## Summary

The machine learning part uses a number of different classifiers – logistic regression, SVMs with RBF and polynomial kernels, random forests, and gradient boosting. After analyzing the results of each individual model, I combine all models into an ensemble using a hard voting classifier. Since some of these models became computationally expensive (could not be trained within 1-2 days on my laptop), I pursued several strategies to make estimating these models feasible: Firstly, I used Bayesian optimization to tune the hyperparameters, which takes less iterations than a brute-force grid- or randomized search. Secondly, I ran the more computationally intensive models on AWS EC2 to take advantage of more powerful hardware. Thirdly, I used a single validation set rather than cross-validation where necessary. If this still wasn't enough, I reduced the dimensionality of the features space using principal components analysis, and used only a subset of observations.

## The challenge of how to properly split the data

The first challenge was to decide how to split the data into training and test set. While the data do have a time component, I chose to ignore the time dimension for two primary reasons: Firstly, time-series models are substantially more complex, so I leave this to further iterations of the project. Secondly, if we treat the data as dependent it is not clear if it would be possible to construct valid test and validation sets that come from across the business cycle (see Notebook 2a for more details).

## Hyperparameter optimization

I initially started out using grid search and randomized search, but these brute-force approaches turned out to be too computationally expensive for the size of my data. Thus, I turned to Bayesian hyperparameters optimization, which performs an intelligent search that learns from the performance of hyperparameter combinations tried previously.

For the computationally less expensive models (logistic regression, random forests, and XGBoost), the performance of a given set of hyperparameters will be evaluated using 5-fold cross-validation to make the most efficient use of the available data. However, since I already have to reduce the sample size considerably for the SVMs with RBF and polynomial kernel, I use a single validation set instead. Compared to 5-fold cross-validation, this cuts computational costs by a factor of 5.

The metric I optimize is the average precision score on the (cross-)validation set. This measure is similar to the commonly used metric of the area under the ROC curve, but it is better suited to unbalanced classification problems such as ours. The precise meaning of the average precision score is less straightforward, but it is roughly equivalent to the area under the precision-recall curve. Another common metric when using precision and recall is the F1-score (which corresponds to the harmonic mean between precision and recall). Its disadvantage is that it is based on a particular point on the precision-recall curve (namely the point we get with scikit-learn's default probability threshold of 0.5). By contrast, the average precision is based on the entire precision-recall curve, and thus take into account a classifier's performance for different trade-offs between precision and recall.

**Predictive modeling**

Note that the primary business objective of this project is accurate prediction. Thus, we do not care as much about identifying which variables are most important, or even about the precise effect of particular variables (which falls under the domain of statistical *inference*). Even though a black-box model that does not return a specific reason justifying its prediction may feel less convincing to humans, it is not worth sacrificing predictive accuracy for this.

My strategy is to train an instance of each of the major classification models, unless there were any contrary indications. For example, I did not use deep learning because it is complex to train, so it would have been a whole project on its own. The models I used to include logistic regression with elastic net regularization, SVMs with RBF and polynomial kernel, random forests, and XGBoost. The latter achieves by far the best performance (average precision on the test set of 0.41). All the other models achieved a score around 0.39.

The starting point is a linear model, namely logistic regression with elastic net regularization. Because the training set has about 800,000 observations and 130 features, I used stochastic gradient descent to substantially cut down the computational cost.

The next model is a random forests classifier. Its strength is that it is able to model nonlinear relationships and interactions between predictors. For both models, I get a similar average precision score of 0.39.

I continue with another tree-based ensemble, XGBoost. a particularly good implementation of gradient boosting. It should not be too surprising that it achieves by far the best performance of all models (0.41).

The remaining two models – SVMs with RBF and polynomial kernel, were by far the most were computationally expensive, so I trained them in a separate notebook run on AWS EC2. As mentioned in the beginning, I still had to take additional steps to make the computation feasible. In particular, I reduced the dimensionality of the features space from 130 to 30 by extracting principal components. Since the first 30 principal components explain about 90% of the variance in the features, this seemed like a promising strategy. I still had to reduce the training size to 80,000 observations. Unfortunately, the model performs slightly worse as logistic regression and random forests. I thus tried using the original features instead, which forced me to further cut the training size to 40,000. This yielded a similar but slightly better performance than all other models except XGBoost. Finally, I trained a SVM with polynomial kernel, which performed similar to logistic regression and random forests. Here, I only tried the original features, since these had performed better than the principal components for the RBF kernel.

**Ensemble**

The final step is to combine all these different models into an ensemble in order to take advantage of the insights from the different learners. Ensembles have been shown to be a good way of decreasing both bias and variance of predictions without overfitting. There are different ways of creating ensembles. The one I choose is a **hard voting classifier**. This strategy simply takes a majority vote of the hard predictions (class labels) from the individual models.

The disadvantage of this strategy is that we lose some information by converting probabilistic predictions to binary class labels before the vote. It would thus be better if we could simply average the probabilities (soft voting classifier). However, the problem is that SVMs return the distance from the separating hyperplane rather than probabilities. While it is possible to estimate probabilities from this, this is computationally too expensive .

One important decision we need to make is how to convert the soft predictions into hard predictions. By default, scikit-learn will generate hard predictions by using a threshold of 0.5 for probabilities and 0 for distance to the seperating hyperplane. In other words, if a borrower's default risk is greater than 50%, we'll predict that they will default. However, it may make sense to deviate from this threshold if the cost of false positives and false negatives is asymmetrical.

In our case, we probably want to err on the side of caution, because the cost of one default is much higher than the income lost from making one fewer loan. How to precisely weigh these two different costs is a much more complicated issue than we can solve here, in part because we do not have knowledge about all the costs associated with each. This decision would also be easier if we were able to model the net present value of each loan rather than a simple binary prediction of default or no default, because the cost of default depends in large part on how much a borrower pays back before the default. Unfortunately though, as explained in more detail in the proposal, we did not have the necessary data to compute this more appropriate target variable. Thus, a justifiable choice is a recall of 0.5, which allows us to attain a precision of about 0.35. This means that we are willing to turn away (a little under) two applicants who would not default in order to successfully identify one applicant who will default. For models returning probabilities, this corresponded to turning away applicants whose default risk exceeded between 0.5 and 0.65, depending on the model.

The disadvantage of this strategy is that we lose some information by converting probabilistic predictions to binary class labels before the vote. It would thus be better if we could simply average the probabilities (soft voting classifier). However, the problem is that SVMs return the distance from the separating hyperplane rather than probabilities. While it is possible to estimate probabilities from this, this is computationally too expensive .

I find that the ensemble performs very similar to XGBoost. Both achieve the same F1-scores for identifying defaults and non-defaults. When we take a more detailed look, we see that XGBoost performs slightly better on some measures (recall on non-default). Therefore, we would probably go with its predictions for making loan decisions for now.