized the feature gain graph which is shown in the figure below:



Many columns had the information gain values less than 0.2 but if a feature has a small information gain but is still relevant to the problem being solved, it may be worth keeping in the model. Since we are unable to decide whether the features are relevant or not, we decided to keep them all.

1.3. Classification Models

To classify our data, we used four classification models. As for the hyperparameters used for each model we decided to use a grid search function that outputs the best hyperparameters to use for each model by giving it the hyperparameter's options and a 20% of the whole train dataset as a validation dataset to choose the hyperparameters that give the highest prediction accuracy.

1.3.1. Extreme Gradient Boosting (XGBoost)

The first model we used was XGBoost. In this algorithm, decision trees are created in sequential form. Also, Weights play an important role where weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and the variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model.

we created a confusion matrix with 20% of the data as a validation set to check our accuracy as shown in the figure below.

```
Confusion matrix:
 [[109   1   0   0]
 [  1 154   0   0]
 [  0   5  76   0]
 [  0   0   0  16]]
```

The highest accuracy, precision, recall, and f1-score were:

```
Accuracy: 98.07 %
Precision: 98.84 %
Recall: 98.07 %
F1 score: 98.42 %
```

This accuracy was achieved with the following hyperparameters which were found using a grid search:

```
Best Parameters: {'learning_rate': 0.5, 'max_depth': 5, 'n_estimators': 100}
```

1.3.2. Gradient Boost Classifier

The sixth model we used was Gradient Boost. This model enables us to combine the predictions from various learner models and build a final predictive model having the correct prediction.

we created a confusion matrix with 20% of the data as a validation set to check our accuracy as shown in the figure below.

```
Confusion matrix:
 [[107   1   2   0]
 [  3 152   0   0]
 [  0   3  78   0]
 [  0   1   0  15]]
```

The highest accuracy, precision, recall, and f1-score were:

```
Accuracy: 97.24 %
Precision: 97.9 %
Recall: 96.35 %
F1 score: 97.09 %
```

This accuracy was achieved with the following hyperparameters which were found using a grid search:

```
Best Parameters: {'learning_rate': 0.5, 'n_estimators': 200}
```

1.3.3. Decision Tree Classifier

The third model we tried was Decision Tree. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules, and each leaf node represents the outcome.

we created a confusion matrix with 20% of the data as a validation set to check our accuracy as shown in the figure below.

```
Confusion matrix:
 [[102   3   5   0]
 [  8 146   1   0]
 [  0   7  74   0]
 [  0   1   0  15]]
```

The highest accuracy, precision, recall, and f1-score were:

```
Accuracy: 93.09 %
Precision: 94.56 %
Recall: 93.01 %
F1 score: 93.75 %
```

This accuracy was achieved with the following hyperparameters which were found using a grid search:

```
Best Parameters: {'criterion': 'entropy', 'max_depth': None, 'min_samples_split': 2}
```

1.3.4. Random Forest Classifier

The last model we used was the Random Forest. Random Forest is a classifier that contains several decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset. So, instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

we created a confusion matrix with 20% of the data as a validation set to check our accuracy as shown in the figure below.

```
Confusion matrix:
 [[103   5   2   0]
 [  3 149   3   0]
 [  0   5  76   0]
 [  1   0   0  15]]
```

The highest accuracy, precision, recall, and f1-score were:

```
Accuracy: 94.75 %
Precision: 95.95 %
Recall: 94.34 %
F1 score: 95.11 %
```

This accuracy was achieved with the following hyperparameters which were found using a grid search:

```
Best Parameters: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}
```

CONCLUSION

In our project we tried four different classification models which were: XGboost, Gradient Boost Decision Tree, and Random Forest. For these models hyperparameters' we used a grid search function to find the best hyperparameters to achieve the highest accuracy using 20% of the train data as a validation dataset.

Out of the four models we used the **XGboost** and the **Gradient Boost Classifier** which managed to get the highest prediction accuracies.