Predicting Loan Defaults

Predicting loan defaults is an extremely common use case for machine learning in banking, one of DataRobot's main target industries. As a loan officer, you are responsible for determining which loans are going to be the most profitable and worthy of lending money to. Based on a loan application from a potential client, you would like to predict whether the loan will be paid back in time.

Data

You will be working with a loan dataset fromLendingClub.com, a US peer-to-peer lending company. Your classification target is **is_bad**.

Task

Partition your data into a holdout set and 5 stratified CV folds. Pick any two machine learning algorithms from the list below, and build a binary classification model with each of them: * Regularized Logistic Regression (scikit-learn) * Gradient Boosting Machine (scikit-learn, XGBoost or LightGBM) * Neural Network (Keras), with the architecture of your choice

Both of your models must make use of numeric, categorical, text, and date features. Compute out-of-sample LogLoss and F1 scores on cross-validation and holdout. Which one of your two models would you recommend to deploy? Explain your decision. (Advanced, optional) Which 3 features are the most impactful for your model? Explain your methodology.

Preprocessing & Exploratory Data Analysis

The Python code is configured to prune the dataset first so that redundant or uninformative features are not used in the training phase. By inspecting the data in a pandas dataframe, the following observations were made. : * Column 'Id' only contains sequential numbers and has no impact on prediction. * Column 'pymnt__plan' has only 2 'y' out of 10000 records. The rest of them are 'n'. * Column 'initialliststatus' has only 17 values 'm' from 10000 records. The rest of them are 'f'. * Column 'purpose' is a textual representation and largly overlaps the information found in 'purpose_cat' so it was removed. The remaining are different only in few cases. * Column 'collections_12_mnths_ex_med' was removed as it only has values 0 and NA

Therefore, these columns are removed by the code as a preprocessing step: * Id * pymnt_plan * initial_list_status * purpose

Categorical & Textual Data

The next step was preprocessing categorical and textual columns. All text was converted to lower case and punctuation was removed. Words like 'company', 'corporation' were removed from the emp_title value as they correspond to different ways of saying the same thing. In *emp_title* column variations of same employer were grouped together to form more representative categories for instance ('us army' and 'us military'). Only categories with enough representation (more than 40 items per group) were kept for *emp_title*. For each record in *Notes* punctuation were removed and text was into words. Stop word list was generated to remove

irrelevant wods. Words indicating various client parameters were left (stability of the job, education, missing payments) were kept as the reason for the loan is sometimes reflected in those words. Words with frequency higher than 400 were kept, each representing a separate feature. The feature is encoded by 1 if word is present and by 0 if word is not available. Column *earliest_cr_line* which is encoded as data were converted to numeric feature - number of days passed since that data. For all the features on the training and prediction stage the following transformations were done:

- Numeric features were normalized by subtracting mean and dividing by standard deviation. In the encoded state missing values were replaced with 0 which represents mean value after encoding.
- Binary features (target column is_bad and all the columns representing words were kept as is.
- Categorical features were encoded with oneHot encoding (each category has its own column assigned to 1 for this category). Missing values has all 0s in those columns.

Cross Validation

Two types of cross-validation were implemented:

- k-fold validation (value of folds can be selected in the config.py) and
- hold-out (percentage of the file used for training can be also set up in the config.py)

The algorithm first generates files which store corresponding training and prediction files. Next each fold algorithm is trained. Generated models are applied to remaining data for prediction. Results produced by each prediction step are combined together to form overall metrics for algorithm with this kind of parameters: accuracy, precision, recall, F1-score, logLoss. For accuracy, precision, recall and F1-score values are also generated for different thresholds of probability to see how these will affect prediction results.

Importance Estimation

Three methods were used for importance estimation:

Ranom Tree Leafs

Generation of the value based on variations in random tree leafs https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html

After model is trained on the prediction stage each column (values of the feature) are randomly permuted. Accuracy of prediction on the permuted data is compared to the accuracy of prediction on the original data. Importance value is recorded as percentage of accuracy drop. After the model is trained on the prediction stage each column(values of the feature) are randomly permuted. LogLoss of prediction on the permuted data is compared to the LogLoss of prediction on the original data. Importance value is recorded as percentage of logloss change. It turned out that this method is not working well when working with the whole dataset. Features combined together replace each other and the change in the accuracy is not visible. LogLoss can change but still insignificantly. In order to select 3 most important features in this case sequential feature selection was applied. First combination of 1 feature was tested, logLoss was estimated and the value with the best logLoss was kept. Next pair of features including this previously selected one was tested. Finally, combinations of 3 most important features was generated.

For logistic regression algorithm that is ['revol_util', 'verification_status', 'annual_inc'] For gradient boosting that is: ['annual_inc', 'open_acc', 'total_acc'] Tests show that gradient boosting would be the best algorithm as it shows better values of recall.

Importance Tree

Configuration & Running the Code

It is easy to run the code from the command line:

> python main.py Please make sure your Python environment satisfies the following requirements: numpy >= 1.14.5 scipy >= 1.1.0 pandas >= 0.23.1 scikit-learn >= 0.19.1 matplotlib >= 2.2.2 which can be found in requirements.txt. The main.py file contains all the configuration for the run in the main section. Several things can be customized there. After running main.py with the default configuration, the output is the results of the prediction for the top three features. These are pasted at the end of this notebook. The classifiers are stored in the loan_analysis/data folder at the root level of the directory. The processed data is also saved there.

Function Architecture

I have tried to avoid using the sklearn libraries unless necessary. I did this so you can get a flavor of my coding style and to give you an idea of how I would approach a problem. Obviously, one can just use sklearn.model_selection.StratifiedKFold() instead of writing crossValidation.py. Here is a list of files with the functions they implement: axil.py clean_files() utility to keep things tidy. check_extra_values() checks if there are extra values in a column normalize_numeric() normalizes a numeric column substracting its mean and dividing the result by standard deviation config.py the configuration file where the user is able to set the processing configuration parameters. crossValidation.py full_file_processing() file processing utility for logistic regression or gradient boosting. crossValidation() exactly as named, performs cross validation. cross_modelEvaluation() evaluation function for logistic regression and gradient boosting. data_reader.py read_Nstr_from_Csv() reads specified number of rows from the csv file starting from selected position. encoders_generation.py generate_settings() get_categorical_encoders() Returns categorical encoders according to data type prediction. get_missing_value_name() Returns missing value name. get_function_by_name.py get_function_by_name() Returns function (as a source) by its name and module name. gradient_boosting_classifier.py gradient_boosting_classifier_train() Performs training using GradientBoostingClassfier() gradient_boosting_classifier_predict() Performs prediction using GradientBoostingClassfier() logistic_regression.py logistic_regression_train() Performs training using LogisticRegression() logistic_regression_predict() Performs prediction using GradientBoostingClassfier() main.py check_stratification() display_importances() main preliminary_statistics.py filter_employment() filter_notes() word_frequency() clear_notes_using_dictionary() clear_employment_using_dictionary() purpose_handling_small_business() date_conversion() make_lower() filter_text_and_categorical_values() refine_input_data.py replace_name_property() refineInputData() save_load.py save_obj() save_objects() load_obj() scores_computations.py get_confusion_matrix() accuracy_confusion_binary() log_loss() split_for_validation.py split_for_validation() get_base_name() generate_names() correct_path() split_data() write_Partitions_Two() write_Partitions_K() write_To_Csv() tree_importance.py get_tree_importance()