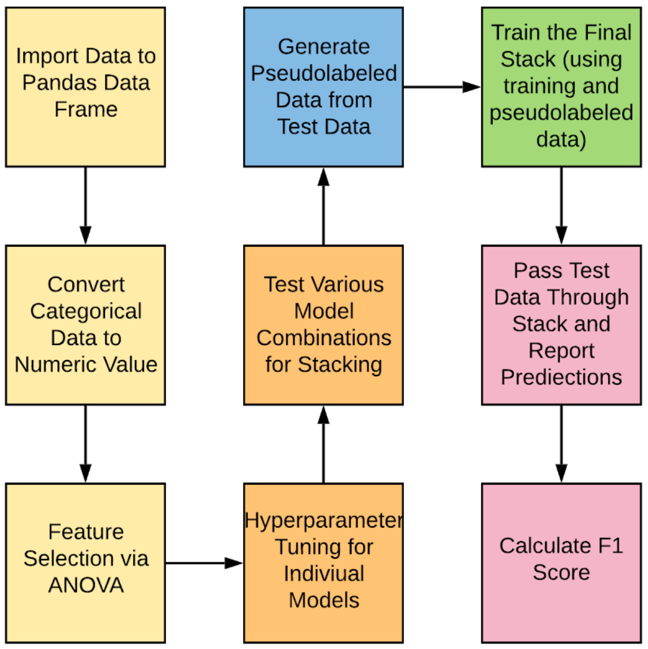
Wells Fargo Campus Analytics Challenge 2020
Deliverable 2: Methodology

Contributors:
Haoxu Huang (huan1780@umn.edu), Nathan Briese (bries042@umn.edu)

Data Pipeline


Import Data to Pandas Data Frame
Convert Categorical Data to Numeric Value
Feature Selection via ANOVA
Hyperparameter Tuning for Indiviual Models
Test Various Model Combinations for Stacking
Generate Pseudolabeled Data from Test Data
Train the Final Stack (using training and pseudolabeled data)
Pass Test Data Through Stack and Report Prediections
Calculate F1 Score

Preprocessing

The first section of the pipeline is the preprocessing. The training and testing data are imported into the program using Pandas. The pandas dataframe is a standard way to import data from .xlsx or .csv files (Pandas, 2020). Additionally, the data frame object in the Pandas library keeps more information from the source, such as feature titles, than Numpy ndarrays (Oliphant, 2006). The next step is to convert the categorical predictor XC to a numerical value with Label Encoding. Specifically, assign A to 1, B to 2, C to 3, D to 4, and E to 5. We also tried Dummies Encoding, excluding the categorical XC and applying z-score normalization after Label Encoding; however, we didn't find performance improvement on all models we tested **(See "Plot Model Performance Changes for Different Methods of Transforming Dataset" Section from "Feature Engineering + Hyper-parameters Tuning" Notebook for details)**. After this, we need to select a subset of features to use in our predictive models. We assume here that the data points are drawn from a distribution. In other words, the probability of a certain classification is not independent of the features of that given data point. This is a reasonable assumption, because otherwise the task of classifying them would not be possible. From this assumption, we can assume that not all of the features are good indicators of the classification. Features with less variance are worse predictors than those with high variance. In order to select the best feature subset to use, analysis of variance (ANOVA) using the f_classif score function from sklearn was used (Pedregosa *et al.*, 2011). Features with higher variance across the dataset are better indicators of the label (Alpaydin, 2014). Additionally, we selected features using recursive feature elimination with cross validation and via Random Forest and XGBoost feature importances, but got the same results as ANOVA. We decided to use the best 16 features. This choice was based on the difference in variance between each feature. When sorting the selected features by their variance and Random Forest/XGBoost feature importances, we see a steady decline until the 17th feature where there is a significant drop in feature importances (approximately 40% decrease) and an even larger drop in variance **(See "Feature Selection" Section from "Feature Engineering + Hyper-parameters Tuning" Notebook for details)**. We also tested to make sure that there was little to no correlation between our selected features. Using Spearman correlation, we made a correlation matrix and showed it through matplotlib (Hunter et al., 2007) **(See "Correlation Matrix" Section from "Feature Engineering + Hyper-parameters Tuning" Notebook for details)**. None of the correlations have values greater than 0.1, meaning that there is no correlation between any of our selected features.

Model Selection and Stacking

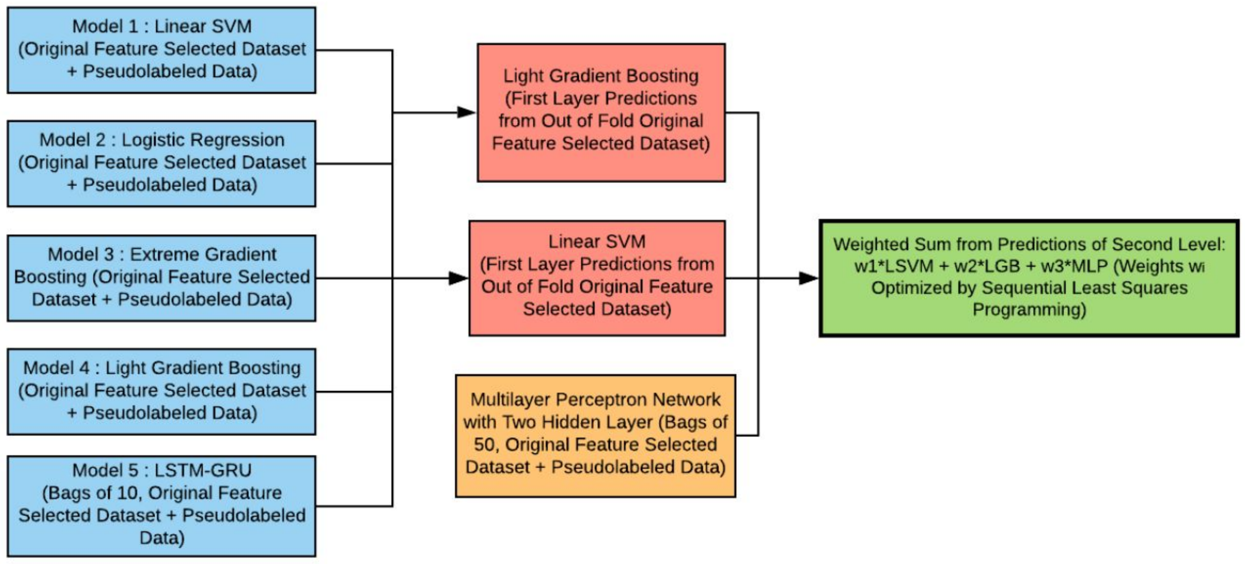
The next section of the pipeline is model selection. In our preliminary testing, we were not satisfied with the performance of the individual models we had chosen. Given the observation that the best model so far was linear SVM and all linear classifiers (Logistic Regression, Linear Discriminant Analysis, Perceptron, etc.) perform fairly well, we assume that the data on these features is close to linearly separable. The second best was a bidirectional long short term memory neural network, which suggests the data might have a sequential relationship (Hochreiter et al., 1997). To get marginally better performance, we decided to find a way to combine the predictions of multiple models. Individual models have their own strengths and weaknesses; so, combining their predictions will ideally amplify their strengths and produce better results. Thus, we will stack multiple models together. The idea of stacking is not necessarily novel, but our particular stacking structure is novel and tailored to this dataset. We selected a diverse set of prediction algorithms to potentially use in our stacked model. For hyperparameter tuning, we firstly used grid search and random search to narrow down the search space, then we used bayesian optimization to tune the hyperparameters for each model in the narrowed search space. While hyperparameter tuning, we experimented with different types of class rebalancing in an effort to increase performance. Rebalancing methods tested include reweighting, SMOTE rebalancing, ADASYN rebalancing, and borderline SMOTE rebalancing. We found that models get marginal better performance with rebalancing. See the source code for details of the final hyper-parameter configurations of each model **(See “Hyperparameters Tuning” Section from “Feature Engineering + Hyper-parameters Tuning” Notebook for details)**. Our final stack structure was derived from empirical experimentation over the various models. Ultimately, we’ve decided to use the models with the highest F1 score on CV after stacking. The final used models include Light Gradient Boosting (Abbr. LGB) (Ke et al., 2017), Linear SVM, Logistic Regression (Abbr. LR), Extreme Gradient Boosting (Abbr. XGB) (Chen, 2016), a neural network that concatenates LSTM(Long Short Term Memory) and GRU (Gated Recurrent Unit) (Abbr. LSTMGRU) (Cho et al. 2014) **(See “Get Final Predictions" Notebook and our final architecture graph for details)**. Adding other models outside this set (including MLP, kNN, GNB, RBF kernel SVM, etc.) caused a decrease in performance. Similarly, the models selected for the second layer had the best F1 score performance on training and pseudolabeled data.

PseudoLabeling

Before training the stack, we added a way to give it more training data to work with for the purpose of refining the decision boundary of models. We will supplement the training data with pseudolabeled test data. Pseaudolabeled data is test data that we have assigned labels using a learning algorithm that we have high confidence in based on mean cross validation F1 score. The training data was rebalanced using the borderline SMOTE algorithm, then we used a linear support vector machine to assign pseudolabels to test data points which had prediction probability less than 0.3 (classified as 0) and bigger than 0.7 (classified as 1). The reason why we used this prediction probability threshold is that we observed that the original dataset still preserved 1.0 mean cross validation F1 score with this threshold while threshold of less than 0.4 and bigger than 0.6 cannot preserve 1.0 mean cross validation F1 score **(See “Check for Performance of Selecting Threshold” Section from “Feature Engineering + Hyper-parameters Tuning” Notebook for details)**. Pseudolabeled points are also used to reinforce/supplement the training of the final model.

Training the Stack

Model Stacking


Model 1 : Linear SVM (Original Feature Selected Dataset + Pseudolabeled Data)
Model 2 : Logistic Regression (Original Feature Selected Dataset + Pseudolabeled Data)
Model 3 : Extreme Gradient Boosting (Original Feature Selected Dataset + Pseudolabeled Data)
Model 4 : Light Gradient Boosting (Original Feature Selected Dataset + Pseudolabeled Data)
Model 5 : LSTM-GRU (Bags of 10, Original Feature Selected Dataset + Pseudolabeled Data)
Light Gradient Boosting (First Layer Predictions from Out of Fold Original Feature Selected Dataset)
Linear SVM (First Layer Predictions from Out of Fold Original Feature Selected Dataset)
Multilayer Perceptron Network with Two Hidden Layer (Bags of 50, Original Feature Selected Dataset + Pseudolabeled Data)
Weighted Sum from Predictions of Second Level: w1*LSVM + w2*LGB + w3*MLP (Weights wi Optimized by Sequential Least Squares Programming)

With all the data in hand, we are ready to train the stack. The stack model is trained by first training each model in the first layer. The first layer is trained on the training data in addition to the pseudolabeled data. We examined three different methods of adding pseudolabels: balance original imbalanced dataset with pseudolabels, add all applicable pseudolabels, add all applicable pseudolables with SMOTE rebalancing. After applying these three methods, we found that directly adding all applicable pseudolabels yielded best performance **(See “Pseudo Labeling Check” Notebook for details)**. The second layer consists of two models: linear SVM and multilayer perceptron (MLP) network with two hidden layers. The MLP is trained on the training and pseudolabeled data. The MLP uses the PReLU activation function instead of the usual ReLU (He et al., 2015). The linear SVM model is trained on the training data using the predictions from the first layer as added features in addition to the selected features. In order to reduce variance of our neural network models, the LSTMGRU model uses bagging with bags of 10 and the MLP uses bagging with bags of 50. 95% bootstrapped samples in each bag from the original training dataset are used for both models.

At the end of the data pipeline, the test data is passed through the stack. We attempted three different methodologies to get full prediction matrices of each model:

1. The test set is broken into 10 cross-validation folds. For each fold for each model, train the first layer and apply folded test data. Each model in the first layer ultimately gives 10 prediction matrices from their corresponding out of the fold data point in the test set. Then, the 10 prediction matrices are stacked vertically to get full prediction for each model.

2. Get accumulated predictions from the test set by predicting the whole test set with all trained models from 10-fold cross validation for each model. Then, the predictions are averaged and sent to the second layer.

3. Each model is trained on all training and pseudolabeled data. Then each model predicts on the whole test set. There is no folding.

The third method is applied to our submitted pipeline because it shows consistently higher performance.

The final prediction is given by a weighted sum for predictions from linear SVM, LGB and MLP in the second level. The weights are optimized by Sequential Least SQuares Programming (Kraft, 1988), where the objective function is the mean F1 score from 10-fold cross validation with l2-regularization on the weight of MLP. The regularization is added because we apply early stopping when training MLP with cross validation, which will cause slight information leakage. **(See "Optimize Blending Weight" Notebook for details)**

Interpretation of score

The F1 score is the harmonic mean of precision and recall (Shalev-Shwartz et al., 2014). F1 score balances the cost of false positives and false negatives. If the dataset is sourced from a situation where false negatives (or visa versa false positives) are more important, then it would not be a great evaluation metric. Examples of this kind of situation include medical diagnosis and fraud detection where false negatives are very costly. Since our model scored very close to 1, it has both high precision and high recall. This means that nearly all of the selected items are relevant and nearly all the relevant items are selected.

Concluding Thoughts

This stacking model has its strengths and weaknesses. It has high prediction accuracy. The combination of models offers higher accuracy than any of the models individually. It tends to have less overfitting. The diversity in the model selection in the stack reduces overfitting. It is not very interpretable or insightful. This is essentially a black box algorithm. Each individual model votes for a classification and then another learning algorithm learns which votes are most important. The decision of an amalgamation of various learning algorithms does not produce actionable insight. Lastly, it is relatively inefficient. Since we have to train so many models individually, training and evaluation times are much longer for the stack than for any single model.

This model has some real world benefits. The stacking idea can be adapted to many different datasets/distributions. Since this model combines the strengths and weaknesses of the various models, a stack can be fitted to both complex and simple data sets, making them very versatile. Stacking can even be adapted to multiclass classification. The linear algorithms would have to be adapted significantly, but it is possible nonetheless.

References

Alpaydin, E. (2014). *Introduction to Machine Learning* (pp. 110-112). Cambridge, MA: MIT Press.

Chen, Tianqi; Guestrin, Carlos (2016). *XGBoost: A Scalable Tree Boosting System*. In Krishnapuram, Balaji; Shah, Mohak; Smola, Alexander J.; Aggarwal, Charu C.; Shen, Dou; Rastogi, Rajeev (eds.). *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*. ACM. pp. 785–794. arXiv:1603.02754. Doi: 10.1145/2939672.2939785.

Cho, Kyunghyun; van Merrienboer, Bart; Gulcehre, Caglar; Bahdanau, Dzmitry; Bougares, Fethi; Schwenk, Holger; Bengio, Yoshua. (2014). *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. arXiv:1406.1078

Dey, Rahul; Salem, Fathi M. (2017). *Gate-Variants of Gated Recurrent Unit (GRU) Neural Networks*. arXiv:1701.05923

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, Tie-Yan Liu. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". Advances in Neural Information Processing Systems 30 (NIPS 2017), pp. 3149-3157.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.* 2015 IEEE International Conference on Computer Vision (ICCV), 1026–1034. https://doi.org/10.1109/iccv.2015.123

Hochreiter, S., & Schmidhuber, J. (1997). *Long short-term memory.* Neural computation, 9(8), 1735-1780.

Hunter J. D. (2007). *Matplotlib: A 2D Graphics Environment.* Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95.

Kraft, D. *A software package for sequential quadratic programming*. (1988). Tech. Rep. DFVLR-FB 88-28, DLR German Aerospace Center - Institute for Flight Mechanics, Koln, Germany.

McKinney, W. (2010). *Data Structures for Statistical Computing in Python.* Proceedings of the 9th Python in Science Conference. Pages 56 - 61. 10.25080/Majora-92bf1922-00a

Oliphant, T. E. (2006). *A guide to NumPy.* Trelgol Publishing.

The pandas development team. (2020). *pandas-dev/pandas: Pandas.* Zenodo. 10.5281/zenodo.3509134 https://doi.org/10.5281/zenodo.3509134

Pedregosa F., Varoquaux G., Gramfort a., … Duchesnay E. (2011). *Scikit-learn: Machine Learning in Python.* JMLR 12, pp. 2825-2830, 2011.

Shalev-Shwartz, S., Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms.* pp. 244-245. Cambridge University Press.

Stéfan van der Walt, S. Chris Colbert, and Gaël Varoquaux. (2011). *The NumPy Array: A Structure for Efficient Numerical Computation.* Computing in Science & Engineering 13, 22-30, DOI:10.1109/MCSE.2011.37