Porto Seguro’s Safe Driver Prediction

Shihao Li (shihaoli@usc.edu)

Dec.01,2017

Abstract

This problem is based on a Kaggle Competition: Porto Seguro’s Safe Driver Prediction. There are about 500,000 training data and 800,000 testing data, with a description of only data format (but without specific meaning) for each feature. We need to give predictions on whether driver will initiate an auto insurance claim in the next year.

In order to deal with the problems, the procedure is divided into three parts:

```
For preprocessing, I've done EDA, fill Nan values, dummy/encoding catagories, standard
lization, feature selection.
```

```
For modeling, I used decorator in python to implement a aspect-oriented programming on
train/predict and grid search for cross validation data generated from last step
```

```
For evaluation, since the problem is asked to be evaluated by gini-index, a simple gin
i-score function is implemented for criteria in Classifier training.
```

With the evaluation on gini_index, the validation score reaches 0.290, public LB reaches 0.285 and private reaches 0.289, reach 20% on the Private LeaderBoard

Problem Statement and Goals

Problem Statement

Porto Seguro, one of Brazil's largest auto and homeowner insurance companies, is working on the price strategies for the insurance costs on drivers. They want to make the insurance prices reasonable enough to let clients initiate auto insurance claim.

Main Goal

In this Competion, we need to predict if a single person would initiate auto insurance claim, given several features.

Difficulties

High dimensionality of feature space

The original feature space is 57(without id), but it will exceed 200 after dummy. Which is not a acceptable amount considering memory usage.

Lack of feature description

In this competition, all of the features are only given with name like 'ps_ind_05', without any deeper description that provides insight. We can only know if it is catagorical or binary.

Nonlinear behaviors

Although it is not specified, nonlinearity exists in amount of features, so we need to justify through linear and non-linear models.

Imbalanced samples

Only 3% percent of the samples are positive, indicating that the dataset is highly imbalanced

Prior and Related Work

As this assignment has been announced, it picked this Kaggle Competition as the title. So I am working on it for both Kaggle and EE660 project.

No other related or prior work exist

Project Formulation and Setup

I'm using XGBoost and LightGBM as the meta classifier, and Logistic Regression as the Stacker(for stacking meta classifiers).

Boosting Trees

Both XGBoost and LightGBM are engineering implementations of boosting trees. So I'd like to introduce boosting trees at first, and then specify the characteristic for both of them.

Boosting is one of the model ensembling method used in machine learning. Unlike bagging, it is a serialized procedure(Forward Stagewise Additive Modeling.

The general procedure is as follows:

1.initialize f_0 = 0

2.for m = 1:M:

(i) $(\beta_m, \gamma_m) = argmin_{\beta_m, \gamma_m} \sum_{i=1}^{N} L[\widetilde{y}_i, \hat{f}_{m-1}(x) + \beta'_m \phi(x_i, \gamma'_m)]$

(ii) $\hat{f}_m(x) = \hat{f}_{m-1}(x) + \beta_m \phi_m(x, \gamma_m)$

3.$\hat{f}_M(x) = \sum_{m=1}^{M} \beta_m \phi_m(x, \gamma_m)$

$\phi_m(x, \gamma)$: *meta classifier(WEAK LEARNER) with x as input and* γ_m *as parameter*

β_m : *weight for* ϕ_m *during ensembling*

$\hat{f}_m(x)$: *ensembled model at m step*

Since we train each tree based on the previous results, those camples which previously predicted incorrectly will be weighted more.

CART

CART is the meta learner for the ensembled model. So it is better to introduce its principle first.

The trained model is $\hat{f}(x) = \sum_{m=1}^{M} w_m I(x \in R_m)$

Tree regions R_m came from recursive splitting of a region into 2, with each decision(node) based on one coordinate variable(feature).

At each iteration(ea. Node of the tree), we divide one region R_m into 2, by thresholding one feature x_j.

Thus, we minimize: $_{m,j,t_k,w_{m_1},w_{m_2}} \{f_{obj}^{(k)}(w_{m_1}, w_{m_2}, D_{ij}, t_k, m)\}$

in which $f_{obj}^{(k)} = cost_k\{(x_i, y_i) \in D\}_{after\ split\ of\ R_m}$

The cost function above for classification problem is: $cost\{(x_i, y_i) \in R_{m'}\} = \frac{1}{N_{R_{m'}}} \sum_{x_i \in R_{m'}} I[y_i \neq \hat{y}_i(R_{m'})]$ = Classification error rate in $R_{m'}$

GBDT$^{[i]}$

Gradient Boosting Decision Tree is the kind of boosting method using gradient through learning.

Compared with AdaBoost(Adaptive Boosting), who modifies the weight for misclassified samples,each new meta classifier is trained using gradient descent.

The general procedure is as follows:

1.initialize $f_0 = armin_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$

2.for m = 1:M:

(i) $\widetilde{y}_{im} = -[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}]_{f=f_{m-1}}$

(ii) $\gamma_m = argmin_w \sum_{i=1}^{N} L_1[\widetilde{y}_i, \phi(x_i, \gamma'_m)]$

(iii)$\beta_m = argmin_\beta \sum_{i=1}^{n} L_2(y_i, F_{m-1}(x_i) + \beta\phi_m(x_i, \gamma_m))$

3.$\hat{f}_M(x) = \sum_{m=1}^{M} \beta_m \phi_m(x, \gamma_m)$

L_1 : *Loss function for training weak learner*

L_2 : *Loss function for line searcing on weight* β

XGBoost

XGBoost is the most popular model used in Kaggle competitions. It always help the Kaggler earn higher rank. Compared with traditional GBDT, it has following characteristics

Support different kinds of meta classifier

Since there's no 'tree' in the name of XGBoost, it supports not only CART as weak learner, but also linear classifier(regressor)

2nd order derivative is taken into consideration

Traditional GBDT only use 1st order derivative. In XGBoost, Taylor series are listed till 2nd order. Making the result more accurate.

Adding regularization term inside meta learner

Shrinkage

After each iteration, a constant is multiplied to the weight, in order to let trees later have more residual to learn, which is, replace step 2 with:

2.for m = 1:M:

(i) $\widetilde{y}_{im} = -[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}]_{f=f_{m-1}}$

(ii) $\gamma_m = argmin_w \sum_{i=1}^{N} L_1[\widetilde{y}_i, \phi(x_i, \gamma'_m)]$

(iii)$\beta_m = argmin_\beta \sum_{i=1}^{n} L_2(y_i, F_{m-1}(x_i) + \beta\phi_m(x_i, \gamma_m))$

(iv)$\beta_m = shrinkage * \beta_m$

Column subsampling

xgboost make a refernce on Random Forest, who subsample features for each meta learner. It could also improve computational speed.

LightGBM

LightGBM is a package developed by MicroSoft. It has a faster training process compared with XGBoost, with only a little loss on performance.

The improvement from LGBM include leaf-wise split and support on catagorical features

Parameters for Boosting trees

All the above algorithms are part of or based on Boosting trees(learners). The following parameters are for one or multi algorithms:

n_estimators

number of meta learners. The algorithm could also stop when accuray on validation set increase within a number of iterations(which indicate overfitting)

eta

parameter for shrinkage(introduced in XGBoost above)

max_depth

max depth for each base learner(CART)

min_child_weight

the threshold weight splitting a node. If the weight doesn't exceed this threshold, the node becomes a leaf

subsample

a ratio of samples used to grow each trees, in order to avoid overfitting

bagging_freq

The algorithm will run bagging when a number of meta-learners is generated

colsample_bytree

a ratio of features used to grow each trees, in order to avoid overfitting

max_bin

the threshold of boundaries that CART will split on one single feature

alpha

L1 regularization term on weight β

lambda

L2 regularization term on weight β

gamma

the threshold for cost function. The split will execute only if the cost function reduce at least gamma

objective

cost function for CART, for classification of 2 classes, binary:logisitic could be used as default

eval_metric

evaluation function for print out temporary score at each iteration

seed

control the randomness in the training process, used for keeping the result same between each run

scale_pos_weight

give a certain weight for samples whose label is positive, it is a method to resolve unbalanced class distribution.

why choosing GBDT

Tree model is the most popular model used in Kaggle competition. It has the following advantages:

1.nonlinearity

The nonlinearity of tree model would avoid the time consuming on make each feature linear(e.g. square each feature).

2.Stable

When adding/removing a feature, the ensembled tree won't be effected so much that a great amount of parameters need to be re-calculated.

3.fast(relatively)

A fine-tuned neural network could also perform well, but it needs careful tuning on network structure and slow training procedure.

Logistic Regression

Logistic Regression is a classification model based on linear regresssion, the trained model is like:

$$\hat{f}(x) = sigmoid(w^T x) = \frac{e^{w^T x}}{1+e^{w^T x}} = \theta(w^T x)$$

Likelihood $p(D|w) = \prod p(\widetilde{y}_i|x_i, w) = \prod_{i=1}^{N} \frac{e^{\widetilde{y}_i w^T x_i}}{1+e^{\widetilde{y}_i w^T x_i}} = \prod_{i=1}^{N} \frac{1}{1+e^{-\widetilde{y}_i w^T x_i}}$

cost function $J(w, D) = NLL(w) = \sum_{i=1}^{N} ln(1 + e^{-\widetilde{y}_i w^T x_i})$

Gradient Descent along w is used to minimize J

Parameters for Logistic Regression

I didn't cast parameter tuning on Logistic Regression, since the performance of stacking is quite stable compared with the 1st stage models. And testing score could only be accessed 5 times each day due to the limit of Kaggle. But I'd like to list some important parameters for it.

penalty

used for weight regularization

C

intensity of regularization

other parameters has appeared in GBDT

Why choosing Logistic Regression

Since we are using the result of for predictions to estimate the real prediction, what we are trying to learn is quite a linear stuff(weight) within each feature. So a linear model would be suitable(enough) for the 2nd stage of problem solving. Besides, it has little parameters and letting most of them as default will not hugely affect the final result. It also has a fast running speed.

Methodology

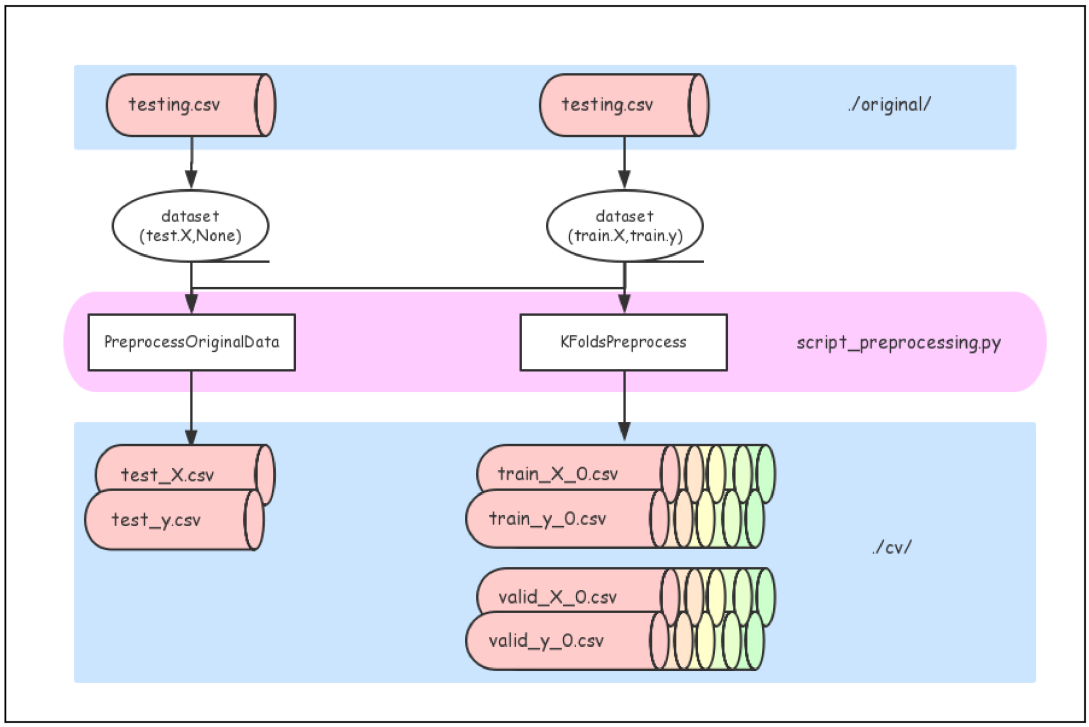
In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

Preprocessing

595212 samples given by Kaggle is labeled(training.csv), while 892816 is given for prediction without label.

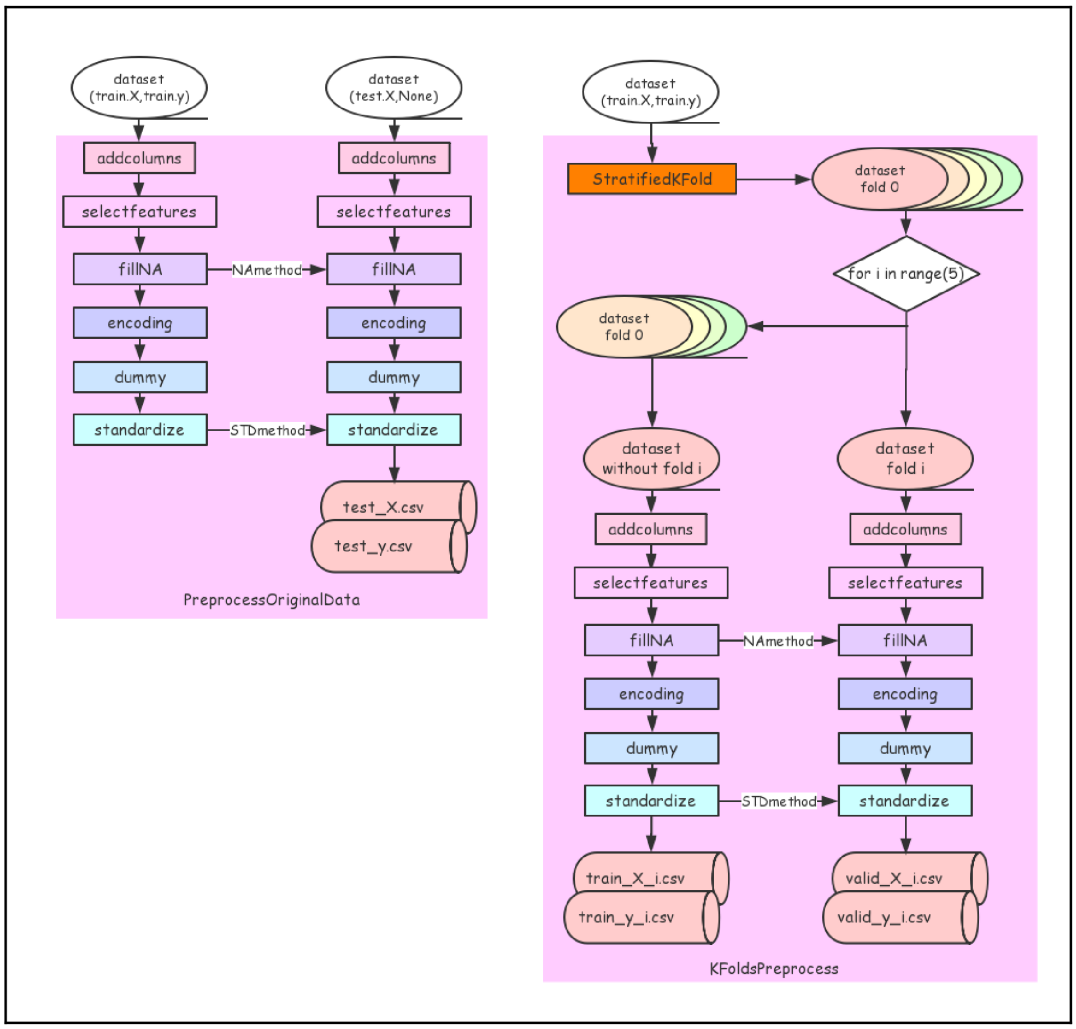
After preprocessing, 5 dataset used for cross validation as well as meta-learner for stacking are generated, and datasets of preprocessed training and testing set is generated. The test_y.csv only contains ids for testing set which is needed when uploading to Kaggle leaderboard.

In [10]:
```
plt.figure(dpi=400, figsize=(10, 6))
plt.xticks([])
plt.yticks([])
imgplot = plt.imshow(mpimg.imread('img/prep.png'))
```



When preprocessing for the testing set, I implement feature augmentation and feature selection first, then dealing with Nan values with fillNA(), dealing with catagorical features with encoding and dummy variables, finally standadize each feature and save the testing set.

In [7]:
```
plt.figure(dpi=400, figsize=(10, 6))
plt.xticks([])
plt.yticks([])
imgplot = plt.imshow(mpimg.imread('img/prep_in_detail.png'))
```



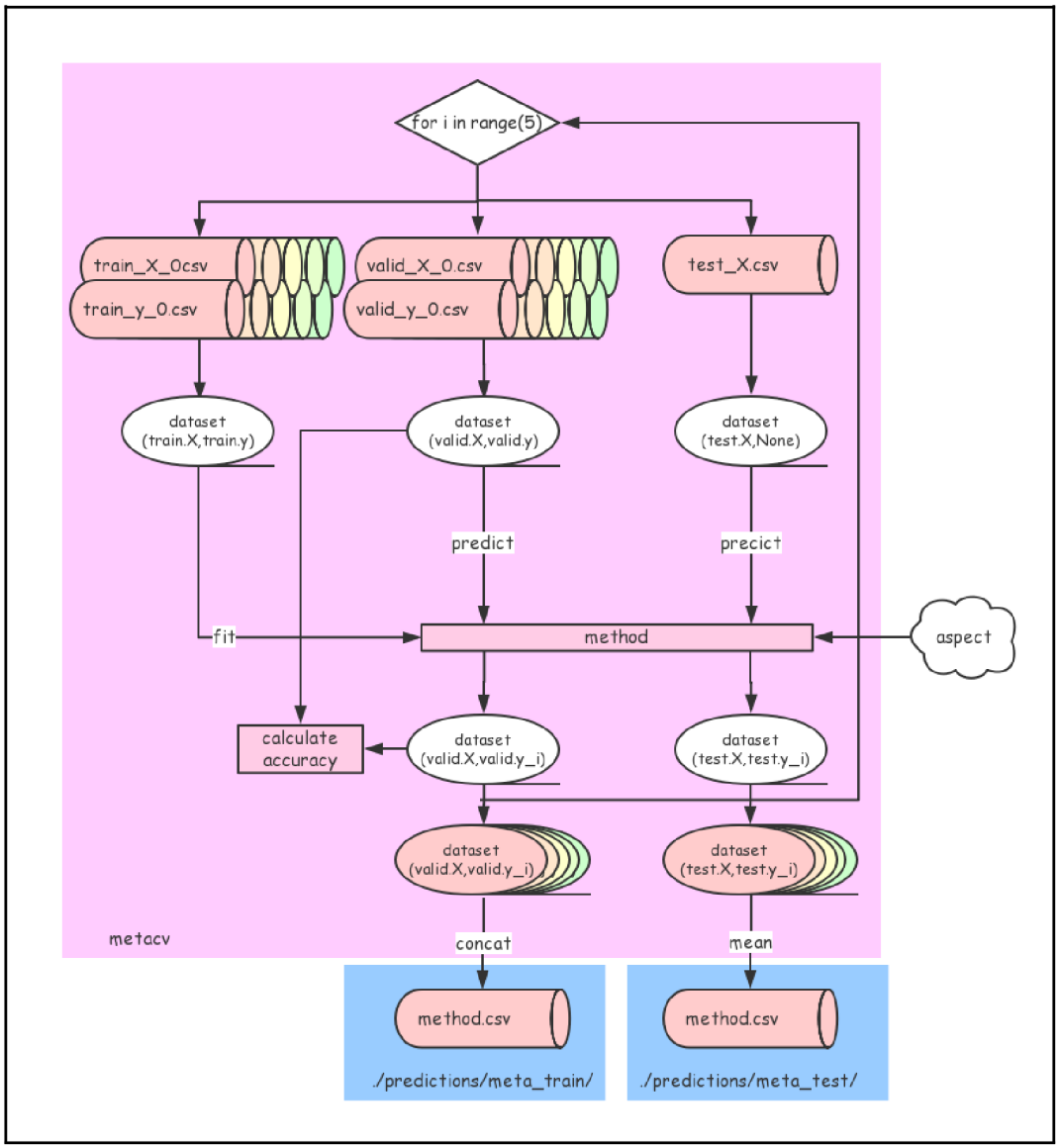
first level model(boosting)

Since the first level model are all about boosting trees, aspect-oriented programming could be used to build outside iteration for training/testing with only fit()/predict() being changed.

metacv() is defined as a wrapper of core train/predict process for each boosting model. For each iteration in 5-fold operation. About 476170 training samples, about 119042 validation samples, and 892816(all) testing samples are used.

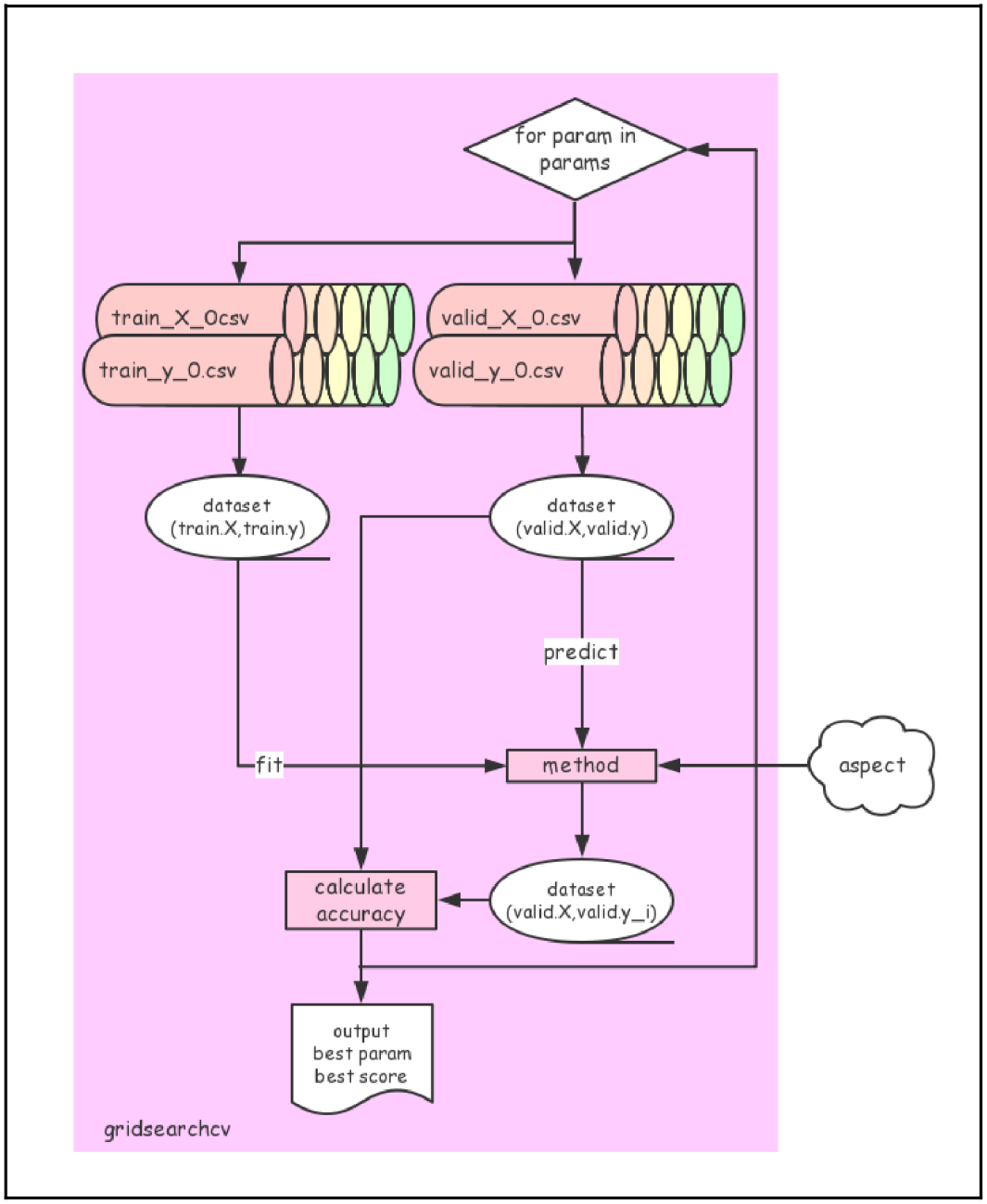
After metacv() we can get predictions for testing set 5 times, and 5 different validation sets 1 time. I concat the 5 validation predictions to get the predictions on original training samples, and take mean value of 5 predictions on testing set and save it as the final predictions for whole testing set.

In [4]:
```
plt.figure(dpi=400, figsize=(10, 6))
plt.xticks([])
plt.yticks([])
imgplot = plt.imshow(mpimg.imread('img/metacv.png'))
```



GridSearch is a tool for searching best parameters for a single model. After all possible values for each parameters is made as input, all combination of values is put into the loop, only score is recorded, and finally best score and best parameter is printed.

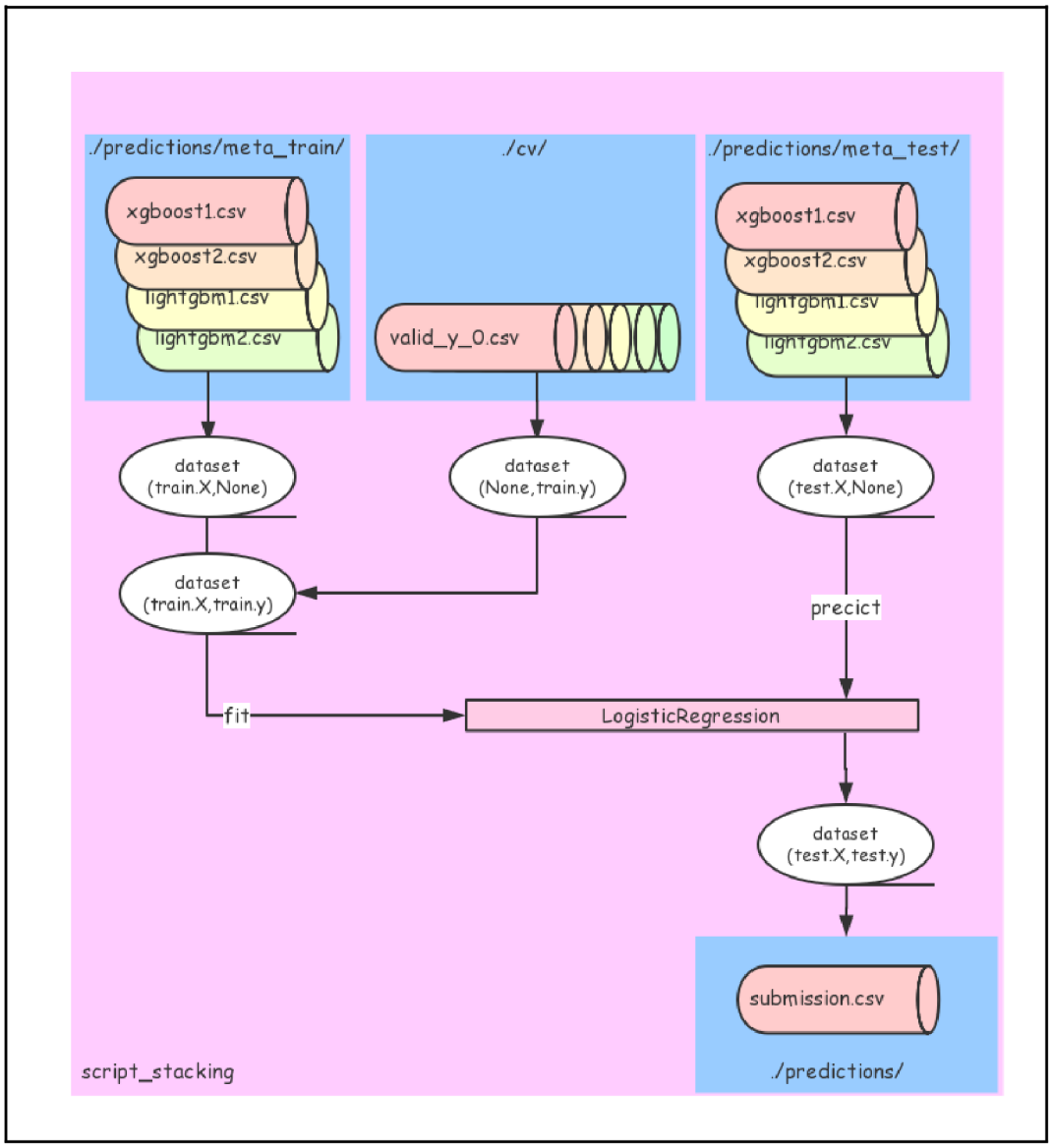
In [5]:
```
plt.figure(dpi=400, figsize=(10, 6))
plt.xticks([])
plt.yticks([])
imgplot = plt.imshow(mpimg.imread('img/gridsearchcv.png'))
```



second level model(stacking)

The image below describe the coding-level procedure of stacking method. It takes the predictions from first level as features, and takes the labels as labels. Logistic Regression is chosen as model. After model is trained, the predictions on testing set from each model is the input and the final output is the final result.

In [11]:
```
plt.figure(dpi=400, figsize=(10, 6))
plt.xticks([])
plt.yticks([])
imgplot = plt.imshow(mpimg.imread('img/stacking.png'))
```



Implementation

Featrue Space

The dataset I used is from a kaggle competition:Porto Seguro’s Safe Driver Prediction

https://www.kaggle.com/c/porto-seguro-safe-driver-prediction/data (https://www.kaggle.com/c/porto-seguro-safe-driver-prediction/data)

Only the type of the data is provided, the name and description of each feature is not provided in the original dataset

features	missing_per	type	cardinality
ps_ind_01	0.00%	real	
ps_ind_02_cat	0.00%	catagory	5
ps_ind_03	0.00%	real	
ps_ind_04_cat	0.04%	catagory	3
ps_ind_05_cat	0.00%	catagory	8
ps_ind_06_bin	0.01%	binary	
ps_ind_07_bin	0.98%	binary	
ps_ind_08_bin	0.00%	binary	
ps_ind_09_bin	0.00%	binary	
ps_ind_10_bin	0.00%	binary	
ps_ind_11_bin	0.00%	binary	
ps_ind_12_bin	0.00%	binary	
ps_ind_13_bin	0.00%	binary	
ps_ind_14	0.00%	real	
ps_ind_15	0.00%	real	
ps_ind_16_bin	0.00%	binary	
ps_ind_17_bin	0.00%	binary	
ps_ind_18_bin	0.00%	binary	
ps_reg_01	0.00%	real	
ps_reg_02	0.00%	real	
ps_reg_03	0.00%	real	
ps_car_01_cat	0.00%	catagory	13
ps_car_02_cat	18.1%	catagory	3
ps_car_03_cat	0.02%	catagory	3
ps_car_04_cat	0.00%	catagory	10
ps_car_05_cat	69.1%	catagory	3
ps_car_06_cat	0.00%	catagory	18
ps_car_07_cat	44.8%	catagory	3
ps_car_08_cat	0.00%	catagory	2
ps_car_09_cat	1.93%	catagory	6
ps_car_10_cat	0.00%	catagory	3
ps_car_11_cat	0.10%	catagory	104
ps_car_11	0.00%	real	
ps_car_12	0.00%	real	
ps_car_13	0.00%	real	
ps_car_14	0.00%	real	
ps_car_15	0.00%	real	
ps_calc_01	7.16%	real	
ps_calc_02	0.00%	real	
ps_calc_03	0.00%	real	
ps_calc_04	0.00%	real	
ps_calc_05	0.00%	real	
ps_calc_06	0.00%	real	
ps_calc_07	0.00%	real	
ps_calc_08	0.00%	real	
ps_calc_09	0.00%	real	
ps_calc_10	0.00%	real	
ps_calc_11	0.00%	real	
ps_calc_12	0.00%	real	
ps_calc_13	0.00%	real	
ps_calc_14	0.00%	real	

Pre-processing and Feature Extraction

The rough order of my work is as follows:

1.A simple prepreocessing to make data available for model

2.Build 1st stage model

3.Build 2nd stage model

4.Pamameter tuning

5.preprocessing in detail

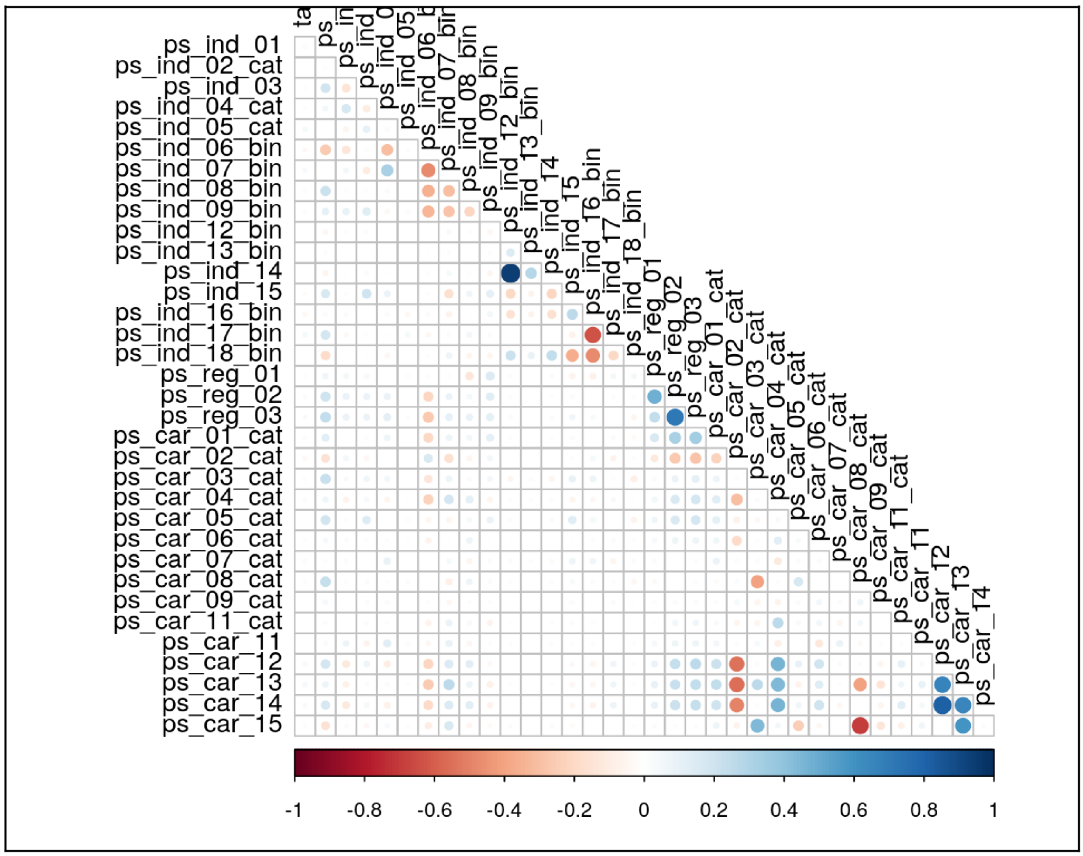
6.feature engineering in detail

7.Parameter tuning in detail

Feature Augmentation

Since there's no explanation on what each feature represent, I cannot extract features based on semantic meaning. Instead, the correlation between features is observed and shown as a heat map. Correlation involved with binary/catagory columns should be excluded, since it is easy for CART to split them.

In [7]:
```
plt.figure(dpi=400, figsize=(10, 6))
plt.xticks([])
plt.yticks([])
imgplot = plt.imshow(mpimg.imread('img/correlation.png'))
```



I've tried adding new features by multiplying pair of features whose correlation are very strong, and the feature importance ranks high as well

```
which are
    'ps_car_13' and 'ps_reg_03'
    'ps_reg_02' and 'ps_reg_03'
    'ps_car_12' and 'ps_car_13'
    'ps_car_13' and 'ps_car_15'
```

And I think numbers of missing values for each row could also be regarded as information, so I put that feature into dataset as well

However, all of the approach above fail to imporve the score. So the final version of my submission this part is commented out.

Feature Selection

At first, all features are put into the model. After training, feature importance can be attached by the attribute of GBDT. Then I iteratively drop the last important feature and run cross validation, until the performance drop by a certain threshold. I manually implement this method under LightGBM, since XGBoost is really slow.

Principle for Boosting Trees feature selection[iii]

Importance is calculated for a single CART by the amount that each attribute split point improves the performance measure, weighted by the number of observations the node is responsible for. The performance measure may be the purity (Gini index) used to select the split points or another more specific error function. The feature importances are then averaged across all of the the decision trees within the model.

Final Appraoch[iv]

The final feature selection criteria is from post on Kaggle. Oliver shared his feature selection strategy, without specific method provided but do perform well than the algorithm-integrated method.

Fill Nan values

After feature selection, there's still some remaining features who has empty values. Methods below has been tried:

drop column

Drop all column that has missing values

fill with values

While implementing the base model, I fill all Nan with -1. But it doesn't make sense when the features distribute as real number. So I give non-binary/non-catagory features with mean value. As a final approach, samples are first grouped by high-importance binary-catagorical features, then the local mean is assigned for each group.

All validation and testing samples are filled with mean values(grouped mean values) generated from training samples.

Empirical Bayesian Encoding[ii]

Since ps_car_11_cat, who has 104 catagories, is added to the feature space, it is a heavy load if it is dummied. Fortunately, A method of changing catagorical feature to real feature is put forward. For each selected catagorical features

The basic method is, for a single catagorical feature, we want to map each of its value to a certain probability leading to its target label:

for each $instance$ in $samples$:

$$instance[encoded] = \hat{P}(label = instance[label] \mid variable = instance[original])$$

And the above model is generated by a empirical bayesian approach

$$\hat{P}(target = y) = \frac{\#\ of\ target = y}{\#\ of\ instances}$$

$$\hat{P}(target = y | variable = k) = \frac{\#\ of\ target = y\ and\ variable = k}{\#\ of\ variable = k}$$

Finally, only features who has cardinality larger or equal than 13 is implemented, the other features are dummied.

For testing/validation samples, the model trained from training samples is reused for encoding

Dummy Variables

Dummy variable is a traditional method for catagorical features. It could transfer a catagorical column to a sparse matrix, with each column represents whether a specific value for that feature appears. For features with less catagory, it performs better than empirical bayesian encoding in practice.

Stardardlization

Stardardlization is the method to modify the value of each column to a certain range. Since gradient descent is used in GBDT only on residue part, there's actually no difference. Principlely it does improve the speed of Logistic Regression, but it is fast enough compared to other model.

min-max standardization is used:

for column in training_set:

$$min[column] = training\_set[column].min()$$

$$max[column] = training\_set[column].max()$$

for column in dataset(training/validation/testing):

$$dataset[column] = \frac{dataset[column] - min[column]}{max[column] - min[column]}$$

Training Process

Parameters for XGBoost

two instances of XGBoostClassifier is trained

```
param1 = {
    'objective': 'binary:logistic',
    'eval_metric': 'auc',
    'eta': 0.02,
    'max_depth': 4,
    'subsample': 0.9,
    'colsample_bytree': 0.9,
    'seed': 99,
    'silent': True,
    'scale_pos_weight':1.6,
    }
```

```
param2 = {
    'objective': 'binary:logistic',
    'eval_metric': 'auc',
    'eta': 0.04,
    'max_depth': 5,
    'min_child_weight': 9.15,
    'gamma': 0.59,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'alpha': 10.4,
    'lambda': 5,
    'seed': 2017,
    'nthread': 5,
    'silent': True,
    'scale_pos_weight':1.6,
    }
```

complexity of hypothesis set

All input matrix has 62 features.

$\frac{4}{5}$(about 476170) of the original training set is used to train each XGBoost instance, $\frac{1}{5}$(about 119042) of the original training set is used as validation set as well as the input of 2nd stage model, 892816 samples are used as test sets.

$119042 * \frac{1}{5}$ * 5 folds * 2 parameters = 2 * 119042 samples is used as 2 features for stage2 model

892816 * 5 folds * 2 parameters = 2 * 892816 samples is used as 2 features for stage2 model(output between each fol

Parameters for LightGBM

two instances of LightGBMClassifier is trained

```
param1 = {
    'learning_rate' : 0.02,
    'n_estimators' : 650,
    'max_bin' : 10,
    'subsample' : 0.8,
    'subsample_freq' : 4,
    'colsample_bytree' : 0.8,
    'min_child_samples' : 700,
    'seed' : 99,
    'scale_pos_weight': 1.6,
    }
```

```
param2 = {
    'learning_rate' : 0.02,
    'n_estimators' : 1100,
    'subsample' : 0.7,
    'subsample_freq' : 1,
    'num_leaves' : 18,
    'seed' : 99,
    'reg_lambda': 15,
    'scale_pos_weight': 1.6,
    }
```

complexity of hypothesis set

All input matrix has 62 features.

$\frac{4}{5}$(about 476170) of the original training set is used to train each LightGBM instance, $\frac{1}{5}$(about 119042) of the original training set is used as validation set as well as the input of 2nd stage model, 892816 samples are used as test sets.

$119042 * \frac{1}{5}$ * 5 folds * 2 parameters = 2 * 119042 samples is used as 2 features for stage2 model

892816 * 5 folds * 2 parameters = 2 * 892816 samples is used as 2 features for stage2 model(output between each fol

Parameters for Logistic Regression

paramters for logistic regression is chosen by heuristics, since grid searching needs validation set, but I didn't prepare validation set for this 2nd stage ensembled model. Besides, parameters for linear model would have less effect compared with 1st stage ones.

```
param = {
    'penalty' : 'l2,
    'C' : 1.0,
    'solver' : 'liblinear',
    }
```

complexity of hypothesis set

595212 samples with 4 features(2 models with 2 parameter settings) is used as input of training, and the trained model is used to predict 892816 samples with 4 features(2 models with 2 parameter settings).

avoid overfitting and underfitting

To avoid underfitting, boosting trees itself is a kind of method to minimize the residual bias, so it is easier to get low bias compared with random forest.

To avoid overfitting, random feature selection and sample selection is done for each meta-learner. alpha, lambda is set to regularize β, min_child_weight, max_depth is set to avoid overfitting for each meta-learner. Stacking is implemented as a higher-level ensembling to avoid overfitting.

Testing, Validation and Model Selection

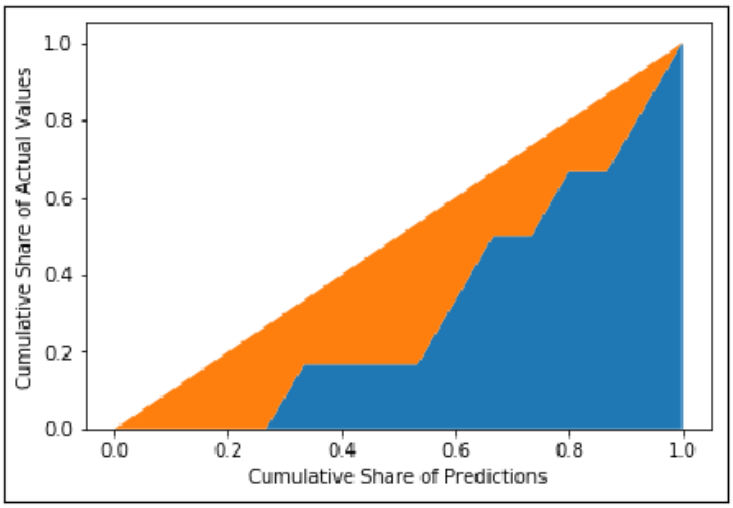
Score criteria

Using traditional error rate is meaningless since the dataset is highly unbalanced, Kaggle use normalized gini-index to evaluate the prediction.

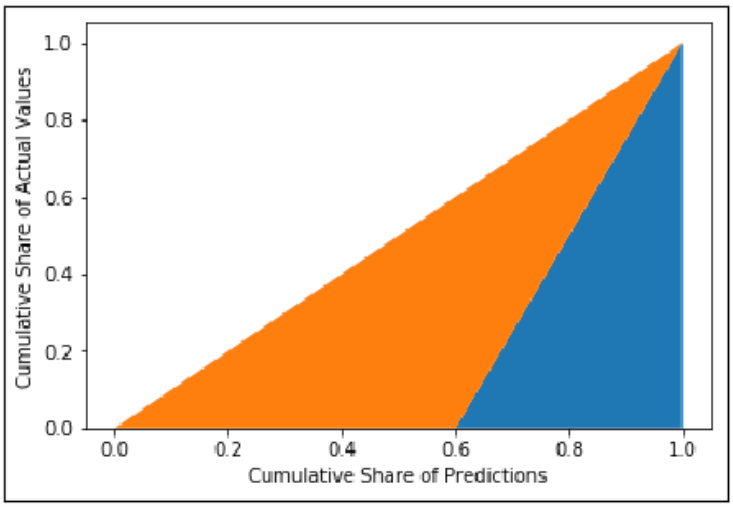
$GiniIndex = 2 * AUC - 1$, where AUC means area under curve.

The curve is drawn with x-axis as cumulative number of predictions and y-axis as cumulative actual values

In [17]:
```
plt.figure(dpi=100)
plt.xticks([])
plt.yticks([])
imgplot = plt.imshow(mpimg.imread('img/gini1.png'))
```



In [18]:
```
plt.figure(dpi=100)
plt.xticks([])
plt.yticks([])
imgplot = plt.imshow(mpimg.imread('img/gini2.png'))
```



Use the two image above as examples. First image is my prediction on the dataset, while the second image reflects the fact(True or False)

predictions is sorted by value, take x-axis as percentage of samples, y is the cumulative value of predictions. Methods are the same as for fact, except that since only 0,1 are in the label, the orange area shape like a triangle.

Finally, the ratio of areas of two orange triangles are used as the score.

So we should use gini-index(auc) as the loss function and the output should be a probabilty but not thresholded binary value.

Model Selection

The first stage model is selected by the cross validation scores. For each model, a script with aspect-oriented module was programmed to fast establish the function for grid search. Then the average score could be obtained. First, parameters need to be fine-tuned to a acceptable level. Then, if its score has huge gap between my leading models, just throw them away, otherwise, if it is around the average performance, try to add it as the feature of the stacking model and test it on Public LB. If it increases or alomst the same, keep it for further testing.

e.g. Random Forest is considered as a meta-learner first. But after fine-tuning and stacking, there's a gap between its peformance and that of boosting trees, and it negatively affects performance of stacking, so it is abandoned finally.

Parameter tuning

the methods came up by searching from forum/internet or came up by myself, validation scores and Public LB scores are referenced to decide whether the approach should be used finally.

Grid Search

GridSearch is a method to tune parameters for models. First some guess for each paramter is put into a map. And for each combo in that list, cross validation is done and a final score is given. The final output is the parameter with the best score.

Procedure

Since we have four boosting tree models, to avoid high variance, the parameter setting for each model should be different. At first, four different starting points are selected.(e.g. XGBoost parameter stops by the decrease on validation scores, but LightGBM is set to stop at certain iteration). And then I do fine-tuning by grid search on each parameter set and let them converge to their local minimum.

Attempts

A normalized gini-index score is the evaluation for each model. For each attempt, a validation score for each stage1 model is given, and public leaderboad(aka. testing) score might be given, due to the limit of sumbission times.

I didn't record my training score, it is between 0.32 to 0.35 for each single boosting trees.

In [8]:
```
plt.figure(dpi=400, figsize=(10, 6))
plt.xticks([])
plt.yticks([])
imgplot = plt.imshow(mpimg.imread('img/attempts.png'))
```

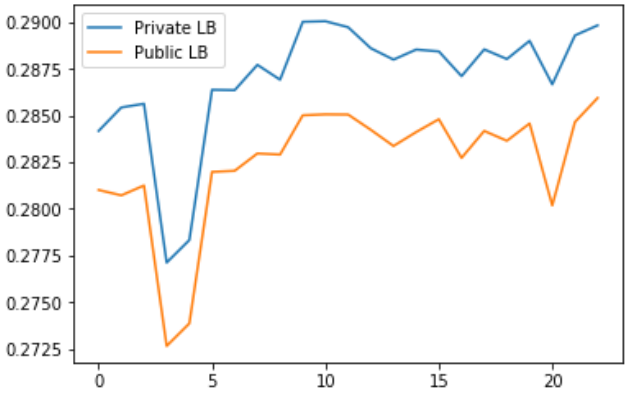
method	XGB1	XGB2	LGBM1	LGBM2	PublicLB	other
XGB	0.2835				0.280	
LGBM1	0.2844				0.281	
LGBM2	0.2833				0.281	
RF						0.2589
stack(balanced)					0.272	
stack(weighted)					0.273	
stack(balanced_withoutRF)					0.281	
stack(weighted_withoutRF)					0.282	
dum_all	0.2840		0.2842	0.2838	0.283	
augmen	0.2844		0.2851	0.2839	0.282	
Fea_Sel	0.2856		0.2871	0.2854	0.285	
rmNanRo	0.2529		0.2502	0.2510		
std_No	0.2856					
std_mea	0.2856					
srd_Mm	0.2856					
mean-1	0.2866		0.2881	0.2868	0.285	
Ecod1	0.2881		0.2882	0.2880	0.285	
Ecod>12	0.2882	0.2895	0.2892	0.2881		
Ecod>9	0.2884		0.2892	0.2883	0.285	
Ecod>3	0.2843	0.2858	0.2850	0.2849		
w1:1	0.2884	0.2895	0.2892	0.290		
w1.6:1	0.2886	0.2900	0.2891	0.2898	0.284	
gmean-1	0.2892	0.2896	0.2897			
gmean-1	0.2896	0.2891	0.2897			

Final Results

submission	Private LB	Public LB
1.	0.28986	0.28596
2.	0.28932	0.28467
3.	0.28669	0.28019
4.	0.28903	0.28459
5.	0.28805	0.28366
6.	0.28857	0.28419
7.	0.28713	0.28274
8.	0.28846	0.28482
9.	0.28856	0.28413
10.	0.28802	0.28338
11.	0.28863	0.28425
12.	0.28976	0.28507
13.	0.29008	0.28508
14.	0.29005	0.28503
15.	0.28694	0.28292
16.	0.28774	0.28298
17.	0.28638	0.28205
18.	0.28640	0.28199
19.	0.27833	0.27385
20.	0.27712	0.27264
21.	0.28565	0.28126
22.	0.28545	0.28073
23.	0.28419	0.28102

In [2]:
```
A = [0.28986,0.28932,0.28669,0.28903,0.28805,0.28857,0.28713,0.28846,
0.28856,0.28802,0.28863,0.28976,0.29008,0.29005,0.28694,
0.28774,0.28638,0.28640,0.27833,0.27712,0.28565,0.28545,0.28419]

B = [0.28596,0.28467,0.28019,0.28459,0.28366,0.28419,
0.28274,0.28482,0.28413,0.28338,0.28425,0.28507,
0.28508,0.28503,0.28292,0.28298,0.28205,0.28199,
0.27385,0.27264,0.28126,0.28073,0.28102]
plt.plot(A[::-1], label = 'Private LB')
plt.plot(B[::-1], label = 'Public LB')
plt.legend(loc='upper left')
plt.show()
```



The result I have chosen ranks 20% of this Kaggle Competition.

Interpretaion

Understanding from my approach

Since the competition on Kaggle has ended, I want to analyze the Pattern Recognition method with the effect on the Final result

1.Tree Model could outperform much more than other model, and easy for parameter tuning and faster for training, so it should be considered first when paticipating in such jobs

2.Empirical Bayesian Encoding would speeds up the training process, especially when there are catagory-type features with a great amount of values. But it doesn't make the performance better. Next time I would use it with cautiously.

3.The share from top-ranked Kagglers indicates that fine-tuned Neural Network can perform as well as Boosting Trees. It's reasonable taking it as meta classifier when time allows.

4.While modifying the stage 1 model, trust CV score need to be taken as important as public LB, since public LB is only part of private LB, overfitting to public doesn't lead to a higher score finally

Understandinf from top-ranked kagglers

Both the 1st place and 2nd place has shared their method.

The first place used blending of 1 LGBM and 5 NN. without deep feature engineering, but with a autoencoder. A autoencoder could remove the noise and automatically fill the Nan values. And the 2nd ranked Kaggler is also doing the similar thing. They use cluster of features to predict the value of other cluster. It is also a kind of encoding method.

In addition, by using this method, all the testing samples could also be used, since the label is not necessary, in order to get a higher rank.

Summary and conclusions

In this competition, I've learned the principle of empirical bayesian encoding. I've heard about auto encoder, mainly in the region of image processing, but never heard it could be used in this approach. Next time in Kaggle, I'll put my time struggling with feature engineering to Neural Network, I believe I could reach a higher score next time.

Reference

[i]Friedman, *Greedy Function Approximation: A Gradient Boosting Machine*

[ii]Daniele Micci-Barreca, *A Preprocessing Scheme for High-Cardinality Categorical Attributes in Classification and P*

[iii]Trevor Hastie, Robert Tibshirani, Jerome Friedman, *The Elements of Statistical Learning: Data Mining, Inference, a*

[iv]Oliver, *https://www.kaggle.com/ogrellier/xgb-classifier-upsampling-lb-0-283/code*

[v]Michael Jahrer, *https://www.kaggle.com/c/porto-seguro-safe-driver-prediction/discussion/44629*