We build a Gradient Boosted Machine (GBM) classifier that yields the risk of a title defecting. In doing so we use H2O library and apply hyper-parameter tuning techniques and cross validation techniques to yield optimal and unbiased out-of-sample performance. From this we draw Receiver Operating Curves (ROC) and Income-Loss curves to investigate the optimal risk threshold for classifying titles into pass or fail. We demonstrate our approach has an out- of-sample true positive rate (TPR) and false positive rate (FPR) of 82% and 46% respectively, and yields a profit of $4.95mn.

Choice of Estimator and Library

It was posited that the data inherits complex non-linear relationships between the independent and dependent variables. In addition to this, a major advantage of decision tree models is that they are able to operate on both continuous and categorical variables directly, as opposed to one-hot-encoding categorical variables which increases the variables and induces large sparsity in the data set. As such tree-based approaches were chosen. The choice of GBM over Random Forests was made due to the following reasons. Given the unbalanced nature of the data set a GBM has a better performance since it strengthens the impact of the positive class. Also, due to the presence of categorical variables with different number of levels, random forests are less favored because to their bias towards those attributes with more levels. Therefore, despite being harder to tune and having a higher risk of over-fitting, the GBM estimator was selected.

The choice of H2O library over, say Scikit-learn, was made by recognizing that the latter library uses one-hot-encoding for categorical variables, where as H2O does not. The binary variables generated by one-hot-encoding are automatically disadvantaged, due to their low-cardinality, over other variables.

Methodology and Results

To this end we underline the process undertaken to build the estimator and provide snippets code developed in python code using the MacVim ide.

1. Load Data
   To accelerate loading of large sets of data we save the csv files into HDF5. This reduces read execution times during code development.

```
def get_data():

    # ----- Convert data to storable HDF5 objects
    # file_location = '/Users/ranajikrishna/invoice2go/git_code/state/default_notices.csv'
    # default_notices = pd.read_csv(file_location)
    # universe = pd.HDFStore('universe.h5')
    # universe['default_notices'] = default_notices
    #
    # file_location = '/Users/ranajikrishna/invoice2go/git_code/state/train_property_data.csv'
    # train_property_data = pd.read_csv(file_location)
    # universe['train_property_data'] = train_property_data
    #
    # file_location = '/Users/ranajikrishna/invoice2go/git_code/state/test_property_data.csv'
    # test_property_data = pd.read_csv(file_location)
    # universe['test_property_data'] = test_property_data
     # -----

    universe = pd.HDFStore('universe.h5')
    return universe['default_notices'], universe['train_property_data'], universe['test_property_data']
```

2. Feature Engineering
   We tie the history of default notices to train and test property data by left-joining *default_notices* to *train_property_data* and *test_property_data* using 'house_id'. We ensure to include only data where the 'record_date' of the default notice is before the 'title_check_date'. We also develop two variables, 'event_sum' and 'def_sum', to understand the net position of the title when the check was done and to capture the number of times the title had NOD in the past respectively. We posit

that these would provide significant separation between titles with defect and no defect. The function *feat_eng* was applied to both *train_property_data* and *test_property_data.*

```
def feat_eng(data, default_data):

    # Set appropriate index
    data.set_index('house_id', inplace=True)

    # Left join defaults to data.
    data = data.join(default_data)

    # Only include instances where record date is before the title check date.
    data = data[data.record_date <= data.title_check_date]
    data.drop(['record_date'], axis=1, inplace=True)

    # Design variable `event_type_sum` to show outstanding position of NOD
    data.event_type.replace('default_notice',1,inplace=True)
    data.event_type.replace('default_rescind',-1,inplace=True)
    data.event_type.fillna(0)

    event_sum = data.groupby(data.index)['event_type'].agg('sum')
    data = data.join(event_sum, rsuffix='_sum')

    # Design variable `event_type_def` to show no. NOD
    data.event_type.replace(-1,0,inplace=True)
    def_sum = data.groupby(data.index)['event_type'].agg('sum')
    data = data.join(def_sum, rsuffix='_def')

    data.drop(['event_type'], axis=1, inplace=True)

    return data.drop_duplicates()
```

3. Hyper-parameter tuning and cross-validation

   We tune the following parameters using the grid-search technique: learning rate, maximum depth of trees, the row- and column- sample rates. We tune for the row- and column- sample rates to combat over-fitting since these improve generalization and lead to lower validation and test set errors. It is also noted that due to this sampling nested cross validation techniques was not necessary in the hyper-parameter tuning process. The number of trees can be tuned for as well, but it was excluded in this exercise due an exponential increase in tuning time.

   The parameter-tuning is performed over 3-folds cross-validation. The best parameters are selected to maximize the area under the curve (AUC). The best model is then used to generate ROC and Income-Loss curves to determine the cut-off threshold.

```
def gbm_model(train, valid, test, ind_col, dep_col):

    # GBM hyperparameters
    gbm_params = {'learn_rate': [0.01, 0.1],\
                  'max_depth': [3, 5, 9], \
                  'sample_rate': [0.8, 1.0],\
                  'col_sample_rate': [0.2, 0.5, 1.0]}

    # Train and validate a cartesian grid of GBMs
    gbm_grid = H2OGridSearch(model=H2OGradientBoostingEstimator, \
                             grid_id='gbm_grid', hyper_params=gbm_params)

    gbm_grid.train(x=ind_col, y=dep_col, training_frame=train, nfolds=3,\
                   ntrees=100, keep_cross_validation_predictions=True, seed=1)

    #gbm_grid.train(x=ind_col, y=dep_col, training_frame=train, \
    #                           validation_frame=valid, ntrees=100, seed=1)

    #Get the grid results, sorted by validation AUC
    gbm_gridperf = gbm_grid.get_grid(sort_by='auc', decreasing=True)
    gbm_gridperf

    # Grab the top GBM model, chosen by validation AUC
    best_gbm = gbm_gridperf.models[0]

    # Now let's evaluate the model performance on a test set
    # so we get an honest estimate of top model performance
    test_hf = h2o.H2OFrame(test)
    test_predict = best_gbm.predict(test_hf)
    test_risk = h2o.as_list(test_predict)
    test.ix[:,'_risk'] = test_risk.p0

    return
```

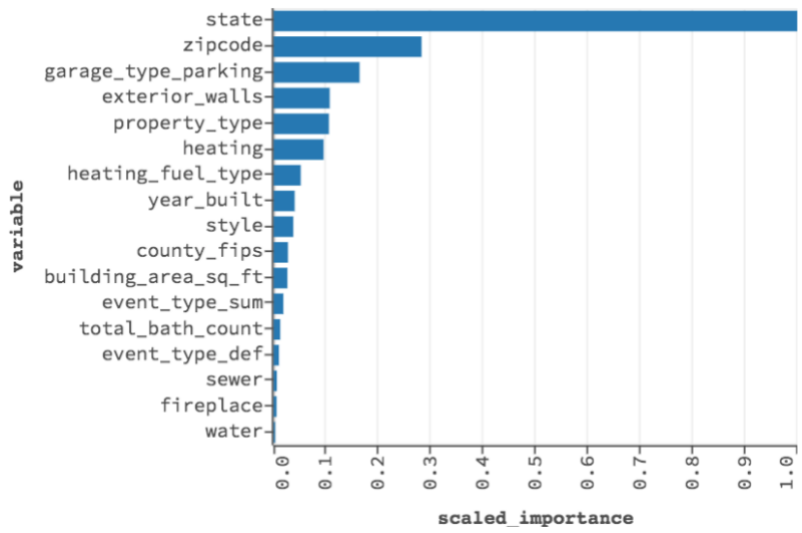
We observe the low standard deviation of the AUC which is indicative of the stability of the coefficients and the tuned parameters.

▾ OUTPUT - CROSS-VALIDATION METRICS SUMMARY

	mean	*sd*	*cv_1_valid*	*cv_2_valid*	*cv_3_valid*
accuracy	0.81271243	0.005928605	0.80882466	0.8243573	0.8049553
auc	0.7550118	0.0025298619	0.7512156	0.7598068	0.75401306
err	0.1872876	0.005928605	0.19117533	0.17564274	0.19504473
err_count	2382.0	74.144005	2448.0	2234.0	2464.0
f0point5	0.2875031	0.0056759217	0.2806051	0.29875988	0.28314427
f1	0.33792827	0.004325868	0.32931507	0.34294116	0.3415286
f2	0.41039696	0.00998999	0.39848825	0.40245754	0.43024507
lift_top_group	7.1714296	0.2529387	6.8879886	6.950273	7.6760273
logloss	0.2876319	0.0025404817	0.2908469	0.2894317	0.28261706
max_per_class_error	0.52038383	0.020510271	0.536623	0.5448868	0.4796417
mcc	0.25520647	0.006570298	0.24238642	0.25911826	0.2641147
mean_per_class_accuracy	0.66465265	0.006808045	0.6555675	0.660412	0.67797834
mean_per_class_error	0.33534735	0.006808045	0.34443244	0.33958802	0.32202163
mse	0.0812924	9.818343E-4	0.08241178	0.08212993	0.07933548
precision	0.261575	0.0067868656	0.25541863	0.27512977	0.25417662
r2	0.094611704	7.957149E-4	0.094666444	0.093206935	0.09596174
recall	0.47961617	0.020510271	0.46337703	0.4551132	0.5203583
rmse	0.28510776	0.0017269367	0.2870745	0.28658321	0.28166556
specificity	0.8496891	0.008746147	0.84775805	0.8657108	0.8355984

We plot the variable importance of the best model to get an essence of the predictive powers of variables.

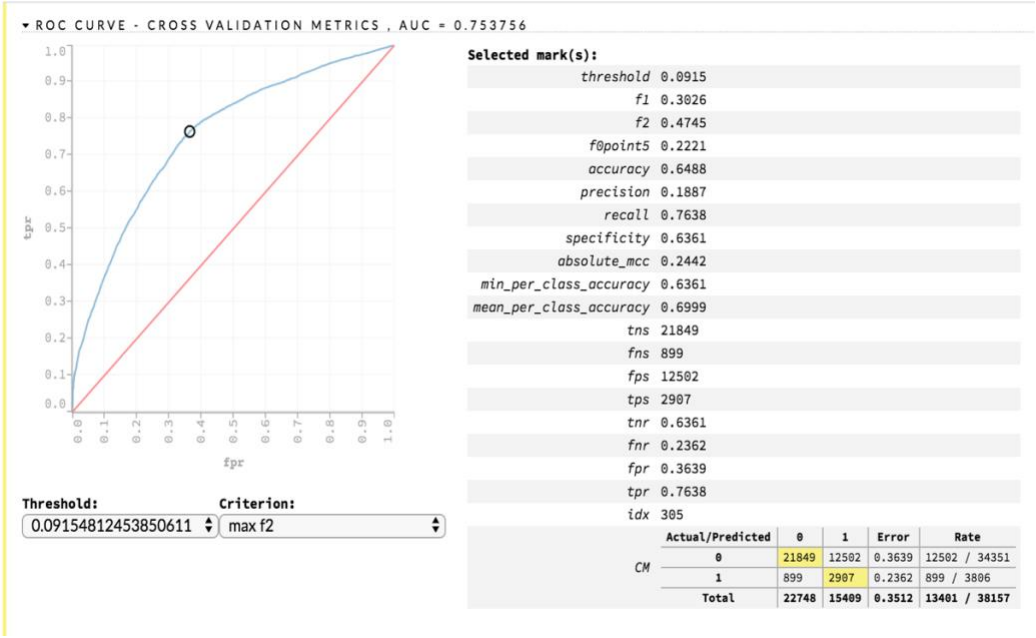
▾ VARIABLE IMPORTANCES



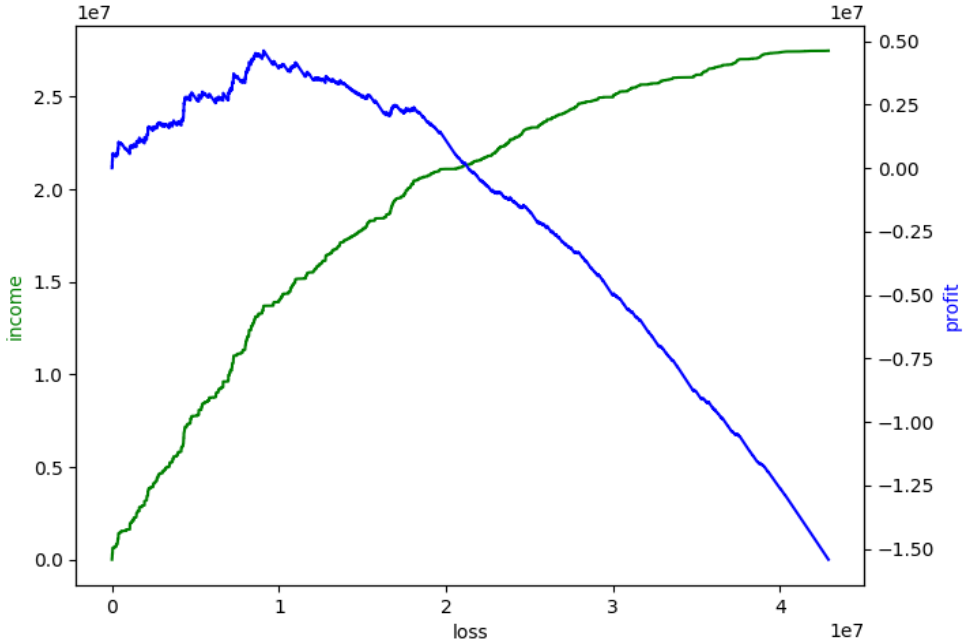
4. ROC and Income-Loss curve
   We simulate ROC and Income-Loss curve yielded by the model.

```
def auc_rev(train, ind_col, dep_col, train_data):

    best_model=H2OGradientBoostingEstimator(col_sample_rate=1, \
            learn_rate=0.01, max_depth=5, sample_rate=0.8, ntrees=100, \
            keep_cross_validation_predictions=True, nfolds=4, seed=1)

    best_model.train(x=ind_col, y=dep_col, training_frame=train)

    predictions = best_model.cross_validation_holdout_predictions()
    risk = h2o.as_list(predictions)

    train_data['_risk']=risk.p1.values

    # Confusion matrix stats.
    fpr, tpr, thresholds = roc_curve(train_data._lien, train_data._risk)

    # Revenue stats.
    income=[sum(1-train_data._lien[train_data._risk <= thr])*800 for thr in thresholds]
    loss=[sum(train_data.lien_amount[train_data._risk < thr]) for thr in thresholds]
    profit=[income[i] - loss[i] for i in range(0,len(thresholds))]

    # Cut-off for maximum profit.
    cut_off = thresholds[np.argmax(profit)]

    # TPR and FPR at maximum profitability.
    true_pos = tpr[thresholds==cut_off]
    flase_pos = fpr[thresholds==cut_off]

    # Plot revenue curves.
    fig, ax1 = plt.subplots()
    ax2 = ax1.twinx()
    ax1.plot(loss, income, 'g-')
    ax2.plot(loss, profit, 'b-')
    ax1.set_xlabel('loss')
    ax1.set_ylabel('income', color='g')
    ax2.set_ylabel('profit', color='b')

    pdb.set_trace()

    return
```

The out-of-sample ROC generated by the best model is drawn out of 3-fold cross validation. The results generated from each of the fold is shown below.



The Income-Loss curves compare the revenue and losses incurred enabled by passing the title at varying threshold values. The cut-off threshold maximizes the profit realized.



A cut-off threshold of 0.07 yields a TPR = 82% and FPR = 46%. This yields a net profit of $4.95mn.