1. ABSTRACT

Home Credit wants to broaden its business by lending to customers not served by traditional banks. We built a risk prediction model for lending that can use alternative data, including telco, housing, and transactional information. This task involved several steps:

* imputing missing values
* engineering features
* building logging functions for scores and model serialization from test runs
* building pipelines to ensure repeatable processes
* running experiments iteratively
  + seeking optimal hyper-parameters for prediction models
  + fitting model parameters
  + verifying model scores against a holdout test set
* comparing scores and qualities to identify the optimal model
* presenting findings

The project has enabled profitable lending to customers who are currently underserved.

2. INTRODUCTION

Home Credit has supplied a set of training data related to approximately 300,000 customer loans. The data set predictors include loan application fields, credit bureau data, fields from previous applications, credit card history, and balance and payment history from previous Home Credit loans. The target binary variable indicates whether the customer defaulted on the loan. The availability of the target variable allows this project to use supervised training to fit a model.

Home Credit has also supplied a set of test data and a mechanism for calculating scores by submitting probabilistic predictions to a competition website hosted by Kaggle. Submissions are scored by the receiver operating characteristic area under the curve (AUC) metric.

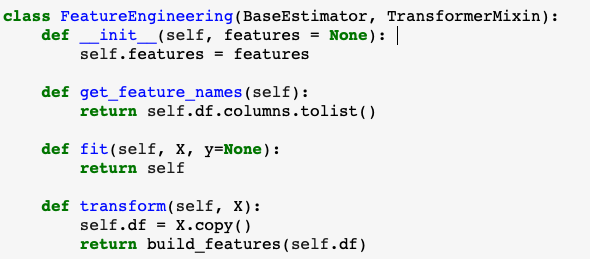
Predicting whether an applicant will default on a loan is a classification task. Therefore our project team has run a series of experiments to identify an optimal classification model. Candidate model algorithms have included logistic regression, random forest, and several varieties of gradient boosting.

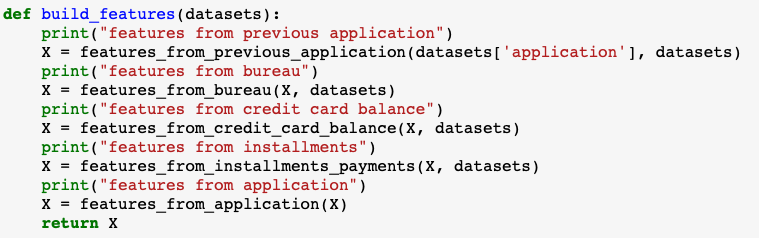
Phases of the project have included:

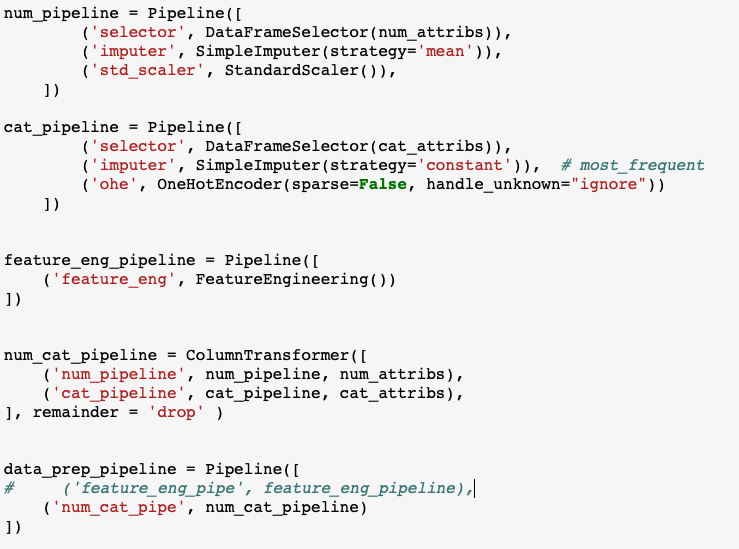
* Phase 0: project planning, exploratory data analysis, and baseline model
* Phase 1: feature engineering, benchmark model, and pipelines for pre-processing
* Phase 2: logging functionality, model experimentation, score comparison
* Phase 3: exploration of two additional classification (SVM and neural networks) and delivery of a report, visualizations, code, an executable pipeline and an executable, optimized classification model. The availability of executable artifacts is assured by the use of a bespoke logging function that stores metrics, pipeline parameters, and the serialized best estimator from each experimental run.

3. Feature Engineering and transformers

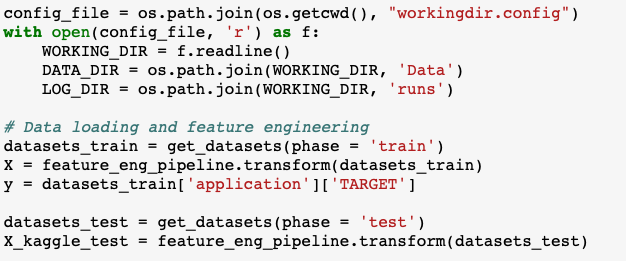
In phase II of our project, we developed and optimized a comprehensive analytical pipeline of data preprocessing. Specifically, we created a FeatureEngineering class and a series of functions that integrate feature engineering, data cleaning & standardization, transformation, feature extraction & selection all within the framework of pipelines.







Below shows an application of the feature engineer pipeline on raw datasets to create and select new features, where the whole feature engineering process was conducted in a featuren\_eng\_pipeline pipeline.



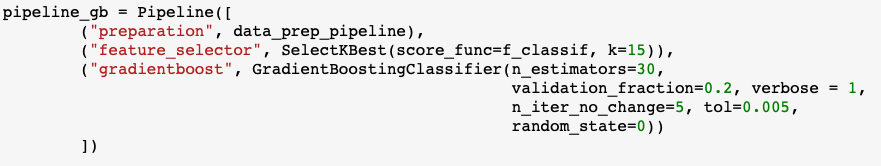
4. Pipelines

A number of pipelines have been created in this project for various purposes including data preprocessing, feature selection, dimensionality reduction and the use of logistic regression and ensemble-based classifiers.

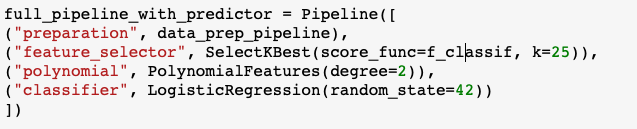
For example, we used ('pca', PCA()) immediately before the logistic regression to help reduce data dimensionality.



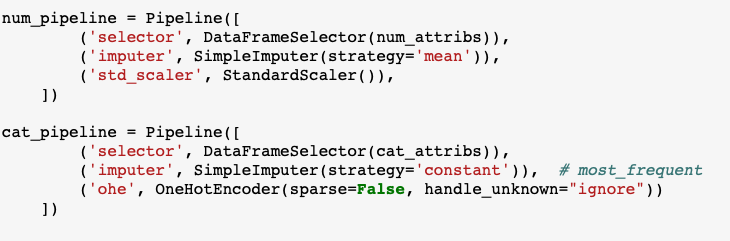
Pipeline ("feature\_selector", SelectKBest(score\_func=f\_classif, k=15)) was used for selecting k best features based on ANOVA F-value computed by `f\_classif`.



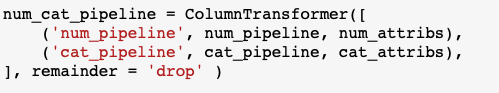
Pipeline ("polynomial", PolynomialFeatures(degree=2)) was used for creating new feature matrix of higher-degree polynomials.



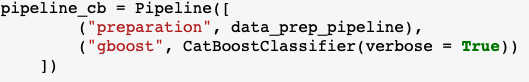
Pipelines num\_pipeline and cat\_pipeline were used for numerical and categorical data pre-processing including imputation, standardization and/or one hot encoding.

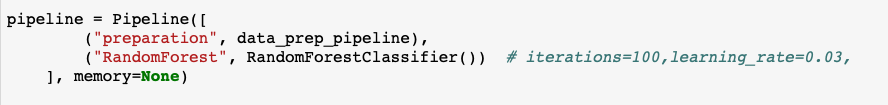
****

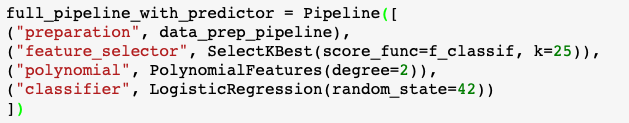
A ColumnTransformer pipeline was used to combine existing pipelines to create new ones with pre-specified feature columns.

****

Lastly, a series of pipelines that encompass various logistic regression and ensemble-based classifiers have been created for building predictive classification models.







5. Experimental results

We tried the following things in Phase 3:

1. SVM

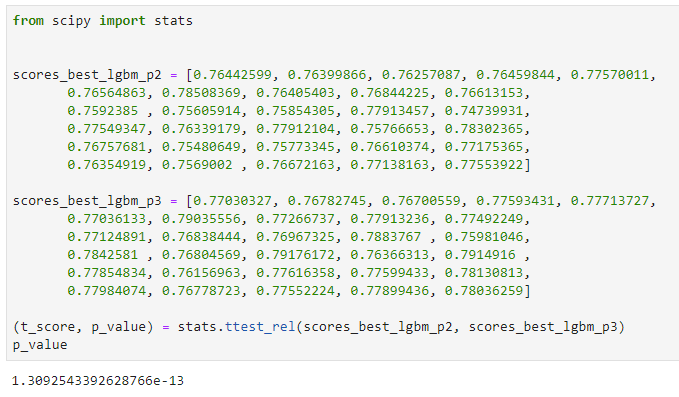
2. Neural Network

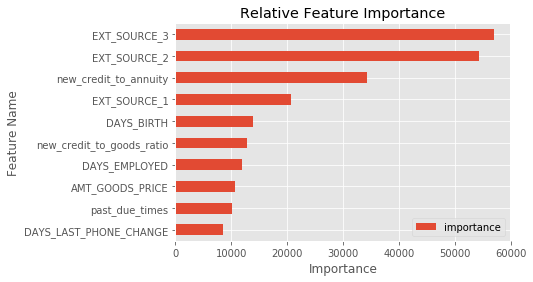
3. LightGBM

The best scores from all project phases are as follows:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  | Experiment | Features | Train AUC | Valid AUC | Test AUC |  |
|  | Baseline | 14 | 0.7359 | 0.7361 | 0.7362 |  |
|  | Benchmark logit | 29 | 0.7504 | 0.7474 | 0.7488 |  |
|  | Best logit | 131 | 0.7543 | 0.7547 | 0.7542 |  |
|  | Best random forest | 29 | 0.8102 | 0.7418 | 0.7415 |  |
|  | Best sklearn GradientBoost | 29 | 0.7232 | 0.7205 | 0.7157 |  |
|  | Best catboost | 29 | 0.7752 | 0.7539 | 0.753 |  |
|  | Best SVM | 140 | 0.6383 | 0.6373 | 0.6354 |  |
|  | Best NN | 131 |  | 0.752 | 0.755 |  |
|  | Best LightGBM | 140 | 0.8614 | 0.7783 | 0.7772 |  |
|  |  |  |  |  |  |  |

We also performed a t-test to compare the scores from the 30-fold cross-validation results from best LightGBM from Phase 2 and best LightGBM model from Phase 3 with additional new features and get a very small p\_value, indicating that the LightGBM does perform better. (We have shown in Phase 1 report that the best logit model performs better than the baseline model).



Feature importance in Best LightGBM model

## 6. Deep learning model

Our experimentation with feed-forward neural networks comprised several stages:

**1. Setup/Configuration of tensorflow-gpu and keras**

This was surprisingly difficult:

* Much of the documentation is out of date, or assumes you want to \*build\* the components rather than just install them, or omits key information.
* Component version dependencies are unclear. Moreover, backwards compatibility is not preserved in the latest CUDA runtime, which means that you cannot simply install the latest versions of the various components.
* Runtime error messages are suppressed or unclear.
* The CUDA runtime installers display messages stating that they are updating the NVidia graphics driver. These messages are misleading; the previous driver is not replaced.

**2. Baseline Model** - We built a simple feed-forward neural network with hidden layers 128-128 and rectified linear unit (RELU) activation. We used the keras.wrappers.scikit\_learn.KerasClassifier class to facilitate grid searches, and keras.callbacks.EarlyStopping in order to minimize training time. Below is a code excerpt from our baseline model testing.

def get\_keras\_model(architecture = [298, 128, 128],

activation\_fn = 'relu',

drop\_fraction = None):

model = Sequential()

# LAYERS

# input

model.add(Dense(architecture[1], input\_shape = (architecture[0],), activation = activation\_fn))

if drop\_fraction:

model.add(Dropout(drop\_fraction))

# features/weights

for i in range(2, len(architecture)):

model.add(Dense(architecture[i], activation = activation\_fn))

if drop\_fraction:

model.add(Dropout(drop\_fraction))

# output

model.add(Dense(1, activation = 'sigmoid'))

# other hyper-parameters

opt = optimizers.Adadelta()

loss\_fn = losses.binary\_crossentropy

mets = [metrics.binary\_accuracy]

# compile and return

model.compile(optimizer = opt, loss = loss\_fn, metrics = mets)

return model

early\_stopping = EarlyStopping(patience = 1, restore\_best\_weights = True)

clf = KerasClassifier(build\_fn = get\_keras\_model,

batch\_size = 10240,

validation\_split = 0.1,

epochs = 20,

verbose = 2)

search\_grid = {'drop\_fraction':[None, 0.25]}

gs = GridSearchCV(

estimator = clf,

param\_grid = search\_grid,

scoring = 'roc\_auc',

cv = 3,

verbose=10,

fit\_params = {'callbacks': [early\_stopping]}

)

Dropout regularization turned out to be not useful for this model.

**3. Exploration of Wide Neural Networks** - Architectures of 256-256, 512-512, and 1024-1024 were tested. Dropout regularization was not useful for the best architecture, 256-256. Validation and test scores were superior to the baseline model.

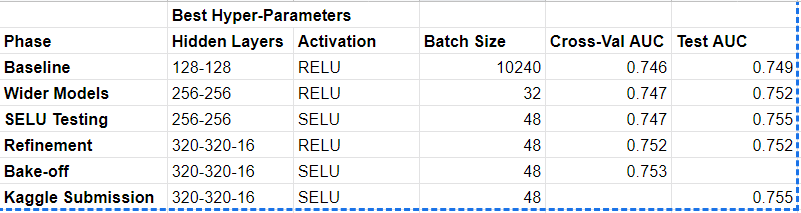
**4. Exploration of SELU activation** - Klambauer, et al. proposed ["self-normalizing neural networks"](https://arxiv.org/abs/1706.02515) that use scaled exponential linear unit (SELU) activation. This activation makes the weight vectors converge toward zero mean and unit variance, which allows training to explore the parameter space without encountering exploding or disappearing gradients. Consequently, feed-forward networks can in theory explore a larger parameter space for more epochs. Architectures of 256-256, 512-512, 256-256-128, and 512-512-256 were tested. A hidden layer architecture of 256-256 was nevertheless still the top performer, and slightly better than similar model with RELU activation.

**5. Exploration of architectures in vicinity of best so far** - Having identified a very approximate best architecture (256-256) and regularization (none), we conducted a search over more similar architectures (320-320, 192-192, 320-320-16, 256-256-16, and 192-192-16). RELU activation was used in order to minimize training time. The best hidden architecture was 320-320-16.

**6. Selection of best activation function** - We conducted a run-off between RELU and SELU activations with a hidden layer architecture of 320-320-16. RELU activation was infinitesimally better in validation AUC, but SELU activation was modestly better in test AUC.

Since test verification is performed with models trained on the entirety of the train split, a plausible explanation of the result is that SELU activation works better with more data. It was important to test this explanation, since the final model would be trained with all 307,511 training records. Therefore we performed a round of 10-fold cross-validation testing for both activations using the entire training set. The mean of SELU model scores was better than the mean of RELU model scores, so SELU was deemed best.

**7. Creation of final model with best hyper-parameters** - Since we wanted to train a neural network with hidden architecture of 320-320-16 and SELU activation with the entirety of the training set, we decided to use Kaggle public scores to determine when the overfitting threshold was crossed. Our first Kaggle submission was after 3 training epochs, then we increased the number of epochs by one until Kaggle scores started to fall. The optimal AUC score of 0.7549 was achieved with 6 epochs of training.



7. Discussion

Our result shows that LGBMClassifier, which is a version of Gradient Boosting algorithm, has the best performance with the highest AUC for both validation and test data sets. The three other tree-based methods, namely RandomForestClassifier (from sklearn), GradientBoostingClassifier (from sklearn), and CatBoostClassifier do not perform as well as the LGBMClassifier, probably because they used much fewer features. In addition, the random forest method has much greater AUC for train data set than for valid and test data sets, indicating we are overfitting the train data.

The impact of the number of features used is also shown in logistic regression models: the models with more features tend to have better AUC. Because L1 and L2 regularization is used in the logistic regression models, they do not seem to overfit the data as we get very similar AUCs for all three data sets.

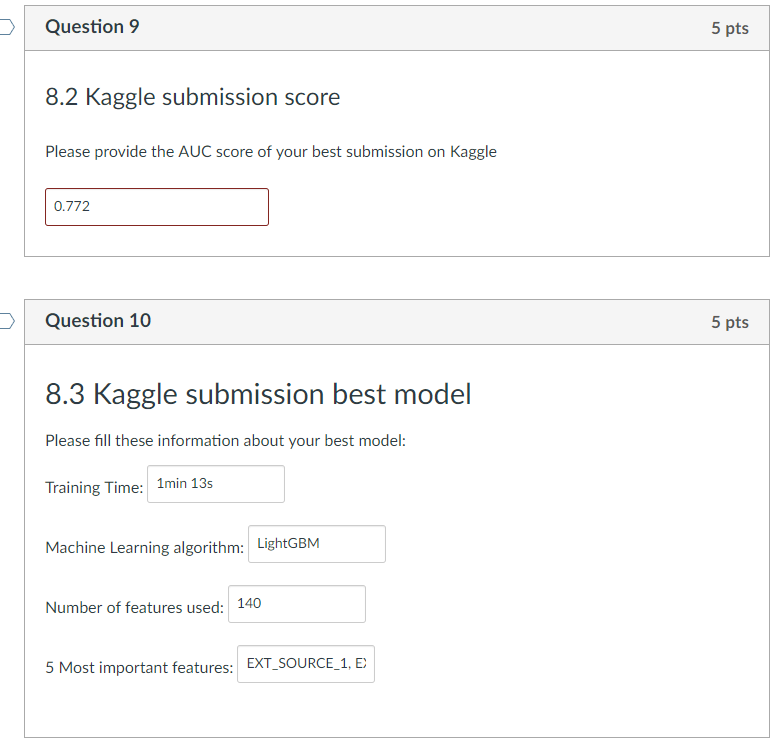
Compared to logistic regression, tree-based methods typically takes much longer time to train. This problem can be mitigated by using early stopping. We used early stopping in GradientBoost (from sklearn) and LightGBM and observed significantly reduced training time and the avoiding of overfitting.

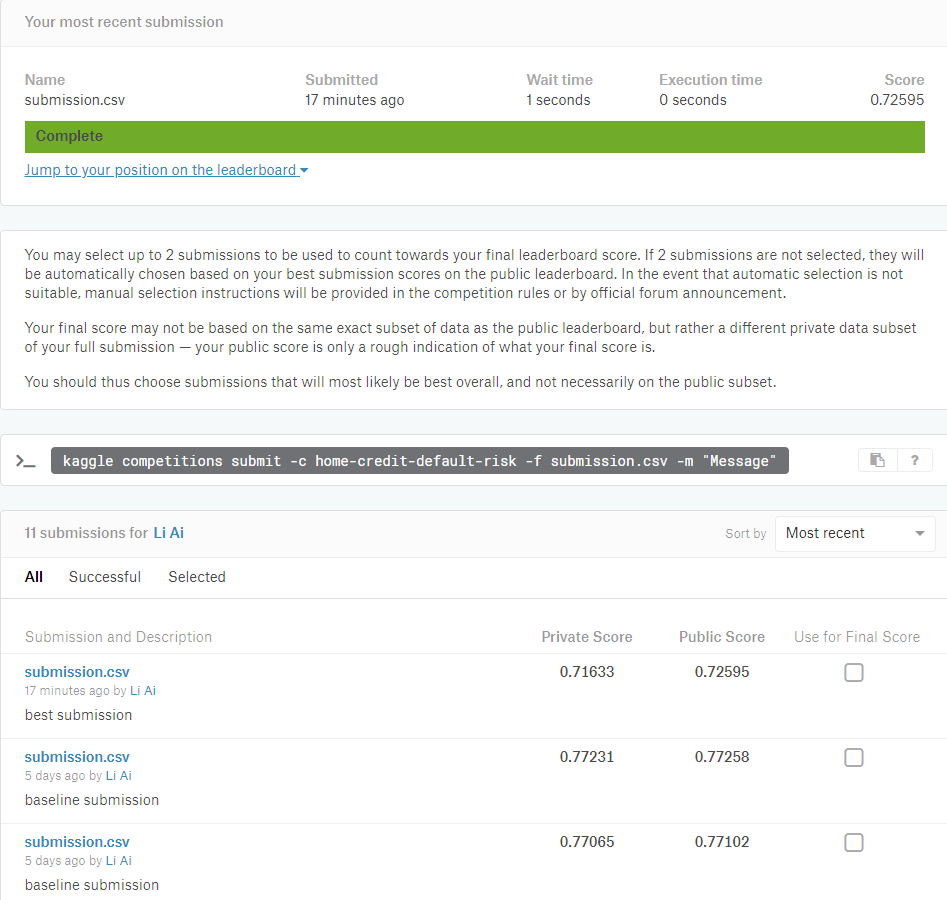
In the last phase, the use of a support vector machine algorithm (SVM) in solving the classification problem was explored as planned. We attempted to train a Sklearn support vector classifier (SVC) and used RandomizedSearchCV for hyperparameter tuning. Because the dataset has sample size (300,000) far exceeding the feature dimension (< 300), a linear kernel was chosen for the final optimization. The SVM as a non-parametric model presumably takes a long time to train. In practice, the grid search spent > 60 hours on model optimizing but still yield no satisfactory results (the best performance score (AUC score) < 0.7). Theoretically, the classification performance could be improved by using an SVC with other non-linear kernels such as Polynomial, Gaussian RBF and *etc*. The downside, however, is that an SVC with a non-linear kernel typically takes O(N^2) ~ O(N^3) to train and is generally not scalable to large datasets.

We have also tried the forward-feeding neural network models. As can be seen in the previous section, the results are comparable with our best LightGBM model from Phase 2, which doesn’t use any additional features from other teams. It is possible that developing new useful features is more important than using a particular model for this problem. NN is know to be the best choice for problems related to image, voice and etc., and thus may not outperform gradient boost models for problems related to relational database.

8. Kaggle Submission

Here is the our best result (with LightGBM):





9. Conclusion

1. LightGBM with full features gives the best result with a test AUC of around 0.77 and a similar Kaggle submission score.

2. In this particular problem, feature reduction/selection does not appear to be necessary, as using the full features gives the best result.

3. Early-stopping reduces the training time and helps prevent overfitting.

4. Using regularization methods also helps prevent overfitting.

5. The non-linear support vector machine classifier is not scalable to large datasets.

6. The performance of neural network models without additional features is on par with other machine learning models.

10. A final thought

For applied machine learning projects for problems with tabular data, feature engineering is probably the most important step. If important new important features can be created, the performance of the models can be greatly enhanced. Domain knowledge is also important as it can often lead to a more efficient treatment of the data and methodology.

Feature reduction approved to be not important in this project. We do not have a very large number of features to start with and none of the features are dominating the final models. However, for some other types of problems, feature reduction and selection may be critical.

It has been a challenge to collaborate with the team members to work on a data science project without spending time, effort (and probably resources). GitHub is very useful but it does not provide a good way of collaborating with Jupyter Notebook. Google Colab is good for Jupyter notebook but has its own problems, e.g. can only save the results to gdrive.

Oftentimes, we found that the experiment for hyperparameter tuning was conducted by a repeated process of trial and error. In addition, an optimal set of hyperparameters obtained on one dataset cannot be automatically generalized to other datasets even when the same machine learning algorithm is used. Presumably, the best model is guaranteed to be achieved if all the hyperparameter space is exhaustively explored. However, this becomes a challenging and nearly infeasible task in practice, because 1）hyperparameter space could be infinitely large, and 2) it’s highly time-consuming and computationally intensive. Developing a principled and systematic approach for hyperparameter optimization would be a very interesting aspect to explore in the future study.

.