Loan Default Prediction
Internal Report for Lenders "R" Us

Katie Carlson (kc594) Michael Luo (mrl233)

Abstract

Here at Lendors "R" Us we have seen great success over the years, thanks to the autonomy and empowerment we provide to our loan agents. However, our new hires often don't have the intuition or experience to make optimal loan approval decisions. We wish to identify lower-performing loan agents early, so that we may educate and assist them with early intervention. In an effort to identify these lower-performing loan agents in a principled manner, our goal is to predict whether or not the loans our agents have approved will default.

In this research, we investigate and model which factors describing loans best predict loan defaults. We use a big messy dataset from Lending Club, and apply a variety of techniques for learning from such data. These techniques include: *data imputation, feature engineering, feature selection, parameter tuning via validation, regularized logistic regression, and random forest regression*. Our results indicate that we can predict whether a loan will default with 81% accuracy.

1. Introduction

One of the founding beliefs of Lenders "R" Us is that our loan agents deserve to work with autonomy and personal empowerment. This has been crucial to our success in the past. However, as recent events in our San Diego office have revealed, our new hires often don't have the intuition or experience to make optimal approval decisions. Last quarter's report shows our San Diego new hires have approved eventually-defaulting loans at a significantly higher rate than our veteran employees. This is cutting significantly into our bottom-line, and in order to stay solvent, we need to pro-actively educate our lower-performing loan agents.

One key issue is to identify lower-performing loan agents early. As they are approving many loans per week, we cannot afford to wait for a significant proportion of their approved loans to default. Instead, we wish to predict whether or not the loans they have already approved will default. Thus, a loan default predictor is a crucial tool in our effort to identify lower-performing employees. In the following report, we describe the data cleaning, feature engineering, and model selection that went into designing our loan default predictor.

Additionally, because we wish to educate our lower-performing loan agents on how to improve, we must identify what aspects of their portfolio have been flagged. This means that **a key focus of our loan default predictor is interpretability**. First, the features used in the model must have clear meanings in regards to the loans. Secondly, the model must be able to identify which of the features contributed to a given loan being flagged. Thus, as we discuss later in this report, we have focused on interpretable features such as borrower annual income and interpretable models such as logistic regression.

This report describes the research we have conducted to investigate and model which factors describing loans best predict loan defaults. It includes explanations for our dataset selection, data preprocessing, and model selection. Our results indicate that we can predict loan defaults with high accuracy. Additionally, we do so with interpretable features and interpretable models. This allows us to effectively and pro-actively identify which of our new loan agents are lower-performing, and then educate them on what aspects of their approved loans contributed most to the flagged high risk.

Related Work There is existing work regarding loan default prediction for loan applications. However, this problem setting differs substantially from ours. Whereas those models need to predict the propensity for a loan to default before it is approved, our models are designed to predict the propensity for a loan to default after is has been approved. This is because application-time loan default predictors are designed to make loan approval decisions themselves, while our loan default predictor is designed to help identify lower-performing loan agents. Thus the existing models do not consider features of active loans such as *funded_amnt*. Additionally, application-time loan default

Project Report for *ORIE 4741: Learning with Big Messy Data* with Professor Madeleine Udell, Cornell University, Fall 2017.

predictors may sacrifice interpretability for accuracy. That is not acceptable for us, as we prioritize the ability to educate and assist our loan agents. Practically, this means that the **existing work for *application-time* loan default prediction uses stronger, non-interpretable models** such as gradient boosted decision trees with polynomial feature expansion.

2. Dataset

Our dataset consists of features for all loans issued by Lending Club for an eight year span. It is an open sourced dataset which can be found at:

`https://goo.gl/KfxsBY`

This dataset contains information about 890k loans during this period. For each loan, there are 75 raw variables. This includes, most importantly, the current status of the loan (e.g. default, fully paid, etc.) which we use to source the predicted label. Additionally, there are 74 other raw variables which we use to source the predictive features. These include raw variables such as:

- *annual_inc*: The self-reported annual income provided by the borrower during registration.
- *home_ownership*: The home ownership status provided by the borrower during registration obtained from the credit report.
- *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.

This dataset is very well suited for our needs. It is a huge dataset containing descriptions of real approved loans, exactly as we wanted. Though it is messy (as we describe below), it has substantial usable data. Additionally, it has many examples of both loans that do default and loans that do not default. Thus it is ideal for investigating and modeling the factors which best predict whether a loan will default.

3. Data Cleaning

3.1. Missing Data

A major roadblock for using this dataset is that many of the values are missing. The majority of the columns in this dataset have at least some values missing. In fact, for some columns the entries for almost all of the loans are missing. In Figure 1 we plot the amount of data missing in the dataset. For each of the 75 columns, we plot the proportion of loans that are missing data for that column.

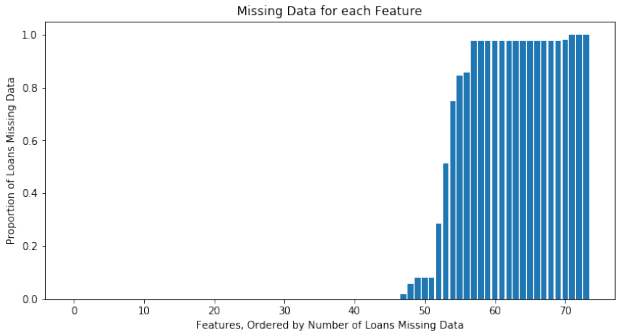

Figure 1. Features in the dataset and the proportion of examples which are missing data. The y-axis represents the proportion of examples which are missing data for a given feature. The features are ordered by the amount of missing data.

From Figure 1 we learn that, even though 40 columns are missing data, only about 20 columns seems to be missing data for a significant proportion of the loans. This is a promising result, as **many columns are still usable, despite missing some data**. We accomplish this by keeping the columns for which 80% of the loans have data. The remaining 20% of the loans simply receive the median value in that column – a standard imputation technique for dealing with missing data.

We note here that we did consider other imputation techniques. Some such alternatives include zero-value imputation, mean-value imputation, and mode-value imputation. These alternatives are not as well suited for our data, however, because they undesirably skew the values of the features. For instance, the column titled *total_acc* has no value below 1.0, so zero-value imputation would be inappropriate, as it assigns a value which otherwise would not exist in the column. Additionally, the column titled *delinq_2yrs* is significantly rightward skewed, so mean-value imputation would assign an abnormally high value to the missing loans. Finally, for columns which are dates it would not make sense to use the mode value, as this would use non-central values which would unnaturally skew the distribution of these columns. In all of these cases, **median-value imputation is the most appropriate technique** for our particular features. The imputed values appear naturally elsewhere in the column, and they are desirably central, so as to avoid skewing the distributions of the columns.

Additionally, we also considered using low rank models and other matrix factorization techniques in order to impute our missing data. These techniques have the desirable quality that they are robust to bias in the samples and are able to assign different values to different loans during imputation. However, these techniques are not appropriate for our dataset because they would compromise the interpretability

of our one-hot encoded categorical variables. As we will discuss below, many of our features needed to be transformed into groups of binary indicator variables. With matrix factorization, our low-rank representation of the data matrix would allow these binary variables to take on values which are not 0 or 1. Even worse, the indicator variables could, for instance, indicate that a given loan has a term length of both 36 months and 60 months. Such a representation has no interpretable value. Thus, we realize that even though they are powerful tools for dealing with missing data, **matrix factorization and low rank models are not appropriate for our use-case** where we demand interpretable one-hot encodings for our categorical variables.

3.2. Mistyped Data

Another issue with our raw dataset is that many of our features are dirty. This means that the values are corrupted and mis-typed by our dataframes package, Pandas, in Python. Most often, a column which should contain a categorical variable instead is typed as a column of arbitrary strings. As an example, the column *term* contains strings such as '` 36 months`' and '` 60 months`'. This is undesirable for a number of reasons, including the fact that the strings have random leading whitespace.

In order to deal with the dirty features, we had to conduct manual audits of the different levels in these features. This entailed printing out every unique value in these columns and sifting through them to see which ones lined up with which. With the levels determined, we proceeded to transform these string columns to categorical columns, with different values representing the different levels the feature could take on. Doing this, we successfully **re-typed ten of our features from string-type to categorical-type**.

3.3. Dirty Labels

The next major hurdle for using this dataset is that the column we intend to use as a label, *loan_status*, is not a simple indicator variable for defaulted or not. Instead, the loan status can take on extraneous values (such as '`In Grace Period`') which do not necessarily indicate whether the loan will or will not default. In Figure 2 we graph the proportion of loans corresponding to each of the raw levels for loan status.

From Figure 2 it becomes immediately clear that approximately 0% of the loans have the label '`Default`'. This would be a huge concern for us; if we did not have any examples of defaulting loans, it would be massively difficult to predict which loans would default. Thankfully, once alerted to this issue, we learned that '`Charged Off`' is a stage strictly after '`Default`'. Thus we consolidated the two label values into a unified defaulted value.

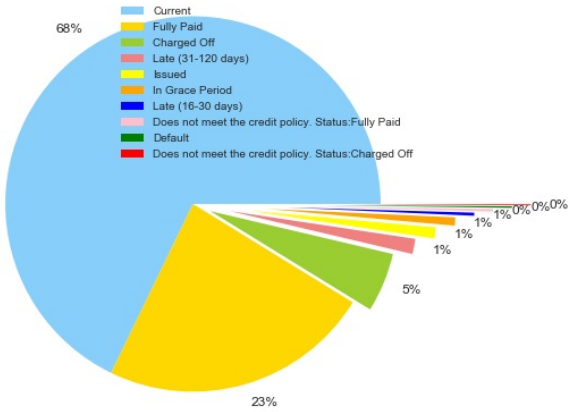

Figure 2. Proportion of the dataset belonging to each raw type of loan status. Each pie slice represents a unique level in the dataset column titled *loan_status*.

Additionally, we note that the majority of loans in the dataset have a loan status of '`Current`' meaning that they are still in the process of being paid off. This label, allowing with the other '`Current`'-esque labels, do not tell us definitively whether a given loan results in a default. Without a definitive label for these examples, it becomes impossible to train a model on these values. We did consider using semi-supervised approaches to include this subset of our data, such as assigning soft-labels to these examples. However, these approaches would force us to train on artificial "true" labels – potentially mis-characterizing a good loan as a bad one and vice-versa. This potential corruption of our labels is unacceptable to us, so in our analyses **we drop the loans which have these intermediate statuses**.

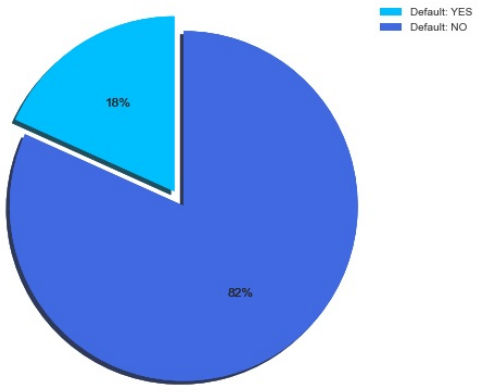

Figure 3. Proportion of the dataset belonging to each processed type of loan status. The two possible values are '`YES`' the loan defaulted or '`NO`' it did not.

Altogether, **our cleaned labels are 'YES' and 'NO'**. These two labels represent whether a given loan definitively resulted in a default (i.e. 'Default' or 'Charged Off') or not (i.e. 'Fully Paid'), respectively. This cleaned dataset still has 250k example loans, which is sufficiently large to build our models with. Even better, we note that the class-imbalance of default-to-nondefault loans is just 18:82, which is illustrated in Figure 3. This is promising because we have plenty of examples from each class of loans, and thus we can effectively train to predict loan defaults.

4. Feature Engineering

In this section we describe the steps of feature engineering we implemented, which ones we considered, and which ones we decided to use. Though we considered many different techniques for feature engineering, we decided to remain principled in which techniques we eventually chose. In particular, **we decided to focus on engineering features which are interpretable**. That is, knowing that a particular feature changes should tell us that some meaningful quality of the loan changes. Interpretable features include *loan_amnt* which represents the dollar amount requested by the borrower. In contrast, a non-interpretable feature is one which is not related in a meaningful or direct way to the characteristics of the loan. Non-interpretable features include *loan_amnt * annual_inc* which represents the product of the loan amount with the annual income of the borrower. In this second case, an increase in the feature does not specifically indicate increased loan amount nor increased annual income.

4.1. Categorical Variables

A significant part of our feature engineering involves dealing with categorical variables. With these types of variables, it is difficult to attribute a numerical value to each level of the variable. Thus instead of encoding our categorical variables as numerical values, **we one-hot encoded our categorical variables**. That is, we split each categorical variable into indicator variables representing each level. For example, instead of having a single variable called *verification_status*, we included *verification_status_Verified*, *verification_status_Source_Verified*, and *verification_status_Not_Verified*. For a given loan, only the indicator variable corresponding to the actual value of the raw feature takes on the value 1, while the other indicator variables take on value 0. The variables we needed to encode are *home_ownership*, *verification_status*, *purpose*, *term*, *initial_list_status*, and *application_type*.

One-hot encoding is a desirable encoding for our categorical variables because the resultant features are interpretable. If a model states that a given loan is flagged because of *verification_status_Not_Verified*, we immediately understand that to mean the loan was flagged because the borrower income was not verified. In contrast, another encoding we considered, numerical encoding, is not interpretable. In this encoding, each level of the categorical variable is assigned an arbitrary numerical value. If a model states that a given loan would be flagged if the feature were to increase, we would have no way to interpret that increase. This improved interpretability is why we decided to use one-hot encoding.

4.2. Ordinal Variables

One of our features, *emp_length*, is ordinal. The unique values this variable can take on are illustrated in Figure 4, along with their associated prevalence in the dataset.

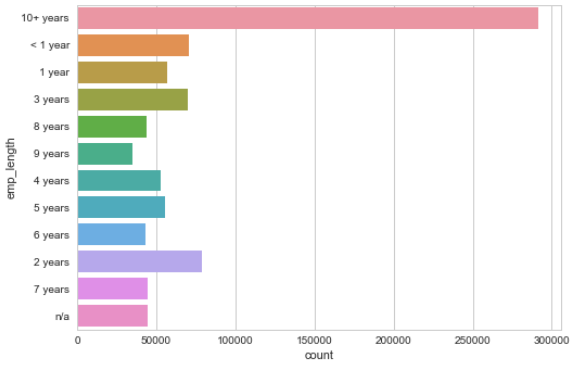

Figure 4. The frequency with which the values corresponding to *emp_length* occur in our data set. '10+ years' has a clear majority, as it represents the largest set of values.

Noticeably, most of the levels have a clear fixed distance between them. This means, for instance, that the distance between '2 years' and '3 years' is half the distance between '5 years' and '7 years'. This motivates a numerical encoding in which each level of this variable is simply assigned the number corresponding to the number of years. This encoding has the desirable property that, if the model flags any loans with borrower employment length shorter than 3 years, it can do so with a single inequality.

However, '< 1 year', '10+ years', and 'n/a' are indefinitely far from the others. This means that even though '9 years' may be one greater than '8 years', it is not clear how much greater '10+ years' is. Especially for '10+ years', which consists of most of our dataset, this may be a concern. Because of this, an appropriate encoding may be one-hot encoding, similar to how we encoded our categorical variables. This encoding would allow our models the flexibility to characterize the effect of changing from '8 years' to '9 years' to be different than the effect of changing from '9 years' to '10+ years'.

Due to the ambiguity of which encoding is most appropriate, **we use both numerical encoding and one-hot encoding for our ordinal variable**. Because we planned to do feature selection, as described below, we were not concerned that this may add too many features, or that this may add correlated features. In fact, by using both encodings, we allow our feature selector the flexibility to leverage whichever features are more useful, improving our overall model strength. Finally, this dual encoding maintains interpretability because the transformed features each still have a clear meaning for the loans they describe.

4.3. Feature Expansion

One important type of feature engineering that we implemented – but decided not to use – is feature expansion. This is otherwise known as kernel methods. This is where the points are mapped from the data space into a higher dimensional space using some mapping function. Alternatively, the examples are implicitly mapped to and operated upon in this space using kernel functions. Either way, the model ends up operating upon higher dimensional representations of our loan vectors.

The kernel that we implemented is a full second order polynomial kernel. This kernel multiplies the existing features by each other, in every combination. This process produces d^2 features, where d is the number of first order features. The benefit of using such an expansion is that the model can then easily capture the effects of second order interactions. For example, perhaps *loan_amnt* and *annual_inc* have limited correlation with the target variable, but together *loan_amnt * annual_inc* is strongly correlated with the target variable.

In the end, **we do not use polynomial feature expansion** in our final feature engineering pipeline. The reason is twofold. First, a preliminary correlation analysis showed that the second order features did not have especially improved correlation with the default label variable, when compared to first order features. Additionally, and perhaps more importantly, these second order features are non-interpretable. Consider for example the feature *loan_amnt * annual_inc* mentioned above. An increase in the feature does not specifically indicate increased loan amount nor increased annual income. Because such features do not provide obvious actionable insights for our lower-performing loan agents, we do not include them in our final analyses.

5. Feature Selection

In the previous sections we described how we cleaned, parsed, transformed, and generated our candidate features. Because we have such a strong focus on interpretability, we wish to whittle down this set, in order to find a smaller set of highly-predictive features. Though this would inherently reduce model strength, we believe a crucial characteristic of interpretable models is a small set of features upon which the model makes each decision. This way, a user can easily determine which factors lead most directly to the classification. For us, this means that our lower-performing loan officers can know precisely why our loan default predictor would flag their approved loans, and they can learn to improve from this experience.

5.1. Domain Knowledge

The first step we took for feature selection was to only select the features which would be available when we apply our model to real loans at Lenders "R" Us. Because our goal is to predict which loans in our loan agents' portfolios will default, our model cannot use features which are only available after a loan as defaulted or been fully paid. This includes features such as *acc_now_delinq*, which itself indicates whether loan payments have been missed. Additionally, we omitted other features which are specific to Lending Club loans and datasets. This includes features such as *grade* which describes Lending Club's own assessment of the quality of a loan. We note here that these features are, naturally, highly correlated with whether the loan defaulted. However, we abstain from utilizing these features because any model which uses them could not be applied to Lenders "R" Us data. **We voluntarily restrict ourselves to features which both are available for current loans and are not specific to Lending Club.**

5.2. ℓ_1-regularized Logistic Regression

The next step we took for feature selection was to **apply a ℓ_1-regularized logistic regression to our training dataset, in order to select the most important features**. By using an ℓ_1-regularizer on our logistic model, the weights for the less important features are pushed towards zero – effectively selecting them out. This is very similar to using LASSO (ℓ_1-regularized ℓ_2 loss function) for feature selection in problem settings where the predicted value is continuous. Instead, in this problem setting, the predicted value is binary, so we set our loss function to be the logistic loss function. Additionally, because our dataset is class-imbalanced, we upweight the defaulted loans so that the two classes contribute equally to the loss, as is standard for imbalanced datasets.

One critical hyperparameter of a regularized generalized linear model is the tuning parameter. This hyperparameter adjusts the relative importance of the regularizer versus the loss function. Typically this is called λ, and as it increases the relative importance of the regularizer increases. However, in the machine learning toolkit scikit-learn for Python, the tuning parameter is called c, and as it increases the relative importance of the regularizer decreases. It is not a priori

evident what value of c is most appropriate for our model. If we set c to too large of a value, no features will be selected out; if we set c to be too small, every feature weight will be set to zero.

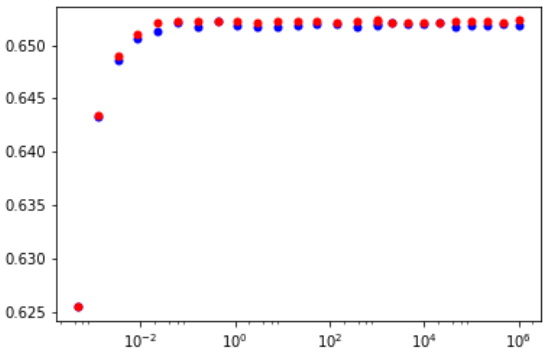

Figure 5. The 5-fold cross-validated accuracy of the ℓ_1-regularized logistic regression, at different settings of tuning parameter c. Each red point represents the average training error, and each blue point represents the average validation error.

In order to resolve this ambiguity, **we apply cross-validation to tune the hyperparameter** c in our ℓ_1-regularized logistic regression. In this process, we train our model on various settings of the hyperparameter and evaluate how it performs on our validation sets. Critically, for each setting of the hyperparameter, we run multiple experiments with different splits set as the validation set. In particular, we split our approximately 200k-example training dataset (which has already been split from the testing dataset) into five equally sized splits of approximately 40k examples. For each setting of the hyperparameter, we evaluate the accuracy on each of the five splits when trained on all the others, and we average the accuracy across splits. Overall this provides us an estimate of the out-of-sample performance of our model, at each of the settings of c. In Figure 5 we show how the accuracy varies with c.

From Figure 5 we see that the validation score remains relatively flat as we remove features, up until c is set to 0.02336. As this is the lowest setting of c which still maintains strong validation performance, this is the setting of c we use in our final feature selector. This final feature selector is trained on the entire training dataset, as is standard once the hyperparameter has been tuned via cross-validation. Using this final ℓ_1-regularized logistic model, we observe which weights are set to 0.0 and drop those features. In the end, the weights that our feature selected learned are shown in Figure 6.

0.2485	loan_amnt	0.0126	purpose_moving
0.1053	funded_amnt	0.0256	purpose_other
-0.3735	funded_amnt_inv	0.0049	purpose_renewable_energy
0.4454	int_rate	0.0906	purpose_small_business
0.1097	installment	0.0006	purpose_vacation
-0.2558	annual_inc	-0.0200	purpose_wedding
0.1835	dti	0.0000	initial_list_status_f
0.0383	delinq_2yrs	-0.0259	initial_list_status_w
0.0862	inq_last_6mths	0.0000	application_type_INDIVIDUAL
0.0951	open_acc	-0.0073	application_type_JOINT
-0.0004	pub_rec	0.0	emp_length_num
-0.0240	revol_bal	0.0940	emp_length_0.0
0.1090	revol_util	0.0083	emp_length_0.5
-0.1634	total_acc	-0.0071	emp_length_1.0
-0.0214	verification_status_Not Verified	-0.0160	emp_length_2.0
0.0	verification_status_Source Verified	-0.0057	emp_length_3.0
-0.0270	verification_status_Verified	-0.0116	emp_length_4.0
-0.0034	home_ownership_ANY	-0.0073	emp_length_5.0
-0.1045	home_ownership_MORTGAGE	0.0005	emp_length_6.0
-0.0035	home_ownership_NONE	0.0002	emp_length_7.0
0.0004	home_ownership_OTHER	0.0	emp_length_8.0
-0.0387	home_ownership_OWN	-0.0003	emp_length_9.0
0.0	home_ownership_RENT	-0.0181	emp_length_10.0
-0.1597	term_36 months	0.0	zip1_0
0.0370	term_60 months	0.0071	zip1_1
-0.0086	purpose_car	-0.0148	zip1_2
-0.0240	purpose_credit_card	0.0266	zip1_3
0.0	purpose_debt_consolidation	0.0041	zip1_4
0.0081	purpose_educational	-0.0137	zip1_5
0.0167	purpose_home_improvement	-0.0101	zip1_6
-0.0067	purpose_house	-0.0335	zip1_7
-0.0070	purpose_major_purchase	-0.0220	zip1_8
0.0211	purpose_medical	-0.0451	zip1_9

Figure 6. The feature weights from the final feature selector. These are the parameters learned by an ℓ_1-regularized logistic regression, trained on the entire training dataset, with tuning parameter set to $c = 0.02336$.

6. Data Modeling

In this section we discuss three of the model classes we used to model our data. These are ℓ_2-regularized logistic regression, support vector machines, and random forest regression. All of these are well suited for binary classification, which is the problem setting in which our problem lies. As we see, however, the performance differs noticeably between the three models. This is true even though we tune the hyperparameters for these models. We also note that the final model from each model class is the only model which is run on our held-out test set of 50k examples. This maintains the generalization properties of our results, as we do not greatly overfit to our test set.

6.1. ℓ_2-regularized Logistic Regression

The first model class we apply is ℓ_2-regularized logistic regression. Though this model class is similar in form to the model we used in our feature selection, it is significantly different in substance. Whereas ℓ_1-regularized logistic regression learns sparse weight vectors because of the Poisson prior distribution it imposes on the parameters, the ℓ_2-regularized logistic regression learns dense weight vectors because of the Gaussian prior distribution it imposes.

Similar to the process we took to tune the ℓ_1-regularized logistic regression, here we use 5-fold cross-validation to set the tuning parameter for our ℓ_2-regularized logistic regression. This parameter adjusts the relative importance of the regularizer versus the loss function, and a higher value indicates lower relative importance of the regularizer. In Figure 7 we graph the relationship between the tuning parameter and the validation score.

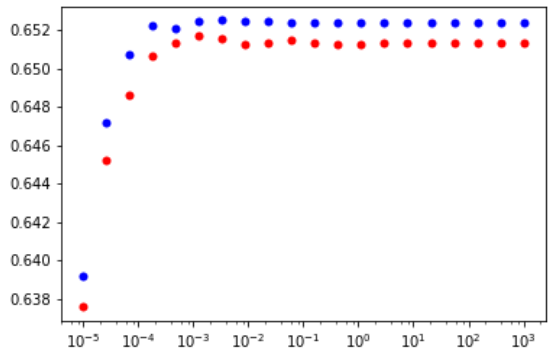

Figure 7. The 5-fold cross-validated accuracy of the ℓ_2-regularized logistic regression, at different settings of tuning parameter c. Each red point represents the average training error, and each blue point represents the average validation error.

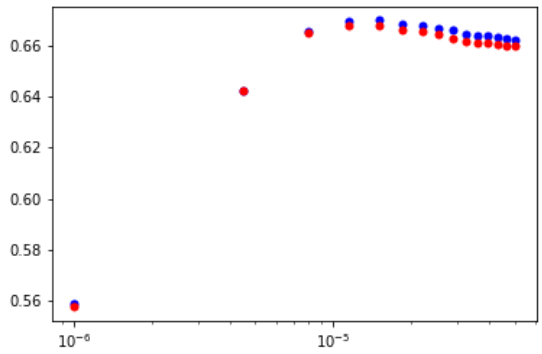

Figure 8. The 5-fold cross-validated accuracy of the support vector machine, at different settings of the tuning parameter c. Each red point represents the average training error, and each blue point represents the average validation error.

From Figure 7 we see that the setting of c for which the validation performance is highest is $c = 0.00127$. We use this as our setting for our final ℓ_2-regularized logistic model. We test this final model on our held-out test set and get a **test accuracy of 65.39%**.

6.2. Support Vector Machine

The second model class we apply is support vector machines. This model class differs from ℓ_2-regularized regression by its loss function. Whereas logistic regression models use logistic loss, support vector machines use hinge-loss.

Support vector machines, too, have an important tuning parameter c. The hyperparameter which most affects model strength is the depth of the trees in the forest. This parameter adjusts the relative importance of the regularizer versus the loss function, and a higher value indicates lower relative importance of the regularizer. In Figure 8 we graph the relationship between the tuning parameter and the validation score.

From Figure 8 we see that the setting of c for which the validation performance is highest is $c = 1.15e - 05$. We use this as our setting for our final support vector machine. We test this final model on our held-out test set and get a **test accuracy of 66.96%**. This is a slight improvement on the performance of the ℓ_2-regularized logistic regression.

6.3. Random Forest Regression

The third model class we apply is random forest regression. This model class is noticeably more expressive than the previous two because it can model nonlinear decision boundaries. Whereas linear models are limited to be linear in the weights, random forest models may be nonlinear in their weights.

Random forests, too, have important hyperparameters to tune. These include tree depth, splitting rule, and number of trees. The hyperparameter which most affects model strength is the depth of the trees in the forest. With deeper and deeper trees, the decision boundary may become more and more nonlinear. We use 5-fold cross-validation to set the tree depth parameter for our random forest regression. In Figure 9 we graph the relationship between the tuning parameter and the validation score.

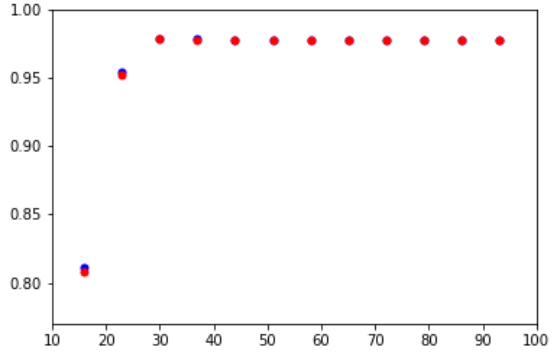

Figure 9. The 5-fold cross-validated accuracy of the random forest regression, at different settings of the tree depth parameter. Each red point represents the average training error, and each blue point represents the average validation error.

From Figure 9 we see that the setting of tree depth for which

the validation performance is highest is 30. We use this as our setting for our final random forest model. We test this final model on our held-out test set and get a **test accuracy of 80.52%**. This is our best test accuracy by far!

7. Conclusion

Our goal is to identify and educate lower-performing loan agents here at Lenders "R" Us. In order to do this in a principled manner, we have developed an automated loan default predictor which effectively predicts whether the loans our loan agents have approved will default. Additionally, we have actively worked to make this model interpretable, so as to clearly identify why a given loan has been flagged.

Our results on the test set indicate that our model achieves good performance accuracy. We are happy to say that our model can predict loan defaults with 81% accuracy. Though this performance seems modest, we note that this is the accuracy when predicting whether a single given loan will default. Given an entire portfolio of loans, the variance of the prediction would decrease, and we would get a strong sense of whether the loans a given loan agent has approved are risky. Thus, for our intended problem of identifying lower-performing loan agents, we believe our loan default predictor is a good tool. **We would be willing to use this model in production**, with the natural caveat that it is applied to entire portfolios, rather than single loans.

Additionally, we conclude that we are very confident in our results. The dataset we use is aligned very well with our needs, the features we use are explicitly selected to not be specific to Lending Club, the results we report are on a held-out test set, and we only test three models on the test set. We made all of these design decisions with generalization in mind. Altogether, they lead us to conclude that our loan default predictor will achieve similar performance on real Lenders "R" Us loans.

We explicitly do not, however, claim that our model is perfect. In fact, we believe there are many aspects that can be improved or modified in future research. In particular we make the following suggestions: First, consider using model-specific feature selectors, where a random forest model will have a different upstream feature selector than a logistic regression. Second, consider using stronger, less interpretable models if our goal changes to application-time loan default prediction. Third, use group lasso to include or exclude entire sets of indicator features, so that the indicators for a given categorical variable are never separated. With this model and possible future improvements, we can lead Lenders "R" Us into a new era of success.