

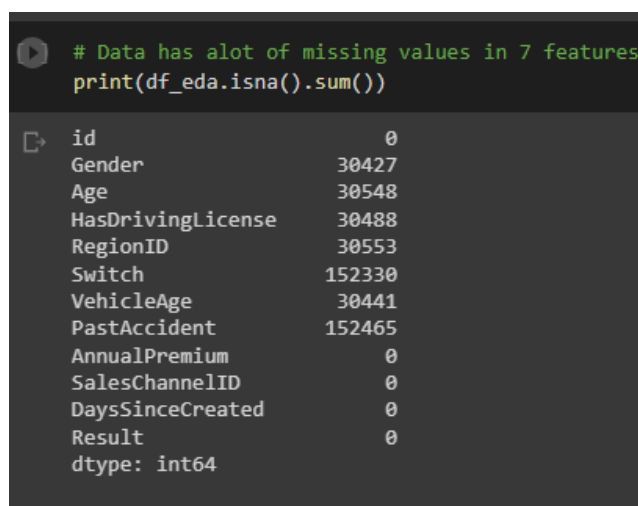
Implementation and Evaluation of a Case Study Using Machine Learning Techniques

Data Exploration

There are several steps to start exploring this data, like, get a sense of the size and structure of your data, visualize data, statistics to identify any unusual values or outliers, checked for missing and look for correlations. Since this is a classification problem, following analysis is performed as well.

- Balance of classes (especially target) in the dataset.
- understand the relationships between different classes in the dataset.

Result of the above mentioned analysis was driving the data processing and models to be used. The 'Result' aka target class was hugely **imbalanced**. 87.8% result were for not buying insurance (result=0) and 12.2% for buying insurance (result=1). The data contained a lot of missing values. Refer to figure.



```
# Data has alot of missing values in 7 features
print(df_eda.isna().sum())
```

id	0
Gender	30427
Age	30548
HasDrivingLicense	30488
RegionID	30553
Switch	152330
VehicleAge	30441
PastAccident	152465
AnnualPremium	0
SalesChannelID	0
DaysSinceCreated	0
Result	0
dtype:	int64

- 'Has Driving License' feature had almost 90% values 'true'.
- 'Region ID': looking at the graph having sharp favour towards particular region with significant difference in count.
- 'Switch' and 'Past Accident' were inversely correlated and had almost 50% missing values each.
- Customers with past accident or vehicle age between 1-2 or previously didn't have insurance, were willing to buy insurance.

Data Pre-processing

To handle the imbalanced data, **SMOTE** is used. SMOTE generated and added the synthetic samples to data to handle the minority class (result = 1). **Simple Imputer** and **KNN Imputer** is used to handle the missing values in features. KNN Imputer uses all available information to estimate the missing values instead of dropping or placeholder like 'N/A', which can lead to more accurate models.

- 'Has Driving License' - imputed using 'most frequent values'. 'Age' - imputed using median.
- Dropped the rows with missing values for 'Region ID' (almost 30k dropped).

- 'Switch' and 'Past Accident': they were important features. KNN Imputer was used to fill in the missing values for 'Switch', 'Past Accident', 'Gender' and 'Vehicle Age'.

'Age', 'Annual Premium' and 'Days Since Created' were the **numerical features**. All other were categorical features. The data types of **categorical features** were set to string and numerical features were set to float for model's easy interpretation features data values.

One-hot encoding was used for categorical features. Labelling couldn't be used here, as it introduces ranking(priority) into the data. **StandardScaler** was done on numerical feature to bring values within model interpretable range.

After data cleaning and zero missing values, we retained ~ 90% of the data. Now, implement pipeline for models: use the tools mentioned above. Use this pipeline with model classifier, with addition of SMOTE to oversample minority class.

Model Implementation

Following classification models were implemented, the reason to choose them is as follows:

XG - Boost:

It is a gradient boosting algorithm using decision trees that can handle imbalanced datasets well, where each subsequent tree is trained on the errors made by the previous tree. It has a lot of hyper parameters to tune to handle imbalanced dataset.

Decision Trees:

Decision trees can be effective for imbalanced datasets because they can learn from the minority class and create rules to predict it. Prone to overfitting, so important to tune the hyperparameters carefully.

Random Forests:

Random forests can handle imbalanced datasets well, as they can learn from the minority class and create a diverse set of rules to predict it. They are less prone to overfitting than decision trees, but are more computationally expensive to train.

Hyper Parameter Tuning

Hyperparameter tuning is important because it can help the model learn from the imbalanced data more effectively and make more accurate predictions. Tuning using metric such as precision, recall, or the F1 score, which are all sensitive to class imbalance. **F1 Score is the optimization metric** used in this project.

GridSearchCV is used for hyper parameter optimization in this project with ± 1 from the default parameters. GridSearchCV is a deterministic method, it return a best parameters from given set of parameters. **RandomSearchCV** is efficient search method, it gives a good set of parameters, but it might miss the best.

Stratified Kfolds cross validation is used instead of Kfolds cross validation. It is a better choice for imbalanced datasets as the folds are created in a way that preserves the class balance within each fold.

Performance Evaluation

As discussed in previous section, metric such as precision (the proportion of positive predictions that are correct), recall (the proportion of actual positive cases that the classifier correctly identifies), or the F1 score – It is the harmonic mean of precision - recall ($2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$), which are all sensitive to class imbalance. Taking this into account, our main focus will be on F1 Score and PR Curve-AUC. Let's look at the following figure from analysis:

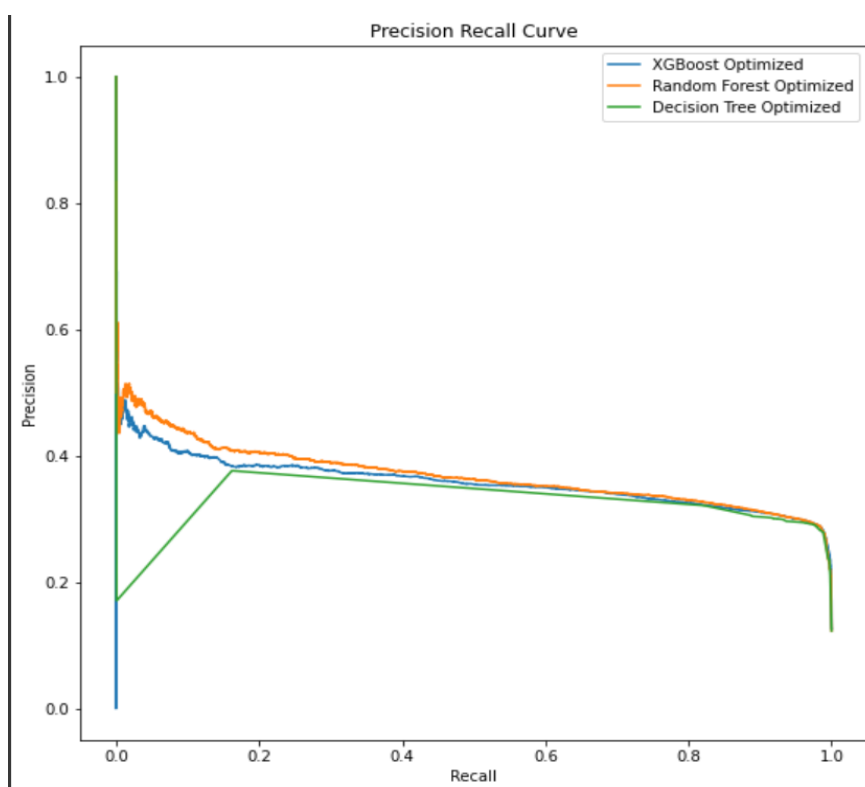


Figure 1 PR Curve - Hyper Parameter Tuned Models

	Models	Accuracy	F1 Score - Major Class	F1 Score - Minor Class	PR-AUC
0	XGBoost Unbalanced	0.877413	0.934695	0.002176	0.375013
1	XGBoost Balanced	0.722421	0.813822	0.454734	0.359321
2	XGBoost Balanced - Hyper Parameter Tuned	0.770489	0.854068	0.462847	0.358223
3	Random Forest Balanced	0.828386	0.899586	0.410124	0.353281
4	Random Forest Balanced - Hyper Parameter Tuned	0.746139	0.833709	0.463746	0.368756
5	Decision Tree Balanced	0.833234	0.904745	0.330977	0.371751
6	Decision Tree Balanced - Hyper Parameter Tuned	0.739663	0.829011	0.454754	0.327750

Figure 2 Scores of All Models

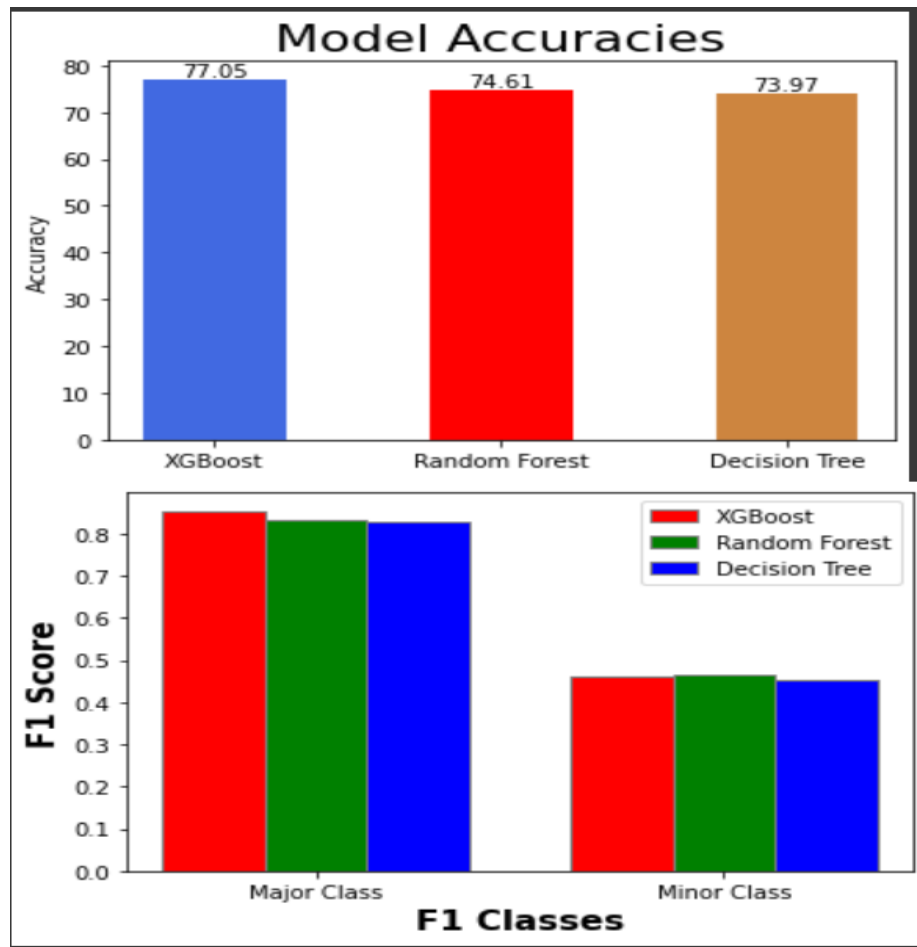


Figure 3 Hyper Parameter Tuned Models

Result Analysis and Discussion

All 3 models were best suited for such unbalanced dataset. Their performance was quite similar with minor differences.

Unbalanced vs Balanced (SMOTE) Dataset

Look at the first row of above image, unbalanced data have higher accuracy for XGBoost as compared to balanced data, but the F1 Score of minor class is very good for balanced data. This is true for other 2 models as well.

Hyper Parameter Tuned – Balanced Dataset Only

All the models showed slight improvements with hyper parameter tuning. Decision Tree improved a lot as per F1 Score, but PR-AUC decreased. Random Forest takes the most time/computation for hyper parameter tuning and it is the most improved. F1 Score of major minor classes for all 3 models were almost similar after hyper parameter tuning.

Best Model

Random Forest is the best model. The F1 score was very good [major=0.89, minor=0.41] and PR-AUC is 0.353. There was improvement in minor class f1-score after hyper parameter tuning [major=0.85 minor=0.46] and PR-AUC is 0.368. XGBoost also performed very well. It was already optimized on default parameters. Not much change in PR-AUC after hyper parameter tuning.

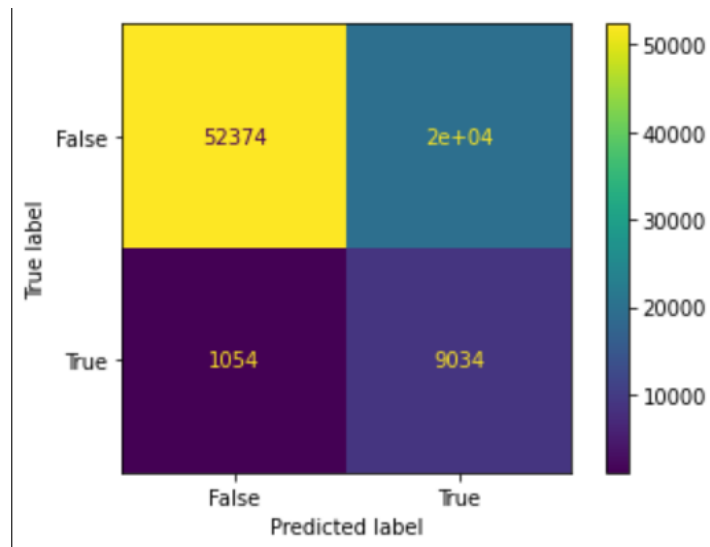


Figure 4 Confusion Matrix Best Model – Random Forest Tuned

Conclusion

- KNN Imputer handles missing values with significant improvement in model performance.
- Dataset was unbalanced, important features had almost 50% missing values.
- Balancing dataset and hyper parameter tuning improves models performance.
- Any single metric cannot tell about the performance of the model. You need to look and compare different metric depending upon the problem.

Future Work

- Explore feature engineering.
- Implement different classification models like Logistic Regression, ADABOOST, etc and artificial neural network.