Firms Defaulted Prediction - Technical Report

Viet Nguyen, Marcell Magda

The code and both reports can be found on this GitHub

1. Data

As described in the summary report, we have to do some file manipulation and feature engineering to obtain the necessary predictors.

For file manipulation, since the original data file is too large, GitHub refuses to accept the file. Hence, we have to split the file into 2 parts and read into 2 dataframes. The final working dataframe is the concatenation of those 2 dataframes.

After importing the data from files, we obtain 287,829 records (N = 287,829).

Determine the default status

Before we start the data manipulation, we first attempt to determine the default status of a company for a given year based on the following criteria: – Existed in year T-1 (sales > 0), but did not exist in year T (sales is 0 or missing)

We define the function to do the job for us:

In [111…
```python
def get_sme_comp_default(df):
    '''
    This function determine the default status for SME firms
    :param df: raw data
    :return: dataframe with default status for only SME firms
    '''
    # add all missing year and comp_id combinations -
    # originally missing combinations will have NAs in all other columns
    df = (
        df.set_index(["year", "comp_id"])
        .unstack(fill_value="toReplace")
        .stack()
        .reset_index()
    )
    df = df.replace("toReplace", np.nan)  # only way I could define it as NaN

    # generate status_alive; if sales larger than zero and not-NA, then firm is alive
    df["status_alive"] = (df["sales"] > 0 & (False == df["sales"].isna())).astype(int)

    # defaults in one year if there are sales in this year but no sales one year later
    # Status_in_one_years: data.groupby('comp_id')['status_alive'].shift(-1)
    df["default"] = (
        (df["status_alive"] == 1)
        & (df.groupby("comp_id")["status_alive"].shift(-1) == 0)
    ).astype(int)

    # filter for SME firms
    return df[(df.sales >= 1000) & (df.sales <= 10_000_000)]
```

Data filtering

As the focus of the prediction is for the small or medium firms (1,000 <= sales <= 10,000,000) in the ‘Manufacture of computer, electronic and optical products’ industry (ind = 26), we filter the data with the appropriate conditions (N = 11,764).

Since we are doing prediction for firms in 2015, we only need the data for the years before 2015 (N = 10,726)

Handling missing values and extreme values

Upon investigation, we found out that columns "COGS", "finished_prod", "net_dom_sales", "net_exp_sales", "wages", "D", "exit_year", "exit_date", "labor_avg" and "founded_year" contains too many missing values. For these columns, we decided to drop them (N = 10,726).

For other remaining columns, we do imputations as follows (N = 10,726):

- Columns "amort", "curr_assets", "personnel_exp", "material_exp", "liq_assets", "inventories", "intang_assets", "fixed_assets", "extra_profit_loss", "extra_inc", "extra_exp", "curr_liab", "tang_assets": impute with 0s to interpret as

non-exist.
- Columns "sales", "profit_loss_year", "inc_bef_tax", "share_eq", "subscribed_cap", "ceo_count", "female", "birth_year": impute with the median as there are fewer missing values and they cannot be 0.
- Columns "intang_assets", "fixed_assets", "curr_assets": replace with 0s for missing values or values < 0 as these values cannot be smaller than 0.

For each imputed observation, we add the "imputed_flag" to mark those observations.

For extreme values, we cap both end of them:

- "extra_profit_loss_pl", "inc_bef_tax_pl", "profit_loss_year_pl", "share_eq_bs": cap between [-1, 1]
- "n_ceo_age": cap between [25, 75]

Label engineering

From the original set of features, we decide to engineer more features to better capture the pattern of each of the features in the dataset:

- "n_day_alive": number of days this company exists (current year - founded date)
- "f_foreign_management": if the company's foreign ratio > 0.5
- "total_assets_bs": sum of "intang_assets", "curr_assets", "fixed_assets"
- Profit & loss variables: P&L related divided by sales
- Balance sheet variables: BS related divided by total assets
- "n_yoy_growth": Year on Year sales growth from previous year
- "n_gross_profit_margin": 1 - ("material_exp" + "personnel_exp") / "sales"
- "n_net_profit_margin": "profit_loss_year" / "sales"
- "n_return_on_equity": "profit_loss_year" / "share_eq"
- "n_debt_equity_ratio": "curr_liab" / "share_eq"
- "n_current_ratio": "curr_assets" / "curr_liab"
- "n_quick_ratio": ("curr_assets" - "inventories") / "curr_liab"
- "n_return_on_assets": "profit_loss_year" / "total_assets_bs"
- "d_ceo_young": if ceo age < 40
- "n_ceo_age" = current year - "birth_year"
- Add square terms for firm age, profit & loss and balance sheet variables to capture non-linearity
- Add log of sales to balance the left skewed sales figures

Train test split

For the holdout set, we use the cross-sectional data in 2014. This set will be used for model performance testing on a simulated live data.

```
In [133…  holdout_metadata
```

Out[133…

	Number of firms	Firms defaulted	Firms stay alive	Mean of sales	Min of sales	Max of sales
n_sales	1037	56	1037	490198.360652	1070.370361	9576485.0

The training set is the rest of the data where the year is before 2014 ("year" < 2014). This will be used for training and cross validation.

---

2. Models

For modeling, we build the 4 models (Logit, Logit with Lasso, Random Forest and Gradient Boosting) as explained in details in the summary report. We use Scikit Learn Pipeline to automatically handles categorical values and interaction terms before fitting the data. All models except for the Logit model are 5-fold cross validated.

Logit model

The logit model is built without any interaction terms. The classifier used is Scikit Learn LogisticRegressionCV.

Logit with Lasso

The logit with lasso model is built with interaction terms as we let the algorithm does the coefficient reduction for us. The classifier used is Scikit Learn LogisticRegressionCV in which cross validation is built in. We use the following lamda values for cross validations:

```
In [129… lamdas = list(10**np.arange(-1, -4.01, -1/3))
          lamdas
```

```
Out[129… [0.1,
          0.046415888336127795,
          0.021544346900318846,
          0.010000000000000005,
          0.004641588833612782,
          0.002154434690031887,
          0.001000000000000001,
          0.0004641588833612782,
          0.00021544346900318867,
          0.00010000000000000021]
```

We then compute the Cs values to fit into the cross validation pipeline:

```
In [130… n_obs = training_set.shape[0]*4/5
          Cs_values = [1/(l*n_obs) for l in lambdas]
          Cs_values
```

```
Out[130… [0.0012901228196924347,
          0.00277948535714713,
          0.005988219673873435,
          0.01290122819692434,
          0.027794853571471285,
          0.05988219673873429,
          0.12901228196924333,
          0.27794853571471284,
          0.598821967387343,
          1.2901228196924321]
```

Random forest

The random forest model takes the same feature set as the Logit model. The classifier used is Scikit Learn RandomForestClassifier with GridSearchCV for cross validation implementation. We use the following tuning grid for cross validation where the criterion is "gini":

- max_features: [10, 12, 15, 18]
- min_samples_split: [8, 10, 15]
- min_samples_leaf: [5, 10, 15]

The best parameters after cross validation are:

- max_features: 18
- min_samples_leaf: 15
- min_samples_split: 8

Gradient Boosting

The gradient boosting model takes the same feature set as the Logit model. The classifier used is Scikit Learn HistGradientBoostingClassifier with GridSearchCV for cross validation implementation. The HistGradientBoostingClassifier should perform faster on large dataset (N > 10,000) compared to the GradientBoostingClassifier; hence we pick it for the gradient boosting model. We use the following tuning grid for cross validation:

- max_depth: [5, 8, 12, 15]
- min_samples_leaf: [5, 7, 10]
- learning_rate: [0.1, 0.01]

The best parameters after cross validation are:

- max_depth: 5
- min_samples_leaf: 10
- learning_rate: 0.01

Performance on training data

Summary table

```
In [115… summary_results['Training time'] = [logit_time, lasso3_time, prob_forest1_time, gbm1_time]
          summary_results
```

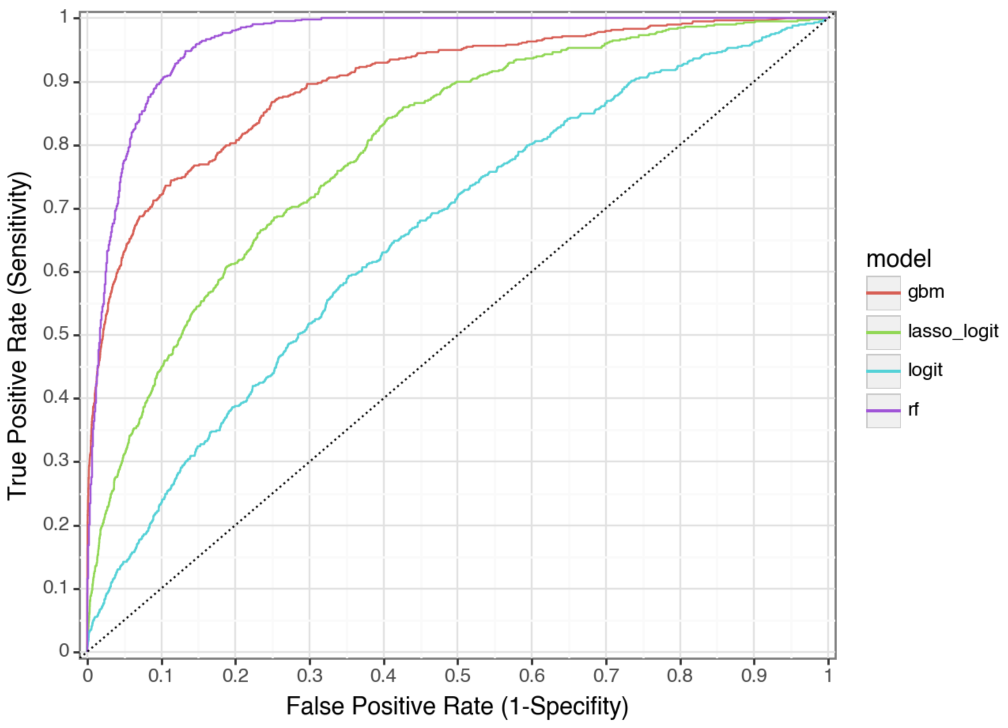
Out[115…

	Model	Number of predictors	CV RMSE	CV AUC	CV threshold	CV expected Loss	Training time
0	logit	107	0.278230	0.652518	0.488771	0.909374	0 days 00:00:03.273379
1	lasso_logit	5778	0.232864	0.777904	0.140603	0.787068	0 days 00:49:56.997905
2	rf	107	0.229560	0.797272	0.166139	0.758891	0 days 00:07:44.125899
3	gbm	107	0.231662	0.781678	0.125166	0.779641	0 days 00:00:19.230873

ROC curves

In [116…

```
# roc plots on training set
create_roc_plots(training_set.default, training_set[categorical_columns + numerical_columns], models)
```



Out[116…

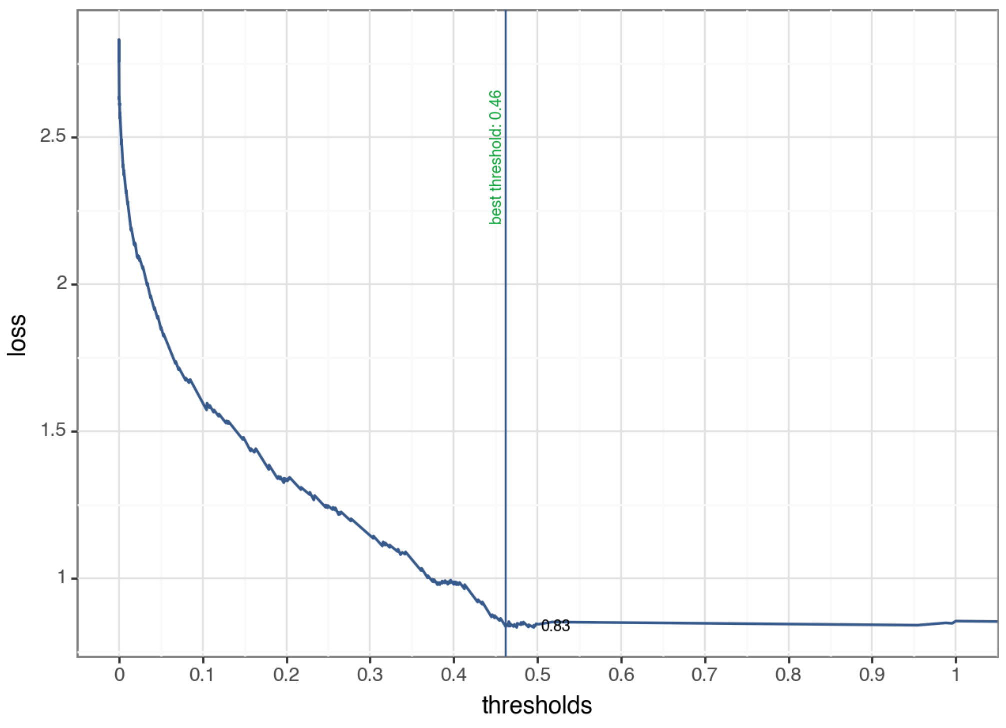
```
<Figure Size: (640 x 480)>
```

Loss plots and ROC plots with best threshold for Fold 5

Logit

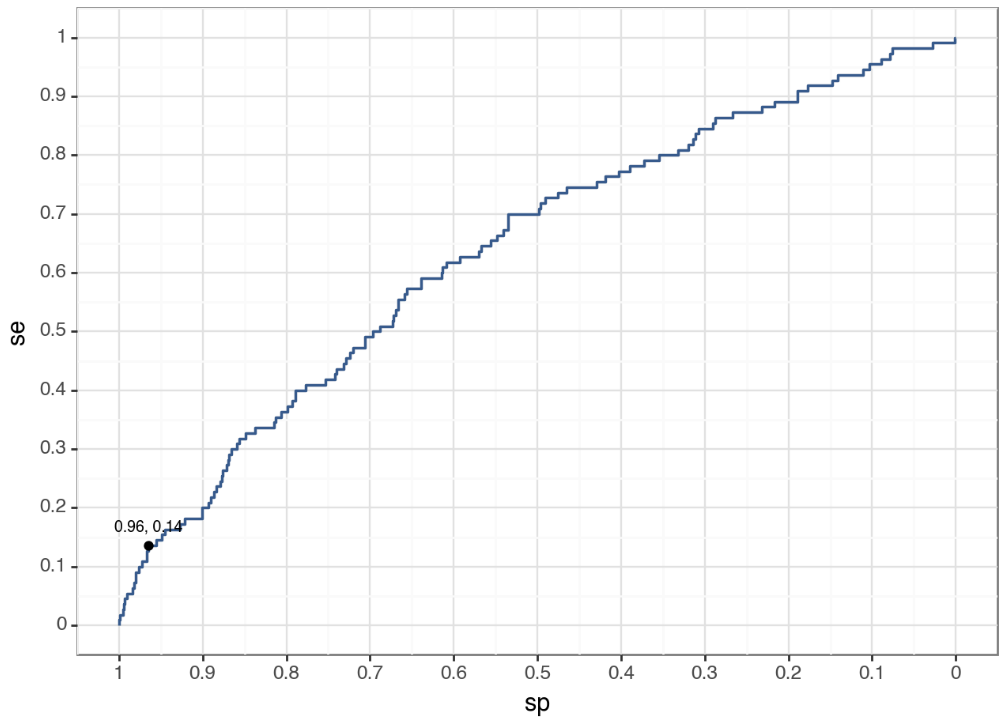
In [117…

```
create_loss_plot(fold5_all_coords['logit'], fold5_threshold['logit'], fold5_expected_loss['logit'])
```



```
Out[117… <Figure Size: (640 x 480)>
```

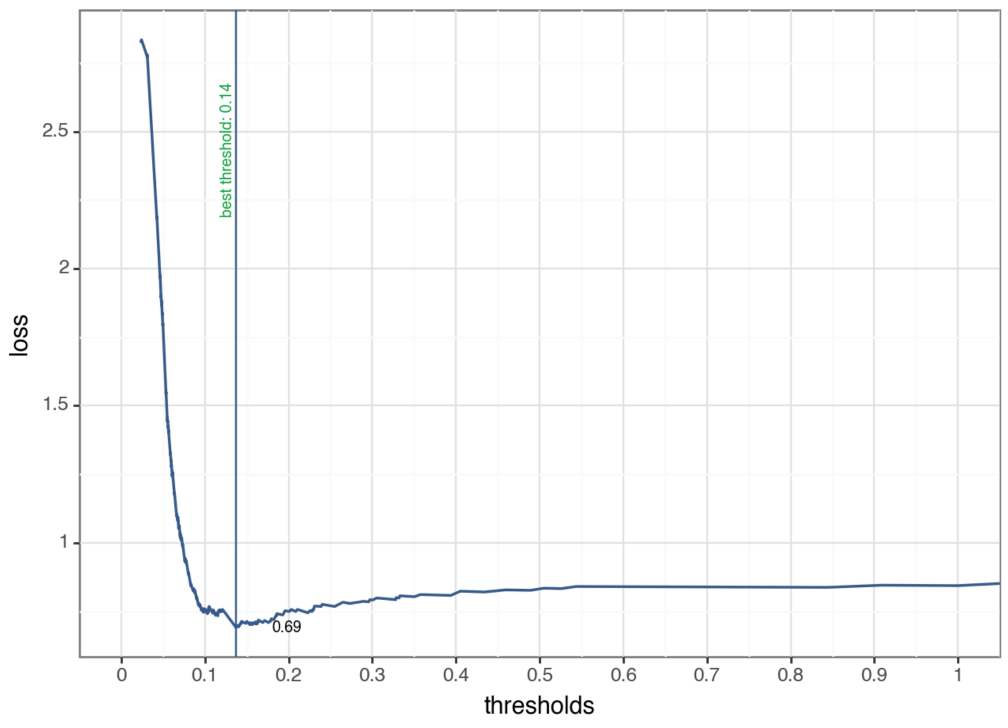
```
In [121… create_roc_plot_with_optimal(fold5_all_coords['logit'], fold5_threshold['logit'])
```



```
Out[121… <Figure Size: (640 x 480)>
```

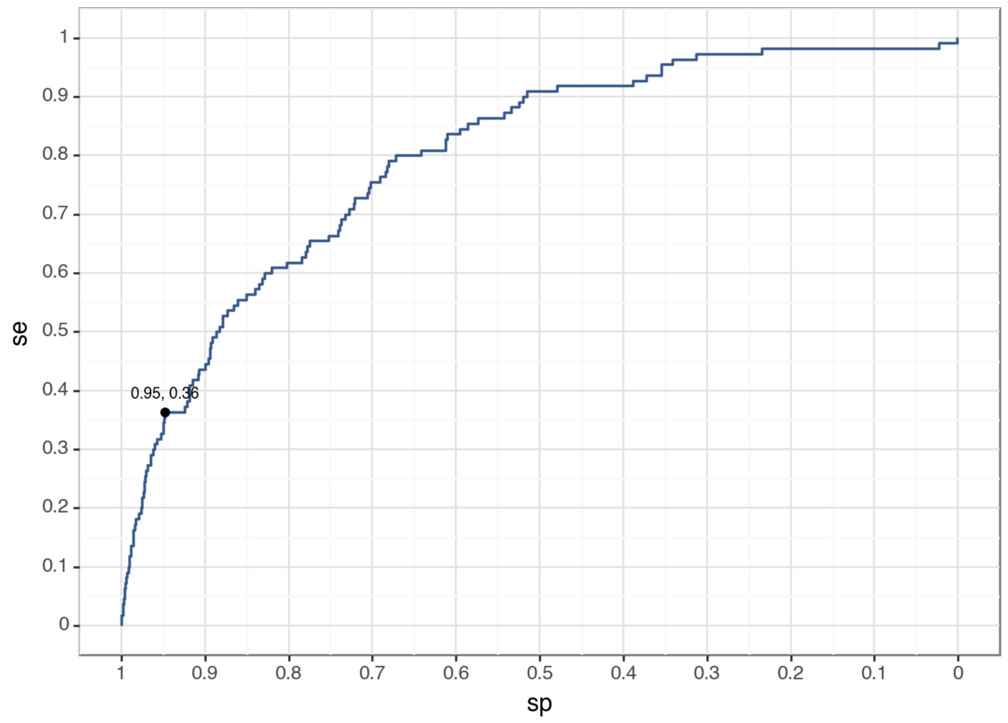
Logit with Lasso

```
In [118… create_loss_plot(fold5_all_coords['lasso_logit'], fold5_threshold['lasso_logit'], fold5_expected_loss['lass
```



```
Out[118… <Figure Size: (640 x 480)>
```

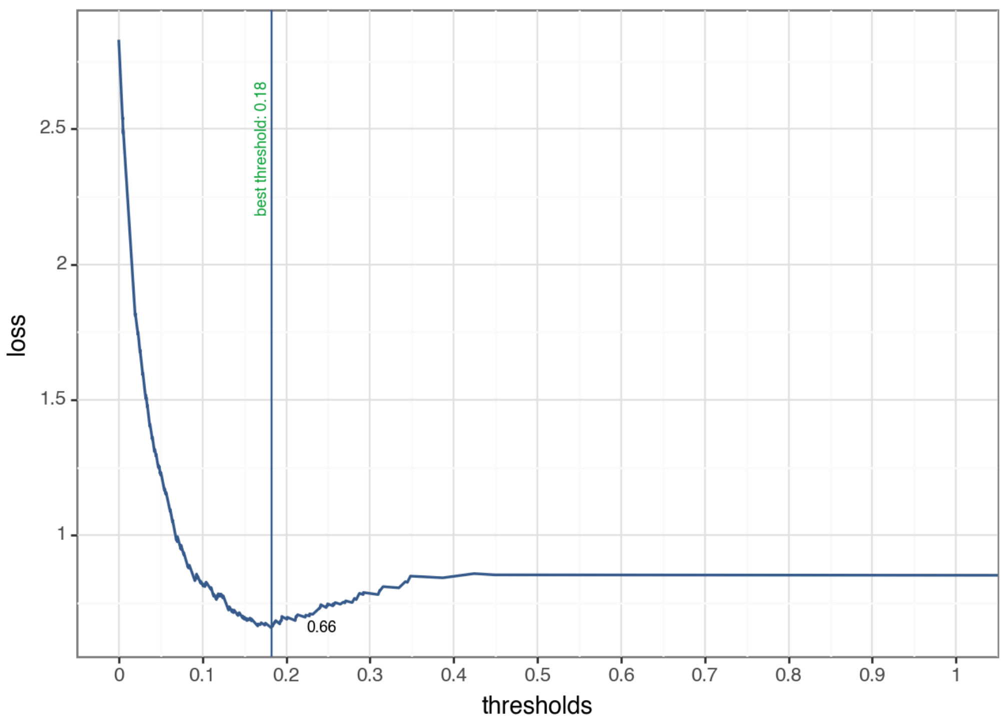
```
In [122… create_roc_plot_with_optimal(fold5_all_coords['lasso_logit'], fold5_threshold['lasso_logit'])
```



```
Out[122… <Figure Size: (640 x 480)>
```

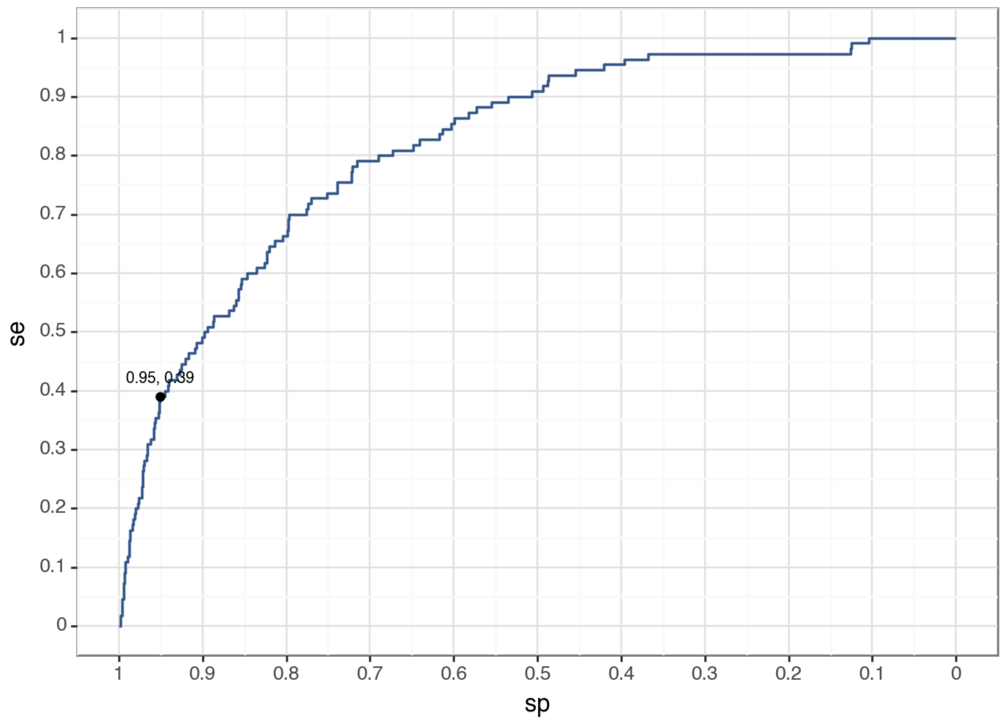
Random forest

```
In [119… create_loss_plot(fold5_all_coords['rf'], fold5_threshold['rf'], fold5_expected_loss['rf'])
```



```
Out[119… <Figure Size: (640 x 480)>
```

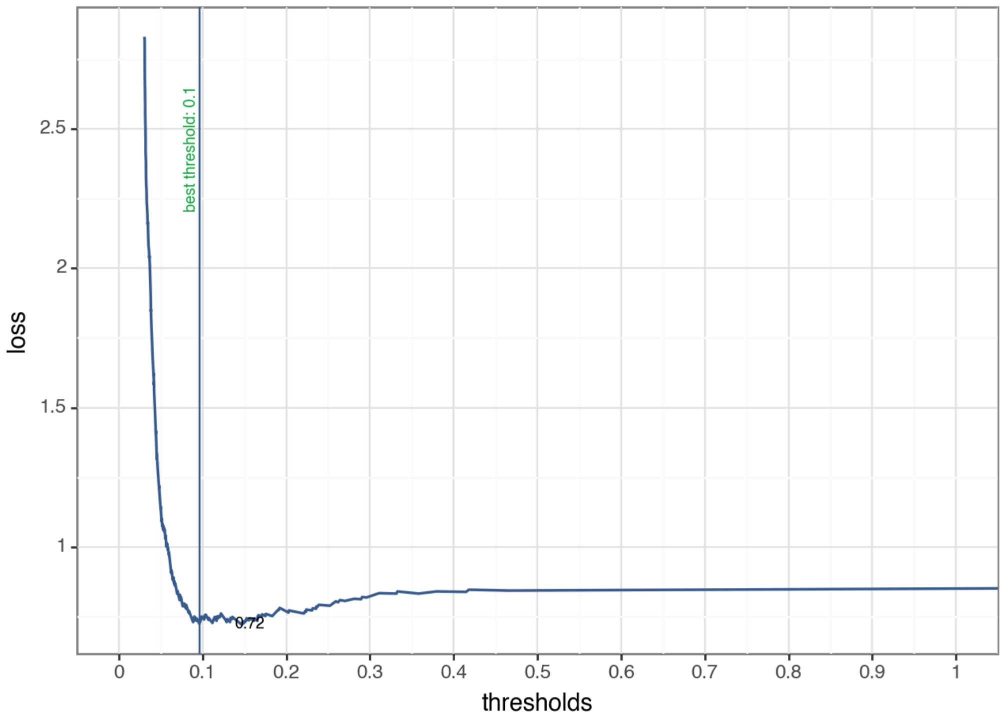
```
In [123… create_roc_plot_with_optimal(fold5_all_coords['rf'], fold5_threshold['rf'])
```



```
Out[123… <Figure Size: (640 x 480)>
```

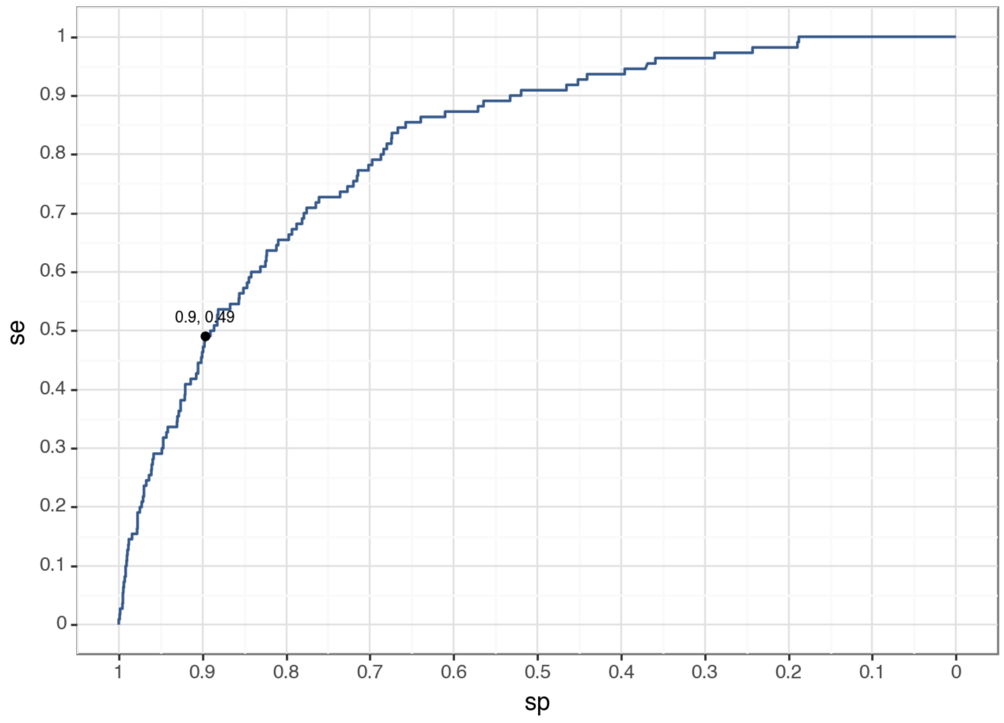
Gradient Boosting

```
In [124… create_loss_plot(fold5_all_coords['gbm'], fold5_threshold['gbm'], fold5_expected_loss['gbm'])
```



```
Out[124… <Figure Size: (640 x 480)>
```

```
In [125… create_roc_plot_with_optimal(fold5_all_coords['gbm'], fold5_threshold['gbm'])
```



```
Out[125… <Figure Size: (640 x 480)>
```

3. Prediction on live data

For performance testing on the hold out set, we use the best threshold (the average of best threshold values among all folds) and best parameters obtained from the cross validations of all model to predict the default status for the holdout set. We then calculate the RMSE (Brier score), AUC, accuracy, sensitivity, specificity and expected loss based on prediction results.

Summary table

```
In [135… holdout_predictions = []
accuracy = dict()
sensitivity = dict()
```

```
specificity = dict()
exp_loss = dict()

for model_name, model in models.items():
    y_pred = model.predict_proba(holdout_set[categorical_columns + numerical_columns])[:,1]
    threshold_prediction = np.where(y_pred < best_thresholds_cv[model_name], 0, 1)
    holdout_predictions.append(y_pred)
    tn, fp, fn, tp = confusion_matrix(holdout_set.default, threshold_prediction, labels=[0,1]).ravel()
    accuracy[model_name] = (tn + tp) / len(y_pred)
    sensitivity[model_name] = tp / (fn + tp)
    specificity[model_name] = tn / (tn + fp)
    exp_loss[model_name] = (fp*FP + fn*FN)/len(y_pred)

holdout_predict_summary = pd.DataFrame({
    'Model': list(models.keys()),
    'RMSE': [np.sqrt(mean_squared_error(holdout_set.default, holdout_prediction)) for holdout_prediction in
    'AUC': [roc_auc_score(holdout_set.default, holdout_prediction) for holdout_prediction in holdout_predic
    'Optimal threshold': summary['Avg of optimal thresholds'],
    'Accuracy': accuracy.values(),
    'Sensitivity': sensitivity.values(),
    'Specificity': specificity.values(),
    'Expected loss': exp_loss.values(),
})
holdout_predict_summary
```

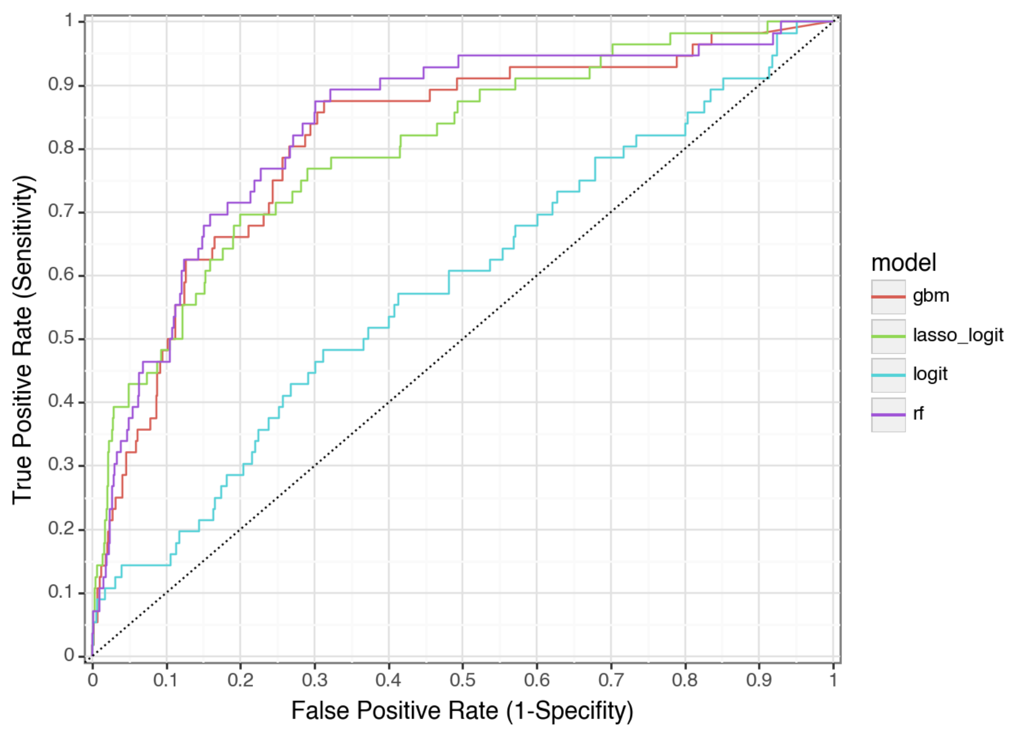
Out[135…

	Model	RMSE	AUC	Optimal threshold	Accuracy	Sensitivity	Specificity	Expected loss
0	logit	0.264424	0.584462	0.488771	0.944069	0.053571	0.994903	0.781099
1	lasso_logit	0.212955	0.802898	0.140603	0.934426	0.392857	0.965341	0.590164
2	rf	0.212542	0.835117	0.166139	0.918997	0.375000	0.950051	0.648023
3	gbm	0.216274	0.814147	0.125166	0.916104	0.321429	0.950051	0.691418

ROC plots for holdout set

In [136…

```
# roc plots on holdout set
create_roc_plots(holdout_set.default, holdout_set[categorical_columns + numerical_columns], models)
```



Out[136…

```
<Figure Size: (640 x 480)>
```