xgboost-1-Introduction

May 8, 2019

1 Extreme gradient boosting with xgboost

This is an online course provided by datacamp

XGBoost (Extreme Gradient Boosting) belongs to a family of boosting algorithms and uses the gradient boosting framework at its core.

- optimized gradient-boosting machine learning library
- Originally written in C++

Why xgboost?

- speed and performance
- core algorithm is parallelizable
- consistently outperforms single-algorithm methods
- state-of-the-art performance in many ML task

you can use the scikit-learn `.fit() / .predict()` paradigm that you are already familiar to build your XGBoost models, as the xgboost library has a `scikit-learn` compatible API!

Boosting is a sequential technique which works on the principle of an ensemble. It combines a set of weak learners and delivers improved prediction accuracy. At any instant t, the model outcomes are weighted based on the outcomes of previous instant $t - 1$. The outcomes predicted correctly are given a lower weight and the ones miss-classified are weighted higher. Note that a **weak learner** is one which is slightly better than random guessing.

```
In [1]: import textwrap
        import os
        import sys
        import pandas as pd
        import numpy as np
        import xgboost as xgb
        import sklearn as sk
        from sklearn.model_selection import train_test_split

        print("Python version: {0}".format(sys.version))
        print("pandas version: {0}".format(pd.__version__))
        print("numpy version:{0}".format(np.__version__))
        print("sklearn version:{0}".format(sk.__version__))
        print("xgboost version: {0}".format(xgb.__version__))
        print("Working directory: \n" + os.getcwd())
```

```
Python version: 3.7.1 (default, Dec 14 2018, 13:28:58)
[Clang 4.0.1 (tags/RELEASE_401/final)]
pandas version: 0.23.4
numpy version:1.15.4
sklearn version:0.20.1
xgboost version: 0.82
Working directory:
/Users/miaocai/Dropbox/RprojectMiao/xgboost
```

In [2]:
```
# from sk.model_selection import train_test_split
train_url = "http://s3.amazonaws.com/assets.datacamp.com/course/Kaggle/train.csv"
train = pd.read_csv(train_url)
train.head()
# y: survived
```

Out[2]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

In [3]:
```
train.columns
```

Out[3]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [4]:
```
test_url = "http://s3.amazonaws.com/assets.datacamp.com/course/Kaggle/test.csv"
test = pd.read_csv(test_url)
test.head()
```

Out[4]:
```
   PassengerId  Pclass                                          Name     Sex  \
0          892       3                              Kelly, Mr. James    male
```

```
1          893       3             Wilkes, Mrs. James (Ellen Needs)  female
2          894       2                    Myles, Mr. Thomas Francis    male
3          895       3                             Wirz, Mr. Albert    male
4          896       3  Hirvonen, Mrs. Alexander (Helga E Lindqvist)  female

    Age  SibSp  Parch   Ticket     Fare Cabin Embarked
0  34.5      0      0   330911   7.8292   NaN        Q
1  47.0      1      0   363272   7.0000   NaN        S
2  62.0      0      0   240276   9.6875   NaN        Q
3  27.0      0      0   315154   8.6625   NaN        S
4  22.0      1      1  3101298  12.2875   NaN        S
```

split into train and test using scikit learn

from sklearn.model_selection import train_test_split xTrain, xTest, yTrain, ySplit = train_test_split(x, y, test_size = 0.2, radom_state = 0)

```
In [5]: X_columns = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch',
                    'Fare', 'Cabin', 'Embarked']
        X_train = train[X_columns].fillna(0)
        X_train = pd.get_dummies(X_train, columns =
                                 ['Pclass', 'Sex', 'Cabin', 'Embarked'])
        y_train = train['Survived']
        X_test = test[X_columns].fillna(0)

        X_train.head()
```

```
Out[5]:     Age  SibSp  Parch     Fare  Pclass_1  Pclass_2  Pclass_3  Sex_female  \
        0  22.0      1      0   7.2500         0         0         1           0
        1  38.0      1      0  71.2833         1         0         0           1
        2  26.0      0      0   7.9250         0         0         1           1
        3  35.0      1      0  53.1000         1         0         0           1
        4  35.0      0      0   8.0500         0         0         1           0

           Sex_male  Cabin_0     ...      Cabin_F2  Cabin_F33  Cabin_F38  Cabin_F4  \
        0         1        1     ...             0          0          0         0
        1         0        0     ...             0          0          0         0
        2         0        1     ...             0          0          0         0
        3         0        0     ...             0          0          0         0
        4         1        1     ...             0          0          0         0

           Cabin_G6  Cabin_T  Embarked_0  Embarked_C  Embarked_Q  Embarked_S
        0         0        0           0           0           0           1
        1         0        0           0           1           0           0
        2         0        0           0           0           0           1
        3         0        0           0           0           0           1
        4         0        0           0           0           0           1

        [5 rows x 161 columns]
```

```
In [6]: xg_cl = xgb.XGBClassifier(objective = 'binary:logistic', n_estimator = 10, seed = 123)
        xg_cl.fit(X_train, y_train)
        preds = xg_cl.predict(X_train)
        accuracy= float(np.sum(preds == y_train)/y_train.shape[0])
        print("Prediction accuracy on train set:", np.round(accuracy*100, 2), "%")
```

```
Prediction accuracy on train set: 87.99 %
```

1.1 Decision tree

Decision trees as base learners:

- Base learner - individual learning algorithm in an ensemble algorithm
- Composed of a series of binary questions
- Predictions happen at the "leaves" of the tree
- Constructed iteratively (one decision at a time)
  - Until a stopping criterion is met (such as predefined tree depth)
  - Individual tree: tend to be low bias but high variance
  - Also, low training error but high test error (overfit)

CART: Classification and Regression Trees

- Decision trees: the leaf nodes always contain decision values,
- CART: contain a **real-valued score** in each leaf, regardless of whether they are used for classification or regression.
  - The real-valued scores can be thresholded to convert into categories for classifiction problems if necessary.

```
In [7]: from sklearn.tree import DecisionTreeClassifier
        dt_clf_4 = DecisionTreeClassifier(max_depth = 4)
        dt_clf_4.fit(X_train, y_train)
        y_pred_4 = dt_clf_4.predict(X_train)
        accuracy= float(np.sum(y_pred_4 == y_train)/y_train.shape[0])
        print("Prediction accuracy on train set:", np.round(accuracy*100, 2), "%")
```

```
Prediction accuracy on train set: 83.05 %
```

1.2 Boosting

- Not a specific machine learning algorithm
- It is actually a concept that can be applied to a set of machine learning models
  - "Meta-algorithm"
- Ensemble meta-algorithm used to convert many weak learners into a strong learner
- Used to reduce any given single learner's variance and to convert any weak learner into an arbitrarily strong learner.

Weak learner and strong learners

- **Weak leaner**: ML algorithm that is slightly better than chance
  - Example: Decision trees whose predictions are slightly better than 50%
- Boosting converts a collection of weak learners into a strong learner
- **Strong learner**: any algorithm that can be tuned to achieve good performance.

How boosting is accomplished

- Iteratively learning a set of weak models on subsets of data
- Weighing each weak prediction according to each weak learner's performance
- Combined the weighted predictions to obtain a single weighted prediction
- The prediction is much better than the individual predictions themselves!

The prediction scores for each possibility are summed across trees, and the prediction is simply the sum of the scores across both trees.

Model evaluation through cross-validation

- Cross-validation: a robust method for estimating the performance of a model on unseen data
- Generating many non-overlapping train/test splits on training data
- Report the average test set performance across all data splits

We can convert the dataset into an optimized data structure that the creators of `XGBoost` made that gives the package its lauded performance and efficiency gains called a `DMatrix`.

If we use the XGBoost cv object, which is a part of XGBoost's learning api, we have to first explicitly convert our data into a DMatrix.

```
In [8]: # churn_dmatrix = xgb.DMatrix(data = churn.iloc[:, :-1],
        #     label = churn.month5_still_here)
        # params = {"objective":"binary:logistic", "max_depth":4}
        # cv_results = xgb.cv(dtrain=churn_dmatrix, params=params, nfold = 4,
        #    num_boost_round=10, metrics="error", as_pandas=True)
        # print("Accuracy: %f" %((1-cv_results["test-error-mean"]).iloc[-1]))
```

```
In [9]: # cv_results = xgb.cv(dtrain=churn_dmatrix, params=params, nfold = 4,
        #    num_boost_round=10, metrics="auc", as_pandas=True)
        # print((cv_results["test-auc-mean"]).iloc[-1])
```

1.3 When should I use XGBoost

Occasions when you should use XGBoost

- A large number of training examples
  - Greater than 1000 training samples and less than 100 features
  - The number of features < number of training samples
- A mixture of categorical and numeric features
  - Or just numeric features

Occassions when you should NOT use XGBoost

- If you can success using other state-of-the-art algorithms or those that suffer from data size issues.
  - Image recognition
  - Computer vision
  - Natural language processing and understanding problems
  - These problems can be much better tackled using deep learning approaches
- When you have very small data sets (<100 observations, or the number of training samples is significantly smaller than the number of features)