ML Project Report: Lend or Lose (Checkpoint 2)

Team Members

1. Trupti Khodwe - IMT2022007
2. Sourav M Dileep - IMT2022033
3. Rishabh Dixit - IMT2022051

Github Link: https://github.com/truptikhodwe/Lend-or-Lose

Task

Training ML models to predict which individuals are at the highest risk of defaulting on their loans is essentially a **binary classification problem**, where the target variable is the likelihood of default. For checkpoint 2, we were supposed to train the dataset on the following models:

- Support Vector Machine (SVM)
- Logistic Regression
- Neural Networks

And also apply skew reduction to reduce bias in the data.

For EDA, pre-processing and results for models of checkpoint 1, please refer to the previous report: Report 1: Lend or Lose

Results for various models (Checkpoint 2)

Sr. no	Model Name	Validation Accuracy	Test Accuracy	Best hyperparameters
1.	SVM without skew reduction	88.59	88.447	Default: `{'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gamma': 'scale', 'kernel': 'rbf', 'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking': True, 'tol':`

				0.001, 'verbose': False}
2.	SVM with skew reduction	67.99	68.241	{'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gamma': 'scale', 'kernel': 'rbf', 'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking': True, 'tol': 0.001, 'verbose': False}
3.	Neural Networks without skew reduction	88.49	88.44	The optimal number of units in the first densely-connected layer is 128, in the second layer is 288 and the optimal learning rate for the optimizer is 0.0001.
4.	Neural Networks with skew reduction	56.74	87.52	The optimal number of units in the first densely-connected layer is 448, in the second layer is 224 and the optimal learning rate for the optimizer is 0.0001.
5.	Logistic Regression without skew reduction	88.48	**88.586 (Highest Accuracy)**	N/A
6.	Logistic Regression with skew reduction	42.06	48.682	N/A

Results for various models (checkpoint 1)

Sr. no	Model Name	Validation Accuracy	Test Accuracy	Best hyperparameters
1.	Decision Tree without feature selection	88.40	87.879	{‘criterion’: ‘entropy’, ‘max_depth’: 10, ‘min_samples_leaf’: 4, ‘min_samples_split’: 10}
2.	Decision Tree with feature selection	88.54	88.388	{‘criterion’: ‘entropy’, ‘max_depth’: 10, ‘min_samples_leaf’: 4, ‘min_samples_split’: 10}
3.	K Nearest Neighbours	88.44	88.214	{'metric': 'manhattan', 'n_neighbors': 11, 'weights': 'uniform'}
4.	Random Forest Classifier	88.81	88.686	{'max_depth': 20, min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 100}
5.	XGBoost without feature selection	88.709	88.774	{'colsample_bytree': 0.8, 'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200, 'subsample': 0.8}
6.	XGBoost with feature selection	88.74	**88.791 (Highest Accuracy)**	{'colsample_bytree': 0.8, 'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 100, 'subsample': 0.8}
7.	Gradient Boosting without feature selection	88.696	88.748	{'learning_rate': 0.1, 'max_depth': 3, 'min_samples_split': 5, 'n_estimators': 100, 'subsample': 0.8}

8.	Gradient Boosting with feature selection	88.714	88.768	{'learning_rate': 0.1, 'max_depth': 3, 'min_samples_split': 5, 'n_estimators': 100, 'subsample': 0.8}
9.	Adaboost	88.672	88.742	{'learning_rate': 1.0, 'n_estimators': 100}
10.	Naive Bayes Classifier	88.505	88.484	N/A
11.	Linear & Polynomial Regression without feature selection	9.96	88.490	N/A
12.	Linear & Polynomial Regression with feature selection	10.14	88.447	N/A

Details about the models used

1. Support vector machine
    a. The preprocessing which I did was the same as before. I applied the SVM model in two cases, one where I did skew reduction and one where I did not do skew reduction.
    b. Results when I did not do skew reduction:
        i. I trained the model using default hyper parameters, which were: `{'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gamma': 'scale', 'kernel': 'rbf', 'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking': True, 'tol': 0.001, 'verbose': False}`
        ii. Model accuracy with default hyper parameters was 88.59

iii. Applied the same model on the test.csv testing data and got a score of 88.447 on kaggle.
iv. Saved the results in the following file: submission_svm_default_without_smote.csv
v. Note: I tried applying GridSearchCV for finding the best hyper parameters but it was taking a lot of time, so I stopped running it after almost 40 hrs.

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits
[CV] END .....................C=0.1, gamma=1, kernel=rbf; total time=35.8min
[CV] END .....................C=0.1, gamma=1, kernel=rbf; total time=37.7min
[CV] END .....................C=0.1, gamma=1, kernel=rbf; total time=39.3min
[CV] END .....................C=0.1, gamma=1, kernel=rbf; total time=101.5min
[CV] END .....................C=0.1, gamma=1, kernel=rbf; total time=37.5min
[CV] END ..................C=0.1, gamma=1, kernel=linear; total time=222.7min
[CV] END ..................C=0.1, gamma=1, kernel=linear; total time=228.8min
[CV] END ..................C=0.1, gamma=1, kernel=linear; total time=227.5min
[CV] END ..................C=0.1, gamma=1, kernel=linear; total time=917.1min
[CV] END ..................C=0.1, gamma=1, kernel=linear; total time=210.9min
[CV] END ...................C=0.1, gamma=0.1, kernel=rbf; total time=36.5min
[CV] END ...................C=0.1, gamma=0.1, kernel=rbf; total time=32.9min
[CV] END ...................C=0.1, gamma=0.1, kernel=rbf; total time=35.8min
[CV] END ...................C=0.1, gamma=0.1, kernel=rbf; total time=35.0min
[CV] END ...................C=0.1, gamma=0.1, kernel=rbf; total time=36.0min
[CV] END ................C=0.1, gamma=0.1, kernel=linear; total time=236.2min
[CV] END ................C=0.1, gamma=0.1, kernel=linear; total time=375.6min
```

c. Results when I did skew reduction:
i. I trained the model using default hyper parameters, which were: `{'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gamma': 'scale', 'kernel': 'rbf', 'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking': True, 'tol': 0.001, 'verbose': False}`.
ii. Model accuracy with default hyper parameters was 67.99
iii. Applied the same model on the test.csv testing data and got a score of 68.241 on kaggle.
iv. Class distribution after SMOTE:

Default:

0 144328

1 144328

v. Saved the results in the following file: submission_svm_default_with_smote.csv

2. Neural Networks:
    a. The pre-processing remains the same as before. The model was trained twice, once with skew reduction and once without.
    b. Without skew reduction:
        i. Before applying the model, I tried hyperparameter tuning using Bayesian optimization. I set the tuner values to:

```
max_trials = 10, executions_per_trial = 2
```

And the tuner search ran for 10 epochs.

        ii. This gave the following best parameters:

```
The optimal number of units in the first
densely-connected
layer is 128, in the second layer is 288 and the
optimal learning rate for the optimizer
is 0.0001.
```

        iii. The model was then fitted on the training data split with epochs = 50.

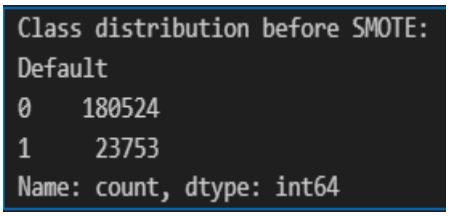
```
Epoch 1/50
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
4086/4086 ━━━━━━━━━━━━━━━━━━━━ 14s 3ms/step - accuracy: 0.7980 - loss: 50.8748 - val_accuracy: 0.8855 - val_loss: 10.6825
Epoch 2/50
4086/4086 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - accuracy: 0.8045 - loss: 23.9302 - val_accuracy: 0.8340 - val_loss: 7.3250
Epoch 3/50
4086/4086 ━━━━━━━━━━━━━━━━━━━━ 10s 2ms/step - accuracy: 0.7998 - loss: 17.4185 - val_accuracy: 0.8854 - val_loss: 31.3737
Epoch 4/50
4086/4086 ━━━━━━━━━━━━━━━━━━━━ 8s 2ms/step - accuracy: 0.8035 - loss: 11.6698 - val_accuracy: 0.6049 - val_loss: 4.3434
Epoch 5/50
4086/4086 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - accuracy: 0.8007 - loss: 8.2261 - val_accuracy: 0.8854 - val_loss: 6.1486
Epoch 6/50
4086/4086 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - accuracy: 0.8050 - loss: 5.7862 - val_accuracy: 0.3283 - val_loss: 3.7733
Epoch 7/50
4086/4086 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - accuracy: 0.8043 - loss: 3.6193 - val_accuracy: 0.8854 - val_loss: 1.6016
Epoch 8/50
4086/4086 ━━━━━━━━━━━━━━━━━━━━ 10s 2ms/step - accuracy: 0.8060 - loss: 2.3307 - val_accuracy: 0.8858 - val_loss: 0.5740
Epoch 9/50
4086/4086 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step - accuracy: 0.8159 - loss: 1.5069 - val_accuracy: 0.8857 - val_loss: 0.5377
Epoch 10/50
4086/4086 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step - accuracy: 0.8270 - loss: 0.8329 - val_accuracy: 0.8854 - val_loss: 1.0164
Epoch 11/50
4086/4086 ━━━━━━━━━━━━━━━━━━━━ 20s 2ms/step - accuracy: 0.8517 - loss: 0.5307 - val_accuracy: 0.8854 - val_loss: 0.3676
Epoch 12/50
4086/4086 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - accuracy: 0.8634 - loss: 0.4775 - val_accuracy: 0.8854 - val_loss: 0.3895
Epoch 13/50
4086/4086 ━━━━━━━━━━━━━━━━━━━━ 11s 2ms/step - accuracy: 0.8681 - loss: 0.4446 - val_accuracy: 0.8854 - val_loss: 0.3634
...
Epoch 50/50
4086/4086 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - accuracy: 0.8828 - loss: 0.3613 - val_accuracy: 0.8854 - val_loss: 0.3560
1277/1277 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8857 - loss: 0.3555
[test loss, test accuracy]: [0.3570156395435333, 0.8849373459815979]
```

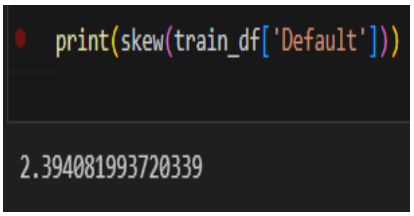
This gave an accuracy of 88.493.

iv. I applied the same model on the data from test.csv and got an accuracy of 88.447 on kaggle.

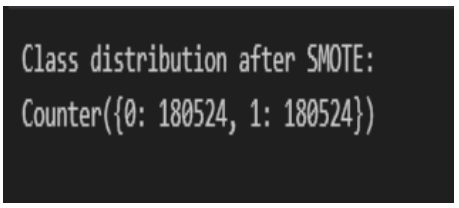
c. With skew reduction:

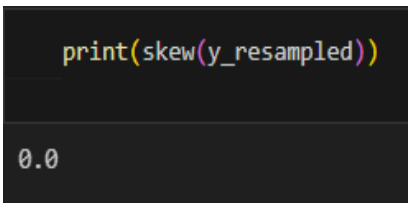
i. First i checked the class distribution of the training data:





ii. I then applied SMOTE to reduce the skewness of the data:





iii. I then did hyperparameter tuning using Bayesian Optimization, with the following tuner values:

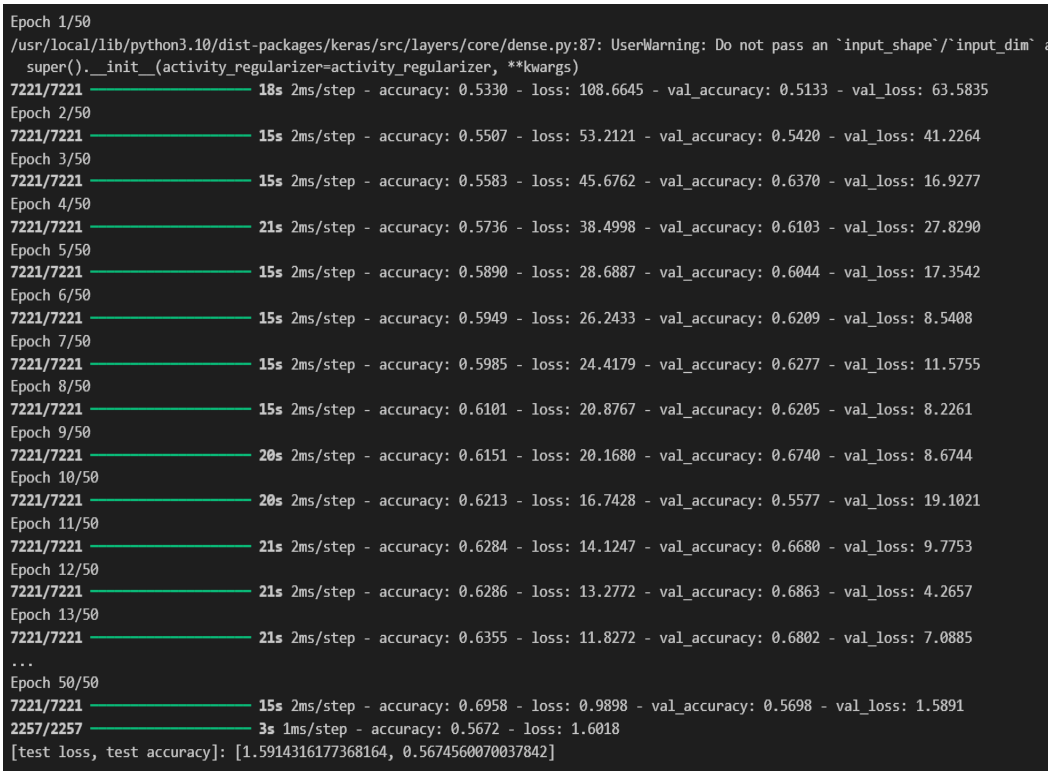
```
max_trials = 5, executions_per_trial = 1
```

And the tuner search ran for 8 epochs.

iv. This gave the following best parameters:

```
The optimal number of units in the first
densely-connected
layer is 448, in the second layer is 224 and the
optimal learning rate for the optimizer
is 0.0001.
```

v. The model was then fitted on the training data split with epochs = 50.



This gave an accuracy of 56.745.

vi. I applied the same model on the data from test.csv and got an accuracy of 87.523 on kaggle.

3. Logistic Regression
    a. Preprocessing was the same as mentioned in the preprocessing section. The model was trained two times, one with skew reduction(SMOTE) and one without skew reduction.
    b. Without using SMOTE:
        i. We got a validation accuracy of 88.48 after applying the LR model.
        ii. After replicating the preprocessing on data of test.csv and applying the model, we got a test accuracy of 88.586.
        iii. This was the highest accuracy we could achieve using these 3 models.
    c. When using SMOTE:
        i. We oversampled the data by adding extra rows using the SMOTE function.
        ii. After the remaining preprocessing steps and application of the LR model, validation accuracy dropped to 42.06.

iii. When this data was submitted on kaggle, the test accuracy dropped to 48.682.

Required Installations:

1. pip install imbalance-learn : to remove skewness from data
2. !pip install keras-tuner : for hyperparameter tuning of NN model (through Bayesian Optimization)