2.4 Using Scikit-learn Interface

February 22, 2017

1 Using Scikit-learn Interface

The following notebook presents the alternative approach for using XGBoost algorithm.

What's included: - load libraries and prepare data, - specify parameters, - train classifier, - make predictions

1.0.1 Loading libraries

Begin with loading all required libraries.

```
In [1]: import numpy as np

        from sklearn.datasets import load_svmlight_files
        from sklearn.metrics import accuracy_score

        from xgboost.sklearn import XGBClassifier
```

1.0.2 Loading data

We are going to use the same dataset as in previous lecture. The scikit-learn package provides a convenient function `load_svmlight` capable of reading many libsvm files at once and storing them as Scipy's sparse matrices.

```
In [2]: X_train, y_train, X_test, y_test = load_svmlight_files(('../data/agaricus.txt.train', '.
```

Examine what was loaded

```
In [3]: print("Train dataset contains {0} rows and {1} columns".format(X_train.shape[0], X_train
        print("Test dataset contains {0} rows and {1} columns".format(X_test.shape[0], X_test.sh
```

```
Train dataset contains 6513 rows and 126 columns
Test dataset contains 1611 rows and 126 columns
```

```
In [4]: print("Train possible labels: ")
        print(np.unique(y_train))

        print("\nTest possible labels: ")
        print(np.unique(y_test))
```

```
Train possible labels:
[ 0.  1.]

Test possible labels:
[ 0.  1.]
```

1.0.3 Specify training parameters

All the parameters are set like in the previous example - we are dealing with binary classification problem (`'objective':'binary:logistic'`), - we want shallow single trees with no more than 2 levels (`'max_depth':2`), - we don't any oupout (`'silent':1`), - we want algorithm to learn fast and aggressively (`'learning_rate':1`), (in naive named `eta`) - we want to iterate only 5 rounds (`n_estimators`)

```
In [5]: params = {
            'objective': 'binary:logistic',
            'max_depth': 2,
            'learning_rate': 1.0,
            'silent': 1.0,
            'n_estimators': 5
        }
```

1.0.4 Training classifier

```
In [6]: bst = XGBClassifier(**params).fit(X_train, y_train)
```

1.0.5 Make predictions

```
In [7]: preds = bst.predict(X_test)
        preds
```

```
Out[7]: array([ 0.,  1.,  0., ...,  1.,  0.,  1.])
```

Calculate obtained error

```
In [8]: correct = 0

        for i in range(len(preds)):
            if (y_test[i] == preds[i]):
                correct += 1

        acc = accuracy_score(y_test, preds)

        print('Predicted correctly: {0}/{1}'.format(correct, len(preds)))
        print('Error: {0:.4f}'.format(1-acc))
```

```
Predicted correctly: 1601/1611
Error: 0.0062
```