mltool Documentation

Release 0.5.1

Maurizio Sambati

November 18, 2015

Contents

mltool is a simple Learning to Rank tool to build a regression tree based ranking model. Currently mltool supports simple CART and Random Forest. The implementation is strongly inspired by rt-rank, but it lacks the support of Gradient Boosting Regression Trees.

CHAPTER 1

Overview

1.1 Features

Despite rt-rank, mltool provides:

- A parameter to set the seed for the random number generator to make the training deterministic
- Serializable model
- Show feature information gain statistics at the end of the training
- Can be used as an API

Some highlights compared to other Random Forest implemented for Python:

- The implementation of the Random Forest makes use of numpy and it is quite optimized
- Parallel Random Forest training

1.2 Installation

Install from PyPI using pip:

```
$ pip install mltool
```

Now you can run mltool:

```
$ mltool -h
```

1.3 Documentation

The documentation is hosted by Read the Docs at the following url: http://readthedocs.org/docs/mltool/en/latest/

1.4 Future

- add support for Stochastic Gradient Boosting Regression Trees
- add support for simple regression and/or classification (i.e. not just focus on ranking)

CHAPTER 2

User Guide

mltool can be used as a command line tools. This part of the documentation covers each command supported by mltool.

2.1 User Guide

mltool is a command line tool that can be used to build ranking model.

2.1.1 Getting Started

Installation

mltool can be easily installed from PyPI using `pip`:

```
$ pip install mltool
```

If you want to install it from the source, you need to install `numpy` first. You can install numpy with `easy_install` or `pip`. This is an example using `pip`:

```
$ pip install numpy
```

Then clone the repository and run `setup.py`:

```
$ git clone git@bitbucket.org:duilio/mltool.git && cd mltool
$ python setup.py install
```

Now you should be able to run mltool:

```
$ mltool -h
```

Building the first model

Let's see how to build our first ranking model. In this example we will use an example dataset provided with svmrank. You can download the dataset here: http://download.joachims.org/svm_light/examples/example3.tar.gz

A dataset contains tuples of query-samples. A sample is a vector with feature values. A dataset for training and testing must provide a label for each sample. The models built with mltool predict labels for unlabelled samples.

Download and unpack the `example3.tar.gz` file:

```
$ wget http://download.joachims.org/svm_light/examples/example3.tar.gz
$ tar zxvf example3.tar.gz
```

The archive contains two files:

train.dat This file consists of 3 queries with some ranked results.

test.dat This file consists of a ranking that we will use to evaluate the model.

mltool uses a different format than svmrank, so we cannot use any of these file. Fortunately the mltool's `conv` command can convert svmrank file format to mltool's one. Here is how to use it:

```
$ mltool conv example3/train.dat train.tsv
$ mltool conv example3/test.dat test.tsv
```

mltool command line requires the first argument to be the command we want to execute. mltool supports several commands, you can see a list of them running `mltool -h`. Each command has it's own list of arguments. We've run the `conv` command that just need two parameters: the input and the output files. You can have a look at the help for the `conv` command with `mltool conv -h`.

Now we have both a train set to build a model and a test set to evaluate it. mltool currently supports only two algorithms to build a model: CART and Random Forest. Let's see how to build a model:

```
$ mltool rf-train -t 2 -s 1 train.tsv test.tsv -o ex3.pkl
2012-04-05 16:17:28,787 Reading training set...
2012-04-05 16:17:28,788 Reading validation set test.tsv...
2012-04-05 16:17:28,788 Training forest...
1   0.866025        0.914208        0.707107        0.718932
2012-04-05 16:17:28,849 Trees 1/2 generated.
2   0.520416        0.953896        0.353553        1.000000
2012-04-05 16:17:28,852 Trees 2/2 generated.
Feature gains:
  f0        4.4444444444444446
  f3        4.3999999999999986
  f4        3.7936507936507944
  f1        3.6666666666666679
  f2        2.1119047619047677
```

`rf-train` builds a model using Random Forest. `-t 2` sets the number of trees of the forest to 2, normally forest should be bigger but for this example that's fine. `-s 1` sets the seed for the random number generator. This will let us to rebuild the same model in the future. The rest of the arguments passed by the command line are the datasets and the output file name which will contain the model.

`rf-train` shows some information about while building the model. The lines starting with a date are logs displayed for your convenience, these messages keep us updated on the building process. We can safely ignore those lines for now.

What's more interesting for our purpose are the other lines. First mltool prints some statistics for each tree added to the model. Random Forest models are built with several trees, each tree add some logic to the model and this generally improve the model. We can measure the improvements looking at some metrics. mltool statistics show the number of trees of the model evaluated then RMSE and NDCG scores for each dataset passed to the command line. The first pair of RMSE/NDCG scores is related to the train data, the second pair is about the test data.

Observing the output we can see that the final model gets an NDCG score of 1.0 for the test data. This means the model provided a perfect rank for each ranking in the test set (just one). RMSE score is about 0.35, this tell us something more about the distance of the predicted label and the effective labels in the dataset. The lower the score for RMSE the better the predicted labels fit the expected ones.

Finally mltool shows a feature gain table. This table give us some hints about how much useful the features used in the train set are. This can be helpful as a quick hint for the usefulness of the selected features.

We want to know now something more about the provided ranking, we want to know the predicted labels. mltool provides an `eval` command that lets us get what we are looking for:

```
$ mltool eval -o preds.txt ex3.pkl test.tsv
2012-04-05 16:19:27,555 Reading dataset...
RMSE: 0.353553390593
NDCG: 1.0
$ cat preds.txt
3.5
3.0
2.5
1.0
$ paste <(echo; cat preds.txt) test.tsv
    qid_    label_  f0      f1      f2      f3      f4
3.5 4       4.0     1.0     0.0     0.0     0.20000000000000001     1.0
3.0 4       3.0     1.0     1.0     0.0     0.29999999999999999     0.0
2.5 4       2.0     0.0     0.0     0.0     0.20000000000000001     1.0
1.0 4       1.0     0.0     0.0     1.0     0.20000000000000001     0.0
```

`eval` just need the model and the file we want to evaluate. `-o filename` must be used to output the predicted labels, otherwise mltool just outputs the RMSE and NDCG scores.

The `preds.txt` file contains one number per line, the predicted labels. The labels are ordered following the same order of the evaluated file. The last command we've used show both `preds.txt` and `test.tsv` files aligned so we can see the differences of the scores.

We have correctly guess the value of the label in two samples over four. The other two samples were predicted with a slighly different score but this doesn't affect the final ranking, that's why our NDCG score is 1.0. We can compute manually the RMSE score:

$$\text{RMSE} = \sqrt{\frac{(4-3.5)^2 + (3.0-3.0)^2 + (2-2.5)^2 + (1.0-1.0)^2}{4}}$$

$$\text{RMSE} = \sqrt{\frac{0.5^2}{2}} \approx 0.35$$

That explains the RMSE score seen in the statistics of the `eval` command.

Now you know everything to build a ranker using mltool. You can start to experiment with your custom rankers. When you want to embed the model built into your app, please have a look to *Embed the model* in the *API Documentation* section.

CHAPTER 3

API Documentation

mltool can be used as a library for prediction or for building your own custom models.

3.1 API Documentation

3.1.1 Embed the model

One of the most common use of the mltool API is to embed the ranker model built with the command line tool into your own application.

The models are pickable file and you can load them using the `pickle` standard module:

```
import pickle
with open('model.pkl', 'rb') as fmodel:
    model = pickle.load(fmodel)
```

Then you can use mltool API to predict the score of a sample:

```
from mltool.predict import predict
pred = predict(model, {'f0': 1.0,
                       'f1': 0.0,
                       'f2': 0.0,
                       'f3': 0.2,
                       'f4': 1.0})
```

Check *predict* and *predict_all* functions for more details.

3.1.2 Prediction

mltool.predict

predict contains functions to predict the target variable score given a model.

---

mltool.predict.**predict**(*model*, *sample*)

Predict the score of a sample

Parameters

- **model** – the prediction model

- **sample** – a dictionary/namedtuple with all the features required by the model

Returns return the predicted score

---

mltool.predict.**predict_all**(*model*, *dataset*)

Predict targets for each sample in the dataset

Parameters

- **model** – the prediction model
- **dataset** – a dataset

Returns return the predicted scores

3.1.3 Model train

mltool implements some algorithms to train regression models. Currently it mainly supports Random Forest and regression trees.

mltool.forest

mltool.forest.**train_random_forest**(*dataset*, *num_trees*, *max_depth*, *ff*, *seed=1*, *processors=None*, *callback=None*)

Train a random forest model.

Parameters

- **dataset** – the labelled dataset used for training
- **num_trees** – number of trees of the forest
- **max_depth** – maximum depth of the trees
- **ff** – feature fraction to use for the split (1.0 means all)
- **seed** – seed for the random number generator
- **processors** – number of processors to use (all if *None*)
- **callback** (*None* or callable) – function to call for each tree trained, it takes the new tree as a parameter

Returns An mltool's model with with a forest of trees.

mltool.decisiontree

mltool.decisiontree.**train_decision_tree**(*dataset*, *max_depth*, *split_func=None*, *seed=None*)

Train a decision tree for regression.

It is possible to customize the function used to find the split for each node in the tree. The `split_func` is a callable that accepts two parameters, an array of labels and an 2d-array of samples. It returns None if no split is found, otherwise a tuple with the index of the feature, the value for the split and a gain score.

The 2d-array of samples has one column for each sample and one row per feature.

Parameters

- **dataset** – the labelled dataset used for training

- **max_depth** – maximum depth of the trees
- **split_func** (*None* or callable) – function to use to find the split. If *None* then a default one is used.
- **seed** – seed for the random number generator

Returns An mltool's model with a single decision tree.

3.1.4 Model Evaluation

mltool.evaluate

The metrics considered for the evaluation are two:

- NDCG
- RMSE

---

mltool.evaluate.**evaluate_preds**(*preds*, *dataset*, *ndcg_at=10*)
Evaluate predicted value against a labelled dataset.

Parameters

- **preds** (*list-like*) – predicted values, in the same order as the samples in the dataset
- **dataset** – a Dataset object with all labels set
- **ndcg_at** – position at which evaluate NDCG

Returns Return the pair RMSE and NDCG scores.

mltool.evaluate.**evaluate_model**(*model*, *dataset*, *ndcg_at=10*, *return_preds=False*)
Evaluate a model against a labelled dataset.

Parameters

- **model** – the model to evaluate
- **dataset** – a Dataset object with all labels set
- **ndcg_at** – position at which evaluate NDCG

Returns Return the pair RMSE and NDCG scores.

3.1.5 Utilities

Handling Dataset

class mltool.utils.**Dataset**
The Dataset class is a `namedtuple` which represents a set of samples with their labels and query ids.

labels
An `array` of labels. Each label is a *float*.

queries
A list of query ids.

samples
A 2d-`array <numpy.array` of samples. It consists of one sample per column, and one row for each feature.

feature_names
A sequence of feature names. Features are in the same order as they appear in the *samples* rows.

---

mltool.utils.**read_input_file**(*fin*)
Read a dataset from a file

Parameters **fin** – a file-like object to read the dataset from

Returns a *Dataset* object.

m

D

E

F

L

M

P

Q

R

S

T