KDDCup 2012 Project

Szymon Jaroszewicz

Institute of Computer Science
Polish Academy of Sciences
Warsaw, Poland

Overview

- KDD Cup 2012 data
- Solution sketch in `scikit-learn` (some things relevant also for TF/PyTorch)
  - SGDClassifier
  - reading data in batches
  - caching data `joblib.mem`
  - feature constructors

KDD-Cup 2012

KDD-Cup 2012

- A log from a Chinese search engine. The task is to predict if the user will click on a given ad
- Tab separated text file with 80mln records, 3GB gzip compressed
- Original competition data 150mln records

- Full description
  http://www.kddcup2012.org/c/kddcup2012-track2

KDD-Cup 2012 – variables

Click	did the user click
DisplayURL	ad url (encoded as huge integer)
AdId	identifier of specific ad (integer)
AdvertiserId	identifier of specific advertiser (integer)
Depth	number of ads displayed in a session (1-3)
Position	position of ad in the list of displayed ads
Gender	1,2 = male/female, 0 = unknown
Age	discretized into 6 intervals
...	...

KDD-Cup 2012 – variables

Text variables:

AdKeyword_tokens keywords for an ad

AdTitle_tokens title of an ad

AdDescription_tokens description of an add

Query_tokens user query

All text variables:

- list of words separated by |
- each word replaced by an integer for anonymity

KDD-Cup 2012

Click	did the user click
DisplayURL	ad url (encoded as huge integer)
AdId	identifier of specific Ad (integer)
AdvertiserId	identifier of specific advertiser (integer)
Depth	number of ads displayed in a session
Position	position of ad in the list of displayed ads
Gender	1,2 = male/female, 0 = unknown
Age	discretized into 6 intervals
AdKeyword_tokens	keywords for an ad
AdTitle_tokens	title of an ad
AdDescription_tokens	description of an add
Query_tokens	user query

```
0 4298118681424644510 7686695 385 3 3 1 3 4133|95|17 4133|95|17|0|4732|95|146|4079 8|81|123|
205|2|95|26|95|60|32|1|17|146|1|991|381|3|1718|2548|2110|3 4133

0 13677630321509009335 3517124 23778 3 1 1 3 4133 145|65|3927|832|93 3683|4990|2793|11589|21|
10741|26|16044|26|316810|1|933|420|26|1395|3927|65|832|93|114 4133
```

Test set

- `D5M_test_x.tsv` is the test set
- You need to score each test set record

- For grading:
- Python script
- a text file with
  - your name on first line (only ASCII please)
  - 5mln real numbers (scores) each on one line

```
SJ
-0.8267939001319954
-0.830694771165716
-0.830167732918347
-0.8999695790422827
-0.9022863852793969
-0.8266344391222575
...
```

KDD-Cup 2012 – tools and hints

Plan

- start with a model taking only 'small' variables such as age, gender, depth
- build and SGDClassifer, draw ROC curves
- play with various parameters: learning, loss, penalty, ...
- add bigger variables, e.g. AdvertiserId
- add text variables

scikit-learn SGDClassifier

- Our main tool will be the `SGDClassifier` class
- Some parameters:
  - loss: e.g. “hinge”, “log”
  - penalty: “l1”, “l2”, “none”, etc.
  - parameters for learning: learning step control, averaging, etc.
  - `n_iter` how many times to iterate over full dataset
  - `class_weight` should classes be balanced
  - .... some others, check the docs
- Model coefficients are in the `.coef_` attribute

Working with large data

- Full data does not fit into main memory
- This is no problem for SGDClassifier
  - read data in chunks e.g. 10000 or 100000 records each
  - use method `partial_fit` to update the model after each batch
- Before a new chunk is used for training it may be used to monitor accuracy

Speeding things up

- reading text data in .tsv is often the most time consuming part
- can be done in parallel, but notice that
  - it can be done once
  - converted data may be stored in some efficient binary format
- Can dump the CSR matrix directly
- An easier solution: `joblib.Memory`
  - can be used to add caching to any Python function

joblib.Memory

An example strategy

- write a function which reads a data chunk
- use the chaching decorator from joblib

```
from sklearn.externals.joblib import Memory

mem = Memory("/tmp/mycache")  # store binary cached
                              # objects here

@mem.cache                    # caching decorator
def load_data(chunk_number):
    X = np.genfromtxt(fname, delimiter="\t",
                      skip_header=1+batch_size*chunk,
                      max_rows = batch_size)
    return X
```

Feature constructors

- scikit-learn contains many data transformation tools
- Typical usage

```
e = OneHotEncoder(...params...)
e.fit(X)          # find out what values are in a file
X = e.transform(X) # transform the data
```

- The above can be shortened

```
e = OneHotEncoder(...params...)
X = e.fit_transform(X) # transform the data
```

Feature constructors – OneHotEncoder

- `OneHotEncoder` class
- Assumes values are integers (use `LabelEncoder` for strings)

X
0
2
0
1
0

$\Rightarrow$

X=0	X=1
1	0
0	0
1	0
0	1
1	0

- Params:
  - `categorical_features` list of numbers of columns to convert or "all"
  - `sparse` whether to return a sparse matrix (default True)

Feature constructors – feature hashing

- Problem with OneHotEncoder:
  - need to know the number of values in advance
  - can do a scan of all data to find how many values each feature has
  - each feature value gets a new variable: many variables can be produced for features such as UserID
- Another solution feature hashing
  - Produces user specified number of features
  - Values are converted to integers then hashed to given range
  - Multiple features may be combined into a single feature

Feature constructors – feature hashing

Example: hash function: $x \mod 4$

X
'John'
'Mary'
'home'
'goes'
'home'

$\Rightarrow$

X
15
27
10
8
10

$\Rightarrow$

F1	F2	F3	F4
0	0	0	1
0	0	0	1
0	0	1	0
1	0	0	0
0	0	1	0

- Here each field contains a singe value
- Each field may contain many values, e.g. a text document

Feature constructors – feature hashing

- `FeatureHasher` class
- Params
  - `n_features` how many features to generate (default 1048576)
  - `input_type` 'string' most useful for us. Each value is a list of strings.

```
>>> from sklearn.feature_extraction import FeatureHasher
>>> f = FeatureHasher(n_features = 4, input_type = "string")
>>> D = [["John", "goes", "home", "Mary", "is", "home"],
         ["John", "and", "Mary", "are", "home"]]
>>> f.fit_transform(D)

array([[ 0.,  0.,  1.,  1.],
       [ 1., -1.,  1.,  0.]])
```

- Negative values present (sign is flipped at random) such that all features have zero expected value

Feature constructors – feature hashing

- Problem with `FeatureHasher`: need to manually split lines into words
- Check `HashingVectorizer`