Bias Correction + XGBoost

37 voters

by Tilii · last run 2 days ago · Python script · 2019 views
using data from Allstate Claims Severity · Public

Code

Download Code

```
import numpy as np
import pandas as pd
from datetime import datetime
from sklearn.preprocessing import StandardScaler
from sklearn.cross_validation import KFold
from sklearn.metrics import mean_absolute_error
from scipy.stats import skew, boxcox
from math import exp, log
import xgboost as xgb


def timer(start_time=None):
    if not start_time:
        start_time = datetime.now()
        return start_time
    elif start_time:
        tmin, tsec = divmod((datetime.now() - start_time).total_seconds(),
60)
        print(' Time taken: %i minutes and %s seconds.' %
              (tmin, round(tsec, 2)))


def scale_data(X, scaler=None):
    if not scaler:
        scaler = StandardScaler()
        scaler.fit(X)
    X = scaler.transform(X)
    return X, scaler


```

```
DATA_TRAIN_PATH = '../input/train.csv'
DATA_TEST_PATH = '../input/test.csv'


def load_data(path_train=DATA_TRAIN_PATH, path_test=DATA_TEST_PATH):
    train_loader = pd.read_csv(path_train, dtype={'id': np.int32})
    train = train_loader.drop(['id', 'loss'], axis=1)
    test_loader = pd.read_csv(path_test, dtype={'id': np.int32})
    test = test_loader.drop(['id'], axis=1)
    ntrain = train.shape[0]
    ntest = test.shape[0]
    train_test = pd.concat((train, test)).reset_index(drop=True)
    numeric_feats = train_test.dtypes[train_test.dtypes != "object"].index

    # compute skew and do Box-Cox transformation
    skewed_feats = train[numeric_feats].apply(lambda x: skew(x.dropna()))
    print("\nSkew in numeric features:")
    print(skewed_feats)
    # transform features with skew > 0.25 (this can be varied to find
optimal value)
    skewed_feats = skewed_feats[skewed_feats > 0.25]
    skewed_feats = skewed_feats.index
    for feats in skewed_feats:
        train_test[feats] = train_test[feats] + 1
        train_test[feats], lam = boxcox(train_test[feats])
    features = train.columns
    cats = [feat for feat in features if 'cat' in feat]
    # factorize categorical features
    for feat in cats:
        train_test[feat] = pd.factorize(train_test[feat], sort=True)[0]
    x_train = train_test.iloc[:ntrain, :]
```

Code Output (3) Comments (21) Log Versions (1) Forks (13) Fork Script

```
    train_test_scaled, scaler = scale_data(train_test)
    train, _ = scale_data(x_train, scaler)
    test, _ = scale_data(x_test, scaler)

    train_labels = np.log(np.array(train_loader['loss']))
    train_ids = train_loader['id'].values.astype(np.int32)
    test_ids = test_loader['id'].values.astype(np.int32)

    return train, train_labels, test, train_ids, test_ids

#################################### Actual Run Code
####################################

# enter the number of folds from xgb.cv
folds = 5
```

```
cv_sum = 0
early_stopping = 25
fpred = []
xgb_rounds = []

start_time = timer(None)

# Load data set and target values
train, target, test, _, ids = load_data()
d_train_full = xgb.DMatrix(train, label=target)
d_test = xgb.DMatrix(test)

# set up KFold that matches xgb.cv number of folds
kf = KFold(train.shape[0], n_folds=folds)
for i, (train_index, test_index) in enumerate(kf):
    print('\n Fold %d\n' % (i + 1))
    X_train, X_val = train[train_index], train[test_index]
    y_train, y_val = target[train_index], target[test_index]

#######################################
#
# Define cross-validation variables
#
#######################################

    params = {}
    params['booster'] = 'gbtree'
    params['objective'] = "reg:linear"
    params['eval_metric'] = 'mae'
    params['eta'] = 0.1
    params['gamma'] = 0.5290
```

Code Output (3) Comments (21) Log Versions (1) Forks (13) Fork Script

```
    params['subsample'] = 0.9930
    params['max_depth'] = 7
    params['max_delta_step'] = 0
    params['silent'] = 1
    params['random_state'] = 1001

    d_train = xgb.DMatrix(X_train, label=y_train)
    d_valid = xgb.DMatrix(X_val, label=y_val)
    watchlist = [(d_train, 'train'), (d_valid, 'eval')]

####################################
#  Build Model
####################################

```

```
    clf = xgb.train(params,
                    d_train,
                    100000,
                    watchlist,
                    early_stopping_rounds=early_stopping)

####################################
#  Evaluate Model and Predict
####################################

    xgb_rounds.append(clf.best_iteration)
    scores_val = clf.predict(d_valid, ntree_limit=clf.best_ntree_limit)
    cv_score = mean_absolute_error(np.exp(y_val), np.exp(scores_val))
    print(' eval-MAE: %.6f' % cv_score)
    y_pred = np.exp(clf.predict(d_test, ntree_limit=clf.best_ntree_limit))

####################################
#  Add Predictions and Average Them
####################################

    if i > 0:
        fpred = pred + y_pred
    else:
        fpred = y_pred
    pred = fpred
    cv_sum = cv_sum + cv_score

mpred = pred / folds
score = cv_sum / folds
print('\n Average eval-MAE: %.6f' % score)
n_rounds = int(np.mean(xgb_rounds))
```



```
#  Make Full Dataset Predictions
####################################

print('\n Training full dataset for %d rounds ...' % n_rounds)
watchlist = [(d_train_full, 'train')]
clf_full = xgb.train(
    params, d_train_full,
    n_rounds,
    watchlist,
    verbose_eval=False,)
y_pred_full = np.exp(clf_full.predict(d_test))

# enter the number of iterations from xgb.cv with early_stopping turned on
n_fixed = 376
```

```
nfixed = int(n_fixed * (1 + (1. / folds)))
print('\n Training full dataset for %d rounds ...\n' % nfixed)
clf_fixed = xgb.train(
    params, d_train_full,
    nfixed,
    watchlist,
    verbose_eval=False,)
y_pred_fixed = np.exp(clf_fixed.predict(d_test))
timer(start_time)

print("#\n Writing results")
result = pd.DataFrame(mpred, columns=['loss'])
result["id"] = ids
result = result.set_index("id")
print("\n %d-fold average prediction:\n" % folds)
print(result.head())
result_full = pd.DataFrame(y_pred_full, columns=['loss'])
result_full["id"] = ids
result_full = result_full.set_index("id")
print("\n Full dataset prediction:\n")
print(result_full.head())
result_fixed = pd.DataFrame(y_pred_fixed, columns=['loss'])
result_fixed["id"] = ids
result_fixed = result_fixed.set_index("id")
print("\n Full datset (at CV #iterations) prediction:\n")
print(result_fixed.head())

now = datetime.now()
score = str(round((cv_sum / folds), 6))
sub_file = 'submission_5fold-average-xgb_' + str(score) + '_' + str(
```

Code Output (3) Comments (21) Log Versions (1) Forks (13) Fork Script

```
result.to_csv(sub_file, index=True, index_label='id')
sub_file = 'submission_full-average-xgb_' + str(now.strftime(
    "%Y-%m-%d-%H-%M")) + '.csv'
print("\n Writing submission: %s" % sub_file)
result_full.to_csv(sub_file, index=True, index_label='id')
sub_file = 'submission_full-CV-xgb_' + str(now.strftime(
    "%Y-%m-%d-%H-%M")) + '.csv'
print("\n Writing submission: %s" % sub_file)
result_fixed.to_csv(sub_file, index=True, index_label='id')
```

show less

Output

Download All

3 files

- submission_5fold-ave...
- submission_full-aver...
- submission_full-CV-x...

submission_5fold-average-xgb_1146.10852_2016-10-13-02-40.csv

Download File

id	loss
4	1474.300048828125
6	2071.08837890625
9	7915.3203125
12	5771.5576171875
15	811.8914794921875
17	2246.956298828125
21	2125.173583984375
28	881.2107543945312

Comments

6

Tilii · last edited 2 days ago by Tilii
2 days ago

This script features **Box-Cox** transformation to correct the skew in continuous variables. XGBoost is used to mop up the mess.

There are 3 different ways of making submission files. One is a simple numeric average of 5 models, each of which is trained on 80% of the data until early stopping criterion is reached. Next, full dataset

Code Output (3) Comments (21) Log Versions (1) Forks (13) Fork Script

Leaderboard:

5fold-average **1118.10987**

full-average 1122.27500

full-CV 1122.42771

At the time of this writing, 5-fold average score is #14 on the leaderboard.

0

modkzs
a day ago

I am little confused why Box-Cox transformation can make such improvement. Can normal distributed feature cause better xgboost performance?

2

LzjtuEr
a day ago

@modkzs, since XGB uses linear regression as the objective, the assumption of linear regression is that the features follow normal distribution.

5

Tilii · last edited a day ago by Tilii
a day ago

> I am little confused why Box-Cox transformation can make such improvement. Can normal distributed feature cause better xgboost performance?

Violations of normal distribution can increase the error in regression analysis. The skew in data is frequently "fixed" by log(N) (or log(N+1)) transformation. However, log(N) transformation is the best only when Box-Cox lambda factor is close to zero. As it is, most of the skewed columns in our dataset have lambda values that are far from zero (see below). There are better **transformations** than log(N) for different values of lambda and Box-Cox procedure finds the best one for a given dataset.

Edit: Below are lambda values of skewed features. log(N) would not be a best transformation for most of them.

-0.585084895197

-0.995197981798

-3.00787936887

-2.2837778627

-2.27901724839

-2.56311666326

-0.613891833259

-0.437372340485

-0.47055182116

-1.44082551851

0

Andrey Shkabko

a day ago

Training full dataset for 366 rounds ...

ZeroDivisionError Traceback (most recent call last) in () 162 n_rounds, 163 watchlist, --> 164 verbose_eval=False,) 165 y_pred_full = np.exp(clf_full.predict(d_test)) 166

ZeroDivisionError: integer division or modulo by zero

^^^ python 2.7

2

Tilii
a day ago

> Training full dataset for 366 rounds ...

I have no idea what causes this error. If you look at the **log** file (scroll all the way down), it runs just fine on Kaggle. The same script runs on my computer as well, and I have python 2.7.

Maybe you have one of the packages (or more) that are not recent enough. My setup is:

numpy==1.11.0rc2 pandas==0.18.0 scikit-learn==0.17.1 scipy==0.17.0 xgboost==0.4

I don't have the latest xgboost, but that shouldn't be a problem as Kaggle does, and this script works on Kaggle.

0

Andrey Shkabko
a day ago

numpy 1.11.1 pandas 0.18.1 sklearn 0.17.1 scipy 0.18.1 xgboost 0.4

17 hours ago

Nice script, I learned a lot. One question, when I train GBMs I usually use subsample of something around .5 because it allows for stable oob error measures and because of this example: http://scikit-learn.org/stable/auto_examples/ensemble/plot_gradient_boosting_regularization.html. Admittedly, I haven't had a lot of experience with them and the example above is just anecdotal. Would you be so kind as to explain the choice of params['subsample'] = 0.9930? Thanks again

2

Tony S
15 hours ago

I've been thinking about the normality assumption that was brought up here behind the reasoning for the transformation and I think there is something more mysterious going on here.

Isn't the normality assumption in linear regression only assuming that the residual errors are normally distributed? If so, that doesn't necessarily imply that the variables themselves need to be normally distributed. For example, a simple linear correlation such as y=2x+epsilon (a small normally distributed error) could have an almost perfect one-to-one correlation and work great in linear regression; x in this case could be any distribution, not just a normal distribution.

I'd love for someone to give me more information or correct me if I am making an incorrect assumption, but it seems to me more likely that you stumbled onto a transformation for one or more of the features that has a better linear correlation with the "loss" variable.

11

Alexandru Papiu
13 hours ago

Nice script and very good score :) I am however a little confused by the comments here. From what I can tell this model is not in any way related to linear regression. It is a boosted model of trees of depth 7. The `['objective'] = "reg:linear"` line is simply telling xgboost that the target is continuous.

Tree-based models do not make any assumptions on the normality of the features. In fact tree based model are **invariant** to monotonic transformations such as taking logs or box-cox. No matter how crazy the distribution of your feature is, the tree will just pick a point to split on.

Out of curiosity I actually ran the script without any log or box-cox transforms and got very similar results: 1122.44270 with full cv, 1118.89063 with 5-fold average.

I am afraid the good score here comes mostly from ensembling a bunch of well-tuned xgboosts.

5

Faron
12 hours ago

@Alexandru I would change it to: "In fact tree based model are **almost invariant** to monotonic transformations".
Why monotonic transforms do affect decision trees

PythonCracker

Code Output (3) Comments (21) Log Versions (1) Forks (13) Fork Script

I have the same question as @AdamWolfeLevin, how did you get the final parameters like gamma=0.5290, min_child_weight=4.2922?

1

Tilii
7 hours ago

> explain the choice of params['subsample'] = 0.9930? I have the same question as @AdamWolfeLevin, how did you get the final parameters like gamma=0.5290, min_child_weight=4.2922?

The parameters were found using global Bayesian optimization as implemented here. See here for an example. Note that most of the code in that example is unnecessary if you run the script locally and have the bayes_opt module installed.

1

Tilii
7 hours ago

> I'd love for someone to give me more information or correct me if I am making an incorrect assumption, but it seems to me more likely that you stumbled onto a transformation for one or more of the features that has a better linear correlation with the "loss" variable.

This is quite possible and I think that Alexandru added similar point of view to your argument.

2

Tilii
7 hours ago

> Tree-based models do not make any assumptions on the normality of the features. In fact tree based model are invariant to monotonic transformations such as taking logs or box-cox. No matter how crazy the distribution of your feature is, the tree will just pick a point to split on.

Funny you should mention this, because I was doing Box-Cox feature manipulations as a preparation for linear models you put to such a good use in this **script**. I tried XGBoost just because it seems to be fashionable these days, and it worked. I am still exploring linear models.

> Out of curiosity I actually ran the script without any log or box-cox transforms and got very similar results: 1122.44270 with full cv, 1118.89063 with 5-fold average.

Yet another thing to try would be a gblinear booster in XGBoost.

> I am afraid the good score here comes mostly from ensembling a bunch of well-tuned xgboosts.

Though I will have a better feel after completing my linear models, this seems like a valid argument.

5 hours ago

> @Alexandru I would change it to: "In fact tree based model are almost invariant to monotonic transformations".

But do they always get better? Or they vary for the same reason they vary when the seed is changed?

0

Tony S
5 hours ago

Thanks @Alexandru, @Faron, and @Tilii for clarification. Makes sense now. The transformation likely has little to no no impact on the gbtree algorithm (or other tree-based algorithms), but it potentially could have some impact on linear models if it provides better linear correlation to loss. I guess I have some food for thought on next steps.

0

algo.ai
3 hours ago

@Tilii

Thanks for another nice contribution!

I would like to know roughly min_child_weight = 4 in XGB is equalivant to what value in sklearn's GradientBoostingRegressor's min_samples_leaf?

0

algo.ai
3 hours ago

1

Tilii
2 hours ago

> I would like to know roughly min_child_weight = 4 in XGB is equalivant to what value in sklearn's GradientBoostingRegressor's min_samples_leaf?

I don't know of any simple conversion procedure between min_child_weight and min_samples_leaf.

It is not that difficult to tune a single parameter, if min_samples_leaf is what interests you. See **here** and **here**.

1

kiddo
2 hours ago

I can confirm that linear regression does not assume predictors are normally distributed. The Box-Cox transformations are nonlinear and can help with the linearity of the relationship between response and predictor in linear regression, but in a regression tree or RF, there should be no difference except for the seed for RF.

Code Output (3) Comments (21) Log Versions (1) Forks (13) Fork Script

Styling with Markdown supported

Enter your comments.

Post Reply

 Our Team Careers Terms Privacy Contact/Support

Code Output (3) Comments (21) Log Versions (1) Forks (13) Fork Script