MACHINE LEARNING ENGINEER TAKE HOME

Name: Viswanathan Chandrashekar

1. The take home assignment asked me to solve the following challenge: https://www.kaggle.com/c/GiveMeSomeCredit

2. I solved the challenge and was able to reach position number 40 on the private leaderboard.

3. Github Link for code: https://github.com/nuschandra/GiveMeSomeCredit-Assignment

4. In addition to the answers given in this report, the python notebook in the Github repository also has detailed points after each step.

5. Firstly, I will give a brief explanation of the models I used to solve this problem before I answer the other questions asked in the take home assignment.

   I used 3 different models for solving this particular problem:
   1. Random Forest Classifier
   2. Gradient Boosting Classifier
   3. Voting Classifier which uses both Random Forest and Gradient Boosting

   - Ensemble techniques are often the go-to technique in prediction-based problems since they combine the predictive power of multiple models and by combining them, get rid of the variance caused by a single model. All the above methods are ensemble methods and produced very good results on the test dataset in the Kaggle competition.

   - In order to tune hyperparameters for each of these models such as n_estimators, max_features, max_samples, etc., I tried different values through trial and error and chose the set of values which produced the best performance on the validation set.

   - Ideally, I would like to run a RandomizedSearchCV/GridSearchCV for different values for the hyperparameters in each model. However, since these search methods often take quite a bit of time, in the interest of time, I relied on trial-and-error methods.

     Note: Either way, I have included the code for RandomizedSearchCV in the notebook. It can be used to tune the hyperparameters further and verify if the final performance can be improved.

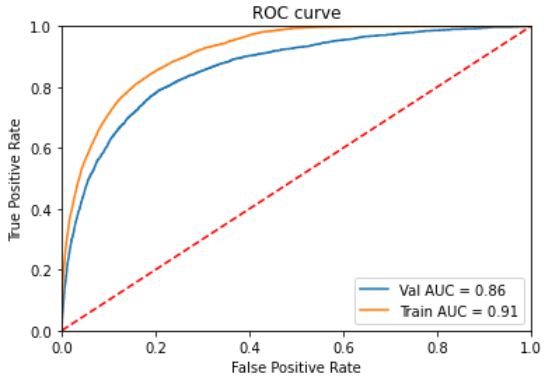
   - Based on performance on the validation set, I used the VotingClassifier as the final model.

4. Tell us how you validate your model, which, and why you chose such evaluation technique(s).

    - In this problem, I used the StratifiedKFold Cross-Validation technique in order to evaluate my models.
    - The reason I use this technique is because it helps give a better idea of how generalized the model is compared to a standard train-test split.
    - Sometimes, it is possible that in the train-test split, the validation set had a lot of easy examples due to which the model performed well on it and hence the performance isn't an accurate reflection of the model. In other words, the validation dataset doesn't fully represent the actual unseen data.
    - Alternately, in K-fold, the dataset is divided into a given number of splits out of which 1 split is used as the validation set in one iteration. In the next iteration, another split is used as the validation set.
    - In the process, each sample is used as training data 4 times in my code and as test data one time. Therefore, the model is able to see the entire training data and also test on different parts of the data. If the model is generalized, then the performance across each validation set will be stable without too much variance.
    - If the model is not generalized, it may perform well on one split and perform poorly on the other split, which will help us identify that an issue needs to be addressed.
    - StratifiedKFold is used to ensure the validation set and training sets are representative of the overall dataset by maintain same ratio of classes. It would be even more useful when there is a dataset imbalance but in this case since we use oversampling, there is no imbalance.

5. What is AUC? Why do you think AUC was used as the evaluation metric for such a problem? What are other metrics that you think would also be suitable for this competition?

    - AUC is referred to as the Area under Curve, specifically the Receiver Operating Characteristic (ROC) curve.
    - The ROC is a plot between the True Positive Rate and the False Positive Rate of a model at different thresholds and the AUC is the area under this curve.
    - If a model is very good at differentiating between 2 classes, then that means it is able to predict both true positives and true negatives well. In other words, the true positive (TP) is high and the false positive (FP) is low.
    - When TP is high and FP is low, the ROC plot will automatically be much higher and therefore, the AUC will also be higher. A sample plot from my code is shown here to depict an ROC curve.



Why does it suit this problem?

- My understanding is that in this problem, it is important for us to separate out the 2 classes – defaulters and non-defaulters – as well as possible and not just focus on one class.
- If we have more False Positives, we will essentially not be giving loans to people who are not defaulters.
- Alternately, if there are many False Negatives, we will be handing out loans to people who are defaulters. Neither of these cases are desirable.
- In such a situation, we want a model to be able to differentiate these 2 classes well and AUC is a good measurement of this because as mentioned earlier, if the curve is higher, we can automatically know that True Positives are higher and False Positives are lower.
- In this problem, the output of the model will be probabilities of the record being a defaulter or not. To plot the ROC curve, multiple thresholds are considered – if the output probability is > the threshold, it is a positive case and if not, it is considered a negative case.
- These can be used to calculate the TPR and FPR at multiple thresholds which is then plotted as an ROC curve.

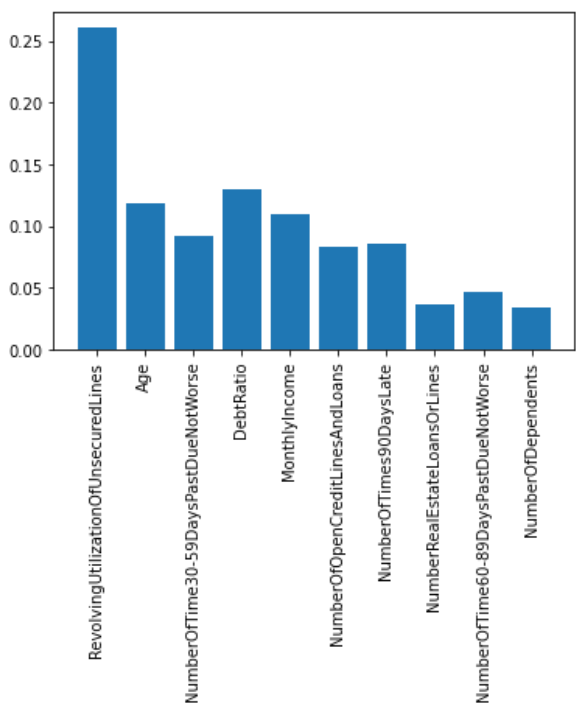
What are other metrics that you think would also be suitable for this competition?

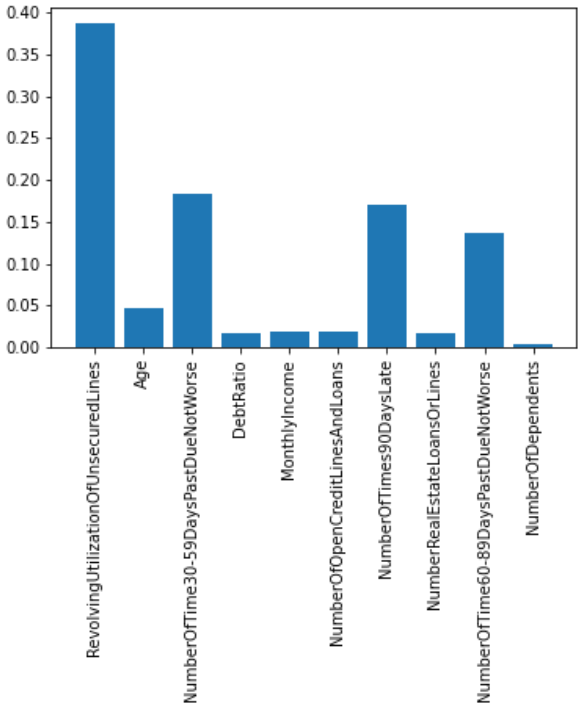
- My suggestion is that F1 score would also be a good metric in order to measure the performance of the model in this case.
- As mentioned earlier, it is not sufficient to focus on just one class (Defaulters or Non-defaulters) but instead differentiate both classes as well as possible.
- If we consider a metric like Precision, it wouldn't consider False Negatives and vice-versa with Recall which wouldn't consider False Positives.
- However, F1 score is a combination of both Precision and Recall and higher the F1 score, better the model.
- Accuracy wouldn't be a right fit for this scenario because it is an imbalanced problem i.e., no. of defaulters << no. of non-defaulters. In such cases, accuracy can be a misleading metric as it could be a high value even if none of the minority class records was predicted correctly.

- Therefore, for this problem where both positive and negative cases have to differentiated correctly, F1 score would also help identify the right model.

6. What insight(s) do you have from your model? What is your preliminary analysis of the given dataset?

    - After fitting the models on the training dataset (both RandomForest as well as GradientBoosting), I measured the feature importance in both these cases. The below bar plot shows the same:





- As can be seen here, the RandomForest and GradientBoosting both considers RevolvingUtilizationOfUnsecuredLines as the most critical feature.
- This definitely seems quite intuitive because this particular feature refers to debt-to-limit ratio. When a person has a very high debt-to-limit ratio, it is quite intuitive that they might be defaulters and may have to be flagged. If they do not, then it is likely that they are not defaulters.
- In the case of GradientBoosting, along with this feature, the next most important features are NumberOfTimes30-59DaysPastDue, NumberOfTimes90DaysLate, NumberOfTimes60-89DaysPastDue. Once again, when we think about it intuitively, these features seem to have a correlation with each other as well as having a correlation with the chance of being a defaulter because the person has not been paying dues on time.
- Alternately, in RandomForest, DebtRatio and MonthlyIncome as well as age are considered important features. Once again, DebtRatio is closely related to MonthlyIncome and intuitively, when a person has a high debt ratio, it is indicative that they are not in a position to make payments on time.

- In the RandomForest, as expected, since different features are considered in different trees, there are a few more features which also have a fairly high level of importance such as age as well as number of times past due features.

PRELIMINARY ANALYSIS OF THE DATASET

- First and foremost, this is a highly imbalanced dataset with very few defaulter records compared to non-defaulter which is expected in the real world as well.
- In order to fix this issue, I tried a couple of oversampling techniques – SMOTE and RandomOversampling. I have added more details about the oversampling techniques in the Python notebook that I have uploaded in the Git repository. Since RandomOversampling gave me the best performance, I went ahead with the same.

The next few points will talk about data cleanup for each feature in the dataset:

- In case of age, I imputed invalid records with age = 0 with the median of the age column.
- I found missing values in columns such as NumberOfDependents and MonthlyIncome which I imputed using the median of the column.
- Interestingly, I also observed that when the values were missing or 0 or 1 or NA in the Monthly Income column, the corresponding DebtRatio values were extremely high and it is possible that the Debts were divided by a dummy value of 1 in order to avoid divide by 0 errors.
- Hence, I assumed these were erroneous data and replaced with the median of MonthlyIncome and the corresponding DebtRatio was calculated using the new MonthlyIncome.
- In case of RevolvingUtilizationOfUnsecuredLines, there were quite a lot of values > 1. This may occur when the balance is greater than the limit and could sometimes indicate fraudulent activity too. However, I have still retained the maximum cap as 10 and imputed values > 10 with the median.
- Lastly, for the columns - NumberOfTimes30-59DaysPastDue, NumberOfTimes90DaysLate, NumberOfTimes60-89DaysPastDue, coded values such as 96, 98 were used which indicate Do not want to answer or Missing values.
- Initially, I removed these values and imputed them with the median and measured the model performance.
- Later, I tried to retain these values 96, 98 without imputing since they might carry some meaning in relation to the target variable. I tried training the model by retaining these values and saw that the validation score improved from before and when tested against the test dataset, the final auc score also improved significantly. Therefore, I decided to keep the coded values as is.
- For the test dataset, I imputed values wherever invalid records or missing records by using the training set median for the appropriate feature.

PLEASE CHECK NEXT PAGE

7. Can you get into the top 100 of the private leaderboard, or even higher?

Classifier	AUC	Rank
RandomForest	0.86560	~200
GradientBoosting	0.86656	~140
VotingClassifier (with both)	0.86766	~68

From the perspective of the contest, since the test data is available as a whole, instead of using training data median, I imputed the test dataset missing/invalid records with the testing data median. This gave a highest AUC score of 0.86836 with the Voting Classifier that ranked me at #40.

Some suggestions to try and improve further are:

- With better imputation techniques instead of just using median (such as using linear regression to estimate the values for records which are invalid), we could check if the model performs better.
- Hyperparameter-tuning to choose the best possible RF or GB classifier.

REFERENCES:

1. https://scikit-learn.org/stable/index.html
2. https://kiwidamien.github.io/how-to-do-cross-validation-when-upsampling-data.html
3. https://stackabuse.com/cross-validation-and-grid-search-for-model-selection-in-python/
4. https://blog.dataiku.com/narrowing-the-search-which-hyperparameters-really-matter#:~:text=We%20again%20found%20the%20most,be%20seen%20in%20figure%203.
5. https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques/
6. https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/