step03-prudential-kaggle-160130

February 6, 2016

In [1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from ml_metrics import quadratic_weighted_kappa
import xgboost as xgb
import datetime as dt
import sklearn
from sklearn.cross_validation import train_test_split
from sklearn.cross_validation import KFold
import functools
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import precision_score
from scipy import optimize

from xgboostmodel import XGBoostModel, ModelPrediction
```

1 Step 03. Optimized classification

1.0.1 Goal 1: Implement class which allows easy combination of boosters to make predictions

1.0.2 Goal 2: Optimize the classify function, that converst scores to categories

We start out by fitting our best `reg:linear` and `multi:softmax` models which we obtained in Step 02:

In [2]:
```
booster = XGBoostModel(nfolds=3)

fold = 1 # the fold to use for train-test split

booster.learn_model(fold, objective='reg:linear', num_round=250,
                    make_plot=False,
                    eta=0.06,
                    max_depth=9,
                    min_child_weight=150,
                    colsample_bytree=0.8,
                    subsample=0.8)

booster.learn_model(fold, objective='multi:softmax', num_round=250,
                    make_plot=False,
                    eta=0.06,
                    max_depth=8,
                    min_child_weight=50,
                    colsample_bytree=0.8,
```

```
                                subsample=0.8)

        booster.get_scores()[['objective', 'train_qwk', 'test_qwk']]
```

Out[2]:

	objective	train_qwk	test_qwk
0	reg:linear	0.674255	0.603402
1	multi:softmax	0.615971	0.550655

2 We define the ComboPredict class

In [3]:
```
#===============================================================================

class ComboPredict:

    def __init__(self, booster):
        """
        Intitialize the combination predictor with an XGBoostModel instance

        Parameters
        ----------

        booster : XGBoostModel

        """

        self.booster = booster

        if (len(self.booster.models) == 0):
            raise ValueError("The XGBoostModel provided does not contain any "
                             "fitted models.")


    def predict_score(self, features, overall_cls_factor):
        """
        Predicts scores for a set of observations.  The score is calculated
        by weighting all of the models present in the XGBoostModel that is
        passed into the class at initialization.

        Note that the scores output by this function need to be coerced to
        category values for a final prediction.

        Parameters
        ----------

        features : array

            Features for which predictions weill be generated

        overall_cls_factor : float

            Relative weight of the classification boosters wrt to the regression
            boosters

        """
```

```
        xg_input = xgb.DMatrix(features)

        weighted_preds = []
        norms = []
        for m in zip(booster.models, booster.scores):
            model, model_fold, model_pred = m[0]
            score = m[1]

            nfeatures = xg_input.num_row()
            X, _ = np.meshgrid(np.arange(8), np.arange(nfeatures))

            if score['objective'] == 'multi:softmax':
                pred_cls = model.predict(xg_input,
                                         ntree_limit=model.best_iteration)
                dummies = pd.get_dummies(pred_cls).values
                weight = model_pred.precisiontrain.reshape(8,1)

                weighted_pred = np.dot(X * dummies, overall_cls_factor * weight)
                weighted_preds.append(weighted_pred)

                norm = np.dot(dummies, overall_cls_factor * weight )
                norms.append(norm)

            else:
                reg_pred = model.predict(xg_input,
                                         ntree_limit=model.best_iteration)
                weighted_preds.append(reg_pred.reshape(nfeatures, 1))
                norms.append(np.ones_like(weighted_preds))

        total = np.sum(weighted_preds, axis=0)
        norm = np.sum(norms, axis=0)

        combo_score = np.squeeze(total / norm)
        return combo_score
```

We demonstrate the use of ComboPredict to calculate scores:

```
In [4]: combo = ComboPredict(booster)

        # All of the models in booster were trained with fold=1:
        assert all(fold == 1 for _, fold, _ in booster.models)

        # We get the features and labels for fold=1
        train, test = booster.make_cv_split(1, returnxgb=False)
        # train, test are each a tuple of the form (features, labels)

        overall_cls_factor = 0.4
        score_train = combo.predict_score(train[0], overall_cls_factor)
        score_test = combo.predict_score(test[0], overall_cls_factor)
```

We then use rounding to the neareset integer to produce categories, and we check kappa for the results:

```
In [5]: def classify(score):
            score = np.asarray(score)
```

```
        return np.rint(np.clip(score, -0.49, 7.49))

    yhcombotrain = classify(score_train)
    yhcombotest = classify(score_test)

    print("train qwk = {:0.5f}".format(quadratic_weighted_kappa(yhcombotrain,
                                                                  train[1])))

    print(" test qwk = {:0.5f}".format(quadratic_weighted_kappa(yhcombotest,
                                                                  test[1])))
```

```
train qwk = 0.67922
 test qwk = 0.60933
```

We recover the results of Step 02, where it was shown that using a weighing ratio 0.4:1 (classification:regression) contribution to the score was beneficial for the quadratic weighted kappa of our predictions.

3 Classification with cutoffs

Rather than rounding to the nearest integer, we implement a function that allows setting score cuttofs which map to each of the eight possible categories:

```
In [6]: def classify_with_cutoffs(yscore, cutoffs):
            """
            Receives a list of seven cutoffs, which will determine the mapping from
            scores to categories.

            Parameters
            ----------

            predicted_score : array

                Array of predicted scores, which will be mapped onto categories
                according to cutoffs

            cutoffs : array

                Array of length 7 (num_categories - 1).
            """
            assert len(cutoffs) == 7
            cutoffs = np.sort(cutoffs)
            return np.digitize(yscore, cutoffs).astype('int')


        print("EXAMPLES OF classify_with_cutoffs:\n")
        cutoffs0 = np.arange(7)+0.5
        for val in list(np.random.rand(3)*9. -2.) + [5.49, 5.51]:
            cat = classify_with_cutoffs(val, cutoffs0)
            print("score={:.2f} => category={}".format(val, cat))
```

```
EXAMPLES OF classify_with_cutoffs:

score=6.25 => category=6
score=5.04 => category=5
```

```
score=0.61 => category=1
score=5.49 => category=5
score=5.51 => category=6
```

3.0.1 We define a function for optimizing the cutoffs:

The cutoffs get "trained" based on the calculated score and the known true labels.

```
In [7]: def optmize_cutoffs_simplex(yscore, ytrue, *, verbose=False):
            """
            Receives an array of predicted scores, and an array of true values.
            Determines which cutoff values make for the best prediction with
            respect to the true values.

            Parameters
            ----------

            yscore : array

                Array of predicted scores

            ytrue : array

                Array of true  values

            verbose : bool (optional)

                When true prints the std dev of y-ypred before and after
                optimization.
            """

            yscore = np.asarray(yscore, dtype=np.float64)
            ytrue = np.asarray(ytrue, dtype=np.float64)


            cutoffs0 = np.arange(7)+0.5

            def error(p):
                errors = classify_with_cutoffs(yscore, p).astype(np.float64) - ytrue
                return np.std(errors)

            just = 15
            if verbose:
                print("{} : {}".format(
                        "start error".rjust(just), error(cutoffs0)))

            #xopt, fopt, niter, funcalls, warnflag, allvecs
            pfit = optimize.fmin_powell(error, cutoffs0, xtol=1e-2, ftol=1e-6,
                                        maxiter=None, maxfun=None)

            if verbose:
                print("{} : {}\n".format(
                        "final error".rjust(just), error(pfit)))

            return pfit
```

```
        pfit = optmize_cutoffs_simplex(score_train, train[1], verbose=True)

        print("OPTIMIZED CUTOFFS:")
        print(pfit)
```

```
start error : 1.704812979226756
Optimization terminated successfully.
         Current function value: 1.696007
         Iterations: 9
         Function evaluations: 839
    final error : 1.6960072842910539

OPTIMIZED CUTOFFS:
[ 0.32229194  1.63951265  2.67249365  3.63436993  4.76973765  5.63572747
  6.37087037]
```

4 Predict for submission

We know go back to the submission samples and make a prediction based on our current model.

```
In [8]: data = pd.read_csv('csvs/data_imputed.csv')
        features = data[data['train?'] == True].drop(['train?', 'Id', 'Response'],
                                                     axis=1)

        labels = data[data['train?'] == True]['Response'].astype('int') -1

        submission_features = data[data['train?'] == False].drop(['train?',
                                                                  'Id',
                                                                  'Response'],
                                                                 axis=1)

        combo = ComboPredict(booster)

        # All of the models in booster were trained with fold=1:
        assert all(fold == 1 for _, fold, _ in booster.models)

        overall_cls_factor = 0.4
        score_submission = combo.predict_score(submission_features, overall_cls_factor)

        best_cutoffs = np.array([ 0.3223,  1.6395,  2.6725,  3.6344,
                                 4.7697,  5.6357,  6.3709])
        yhcombo_submission = classify_with_cutoffs(score_submission, best_cutoffs)

        submission_ids = data[data['train?'] == False]['Id']
        submission_df = pd.DataFrame({"Id": submission_ids,
                                      "Response": yhcombo_submission.astype('int') + 1})

        submission_df = submission_df.set_index('Id')
        submission_df.to_csv('step03_submission.csv')

        submission_df.describe()
```

```
Out[8]:            Response
        count  19765.000000
```

```
mean       5.721882
std        1.804223
min        1.000000
25%        5.000000
50%        6.000000
75%        7.000000
max        8.000000
```

The submission obtained above scores 0.63806 in Kaggle's leaderboard. The best score at the moment (02/06 at 19:08) is 0.68271.

5 Next step: feature engineering

In the next step I will do some exploratory data analysis, and will come up with customized features that may help improve the score.