Progress Report 5

In this report, I'll talk about technical details of modelling and results of the projects.

Why LightGBM?

In previous report, I mentioned that LightGBM is chosen as the baseline model for the project. LightGBM is a gradient boosting framework that utilizes a tree-based learning algorithm to construct powerful ensemble models. It offers several advantages that make it well-suited for imbalanced classification tasks. Why we utilized LGBM?

LightGBM is chosen for several reasons:

- It offers a built-in imbalance technique, automatically adjusting class weights to handle class imbalance during training.
- LightGBM minimizes the need for extensive preprocessing steps, simplifying the workflow.
- It's widely popular and recognized in practice for its efficiency and performance in handling complex datasets.

Modelling: Technical Details

As we talked in previous report, we built 6 models for each dataset:

1. LGBM Baseline
2. LGBM Upsample
3. LGBM Downsample
4. SMOTE LGBM
5. LGBM Balanced Bagging
6. LGBM_Imbalance

For each of the models above, I developed a function, then combined all of the models in one function to in order to make the code DRY. Each of the functions returns the model and the report of the train and test results.

LGBM Baseline

Function developed for baseline LGBM model. This function is also used in downsample, upsample, SMOTE functions

```
def base_lgbm(df_train:pd.DataFrame,
              df_test:pd.DataFrame,
              target_col:str,
              dataset_name:str,
              model_name:str="LGBM Baseline",
              message:str="Building LGBM Model..."):
    # Train data
    X_train = df_train.drop(columns=target_col)
    y_train = df_train[target_col]

    # Test data
    X_test = df_test.drop(columns=target_col)
    y_test = df_test[target_col]

    print(message)
    clf = lgb.LGBMClassifier()
    clf.fit(X_train, y_train)

    prediction_train = clf.predict(X_train)
    prediction_test = clf.predict(X_test)
    prediction_proba_train=clf.predict_proba(X_train)
    prediction_proba_test=clf.predict_proba(X_test)

    cr_test = classification_report(y_test,prediction_test,zero_division=True, output_dict=True)
    f1_test = cr_test['1']['f1-score']
    acc_test = cr_test['accuracy']
    auc_test = roc_auc_score(y_test, prediction_proba_test[:,1])
    cr_train = classification_report(y_train,prediction_train,zero_division=True, output_dict=True)
    f1_train = cr_train['1']['f1-score']
    acc_train = cr_train['accuracy']
    auc_train = roc_auc_score(y_train, prediction_proba_train[:,1])

    report = {
        "Dataset":dataset_name,
        "Model":model_name,
        "f1_test": f1_test,
        "f1_train":f1_train,
        "accuracy_test" : acc_test,
        "accuracy_train" :acc_train,
        "AUC_test" :auc_test,
        "AUC_train" :auc_train
    }
    print("ROC Curve for test data: {}".format(auc_test))
    plot_sklearn_roc_curve(y_test,prediction_proba_test[:,1])

    return clf, report
```

Other Models

For the report, I will not add the functions for other methods, you can find in "Modelling_Final.ipynb" notebook file in code folder. All of the modelling functions return model itself and report of training and test results. All of them have the same input parameters. Here are the function prototypes of upsample and downsample:

```
def upsample(df_train:pd.DataFrame,
             df_test:pd.DataFrame,
             target_col:str,
             dataset_name:str,
             model_name:str="LGBM Upsample",
             message:str="Building LGBM Upsample Model..."):
```

```
def downsample(df_train:pd.DataFrame,
               df_test:pd.DataFrame,
               target_col:str,
               dataset_name:str,
               model_name:str="LGBM Downsample",
               message:str="Building LGBM Downsample Model..."):
```

Other functions are:

```
smote_lgbm, balanced_bagging_lgbm, lgbm_imbalance
```

Final Model Function

By combining all the previous functions, I developed the function "built_experiment" function. This function builds all the models, saves the models and returns the dataframe of combined results.

```
def built_experiment(df_train:pd.DataFrame,
                     df_test:pd.DataFrame,
                     target_col:str,
                     dataset_name:str)
```

Results

In Figure 1. you can see the test and training F1 scores on synthetic datasets. We can see that on test scores for lower minority percentages upsample, smote and downsample performing better than baseline model. But if we look at the train results in those cases we see that they are highly overfitted and can’t be used in production. LightGBM’s built-in balancing technique demonstrates superior data generalization compared to the aforementioned techniques.

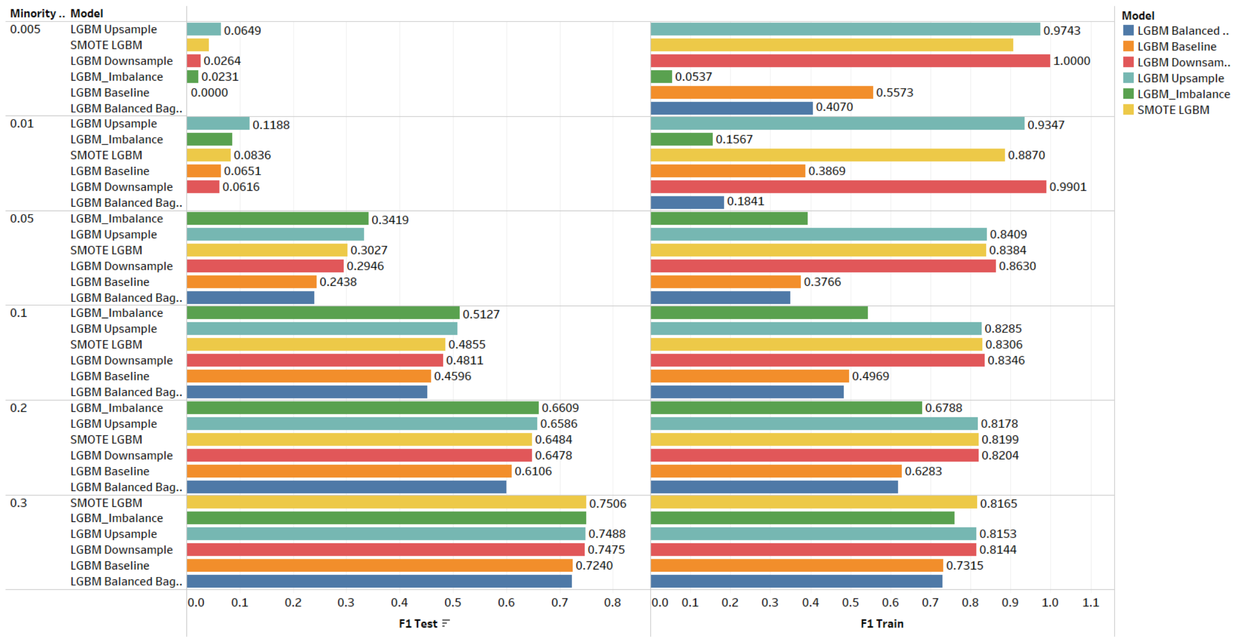

Figure 1.

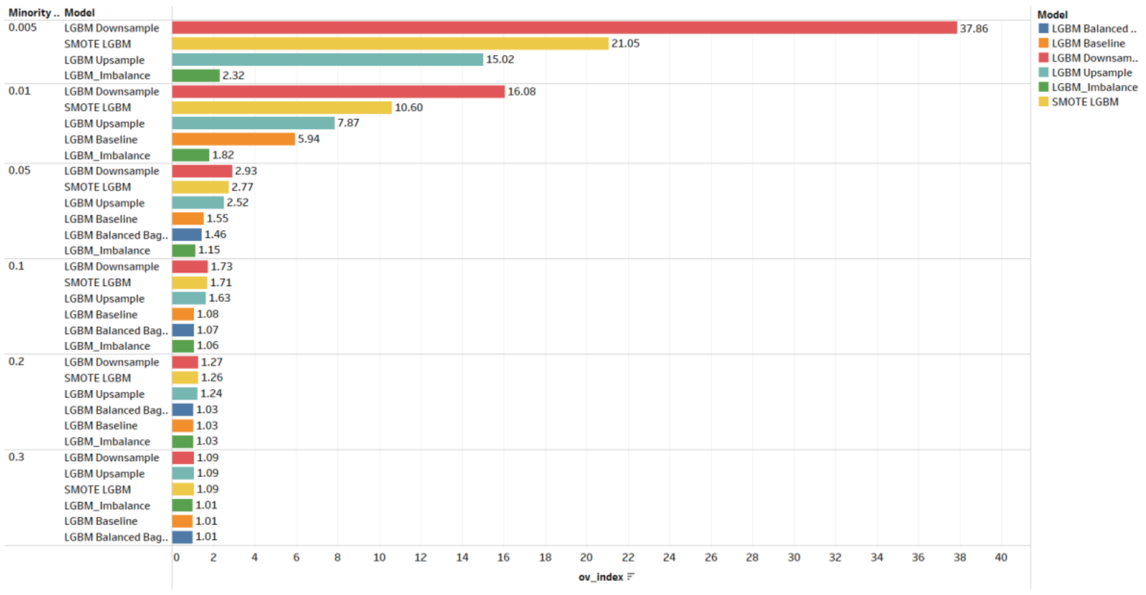

Figure 2.

In Figure 2, we can see which models overfitted mostly. For this plot, we used below metric:

$$\frac{F\_train}{F\_test}$$

We can see that downsample, upsample and SMOTE are most overfitted models. In general, these models overfits below the 10% minority percent.

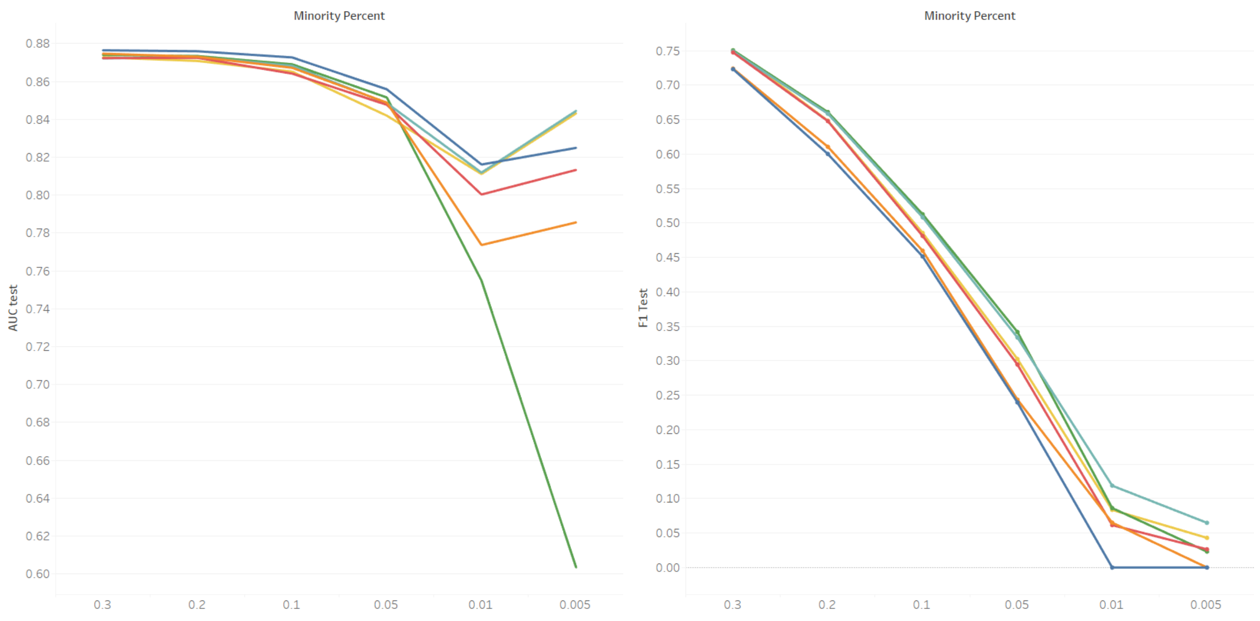

Figure 4.

In Figure 4, we see that from minority class percent 30% to 10%, the performances of balancing techniques and the baseline model are very close. Our observation indicates that balancing techniques tend to be less effective when the class imbalance percentage exceeds 10%.

By now, we only examined the results for the synthetic datasets, we can see the similar results for the other collected datasets. Since we have different classification tasks, results are also dependent on the nature of the data, Therefore, we developed two more metrics,

$$AUC_{Relative} = \frac{AUC_t}{AUC_{baseline}}$$

Where AUC_t indicates the AUC score of specific technique, $AUC_{baseline}$ is the AUC score of baseline model. Similarly,

$$F1_{Relative} = \frac{F1_t}{F1_{baseline}}$$

In Figure 5, we see that after 10%, the performances of balancing techniques and the baseline model are very close.

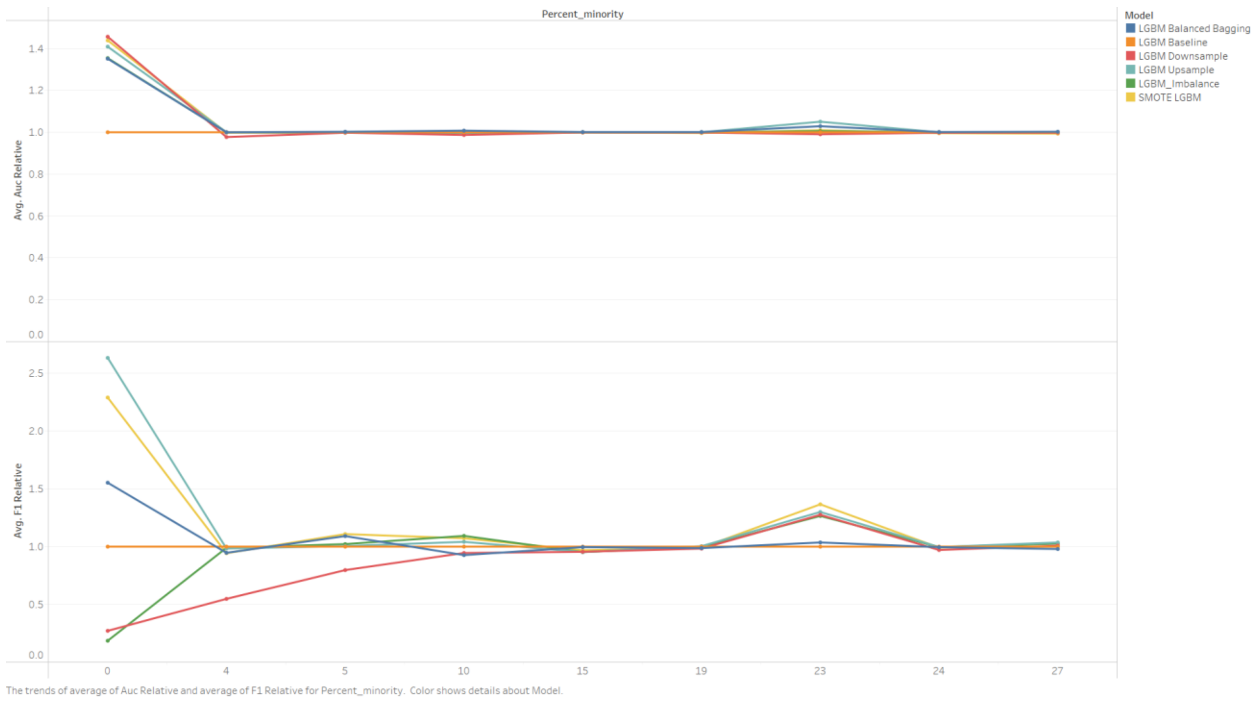

Figure 5.

Conclusion

In conclusion, our research underscores the significance of thoughtful model selection and balancing strategies in addressing class imbalance. By carefully evaluating techniques and considering the degree of imbalance, we can enhance model performance and create more reliable and effective machine learning solutions.

- It is advisable to construct a baseline model without any balancing technique when dealing with an imbalanced dataset.
- Our observation indicates that balancing techniques tend to be less effective when the class imbalance percentage exceeds 10%.
- Certain balancing methods, notably downsampling, SMOTE, and upsampling, have the potential to lead to overfitting.
- LightGBM's built-in balancing technique demonstrates superior data generalization compared to the aforementioned techniques.

Future Work

Extending our research, we envision the following avenues of exploration:

- **More Broad Datasets**: Expanding our analysis to encompass a broader range of datasets from diverse domains will provide a deeper understanding of how balancing techniques perform in various real-world scenarios.

- **More Balancing Techniques**: Investigating additional and advanced balancing techniques beyond those studied in this research can contribute to a more comprehensive comparison and offer practitioners a wider array of options.
- **Tests on Different Algorithms**: Extending our evaluation to include different machine learning algorithms will help ascertain whether the observed conclusions remain consistent across various modeling approaches.
- **Calibrating Models Trained on Balanced Data**: Examining the calibration of models trained on balanced data sets is crucial, as it ensures that predicted probabilities align closely with actual outcomes, enhancing the model's interpretability and reliability.
- **Preventing Overfitting with Alternative Methods**: Exploring alternative techniques to mitigate overfitting risks, beyond feature engineering, such as regularization and ensemble strategies, can help ensure model stability and robustness.
- **Explaining Mathematical Reasons Behind Conclusions**: Delving into the mathematical underpinnings of our conclusions can offer a deeper insight into the observed trends and contribute to a more comprehensive understanding of the interplay between class imbalance and model performance.