Python Package Introduction

This document gives a basic walkthrough of xgboost python package.

List of other Helpful Links

- Python walkthrough code collections (https://github.com/tqchen/xgboost/blob/master/demo/guide-python)
- Python API Reference (python_api.html)

Install XGBoost¶

To install XGBoost, follow instructions in Installation Guide (../build.html).

To verify your installation, run the following in Python:

```
import xgboost as xgb
```

Data Interface

The XGBoost python module is able to load data from:

- LibSVM text format file
- Comma-separated values (CSV) file
- NumPy 2D array
- SciPy 2D sparse array
- Pandas data frame, and
- XGBoost binary buffer file.

(See Text Input Format of DMatrix (../tutorials/input_format.html) for detailed description of text input format.)

The data is stored in a `DMatrix` (python_api.html#xgboost.DMatrix) object.

- To load a libsvm text file or a XGBoost binary file into `DMatrix` (python_api.html#xgboost.DMatrix):

```
dtrain = xgb.DMatrix('train.svm.txt')
dtest = xgb.DMatrix('test.svm.buffer')
```

- To load a CSV file into `DMatrix` (python_api.html#xgboost.DMatrix):

```
# label_column specifies the index of the column containing the
dtrain = xgb.DMatrix('train.csv?format=csv&label_column=0')
dtest = xgb.DMatrix('test.csv?format=csv&label_column=0')
```

> **Note**
> Categorical features not supported
>
> Note that XGBoost does not support categorical features; if your data contains categorical features, load it as a NumPy array first and then perform one-hot encoding (http://scikit-

learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html).

> **Note**
> Use Pandas to load CSV files with headers
>
> Currently, the DMLC data parser cannot parse CSV files with headers. Use Pandas (see below) to read CSV files with headers.

- To load a NumPy array into `DMatrix` (python_api.html#xgboost.DMatrix):

```
data = np.random.rand(5, 10)  # 5 entities, each contains 10 fe
label = np.random.randint(2, size=5)  # binary target
dtrain = xgb.DMatrix(data, label=label)
```

- To load a `scipy.sparse` (https://docs.scipy.org/doc/scipy/reference/sparse.html#module-scipy.sparse) array into `DMatrix` (python_api.html#xgboost.DMatrix):

```
csr = scipy.sparse.csr_matrix((dat, (row, col)))
dtrain = xgb.DMatrix(csr)
```

- To load a Pandas data frame into `DMatrix` (python_api.html#xgboost.DMatrix):

```
data = pandas.DataFrame(np.arange(12).reshape((4,3)), columns=[
label = pandas.DataFrame(np.random.randint(2, size=4))
dtrain = xgb.DMatrix(data, label=label)
```

- Saving `DMatrix` (python_api.html#xgboost.DMatrix) into a XGBoost binary file will make loading faster:

```
dtrain = xgb.DMatrix('train.svm.txt')
dtrain.save_binary('train.buffer')
```

- Missing values can be replaced by a default value in the `DMatrix` (python_api.html#xgboost.DMatrix) constructor:

```
dtrain = xgb.DMatrix(data, label=label, missing=-999.0)
```

- Weights can be set when needed:

```
w = np.random.rand(5, 1)
dtrain = xgb.DMatrix(data, label=label, missing=-999.0, weight=w
```

When performing ranking tasks, the number of weights should be equal to number of groups.

Setting Parameters

XGBoost can use either a list of pairs or a dictionary to set parameters (../parameter.html). For instance:

- Booster parameters

```
param = {'max_depth': 2, 'eta': 1, 'objective': 'binary:logistic
param['nthread'] = 4
param['eval_metric'] = 'auc'
```

- You can also specify multiple eval metrics:

```
param['eval_metric'] = ['auc', 'ams@0']

# alternatively:
# plst = param.items()
# plst += [('eval_metric', 'ams@0')]
```

- Specify validations set to watch performance

```
evallist = [(dtest, 'eval'), (dtrain, 'train')]
```

Training

Training a model requires a parameter list and data set.

```
num_round = 10
bst = xgb.train(param, dtrain, num_round, evallist)
```

After training, the model can be saved.

```
bst.save_model('0001.model')
```

The model and its feature map can also be dumped to a text file.

```
# dump model
bst.dump_model('dump.raw.txt')
# dump model with feature map
bst.dump_model('dump.raw.txt', 'featmap.txt')
```

A saved model can be loaded as follows:

```
bst = xgb.Booster({'nthread': 4})  # init model
bst.load_model('model.bin')  # load data
```

Methods including update and boost from xgboost.Booster are designed for internal usage only. The wrapper function xgboost.train does some pre-configuration including setting up caches and some other parameters.

Early Stopping

If you have a validation set, you can use early stopping to find the optimal number of boosting rounds. Early stopping requires at least one set in `evals` . If there's more than one, it will use the last.


```
train(..., evals=evals, early_stopping_rounds=10)
```

The model will train until the validation score stops improving. Validation error needs to decrease at least every `early_stopping_rounds` to continue training.

If early stopping occurs, the model will have three additional fields: `bst.best_score`, `bst.best_iteration` and `bst.best_ntree_limit`. Note that `xgboost.train()` (python_api.html#xgboost.train) will return a model from the last iteration, not the best one.

This works with both metrics to minimize (RMSE, log loss, etc.) and to maximize (MAP, NDCG, AUC). Note that if you specify more than one evaluation metric the last one in `param['eval_metric']` is used for early stopping.

Prediction

A model that has been trained or loaded can perform predictions on data sets.

```
# 7 entities, each contains 10 features
data = np.random.rand(7, 10)
dtest = xgb.DMatrix(data)
ypred = bst.predict(dtest)
```

If early stopping is enabled during training, you can get predictions from the best iteration with `bst.best_ntree_limit`:

```
ypred = bst.predict(dtest, ntree_limit=bst.best_ntree_limit)
```

Plotting

You can use plotting module to plot importance and output tree.

To plot importance, use `xgboost.plot_importance()` (python_api.html#xgboost.plot_importance). This function requires `matplotlib` to be installed.

```
xgb.plot_importance(bst)
```

To plot the output tree via `matplotlib`, use `xgboost.plot_tree()` (python_api.html#xgboost.plot_tree), specifying the ordinal number of the target tree. This function requires `graphviz` and `matplotlib`.

```
xgb.plot_tree(bst, num_trees=2)
```

When you use `IPython`, you can use the `xgboost.to_graphviz()` (python_api.html#xgboost.to_graphviz) function, which converts the target tree to a `graphviz` instance. The `graphviz` instance is automatically rendered in `IPython`.

```
xgb.to_graphviz(bst, num_trees=2)
```

XGBoost Python Package (index.html) Python API Reference (python_api.html)