Loan Default Predictive Modeling

A Case Study of Classification Algorithms

http://www.github.com/pandrewg/loan-risk-analysis

Pablo Guevara
January 2018

Table of Contents

Executive Summary

Motivation

Loan portfolio managers are faced with the task of overseeing collections of financial instruments and making the appropriate decisions to align their portfolios' characteristics with their investment goals. We study the development of machine learning systems to aid managers in their decision-making by efficiently processing the data representing each loan. An ideal system would process the data and automate decision-making, but the feasible performance levels seen in this study do not support that paradigm. Thus, the systems we develop would be best seen as supplemental to the overall analyses conducted by portfolio managers.

Approach

We explore the prediction of loan defaults within a binary classification framework. The data set used has two important characteristics which cause initial sub-optimal performance: class imbalance and inter-class feature similarity. The models perform particularly poorly with respect to the minority class (the defaulted loans, which are of primary interest), and so much of our study is devoted to exploring methods to improve the systems' performance on these loans.

The machine learning systems can be seen as being composed of two parts: the *data space* (the input data—including the training data, the features chosen, and feature transformations) and the *hypothesis space* (the learning algorithms which map the inputs to outputs). We study variations in the model architecture in both these spaces. Within the *data space*, we study resampling techniques and unsupervised transformations of categorical features. Within the *hypothesis space*, we study variations of classification algorithms, hyperparameter optimization, and ensemble learning architectures.

Along the report, we delve deeper into analyses of the model architecture to understand why the systems are performing as they are and suggest solutions for improvement. However, this is primarily only to illustrate the approach for analysis of the underlying algorithms. This study is far from exhaustive—a complete exploration of classification systems is not within the scope of a single report.

We also develop a pipeline approach to apply variations of the model architecture in combinations and effectively evaluate performances across whole collections of model architectures. Again, this is still not exhaustive and primarily serves to illustrate an approach that could be more effectively used with sufficient computational resources.

Conclusions

We find there is a general tradeoff between precision and recall (i.e. specificity and sensitivity) of the models that must be considered when configuring the systems. Additionally, recall of a model can often be raised arbitrarily high, but precision on test sets has not been seen to be higher than about 0.42. It is this lack of high precision which ultimately leads us to suggest these systems be used as supplemental in analysis rather than as replacements. Still, the models are very effective in processing large data sets to a form that can be easily interpreted by a human manager. We lastly suggest the models output probabilities rather than predictions, so the systems can be utilized more effectively by portfolio managers in quantifying risk.

§1: Introduction

1.1 Background

Risk, in its various forms, underlies the ownership of any asset, and successful risk management is necessary for successful asset management. We consider in this study the problem loan portfolio managers face of default risk behind unsecured loans. Most of this report will be focused on the task of predicting payoffs and defaults with classification algorithms, however the report will conclude with a proposition for using these algorithms to aid in quantifying default risk.

We will see that reliably predicting defaults is an inherently very difficult task, and so the models we develop should be viewed as a system to aid in decision making rather than as the decision maker itself. For loan portfolio managers, the algorithms would serve as engines for flagging unfavorable positions. This, together with other financial analyses, should determine the decision to acquire, retain, or relinquish the loans.

The data analysis, modeling, and model evaluation will be entirely conducted in Python, which is used together with SQL and Excel for data management. We use the NumPy, Pandas, SQLite, imblearn, matplotlib, seaborn, and scikit-learn Python libraries to enable our analysis. We choose a scripting language with the goal of developing applications which can be effectively integrated within a more general business architecture.

1.2 Data

Overview

We use a data set released by Lending Club[1] representing about 880,000 consumer loans over a five-year period. This includes many loans which have not completed their terms (i.e. the statuses of these loans are: current, late 30 days, late 60 days, etc.) which will not be considered in our study of binary classification.[*] After cleaning the data and dropping data points with excessive missing values which cannot be reasonably estimated, we are left with about 250,000 loans to conduct this project on.

The original database contains information regarding over 70 features, but the project uses a curated subset containing 20 features[†]. This reduction in feature space is the result of cleaning (we remove columns which have mostly missing values) as well as real-world application simulation (we remove columns which contain information that would not be available at time of loan origination). The presence of certain features creates a trivial modeling environment, where the models perform excessively well—for example, 'post-default collection amounts' serve as triggers for defaulted loans. We remove these features.

[*] The multi-class problem requires certain changes in the analytical framework—e.g. the 'one-vs-all' paradigm—but can loosely be thought of as a generalization of the binary problem.

[†] We study various subsets and transformations of the feature space. Data dictionary is available in Appendix A.

Data Management

We treat the original database released by Lending Club as a read-only file and write all the processed/modified data to a separate SQL database. The aim with this is to ensure that the original data is not erroneously modified and can always be re-parsed.

We write the processed data into separate tables within the new SQL database after performing various transformations and train-test splits to create another level of organization within the data set. This is to ensure that the modified data sets are available throughout the project without the need to re-run the transformations.

Finally, we write the results of computationally expensive operations to CSV files to also eliminate the need to re-run code. Writing the results to CSV files rather than to the SQL database creates yet another level of organization between the data used for analysis and the results of the analysis. *Figure 1.1* summarizes the data management we implement.

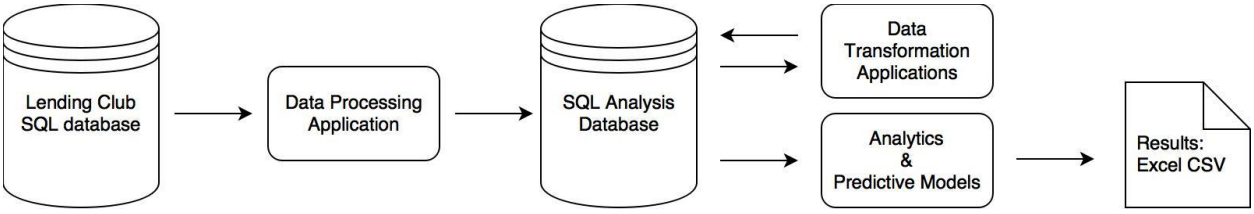

Figure 1.1 *Overview of data flow through this project.*

Data Processing & Cleaning

The data processing steps are mostly comprised of clerical issues involving extraction of numerical data from strings and conversion to appropriate data types. We study the formatting of the original data and choose appropriate methods to efficiently transform the data using vectorized operations. The goal with this is to develop a process which can be scaled to bigger data sets without creating unmanageable computational overhead.

The loan grade feature, which is originally presented alphabetically (A, B, C, etc.), requires a less trivial transformation. We use a simple mapping (A:7, B:6, C:5, etc.) to endow this categorical feature with numerical values, since there is a clear ordering to the loan grades. It is not entirely clear if this is the ideal mapping to use, but we proceed with this approach in our analysis. It remains to be studied if another mapping (e.g. squares -- A:49. B:36, C:25, etc.), or no mapping at all, will help the models perform better.[‡]

Lastly, we estimate the missing values of two features: revol_util and tot_coll_amt. The missing data entries comprise about 0.07% and 25.1% of the data set, respectively, and so we assume we can arrive at reasonable estimates. The approach we take is akin to a simple recommendation engine: we use categorical information to separate the training data into disjoint subsets and use the corresponding averages of each to fill missing values. For some missing values, there exist no other data within the same categorical subset. We take the

[‡] There are many design choices that will not be explored but will be left as suggestions for future work—an exhaustive exploration is not feasible.

average of less specific subsets (for which there exist data) to estimate these. Lastly, we use the 'recommendation engine' developed on the training set to estimate missing data on the test set.

Data Set Characteristics: Class Imbalance and Feature Space Overlap

An inherent characteristic of this data set is its class imbalance. There is an approximately 1:5 ratio of negative (defaulted) to positive (fully paid) data points, and this causes the baseline models to perform sub-optimally.[§] The models are initially overly insensitive to predicting defaults, and they misclassify almost all the defaulted loans in the test set (i.e. the baseline models' recalls on defaults is very poor). We address this issue in further detail with resampling techniques in section 3.

Additionally, this data set has a high degree of feature overlap between classes. Many defaulting loans have feature values that are very similar to feature values of loans which pay off. This high degree of overlap in the feature space may forecast difficulty for the models to predict defaults with high precision.

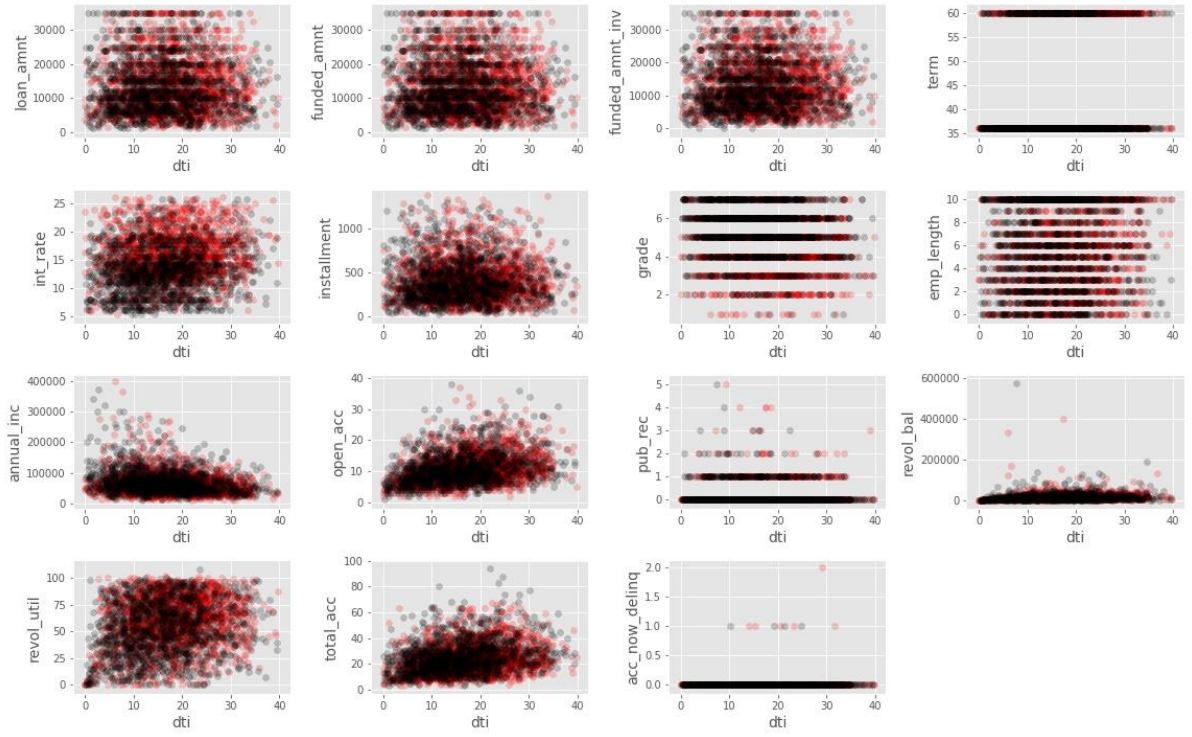

Figure 1.2 *Numerical features plotted against debt-to-income ratio between samples of each class. Defaulted loans are plotted in red. Fully paid loans are plotted in black.*

By plotting different combinations of numerical features onto two dimensional spaces, we see a visual example of this lack of distinction between classes. In *figure 1.2*, random samples of defaulted loans (red) and fully paid loans (black) can be seen to have almost indistinguishable distributions across various features. Interest rate and grade are exceptions, which show more defaulted loans on the higher and lower regions, respectively.

[§] Unfortunately, we have conducted this project with the labeling convention reversed. The defaults are of primary interest, but have been set to be associated with the negative class.

Oddly, debt-to-income ratio seems to bear little relationship to default counts—we observe negligible variation in density of red/black points with respect to variation on the DTI axes.

A similar analysis of kernel density estimation plots on samples of each class reveals that the data is similarly distributed on almost all features. The exceptions again seem to be interest rate, grade, and DTI—for which the distribution estimations deviate from each other. As remarked earlier, however, the deviations in the DTI plots are less pronounced. (In *figure 1.3* below, we remove features which provide little visual interpretability.)

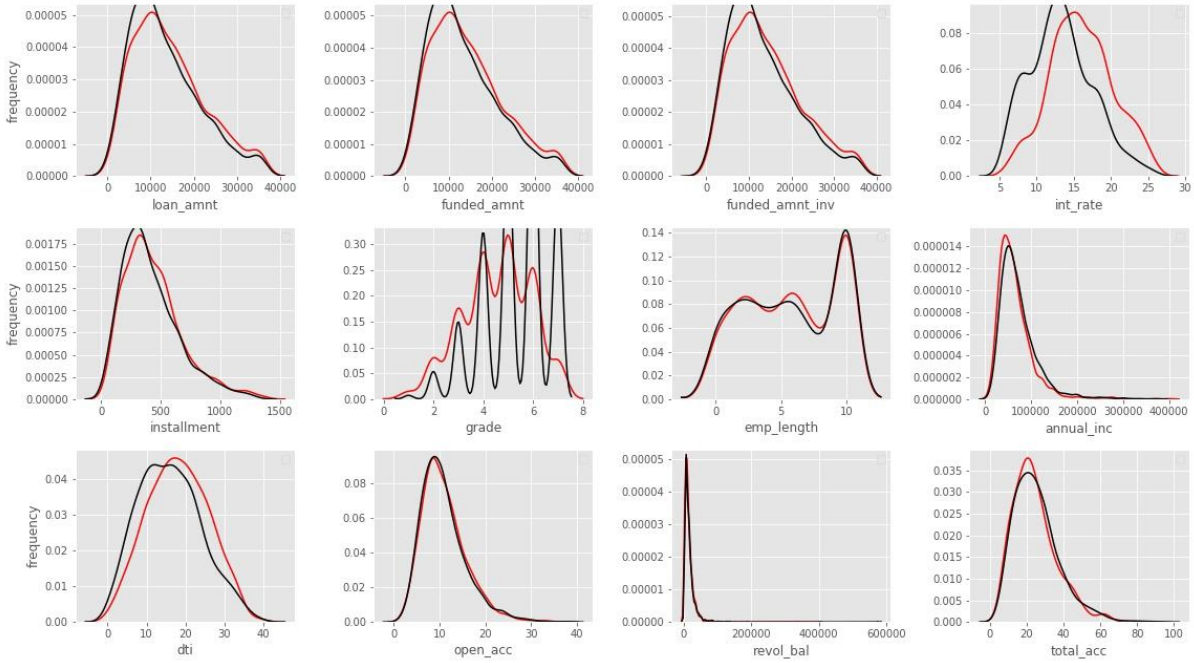

Figure 1.3 *Kernel-density estimation plots on samples of defaulted and fully paid loans.*

Applying PCA to observe a two-dimensional mapping of the entire numerical feature space results in little separation as well. In *figure 1.4*, we see a significant amount of overlap between classes after transforming and plotting the data. The classes may be more distinctly separated in the full dimension of the feature space, but model performance (yet to be seen) suggests this is unlikely.

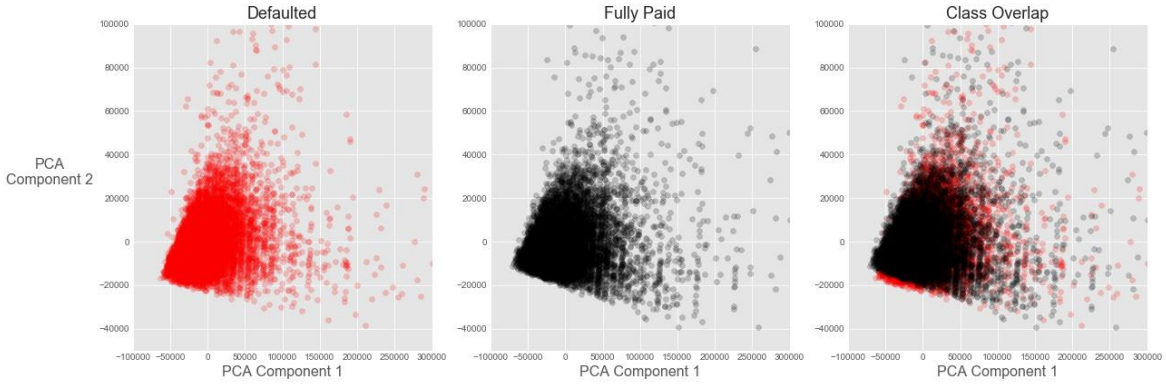

Figure 1.4 *Plots of each class after PCA reduction of numerical features to 2 components.*

1.3 Model Overview

As we progress through this project, we explore techniques to modify different aspects of the model architecture (i.e. we explore models within the 'model space'). These techniques are, in order: variation of classification algorithms, resampling, hyperparameter optimization, inclusion of categorical information, and application of ensemble architectures. Variation of classification algorithms, hyperparameter optimization, and ensemble learning operate in the 'hypothesis space,' whereas resampling and inclusion of categorical information operate in the 'data space.' See *figure 1.5.*

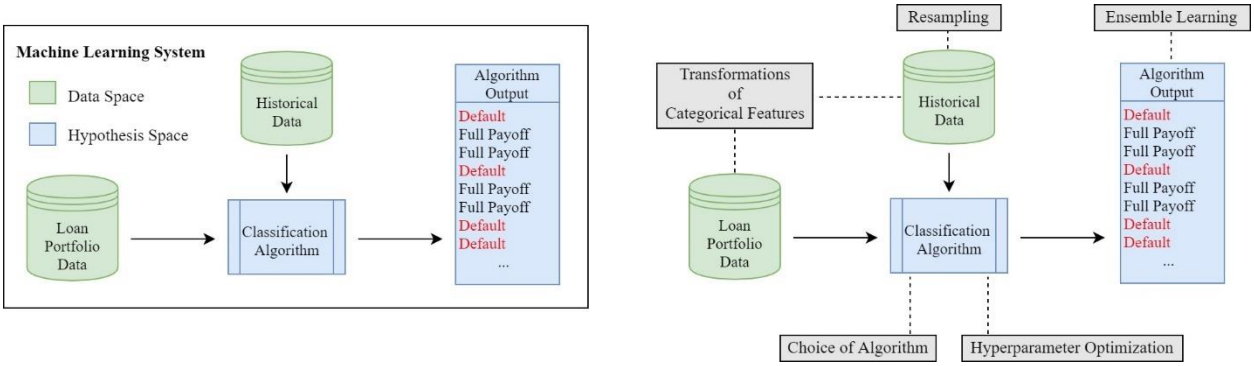

Figure 1.5 *Left: Overview of the general model architecture. Right: techniques explored in this report to modify the model architecture (grey) and their respective dimensions of operation (connected to dotted lines).*

We begin with a baseline evaluation of various classification algorithms and aim to improve performance. The techniques we study can be applied in combination, so we develop a pipeline method in section 5 to evaluate model performance across various model architectures.[**] Lastly, inclusion of categorical features does not guarantee better performance across all architectures (as we will see), so we study that after developing the pipeline—this allows us to study the effect over different model architectures.

[**] Refer to appendix B for a summary and brief discussion of the combinations of techniques explored.

§2: Baseline Evaluation

2.1 Classification Algorithms

We study the performance of four classification algorithms: Logistic Regression, Gaussian Naïve Bayes, k-NN, and Random Forest. These four offer different approaches to finding structure in the data, and so analysis of their combined hypothesis space should give a reasonable estimation to the error bounds of this problem. For this section, the hyperparameters of the models are left as the defaults set in the scikit-learn APIs.

These performance evaluations are all conducted over the same train-test split (80%-20%) to ensure consistency in the evaluations. Running each model several times over various splits and averaging the results may develop a more robust evaluation, but we leave that for future work in the interest of time and for lack of computational resource. Additionally, categorical features will need to be handled specially later in the project, and so baseline evaluations will be conducted over only the numerical features.

For this report, we set aside overall accuracy to focus on precision and recall. The class imbalance results can be misleading if evaluations are based only on accuracy, since the data set is dominated by non-defaulting loans (for which the models perform very well.) As an example, Logistic Regression results in about 82% accuracy on the test set, but we can see below that the model is not a reliable predictor of defaulted loans. Still, it is important to tie accuracy back into final evaluations to gain a better overall picture of model performance.

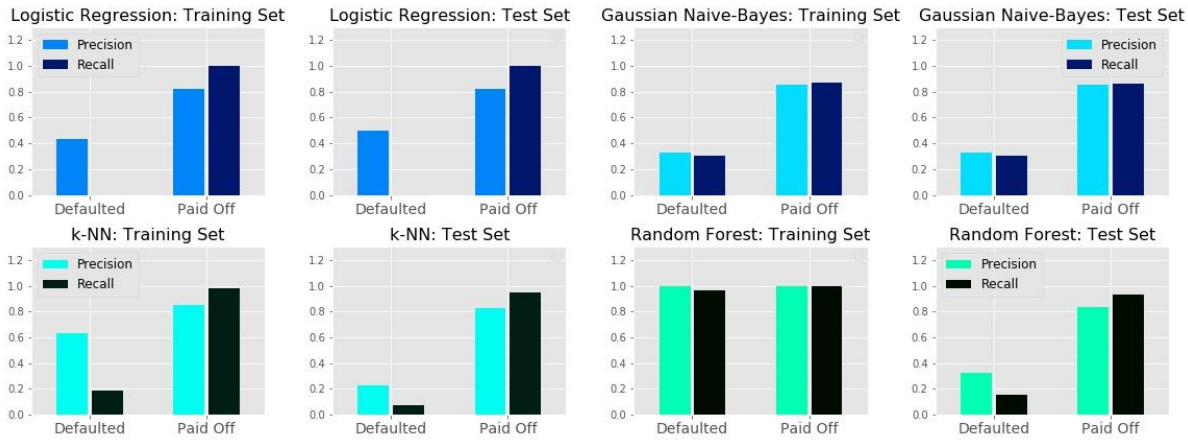

Figure 2.1 *Baseline precision and recall scores on the defaulted and paid off loans.*

Across all algorithms, we see a clear inequality in model performance between the two classes. (Random Forest's performance on the training set is an exception, but the drop in performance on defaults in the test set suggests there is a considerable degree of overfitting.) The models seem especially weak with respect to recall on defaults—hence the aforementioned "insensitivity to predicting defaults." This is especially true for Logistic Regression, which has close to 0% recall on defaults.

Considering that the cost of misclassifying a default (a false positive) is greater than the cost of misclassifying a payoff (a false negative), we see that we want the models to be

more sensitive to predicting defaults—even if this means increasing the likelihood of false negatives.[††] So, our immediate next goal is to explore techniques for improving recall on defaults. Ideally, however, we want to aim for better performance without sacrificing performance in other respects. Thus, we study all metrics together while primarily aiming to improve recall on defaults.

[††] A false positive would encourage a loan portfolio manager to keep a defaulting loan, whereas a false negative would encourage the manager to relinquish a paying loan. The loss on the former case is likely much higher.

§3: Resampling

Our approach to improving model recall on defaults involves applying various resampling techniques. We study undersampling of the majority class, SMOTE,[2] and ADASYN.[3] (We do not study simple oversampling of the minority class, as it has a known tendency to cause overfitting.) All resampling is applied to the training data after performing a train-test split, because we want the test set to approximate the data distribution in practice. Performing resampling before the split would create artificial structure in the test set—which cannot be done on unseen data in practice.

With these techniques, the target ratios of minority to majority class samples are parameters to be adjusted (away from the data set's natural 1:5 ratio). Thus, to study the range of effects a technique has on an algorithm, we vary this ratio and study the resulting precision, recall, and F1 scores on a specified test set. For details of performance on the training set, refer to the Jupyter Notebook: 'analysis-pt2.'

3.1 Undersampling

We begin with undersampling of the majority class, which is intuitively the simplest approach. The technique randomly samples a subset of the majority class to satisfy the specified class ratio. Our method leaves the minority class 'as-is' to achieve varying minority to majority ratios.

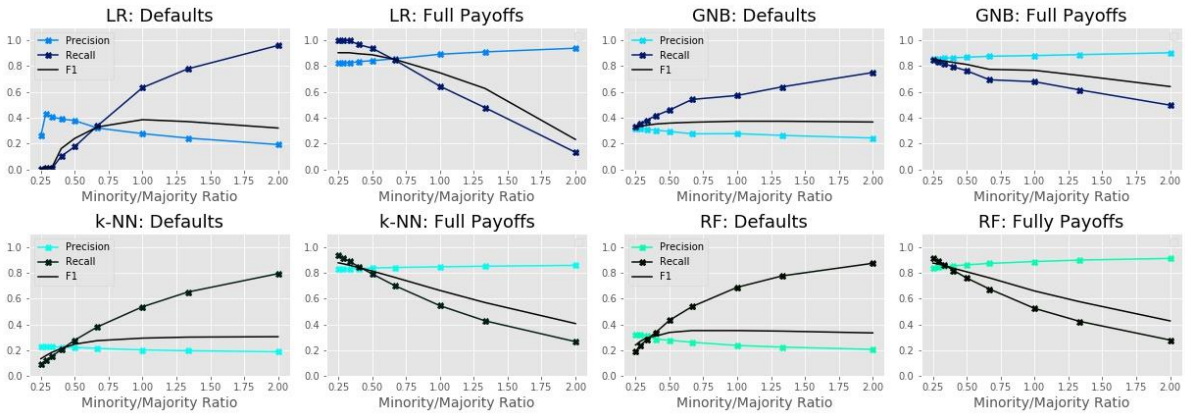

Figure 3.1 *Model Performance on test set after undersampling with varying minority to majority ratios. (Leftmost points of each plot have the values from the test sets in figure 2.1.)*

In *figure 3.1*, we can see there is a tradeoff in performance between classes in all algorithms. As we increase the resampling ratios, increases in recall of defaults are matched with decreases in recall of payoffs. The extent to which undersampling affects model performance also varies from model to model—for example, the Logistic Regression plots show greater change in recall compared to the Gaussian Naïve Bayes plots.

Precision is affected to a much lesser extent, with a general tendency of models to have slightly reduced precision on defaults and slightly increased precision on payoffs as we increase the resampling ratio. It seems precision is a result of the underlying feature values rather than the quantities of each class in the training set. With this assumption, we conclude resampling enables us to artificially change recall of models on a given data set

("alter the sensitivity" of models to outputting a class), but it does not enable us to effectively change precision.

3.2 SMOTE

SMOTE (Synthetic Minority Oversampling Technique) involves generating new data points rather than sampling a subset from the existing data. Minority class samples are randomly chosen, and new minority samples are randomly generated in the feature space along the vectors which connect the samples to their closest minority class samples. We now keep the majority class as-is and vary the quantities of new data to generate to once again achieve varying minority to majority class ratios.

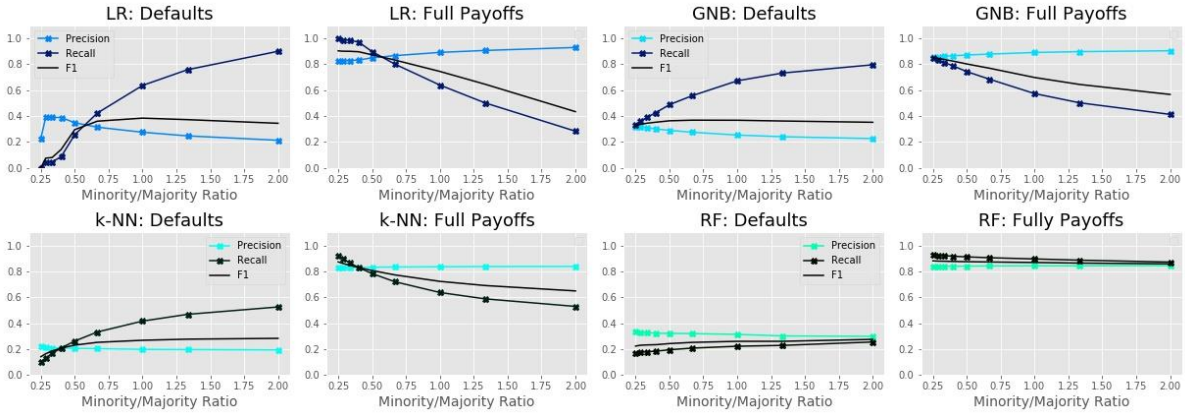

Figure 3.2 *Model performance after applying SMOTE with varying minority to majority resampling ratios.*

With SMOTE, we see a very similar collection of patterns (with the exception of Random Forest), but the extent to which the technique affects model performance varies from model to model. For example, Gaussian Naïve Bayes seems to be affected to a greater extent by SMOTE than by undersampling, whereas k-NN seems to be affected to a lesser extent.

We hypothesize that Random Forest is minimally affected because of the approach SMOTE takes in generating synthetic data: it is generated around the nearest minority neighbors of minority samples. Since Random Forest builds decision trees based on Gini impurity in subsets of the feature space, there is a high probability that the algorithm will build similar trees in the ensemble.

Varying the number of neighbors *k* used to generate synthetic data has a negligible effect on performance across all algorithms and metrics. Further details are available in the Jupyter Notebook 'analysis-pt2,' but we do not explore this idea any further here.

3.3 ADASYN

ADASYN (Adaptive Synthetic Sampling Approach for Imbalanced Learning) is a technique with an approach very similar to SMOTE but includes weights on the minority class samples when generating data. The weights are used to determine regions in the

feature space which contain minority class samples that are 'difficult to learn' and generates more synthetic data in those regions.[‡‡]

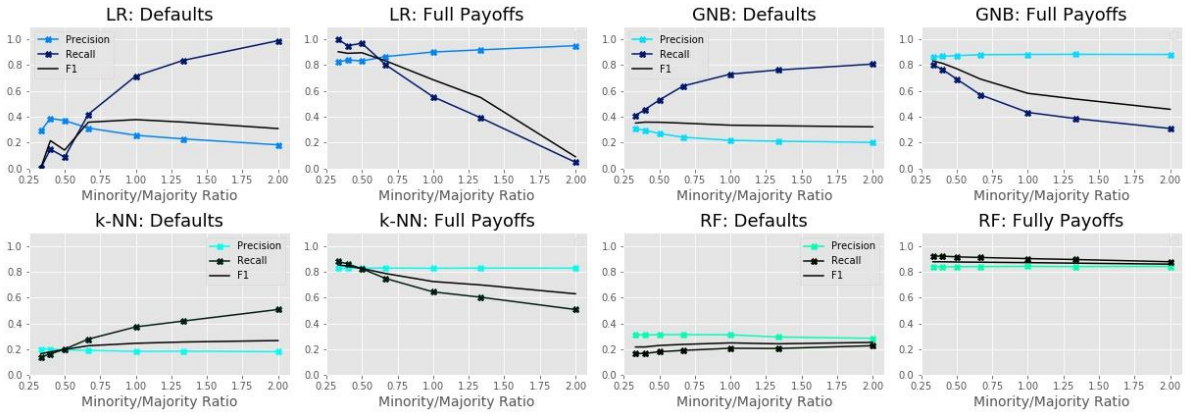

Figure 3.3 *Model performance after applying ADASYN with varying minority to majority resampling ratios.*

Between SMOTE and ADASYN, we see the same pattern of change as between undersampling and SMOTE. Gaussian Naïve Bayes seems affected by ADASYN more than it is by SMOTE, and k-NN seems to be affected less. Random Forest is similarly affected by ADASYN as by SMOTE. This is not surprising, as ADASYN takes a very similar approach to SMOTE in generating synthetic data.

3.4 Resampling Results

We have seen various performance patterns across algorithms as we vary the resampling techniques and resampling ratios. Recall on defaults can be strengthened by changing these model architecture parameters, but this generally comes with a weakening in other performance metrics. Thus, we must study how overall performance (on both classes) is affected as we vary the algorithm, resampling technique, and resampling ratio. To most effectively apply resampling, however, we must conduct deeper analysis of the algorithms (as we have illustrated with oversampling's effects on Random Forest).

[‡‡] The weights are determined as proportions of majority class samples out of the *k* nearest neighbors. That is, the algorithm defines a minority sample as 'difficult to learn' if most of its nearest neighbors are of the opposite class.

§4: Hyperparameter Optimization

We now step away from resampling of the training data and move onto exploration of the models' hypothesis space by variation of hyperparameters. We evaluate model performance using K-Fold Cross Validation over the training set and studying average performance on the folds. Each fold used to compute the metrics is not included in the respective fit of the models, so the metrics are representative of the models' generalization error.

We do not include our analysis of training set error in this report for the sake of clarity in the visualizations, but we simply note we did not observe any significant degree of overfitting in any of the models. We do, however, include performance on the held-out test set to study model performance when the algorithms are fit on the entire training set rather than on the subset K-1 folds. Performance generally does not improve when fitting on the entire training set, suggesting that the performance issues cannot be improved by adding more training data.

We initially explore how variations in hyperparameters can affect model performance while holding resampling ratios/techniques constant. We later explore approaches to combine resampling with hyperparameter optimization to develop models with more desirable performance characteristics. For sub-sections 4.1 to 4.4, however, we use the natural class imbalance to study the effects of hyperparameter variation (i.e. we do not apply any resampling techniques).

4.1 Logistic Regression

Logistic Regression, as implemented in scikit-learn, contains a single hyperparameter to optimize with respect to: C. C is the inverse of the L2 regularization strength—that is, smaller values of C correspond to greater penalization for large parameter magnitudes when fitting the model. In effect, C serves to limit the complexity of the model to prevent overfitting.

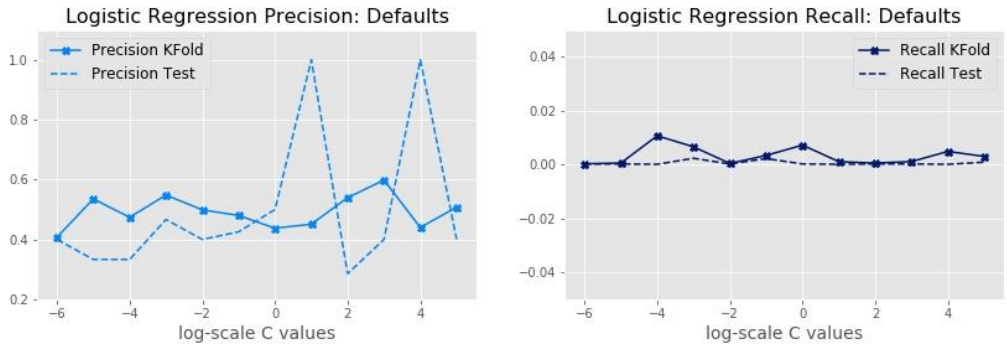

Figure 4.1 *Changes in Logistic Regression performance on defaults, with respect to variations in regularization strength on Logistic Regression. Note that the vertical axes are not equally scaled.*

In *figure 4.1,* we see that adjusting regularization strength has a greater effect on Logistic Regression's precision than on its recall. This may largely be due to the class imbalance and the model's bias against predicting defaults. (Observing the right plot, we see that the recall is in fact close to 0 across all values of C.)

The effect on precision of increasing regularization strength is not entirely clear—this is likely because the model predicts defaults so infrequently when we do not resample, so small changes in the true negative count will produce relatively big swings in precision. This is why we observe two C values which result in 100% precision on the test set. We would expect more stable model performance (with respect to variations in C) with a resampled training set.

In practice, we would search over a greater range of parameter values to gain better insight into the effects of regularization on this model—we leave this as a suggestion for future work in the interest of time and for lack of computational resource.

4.2 Gaussian Naïve-Bayes

Gaussian Naïve-Bayes, as implemented in scikit-learn, automatically computes the estimated prior probabilities of each class from the training data with maximum likelihood estimation. However, these parameters (two in this case of binary classification) can be instead specified as arguments.

With maximum likelihood estimation, it turns out that the prior probabilities are estimated to be the proportion out of the entire data set that each class represents. Since we are dealing with an imbalanced data set, this causes the prior probability of the minority class to be very low. Thus, increasing the prior probability of the minority class should have a very similar effect on Gaussian Naïve-Bayes performance as resampling to increase the minority to majority class ratio. Below, we study variation in minority to majority prior probability ratios in the same way we studied variation in minority to majority resampling ratios when resampling.

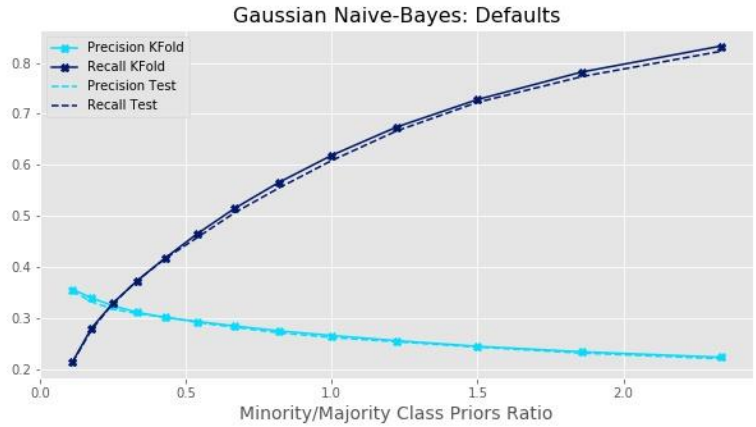

Figure 4.2 *Changes in Gaussian Naïve-Bayes performance on defaults, with respect to variation in the minority to majority class priors ratio.*

Indeed, we see in *figure 4.2* a pattern in precision in recall that is very similar to the patterns in the Gaussian Naïve-Bayes defaults plots in the resampling section (*figure 3.1, figure 3.2, figure 3.3*). We can increase recall on defaults by artificially adjusting its prior probability, however this comes at the expense of reduced precision. Observing the plot above, however, we see that recall is increased a greater rate than precision is decreased.

4.3 k-NN

For k-NN, we study variations of the number of neighbors *k* used in classification, with uniform weights and the Euclidean distance metric. We do not explore the entire hypothesis space for the sake of practicality—we aim to understand the behavior of model performance as we vary *k*. We expect general performance gains with increasing *k*, but we will also consider the increase in computational complexity.

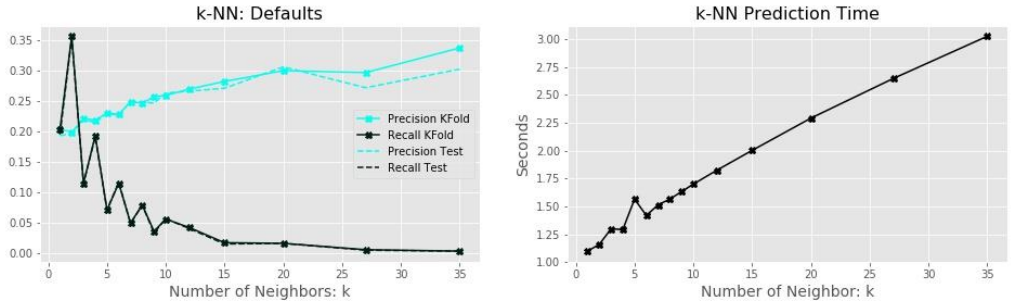

Figure 4.3 *Changes in k-NN performance on defaults, with respect to variation in neighbor counts: k.*

Interestingly, increasing the number of neighbors in k-NN seems to have the opposite effect of increasing the minority to majority class priors ratio for Gaussian Naïve-Bayes. With increasing *k*, the algorithm's precision on defaults is improved while recall deteriorates. The explanation for the increase in precision is intuitive: the algorithm will only predict defaults for data points in the denser default regions of the feature space, i.e. the criteria for default prediction becomes stricter.

When considering the effect on recall however, we must consider the nature of the data set. The imbalance of classes and overlap of feature values results in defaulted loans to be closely surrounded by payoffs in the feature space. As we increase the number of neighbors used in classification, there are decreased counts of defaults which are in dense enough regions of the defaulting class to be classified correctly. Resampling should change this relationship.

The precision of the model appears to be bounded from above by around 0.35 (details available in 'analysis-pt3'), so fitting the algorithm with *k* greater than 35 seems inefficient when considering the linear increase in computational complexity.

4.4 Random Forest

Random Forest contains many hyperparameters available for optimization, but we leave them all as defaults and focus on just one in this report: the number of decision trees used in the ensemble.[§§] We should expect better general performance as we increase the tree count; however, this will come at computational expense. Thus, we will also study computation time to determine a reasonable balance between performance gains and computational overhead.

[§§] 'analysis-pt3' contains results of a grid search over a larger hyperparameter space, but it is still not exhaustive. We leave the complete analysis for future work.

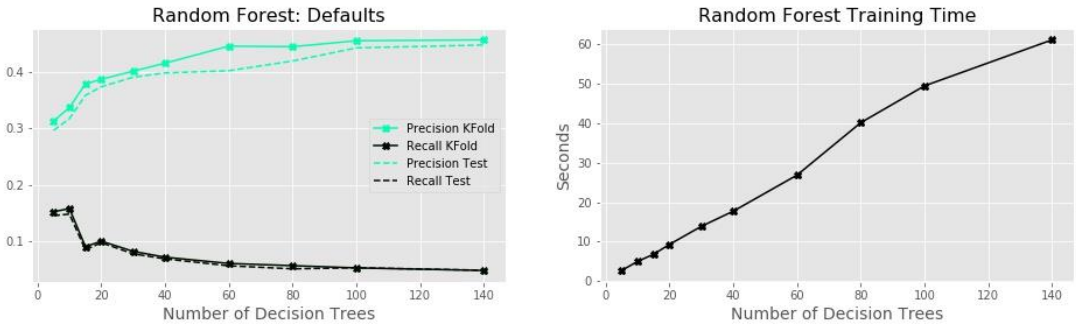

Figure 4.4 Left: *Changes in Random Forest performance on defaults, with respect to variation in decision tree count.* ***Right:*** *Changes in time to train each ensemble, with respect to variation in decision tree count.*

We see above a pattern similar to what we see in k-NN: the model becomes more precise in predicting defaults as we increase the number of decision trees used in the ensemble, but we also observe a decrease in recall. The behavior on recall is likely again due to the class imbalance. The dominance of the majority class causes trees to predict payoffs rather than defaults with greater likelihood. As we increase the count of trees, then, the ensemble will predict payoffs with greater likelihood than it will predict defaults.

Studying the computational time to train the ensembles, we see an approximately linear relationship between the number of trees used and the time to fit the ensemble. However, it seems that a large ensemble is inefficient when we consider the diminishing increases in precision on the left plot. An ideal tree counts looks to be around 60 in this case—however this may change as we alter the training data with resampling.

4.5 Resampling

We now explore the effects of varying hyperparameters after first resampling the training data. We study k-NN and Random Forest, since we have noted in both cases that the class imbalance is a likely cause for decreasing recall as we increase the number of neighbors and tree count in the respective algorithms. This will serve as a segue into the next section, where we explore the effects of both resampling and hyperparameter optimization on all algorithms. We resample here with a 1:1 minority to majority class ratio, with ADASYN for k-NN and under sampling used for Random Forest.[***]

k-NN

In *figure 4.5*, we indeed see that the relationship between the neighbor count and recall has changed. We no longer observe the general decreasing trend in *figure 4.3*, but rather a constant or even slight increasing trend. Unfortunately, this comes at a price of reduced increases in precision. In *figure 4.3*, k-NN has considerable performance gains with respect to precision as we vary the neighbor count, but in *figure 4.5* we do not observe the same increases.

The oscillations observed with low neighbor count is likely a result of the class overlap. With only a few neighbors, a slight change in the count can change the prediction for many inputs since data points are closely surrounded by many instances of both classes. This

[***] This choice is because we have seen in section 3 that over sampling methods do not improve Random Forest's performance due to the nature of the algorithm—very similar trees end up generated in the ensemble.

behavior is also observed in *figure 4.3.* In both cases, we see reductions in the oscillations as we increase the neighbor count.

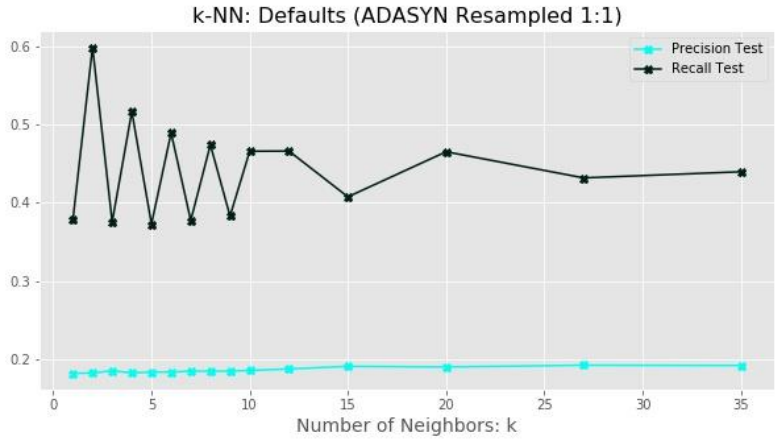

Figure 4.5 *Changes in k-NN performance on defaults after resampling, with respect to variation in neighbors count: k.*

Random Forest

In *figure 4.6,* we see the relationship change again in a very similar pattern. There is no longer a general decreasing trend in recall as we increase the count of decision trees as in *figure 4.4*, however we also see that performance gains in precision are again muted as they were for k-NN.

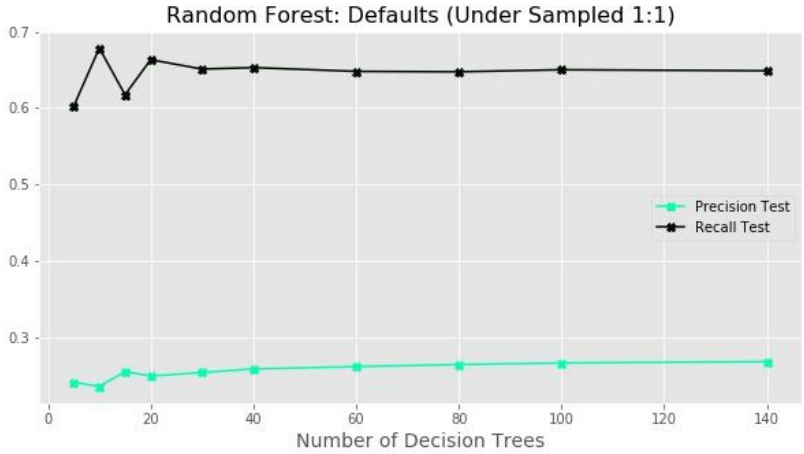

Figure 4.6 *Changes in Random Forest performance on defaults after resampling, with respect to variation in decision tree count.*

In *figure 4.5* and *figure 4.6* we see a similar pattern: model performance becomes almost constant with respect to variation in hyperparameters when we first apply a resampling technique. Resampling allows us to greatly increase recall as well as prevent it from decreasing as we vary hyperparameters, however this comes at the cost of reduced precision as well as prevented performance gains in precision with respect to variation in the hyperparameters. It remains to be seen, however, if varying the resampling ratio will change the extent to which these relationships change.

§5: Model Pipeline

At this point, we have seen various model architectures to approach the classification task at hand—we have varied algorithms, resampling techniques, resampling ratios, and algorithm hyperparameters. The visualizations have been helpful guides in understanding the relationships between model performance as we vary these parameters, but we now step away from visualizations and seek to explore a more exhaustive space of models.

We now develop a pipeline approach to search over resampling techniques and hypotheses spaces to find the optimal model architecture. Yet again, the exploration will not be complete as there are many parameters to vary (as well as parameter values to vary), but we seek to illustrate an approach that could thoroughly explore model architectures with greater computational resources.

In *figure 5.1* and *figure 5.2*, we see a snapshot overview of different models' performances across resampling techniques and optimized algorithms. The resampling ratio is held constant here, so a more exhaustive search over varied resampling ratios would result in multiple tables of this sort—one pair of negative-positive tables for each ratio.

Below, the 're-' prefixed models correspond to those that have been optimized over fixed hyperparameter grids, with respect to negative F1 score. 'N' refers to the negative class (defaults) and 'P' refers to the positive class (payoffs). 'pre' refers to precision and 'rec' refers to recall.

model	N_pre	N_rec	UnderSamp_N_pre	UnderSamp_N_rec	SMOTE_N_pre	SMOTE_N_rec	ADASYN_N_pre	ADASYN_N_rec
LR	0.500000	0.000224	0.273209	0.640721	0.273460	0.630866	0.253973	0.726733
reLR	0.500000	0.000224	0.273283	0.640497	0.274015	0.634786	0.254445	0.719677
GNB	0.325714	0.307985	0.266798	0.607011	0.252503	0.669392	0.217562	0.726173
reGNB	0.263459	0.608915	0.256094	0.664800	0.236426	0.750028	0.208146	0.788106
KNN	0.229395	0.070445	0.203121	0.543734	0.196971	0.413596	0.183698	0.373054
reKNN	0.215069	0.191175	0.194014	0.706350	0.195092	0.370366	0.183156	0.517303
RF	0.335203	0.160600	0.238553	0.688543	0.311747	0.223205	0.304969	0.210326
reRF	0.331065	0.152537	0.259314	0.664128	0.314198	0.207190	0.311830	0.169448

Figure 5.1 *Defaulted (negative class) test set performance. Resampling ratios are set to 1:1 for all resampling techniques.*

model	P_pre	P_rec	UnderSamp_P_pre	UnderSamp_P_rec	SMOTE_P_pre	SMOTE_P_rec	ADASYN_P_pre	ADASYN_P_rec
LR	0.823123	0.999952	0.891379	0.633674	0.889674	0.639764	0.902102	0.541196
reLR	0.823123	0.999952	0.891360	0.633939	0.890530	0.638537	0.900749	0.546781
GNB	0.852989	0.862968	0.883650	0.641473	0.889863	0.574100	0.881718	0.438705
reGNB	0.882964	0.634132	0.890346	0.584956	0.899223	0.479384	0.886475	0.355614
KNN	0.826112	0.949139	0.846681	0.541533	0.834957	0.637598	0.826907	0.643712
reKNN	0.830218	0.850042	0.854058	0.369334	0.832294	0.671585	0.829341	0.504152
RF	0.837757	0.931544	0.887418	0.527645	0.842653	0.894091	0.840893	0.896979
reRF	0.836777	0.933759	0.891364	0.592298	0.841228	0.902804	0.837447	0.919629

Figure 5.2 *Payoff (positive class) performance. Values here correspond to the respective models in figure 5.1.*

We are primarily concerned with improving performance on defaults, but we must also consider the tradeoffs in performance on payoffs. Thus, we must consider *figure 5.2* together with *figure 5.1* to find an acceptable level of performance.

An optimized Gaussian Naïve-Bayes model used in combination with ADASYN returns the highest recall on defaults, but also comes with a relatively low precision on defaults as well as poor recall on payoffs. Logistic Regression together with ADASYN offers a lower recall on defaults, but also has a more balanced performance with respect to all the other metrics. We see there are various tradeoffs with different model architectures, and the choice of which to implement in practice will depend on the desired performance levels across all metrics.

§6: Categorical Features

So far, we have explored model performance on a limited subset of the features which only contain numerical variables. We now seek to utilize the entire feature space—including categorical features—as inputs to the models. Since we are adding more information about each loan, we expect a general performance increase, but we must first transform the data to be effectively input to the algorithms.

The following transformations are developed so they can be fit on training data and saved so that new test data can be transformed with the same fit. The aim with this approach is to develop a method that can be used in practice on unseen data.

6.1 Transformations

Indicator Features

Our approach to transforming categorical features of the data set to a form that can be input to the algorithms involves first developing indicator features for each categorical value.[†††] The transformation will place a 1 in the corresponding dummy feature if the data point takes on that value, and a 0 otherwise. With this transformation, the categorical information is numerically encoded in a form that can be input to the classification models.

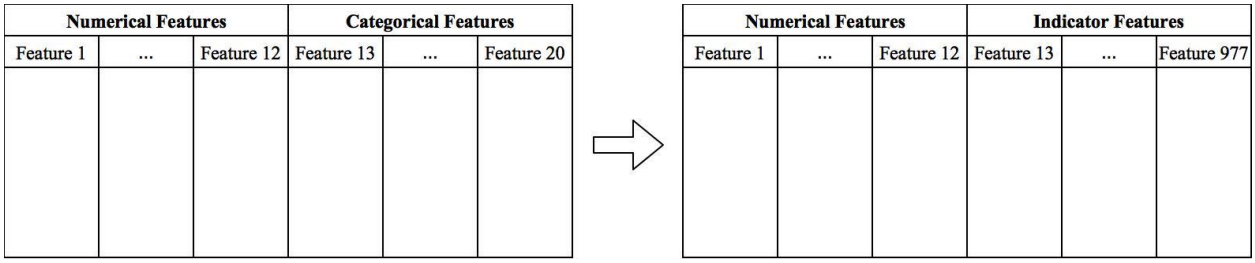

Figure 6.1 *First transformation: categorical features transformed to indicator features.*

With this transformation, however, we are left with 965 indicator features in place of the categorical columns (one for each unique value of each categorical feature). With this large increase in the dimension of the feature space, the algorithms become much more expensive to fit. Running an entire pipeline of algorithms becomes infeasible with this transformed feature space, so we must reduce the dimension to a manageable level.

Principal Component Analysis

We apply PCA to the indicator features (i.e. features 13 to 977 in *figure 6.1*) to reduce the categorical indicator features to a 12-dimensional space. We do not include the numerical features (i.e. features 1 to 12) in the PCA transformation, since the values they take on are of different nature than the indicator variables.

[†††] We use get_dummies with default arguments, as implemented in the Pandas library.

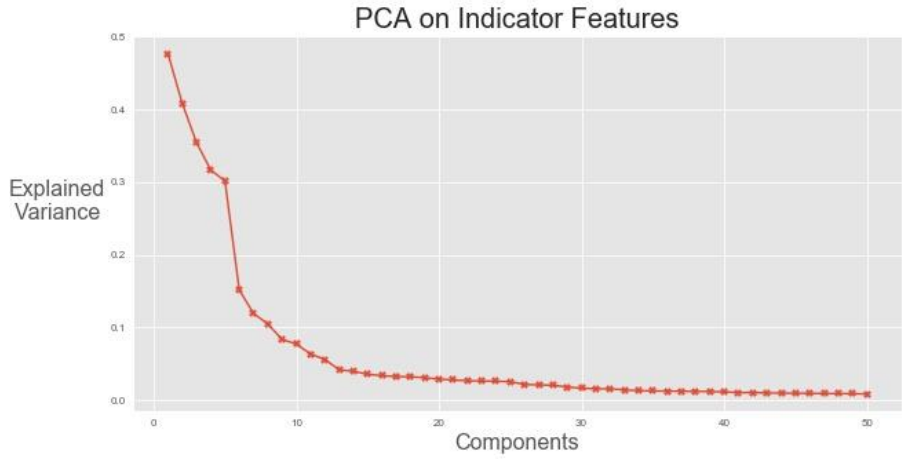

Figure 6.3 *Explained variance on the 50 principal components of indicator features.*

Studying the explained variance of the first 50 principal components, we see there is a steep drop after the 5th component. However, the explained variance continues to drop until after the 12th component, at which point all additional components have approximately the same low level of explained variance. The computational expense of using 12 components over 5 is not significant, so we proceed with 12 components to capture more information about the indicator features.

Numerical Features			Indicator Features		
Feature 1	...	Feature 12	Feature 13	...	Feature 977

⇨

Numerical Features			PCA-Reduced Indicator Features		
Feature 1	...	Feature 12	Feature 13	...	Feature 24

Figure 6.3 *Second transformation: indicator features projected onto 12 principal components.*

At this point in the sequence of transformations, we have a training set that can be effectively input to the classification algorithms. However, we develop an alternative training set that includes additional transformations. We will run the algorithm pipeline with both versions of the training set to evaluate which has more predictive value.

Clustering

We apply K-Means to develop clusters within the PCA-reduced indicator features. Again, we do not include the numerical features since their values are of different natures than the categorical indicators. The goal with clustering on this subset is to group together data points which are categorically similar. (We could apply clustering before the PCA transformation, but applying K-Means on 965 features is very computationally expensive.)

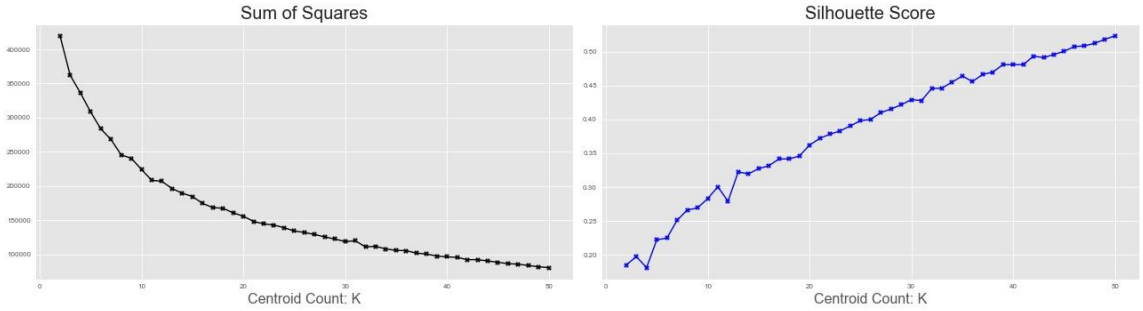

Figure 6.4 *Sum of squares and silhouette score with respect to K-Means ran with varied cluster counts.*

It is not entirely clear from *figure 6.4* what the ideal count of clusters should be. The sum of squares seems to continue to decrease with increased cluster count, and the average silhouette score seems to continue to increase with increased cluster count. We use 30 clusters to not develop too many groupings, and we replace the indicator features entirely with the cluster results.

Numerical Features			**PCA-Reduced Indicator Features**		
Feature 1	…	Feature 12	Feature 13	…	Feature 24

⇨

Numerical Features			**Clusters**
Feature 1	…	Feature 12	Feature 13

Figure 6.5 *Third transformation (alternative training set): indicator features clustered and replaced with results of K-Means.*

At this point along the alternative sequence of transformations, however, we are back to having a categorical feature (the clusters), which cannot be effectively input to the classification algorithms. Thus, the final transformation on this alternative sequence is to again generate indicator features from the clusters. We are now ready to re-run the model pipeline from section 5 on these two alternative versions of the training set which incorporate categorical information.

Numerical Features			**Clusters**
Feature 1	…	Feature 12	Feature 13

⇨

Numerical Features			**Cluster Indicator Features**		
Feature 1	…	Feature 12	Feature 13	…	Feature 42

Figure 6.6 *Final transformation (alternative training set): K-Means clustering results transformed to indicator features*

6.2 Pipeline Results

We study below the results of the pipelines using the alternative training sets. For this report, we limit the study to the defaulting class for the sake of clarity and ease of understanding.[‡‡‡] Indeed, in comparison with the table in *figure 5.1*, we see many performance increases. However, this is not strictly true as some algorithm-resampling combinations have reduced performance when we include categorical information.

model	N_pre	N_rec	UnderSamp_N_pre	UnderSamp_N_rec	SMOTE_N_pre	SMOTE_N_rec	ADASYN_N_pre	ADASYN_N_rec
LR	0.426554	0.017151	0.274980	0.635166	0.274922	0.633689	0.257353	0.714562
reLR	0.402256	0.012154	0.274669	0.632326	0.271399	0.634939	0.255227	0.715470
GNB	0.318875	0.332349	0.274333	0.566333	0.262404	0.618128	0.231312	0.685711
reGNB	0.261056	0.644366	0.260950	0.642890	0.248248	0.696274	0.220014	0.760904
KNN	0.300030	0.111881	0.230889	0.587347	0.225715	0.508065	0.244896	0.290209
reKNN	0.265069	0.251249	0.208885	0.733303	0.223567	0.448887	0.244571	0.283962
RF	0.316195	0.144593	0.230510	0.690482	0.300139	0.196161	0.301639	0.177647
reRF	0.317656	0.149591	0.265471	0.647547	0.347698	0.120968	0.375723	0.066447

Figure 6.7 *Pipeline results using categorical features without clustering (i.e. right-hand side of figure 6.3), with resampling ratio of 1:1.*

model	N_pre	N_rec	UnderSamp_N_pre	UnderSamp_N_rec	SMOTE_N_pre	SMOTE_N_rec	ADASYN_N_pre	ADASYN_N_rec
LR	0.427136	0.019309	0.273490	0.625511	0.271991	0.629373	0.256371	0.708428
reLR	0.404494	0.008178	0.273836	0.625057	0.272186	0.627783	0.256325	0.708882
GNB	0.316061	0.320082	0.270894	0.566220	0.241990	0.727510	0.209674	0.825193
reGNB	0.270347	0.562926	0.256158	0.647319	0.227005	0.800886	0.200532	0.873807
KNN	0.297207	0.111199	0.227424	0.588142	0.223180	0.497160	0.242788	0.286801
reKNN	0.264309	0.251249	0.209091	0.735120	0.223570	0.445820	0.246706	0.287142
RF	0.302279	0.138573	0.227376	0.684348	0.302046	0.199568	0.292909	0.182985
reRF	0.308200	0.137892	0.259681	0.679464	0.328131	0.183894	0.315528	0.108019

Figure 6.8 *Pipeline results using categorical features with clustering (i.e. right-hand side of figure 6.6), with resampling ratio of 1:1.*

Between *figure 6.7* and *figure 6.8*, we also see variable increases and decreases in model performance. It seems that there is no de-facto ideal training set to use between the two alternatives. Furthermore, there still does not appear to be a clear choice in combination of algorithm and resampling technique to use. In *figure 6.8* we see that an optimized Gaussian Naïve-Bayes model used together with an ADASYN resampled training set achieves very good recall on defaults (87%) but still struggles with precision.

In fact, all the algorithms with all the resampling techniques and alternative training sets are still very weak at precisely predicting defaults. This, we hypothesize, goes back to the

[‡‡‡] The full set of results, including payoffs, is available in 'analysis-pt5.'

nature of the data set—the overlap between classes makes it very difficult for the algorithms to make precise predictions.

6.3 Alternative Approach

A potential alternative approach to dealing with categorical features involves first developing a decision tree with leaves corresponding to all the possible combinations of categorical values. Having done this, we can fit a model to each combination of categorical values using a reduced training set with matching categorical values. To predict an outcome, each data point will be input to the corresponding model that has been trained on matching categorical data.

The time to train this collection of models should not be significantly longer than training a single model as we have done in section 6, since each model is trained on a mutually exclusive subset of the training data. However, we would require an efficient (vectorized) way to predict outcomes for a batch of test data without iteratively assigning data points to their respective models. We do not explore this approach in this project and leave this as a suggestion for future work.

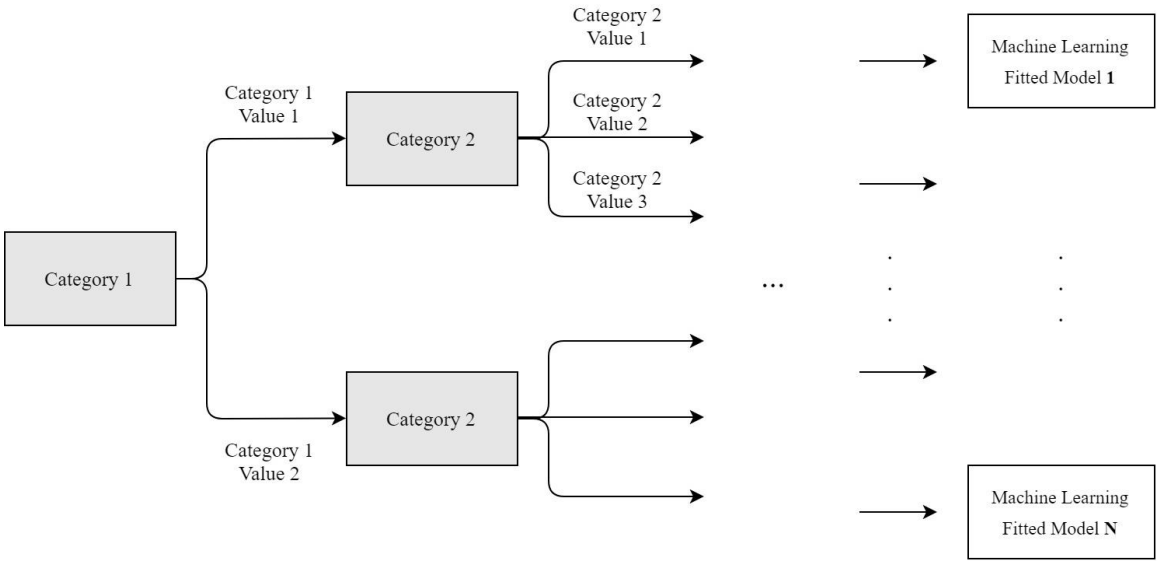

Figure 6.9 *Alternative overarching model architecture.* **N** *is the number of combinations of categorical values across all categories.*

§7: Ensemble Learning

In this penultimate section, we explore ensemble architectures with the goal of improving general performance. There are numerous variations we can consider, but we limit our study to three for the sake of illustration. We first study a 'majority vote' type ensemble, where the ensemble predicts a default only when a sufficient number of individual classifiers predicts defaults. Second, we study the approach of taking joint probabilities of the algorithms and varying the discrimination threshold for prediction. Lastly, we study a 'stacking' approach, where we initially output each algorithm's probabilities for default—followed by another run of a classifier to consolidate the probabilities into a prediction.

We train the algorithms in the ensemble on different re-sampled versions of the training data based on how they best performed through the study so far. Our hope with varying the training data is to instill more randomness in the fitted models to increase generalizability of the ensemble.

7.1 Majority Vote

Our first approach is the most intuitive and simple to interpret. 'Majority vote' is misleading since we will vary the discrimination threshold, but the approach is easy to understand—we output predictions of all classifiers as we have done so far but have the ensemble predict defaults only for those which satisfy the minimum prediction counts.

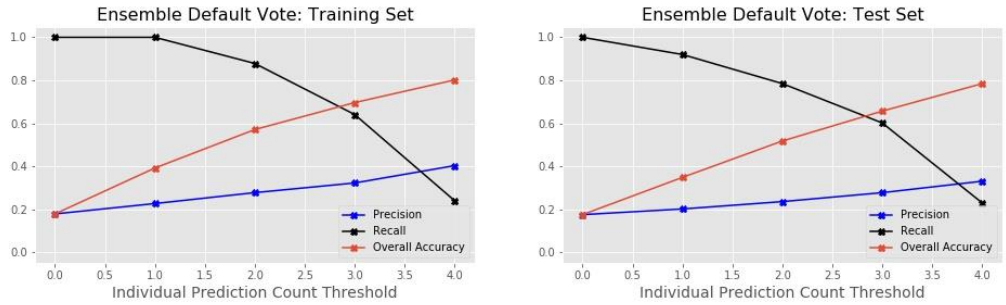

Figure 7.1 *Majority vote ensemble, with varying discrimination thresholds.*

In *figure 7.1*, we vary the number of default predictions needed for the ensemble to predict a default along the horizontal axes. We can see that as the criteria gets stricter (i.e. more individual default predictions are needed for an overall default prediction), the precision of the ensemble increase. However, as we would expect, the recall correspondingly falls as fewer defaults satisfy the criteria. We plot the accuracy as well, since we want to study the model's overall performance on the test set.

7.2 Joint Probability

We now consider an approach which takes the predicted probabilities of each algorithm into consideration, rather than the predictions. In particular, we take the product of the probabilities of default of each algorithm as a joint probability—and again vary the discrimination threshold.

$$P_{ensemble}(Y = Default \mid X) = P_{LR}(Y = Default \mid X)P_{GNB}(Y = Default \mid X)P_{kNN}(Y = Default \mid X)P_{RF}(Y = Default \mid X)$$

This approach is intuitively similar to the majority vote in that it directly consolidates information to produce a single prediction, but this approach incorporates weighted outputs rather than binary predictions.

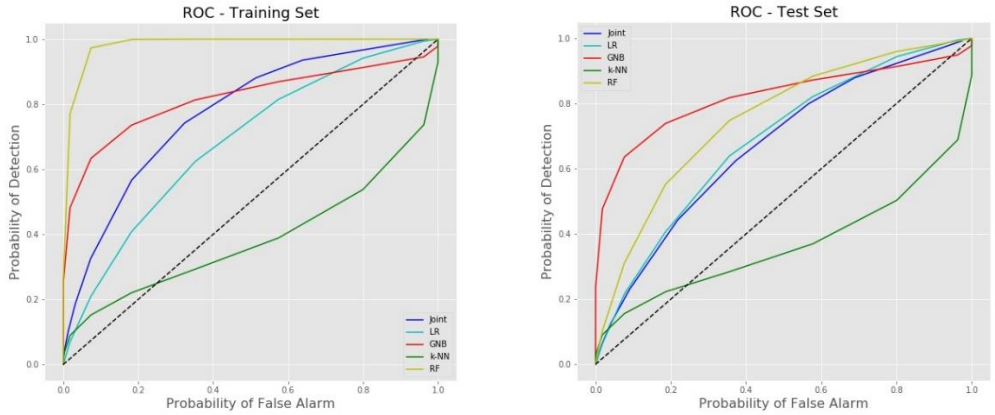

Figure 7.2 *Receiver operating characteristics of the ensemble as we vary the probabilistic threshold for default prediction.*

Studying the ROC curves above, we can see that there does not appear to be much benefit in taking a joint probability over a single best classifier. Particularly, the approach k-NN takes in outputting probabilities (as implemented in scikit-learn) does not appear to be reliable in predicting defaults. k-NN is likely reducing the predictive strength of the ensemble.

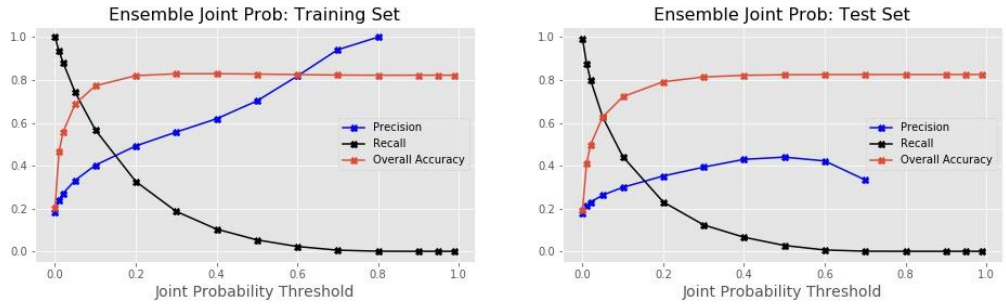

Figure 7.3 *Joint probability ensemble, with varying discrimination thresholds.*

In *figure 7.3*, we see that we can achieve fairly high precision on the training set, but this is not replicated on the test set. As the discrimination threshold on the joint probability is increased past around 0.7, the ensemble does not predict any defaults (thus the precision curve ends prematurely). Again, we see the tradeoff we have seen through this project between precision and recall as we vary the 'strictness' of the prediction criteria.

7.3 Stacking

Lastly, we consider an ensemble approach which applies classification algorithms 'in sequence' rather than taking a simple consolidation of the outputs as in the previous two cases. Specifically, we take the predicted probabilities of default for each algorithm and input the probabilities into each classifier, with the hope that the classifiers will find patterns in the output probabilities of the four algorithms and the target variable.

In *figure 7.4*, we see a general overview of the stacked model architecture. The final classifier ('Stacked Classification Algorithm') uses the predicted default probabilities as features. The final classifier is still fit solely on the training data, and implementation of the model requires an intermediary step where the base algorithms transform the original features into the 'predicted-probability-features.' In *figure 7.5*, the 'Stacked Classification Algorithm' is what varies between pairs of plots.

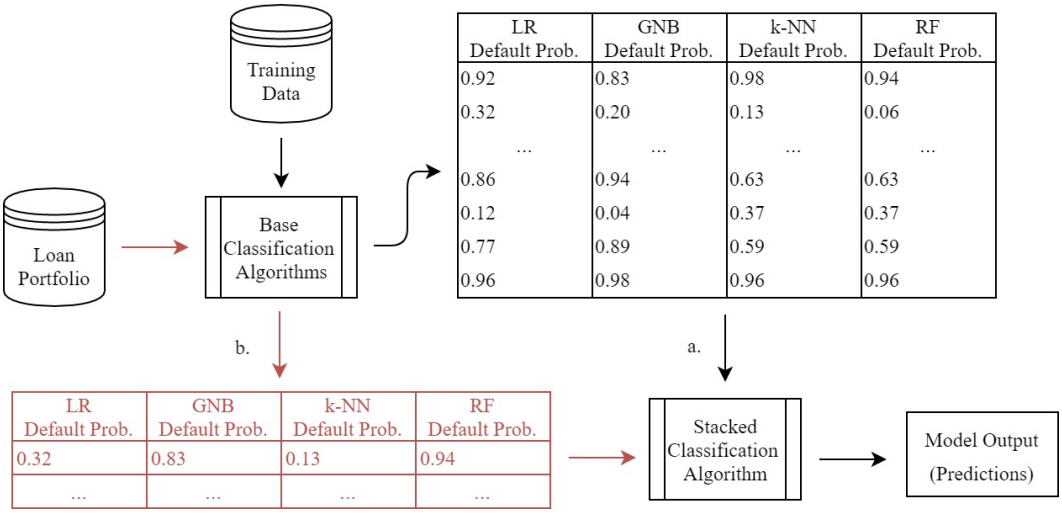

Figure 7.4 *Stacked model architecture. Region a. represents the fitting of a new model on the predicted probabilities of the training data. Region b. represents the intermediary step in implementation, requiring the base algorithms to predict the probabilities of default before inputting into the final classifier.*

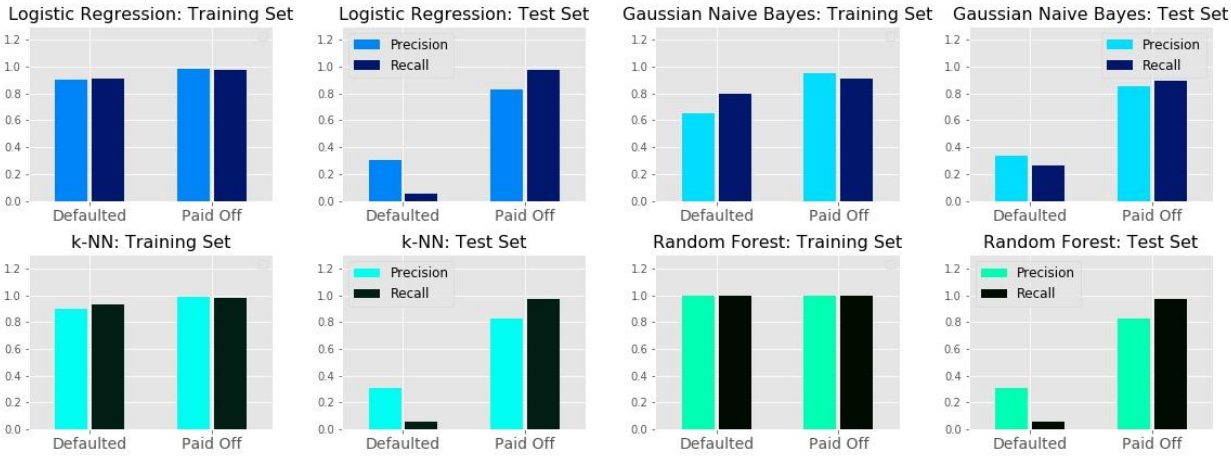

Figure 7.5 *Classifiers stacked on the initial predicted probabilities of each algorithm.*

In *figure 7.5*, we see a large degree of overfitting as the algorithms successfully find patterns between the output probabilities and the target variables within the training set but fail to generalize to the test set. This suggests that the patterns the algorithms find in the probabilities are specific to the set they were trained on and do not represent general patterns.

7.4 Ensemble Conclusions

We have explored just three approaches to combining the predictive power of different algorithms, but we can see there is much room for creativity in developing ensemble architectures. We can incorporate more variations of base learning algorithms (e.g. SVM, neural networks, boosting, bagging, etc.) on top of varying the structure of the ensemble.

As has been the case throughout this project, we have illustrated approaches to analyze and evaluate model architectures, but a comprehensive study is left to be done. There is great depth and breadth to the ensemble approaches we can take and leave further exploration for future study.

§8: Conclusion

8.1 Overview

We have explored different frameworks for predicting defaults on loans, with variations in the model architecture at different points. This includes variations in the classification algorithms, resampling techniques, resampling ratios, algorithm hyperparameters, and the input features. We have not exhaustively explored all methods, but we have illustrated approaches to applying the changes to the model architecture and evaluating results.

Still, this study is not complete—there are entirely different methods to alter the model architecture that we have not considered. As an example, consider the output of the machine learning system. Rather than directly predicting defaults, the model could instead output default probabilities. Alternatively, the model could incorporate the probabilistic outputs of multiple classification algorithms before combining them back to a prediction.

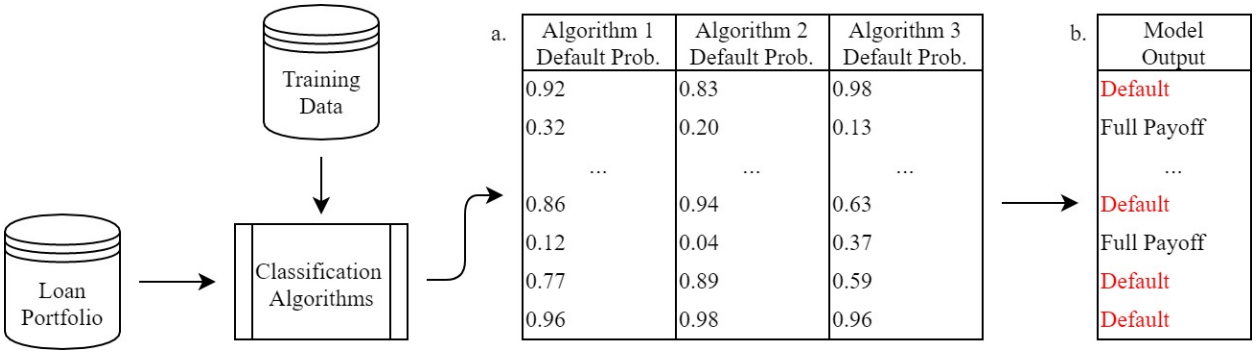

Figure 8.1 *Example potential variation of the model architecture.*

In *figure 8.1*, we see an example of this variation in the model architecture. By leveraging the predictive value of multiple algorithms in combination, it is conceivable that the overall model will perform better than any single algorithm. We can take traditional ensemble approaches to arrive from a to b (e.g. majority vote of each classification algorithm's prediction) or develop new methods to combine ensemble results (e.g. input part a to another classification algorithm).

We do not further explore this idea in this report, but we can see there are many modeling approaches to take and that an exhaustive exploration is infeasible.[§§§] The approaches in this project are chosen based on the characteristics of the data set but really serve to illustrate the process of evaluating machine learning models and intelligently improving them.

The changes in model performance with respect to changes the model architecture are not always easily interpretable (e.g. changing the input features, section 6.2), but careful data analysis and algorithm analysis enables us to improve performance (e.g. resampling before hyperparameter optimization, section 4.5).

[§§§] A brief exploration of heterogeneous ensemble approaches is available in 'analysis-pt6.'

8.2 Implementation and Risk Quantification

Recommendations for the Client

With the performance levels seen thus far, it is apparent that these models are not suited for fully automated loan portfolio management. The models are not a replacement for analysis—they are a *supplement*. We again suggest these models be used together with other financial and economic analysis in determining management decisions.

For example, a loan predicted to have high probability of default may be worth retaining in the portfolio if it has sufficiently high installments and does not overly increase the risk level of the portfolio. So, we see that the models' outputs are not the end of the story—their primary role should be in *supporting* decision making rather than in decision making itself. Still, the models are very effective in interpreting vast collections of data and transforming them to an output that can be easily interpreted by a human manager.

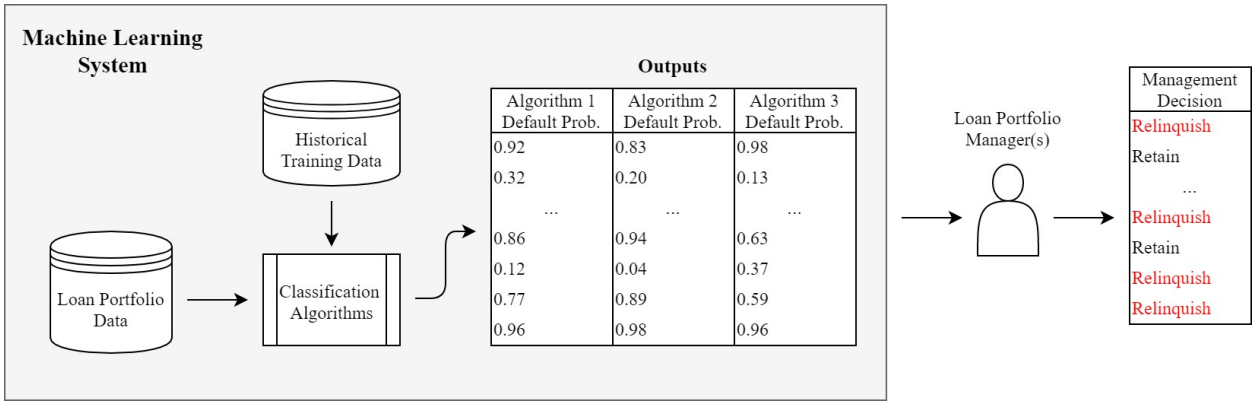

Figure 8.2 *Potential model implementation approach in practice.*

In *figure 8.2*, the machine learning system's outputs represent a concise mapping of the input data and serve the client (loan portfolio manager) to quantify risk much more effectively than the input data itself. The client using the machine learning system will still make management decisions based on additional analysis. Additionally, the machine learning system's outputs could be incorporated within other computational models to aid in risk quantification.

Limitations

We have studied precision and recall through this project, ideally looking to maximize both with respect to the defaulting class. The initial weak precision is a result of the overlap of classes in the feature space, and the initial weak recall is a result of the class imbalance. Our analysis shows there are inevitable tradeoffs to be made as we change the model architecture to improve performance with respect to a metric.

Resampling to improve recall generally hurts precision, whereas hyperparameter optimization to improve precision has been seen to hurt recall. In either case, improving performance on the defaulting class often hurts performance on the payoff class.

Combinations of techniques can be used to lessen the unintended reductions in model performance, but we have seen that certain performance levels are infeasible.

This is particularly true for precision on the defaulting class, which has not been seen to be greater than 50% for any model architecture. The feature overlap between classes is an inherent characteristic of the data that cannot be artificially altered as, for example, the class imbalance on training data can with the application of resampling techniques.

Many defaulting loans have data characteristics nearly identical to those of payoff loans, as we have seen in the preliminary exploration. The underlying causes of default which separate them from the payoffs, then, are not captured within the data set used in this project. It is erroneous to assume that machine learning algorithms, even very powerful ones, will find clear patterns where they do not exist.

To really improve model performance past the limits seen in this project, we require more relevant features to describe each loan—the feature space used in this study is not sufficient to distinctly separate the classes. The models will catch the extreme cases, but they will not reliably predict the boundary points which overlap with the opposing class. Unfortunately, in the context of consumer loan default prediction, this information is difficult to capture and may not be practical.

8.3 Future Work

We have seen that an exhaustive study is infeasible for various reasons. On one hand, there are practical limitations such as a lack of computational resources to evaluate model performance over bigger collections of model architectures. On the other hand, there are domains and creative approaches which have not been touched upon in this project. We can see that the development of better model architectures requires not only the resources to implement the models but also the creativity and ingenuity to develop novel systems.

The approaches studied in this project are a mixture of general machine learning techniques (e.g. hyperparameter optimization) as well as techniques specific to modeling scenarios (e.g. resampling for imbalanced classification). There are many techniques we have not explored: feature selection algorithms, alternative approaches to dealing with categories, different learning algorithms, and so on. We see we can develop increasingly complex model architectures to improve performance but leave this as future work to be done.

Appendix A

Data Dictionary: Features

NUMERICAL

loan_amnt - The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.

funded_amnt - The total amount committed to that loan at that point in time.

term - The number of payments on the loan. Values are in months and can be either 36 or 60.

int_rate - Interest Rate on the loan.

Installment - The monthly payment owed by the borrower if the loan originates.

grade - LC assigned loan grade.

emp_length - Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.

dti – A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower's self-reported monthly income.

open_acc - The number of open credit lines in the borrower's credit file.

revol_util - Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.

tot_coll_amnt - Total collection amounts ever owed.

annual_inc - The self-reported annual income provided by the borrower during registration.

Appendix A

Data Dictionary: Features

CATEGORICAL

home_ownership - The home ownership status provided by the borrower during registration. Our values are: RENT, OWN, MORTGAGE, OTHER.

verification_status - Indicates if the borrowers' joint income was verified by LC, not verified, or if the income source was verified.

pymnt_plan - Indicates if a payment plan has been put in place for the loan.

purpose - A category provided by the borrower for the loan request.

zip_code - The first 3 numbers of the zip code provided by the borrower in the loan application.

addr_state - The state provided by the borrower in the loan application.

initial_list_status - The initial listing status of the loan. Possible values are – W, F.

application_type - Indicates whether the loan is an individual application or a joint application with two co-borrowers.

Appendix B

Within each dimension of model architecture we modify, there are several techniques available for exploration. We can summarize the model architecture with a tuple, with each entry representing a technique within a dimension. Below, we have a summary of architectures studied in this project—in order that they appear in the report.

For resampling and hyperparameter optimization, we do not list the entirety of resampling ratios and hyperparameter values for the sake of brevity. As we can see, there are several architectures to explore within the 'model space.' An exhaustive exploration not contained in this report, but we study these particular combinations to study the feasible performance characteristics within the 'model space.'

(A, R, H, C)

A – Algorithm
R – Resampling
H – Hyperparameter Optimization
C – Categorical Information

Section 2 – Baseline Evaluations
(Logistic Regression, None, None, None)
(Gaussian Naïve Bayes, None, None, None)
(k-NN, None, None, None)
(Random Forest, None, None, None)

Section 3 – Resampling
(Logistic Regression, Under Sample, None, None)
(Gaussian Naïve Bayes, Under Sample, None, None)
(k-NN, Under Sample, None, None)
(Random Forest, Under Sample, None, None)

(Logistic Regression, SMOTE, None, None)
(Gaussian Naïve Bayes, SMOTE, None, None)
(k-NN, SMOTE, None, None)
(Random Forest, SMOTE, None, None)

(Logistic Regression, ADASYN, None, None)
(Gaussian Naïve Bayes, ADASYN, None, None)
(k-NN, ADASYN, None, None)
(Random Forest, ADASYN, None, None)

Section 4 – Hyperparameter Optimization
(Logistic Regression, None, C, None)
(Gaussian Naïve Bayes, None, Prior Probabilities, None)
(k-NN, None, K, None)
(Random Forest, None, Number of Decision Trees, None)

Appendix B

Section 5 – Pipelining
(Logistic Regression, Under Sample, C, None)
(Gaussian Naïve Bayes, Under Sample, Prior Probabilities, None)
(k-NN, Under Sample, K, None)
(Random Forest, Under Sample, Number of Decision Trees, None)

(Logistic Regression, SMOTE, C, None)
(Gaussian Naïve Bayes, SMOTE, Prior Probabilities, None)
(k-NN, SMOTE, K, None)
(Random Forest, SMOTE, Number of Decision Trees, None)

(Logistic Regression, ADASYN, C, None)
(Gaussian Naïve Bayes, ADASYN, Prior Probabilities, None)
(k-NN, ADASYN, K, None)
(Random Forest, ADASYN, Number of Decision Trees, None)

Section 6 – Categorical Information
(Logistic Regression, Under Sample, C, Non-Clustered Categories)
(Gaussian Naïve Bayes, Under Sample, Prior Probabilities, Non-Clustered Categories)
(k-NN, Under Sample, K, Non-Clustered Categories)
(Random Forest, Under Sample, Number of Decision Trees, Non-Clustered Categories)

(Logistic Regression, SMOTE, C, Non-Clustered Categories)
(Gaussian Naïve Bayes, SMOTE, Prior Probabilities, Non-Clustered Categories)
(k-NN, SMOTE, K, Non-Clustered Categories)
(Random Forest, SMOTE, Number of Decision Trees, Non-Clustered Categories)

(Logistic Regression, ADASYN, C, Non-Clustered Categories)
(Gaussian Naïve Bayes, ADASYN, Prior Probabilities, Non-Clustered Categories)
(k-NN, ADASYN, K, Non-Clustered Categories)
(Random Forest, ADASYN, Number of Decision Trees, Non-Clustered Categories)

(Logistic Regression, Under Sample, C, Clustered Categories)
(Gaussian Naïve Bayes, Under Sample, Prior Probabilities, Clustered Categories)
(k-NN, Under Sample, K, Clustered Categories)
(Random Forest, Under Sample, Number of Decision Trees, Clustered Categories)

(Logistic Regression, SMOTE, C, Clustered Categories)
(Gaussian Naïve Bayes, SMOTE, Prior Probabilities, Clustered Categories)
(k-NN, SMOTE, K, Clustered Categories)
(Random Forest, SMOTE, Number of Decision Trees, Clustered Categories)

Appendix B

(Logistic Regression, ADASYN, C, Clustered Categories)
(Gaussian Naïve Bayes, ADASYN, Prior Probabilities, Clustered Categories)
(k-NN, ADASYN, K, Clustered Categories)
(Random Forest, ADASYN, Number of Decision Trees, Clustered Categories)

References

1. https://www.kaggle.com/wendykan/lending-club-loan-data

2. N. Chawla, K. Bowyer, L. Hall, W. Kegelmeyer. 2002. SMOTE: Synthetic Minority Over-sampling Technique
   - *https://www.jair.org/media/953/live-953-2037-jair.pdf*

3. H. He, Y. Bai, E. Garcia, S. Li. 2008. ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning
   - *http://www.ele.uri.edu/faculty/he/PDFfiles/adasyn.pdf*