Data and Systems

Spyros Samothrakis
Research Fellow, IADS
University of Essex

March 14, 2017

About

- We will now turn our attention to
  - Systems
  - Big Data
  - Online visualisation
  - Web
  - Clusters
- Until now we have done mostly algorithms (with the exception of Pandas)
- Tips and Tricks

PyCharm shortcuts

- Double shift - meta-shortcut!

PyCharm Default Keymap

Editing

Ctrl + Space	Basic code completion (the name of any class, method or variable)
Ctrl + Alt + Space	Class name completion (the name of any project class independently of current imports)
Ctrl + Shift + Enter	Complete statement
Ctrl + P	Parameter info (within method call arguments)
Ctrl + Q	Quick documentation lookup
Shift + F1	External Doc
Ctrl + mouse over code	Brief Info
Ctrl + F1	Show descriptions of error or warning at caret
Alt + Insert	Generate code...
Ctrl + O	Override methods
Ctrl + Alt + T	Surround with...
Ctrl + /	Comment/uncomment with line comment
Ctrl + Shift + /	Comment/uncomment with block comment
Ctrl + W	Select successively increasing code blocks
Ctrl + Shift + W	Decrease current selection to previous state
Ctrl + Shift +]/[	Select till code block end/start
Alt + Enter	Show intention actions and quick-fixes
Ctrl + Alt + L	Reformat code
Ctrl + Alt + O	Optimize imports
Ctrl + Alt + I	Auto-indent line(s)
Tab / Shift + Tab	Indent/unindent selected lines
Ctrl + X or Shift + Delete	Cut current line or selected block to clipboard
Ctrl + C or Ctrl + Insert	Copy current line or selected block to clipboard
Ctrl + V or Shift + Insert	Paste from clipboard
Ctrl + Shift + V	Paste from recent buffers...
Ctrl + D	Duplicate current line or selected block
Ctrl + Y	Delete line at caret
Ctrl + Shift + J	Smart line join
Ctrl + Enter	Smart line split
Shift + Enter	Start new line
Ctrl + Shift + U	Toggle case for word at caret or selected block
Ctrl + Delete	Delete to word end
Ctrl + Backspace	Delete to word start

PyCharm Default Keymap

Running

Alt + Shift + F10	Select configuration and run
Alt + Shift + F9	Select configuration and debug
Shift + F10	Run
Shift + F9	Debug
Ctrl + Shift + F10	Run context configuration from editor
Ctrl+Alt+R	Run manage.py task

Debugging

F8	Step over
F7	Step into
Shift + F8	Step out
Alt + F9	Run to cursor
Alt + F8	Evaluate expression
Ctrl + Alt + F8	Quick evaluate expression
F9	Resume program
Ctrl + F8	Toggle breakpoint
Ctrl + Shift + F8	View breakpoints

Navigation

Ctrl + N	Go to class
Ctrl + Shift + N	Go to file
Ctrl + Alt + Shift + N	Go to symbol
Alt + Right/Left	Go to next/previous editor tab
F12	Go back to previous tool window
Esc	Go to editor (from tool window)
Shift + Esc	Hide active or last active window
Ctrl + Shift + F4	Close active run/messages/find/... tab
Ctrl + G	Go to line
Ctrl + E	Recent files popup
Ctrl + Alt + Left/Right	Navigate back/forward
Ctrl + Shift + Backspace	Navigate to last edit location
Alt + F1	Select current file or symbol in any view
Ctrl + B or Ctrl + Click	Go to declaration
Ctrl + Alt + B	Go to implementation(s)
Ctrl + Shift + I	Open quick definition lookup

(From jetbrains blog)

Jupiter/IPython notebook shortcuts

> The Jupyter Notebook has two different keyboard input modes. **Edit mode** allows you to type code/text into a cell and is indicated by a green cell border. **Command mode** binds the keyboard to notebook level actions and is indicated by a grey cell border. ×

Command Mode (press `Esc` to enable)

- `Enter` : enter edit mode
- `Shift` - `Enter` : run cell, select below
- `Ctrl` - `Enter` : run cell
- `Alt` - `Enter` : run cell, insert below
- `Y` : to code
- `M` : to markdown
- `R` : to raw
- `1` : to heading 1
- `2` : to heading 2
- `3` : to heading 3
- `4` : to heading 4
- `5` : to heading 5
- `6` : to heading 6
- `K` : select cell above
- `Up` : select cell above
- `J` : select cell below
- `Down` : select cell below
- `Shift` - `K` : extend selection above
- `Shift` - `J` : extend selection below
- `A` : insert cell above
- `B` : insert cell below
- `X` : cut selected cell
- `C` : copy selected cell
- `Shift` - `V` : paste cell above
- `V` : paste cell below
- `Z` : undo last cell deletion
- `D,D` : delete selected cell
- `Shift` - `M` : merge selected cells
- `S` : Save and Checkpoint
- `Ctrl` - `S` : Save and Checkpoint
- `L` : toggle line numbers
- `O` : toggle output
- `Shift` - `O` : toggle output scrolling
- `Esc` : close pager
- `Q` : close pager
- `H` : show keyboard shortcut help dialog
- `I,I` : interrupt kernel
- `0,0` : restart kernel
- `Shift` - `Space` : scroll up

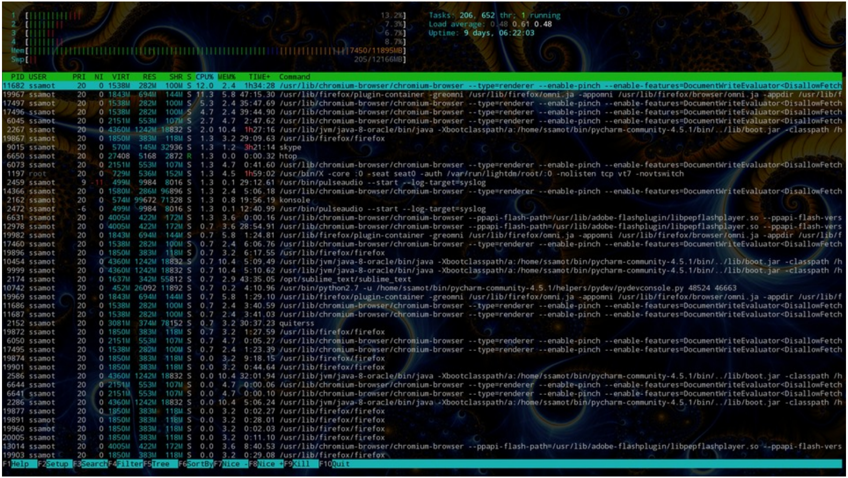
(From stackoverflow)

Unix

- Some basic knowledge of unix will be extremely helpful when it comes to dealing with the systems aspect
- Windows are indeed used for data science (depening on industry)
  - But unix is almost ubiquitous in the server environment
- `cat`
- `cat A B > C`
- `head`
- `tail` / `tail -f`

HTOP

CPUS, Memory, GPUs etc



Putting commands in the background

- Quite often you have long running commands that you need to run in a remote system
- `nohup <command-name> 1>out.txt 2>err.txt &`
- If command already running
  - `ctrl+z`
  - Puts command in the background
  - `disown [-h] [job-spec]`
- You can now exit the shell

Regular expressions and crawling the web

- Collecting data online
- Parsing files
- Example: parsing IRC logs for BobBr

```
find ./ -name "*" | xargs grep "BobBr" cat irc.log |
grep "BobBr"
```

Regular expressions

- `^` start of a line
- `$` end of a line
- `.` any character
- `*` more than zero occurrences
- `\+` more than one occurrences

Let's write some grep commands

Scrappy

```python
import scrapy


class QuotesSpider(scrapy.Spider):
    name = "quotes"
    start_urls = [
        'http://quotes.toscrape.com/tag/humor/',
    ]

    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'text': quote.css('span.text::text').extract_first(),
                'author': quote.xpath('span/small/text()').extract_first(),
            }

        next_page = response.css('li.next a::attr("href")').extract_first()
        if next_page is not None:
            next_page = response.urljoin(next_page)
            yield scrapy.Request(next_page, callback=self.parse)
```

Multi-threading

- Python does not allow native multi-threading
  - Threads can improve IO performance
  - Only one CPU core is used because of GIL
- Multi-processing
  - `copy-on-write` (not on windows)
  - Harder to share state
- scikit-learn classifiers support multi-processing (`n_jobs`)
  - Not distributed
- It actually makes tensorflow/theano slower

Joblib

```
from math import sqrt
k = [sqrt(i ** 2) for i in range(10)]
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]

from joblib import Parallel, delayed
k = Parallel(n_jobs=2)(delayed(sqrt)(i ** 2) for i in range(1000))
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0....]
```

```
[Parallel(n_jobs=2)]: Done    1 out of  181 | elapsed:    0.0s remaining:    4.5s
[Parallel(n_jobs=2)]: Done  198 out of 1000 | elapsed:    1.2s remaining:    4.8s
[Parallel(n_jobs=2)]: Done  399 out of 1000 | elapsed:    2.3s remaining:    3.5s
[Parallel(n_jobs=2)]: Done  600 out of 1000 | elapsed:    3.4s remaining:    2.3s
[Parallel(n_jobs=2)]: Done  801 out of 1000 | elapsed:    4.5s remaining:    1.1s
[Parallel(n_jobs=2)]: Done 1000 out of 1000 | elapsed:    5.5s finished
```

Map

- Performs computation on each element

```
def f(x):
    return x*x

map(f, range(10))
```

Multi-processing map

- Going from map to multi-processing map is trivial
- Problems with `ctrl + c`

```
from multiprocessing import Pool

def f(x):
    return x*x

pool = Pool(processes=16)
pool.map(f, range(10))
```

Filter

- Removes some elements from a list

```
number_list = range(-5, 5)
less_than_zero = list(filter(lambda x: x < 0, number_list))
```

Reduce

- Performs computation on a list
- Returns a single result
- Combines elements iteratively
- Reminds you of anything?

```
reduce((lambda x, y: x * y), [1, 2, 3, 4])
```

MapReduce

- Very commonly used paradigm for processing large datasets on multiple machines
  - Not used as much anymore
- Each machine has a piece of the data
- Map step –> Each machine applies a function to the data it has locally
- Shuffle step –> Data is redistributed to each machine according to a key
- Reduce step –> Data is reduced per key
- So basically, the same stuff you would do locally, but with a key

Question

- Why can't you just sample?

Data trumps algorithms

- It is often tempting to try to find a better algorithm to solve a certain problem
- But it has been shown time and time again that one much better off by adding more data
- Problems with neat solutions are very rare, more data
- *Physics envy* [1]
  - "An informal, incomplete grammar of the English language runs over 1,700 pages"
- We are modelling human perception as much as we are modelling cars or numbers!

[1] Halevy, Alon, Peter Norvig, and Fernando Pereira. "The unreasonable effectiveness of data." IEEE Intelligent Systems 24.2 (2009): 8-12.

Hadoop

- Hadoop Distributed File System (HDFS)
  - Splits large files and move them around different computers
  - Data lake
    - Or more like data dump?
- Hadoop MapReduce
  - A framework for using MapReduce in hadoop
- Hadoop is java, for python you have
  - MrJob

HDFS

- Can be used from the command line like any other programme
- `hdfs dfs <unix-like-command>`
    - `HDFS dfs -get <filename>`
    - `HDFS dfs -put <filename>`
    - `HDFS dfs -ls <filename>`
- Can accept connections remotely

MrJob

- Hadoop is written Java
  - Has something called the streaming API to help use other languages
- MrJob was created by Yelp, to be used on Amazon clusters
  - Elastic MapReduce
  - Hadoop
- You need to have a hadoop client configures in the machine with the appropriate environment variables

```
python mrjob/examples/mr_word_freq_count.py README.rst
-r hadoop > counts
```

MrJob example

```
class MRWordFrequencyCount(MRJob):

    def mapper(self, _, line):
        yield "words", len(line.split())

    def reducer(self, key, values):
        yield key, sum(values)


if __name__ == '__main__':
    MRWordFrequencyCount.run()
```

Spark

- MapReduce is slowing being abandoned
- HDFS still alive
- The cluster still alive
- "...using Spark on 206 EC2 machines, we sorted 100 TB of data on disk in 23 minutes"[2]
- A number of (mostly technical) speed updates over Hadoop involving memory, but most importantly
  - Does not save the results of each map operation to disk

[2] https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html

Spark example

```
datafile = spark.textFile("hdfs://...")
## flatmap first flattens the results of all line.split() s in the file
datafile.flatMap(lambda line: line.split())
        .map(lambda word: (word, 1))
        .reduceByKey(lambda x, y: x+y)
```

Spark Dataframes

- Spark has dataframes
- Like pandas!!!
- But slightly less advanced
  - For example, they can't read command csv files
  - But of course add-ons exist
- Spark dataframes live on the cluster
  - It means that operations on them can run on multiple machines

Spark MLib

- Spark has its own machine learning library
- That runs in a distributed fashion!
- scikit-learn is much faster
    - But your data might not fit in memory
- Avoid unless you absolutely have a really good use case
- Prefer out of core algorithms instead

Example (from the spark tutorial)

```
#...data is somehow loaded

(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Train a RandomForest model.
#  Empty categoricalFeaturesInfo indicates all features are continuous.
#  Note: Use larger numTrees in practice.
#  Setting featureSubsetStrategy="auto" lets the algorithm choose.
model = RandomForest.trainClassifier(trainingData, numClasses=2, categoricalFeaturesInfo={},
                                     numTrees=3, featureSubsetStrategy="auto",
                                     impurity='gini', maxDepth=4, maxBins=32)

# Evaluate model on test instances and compute test error
predictions = model.predict(testData.map(lambda x: x.features))
labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
testErr = labelsAndPredictions.filter(lambda (v, p): v != p).count() / float(testData.count())
print('Test Error = ' + str(testErr))
print('Learned classification forest model:')
print(model.toDebugString())
```

SCIKIT-LEARN OUT-OF-CORE

- Split your data into multiple files
- Read each file individually
    - Through the data into .partial_fit()
    - Not every algorithms supports this
- Use “Dask”

```
df = dd.read_csv('my-data-*.csv')
df = dd.read_csv('hdfs:///path/to/my-data-*.csv')
df = dd.read_csv('s3://bucket-name/my-data-*.csv')
```

Databases

- Pandas can read directly from databases
- The most common pathway is to basically get the data you need from a database
  - Do the analysis locally
- Feed the data back to the database

```
import MySQLdb
mysql_cn= MySQLdb.connect(...)
df_mysql = pd.read_sql('select USER_NAME, USER_AGE from USERS;', con=mysql_cn)

mysql_cn.close()
```

NoSQL databases

- Cassandra, MongoDB, BigTable
  - Wide column stores
- No master-slave relationship
  - Better disaster recovery
- Speed and scalability
- If you have constant streams of data, without much structure
- E.g. chat messages
-
  - and you plan on scaling to a substantial number of users

Data pipeline

- Problem definition
- Data collection
- Data cleaning
- Data coding
- Metric selection
- Algorithm selection
- Parameter optimisation
- Post-processing
- Deployment
- Debug

`https://indico.lal.in2p3.fr/event/2914/session/1/contribution/4/material/slides/0.pdf`

Bokeh

- A python package for rendering online visualisations
- Extends seaborn and renders with the style of D3.js
- Standalone capabilities as well
- Can be combined with pandas

From seaborn to bokeh

```
import seaborn as sns

from bokeh import mpl
from bokeh.plotting import output_file, show

tips = sns.load_dataset("tips")

sns.set_style("whitegrid")

ax = sns.violinplot(x="day", y="total_bill", hue="sex",
                    data=tips, palette="Set2", split=True,
                    scale="count", inner="stick")

output_file("violin.html")

show(mpl.to_bokeh())
```

IMDB movie example

- Let's move to the browser

`http://bokeh.pydata.org/en/latest/docs/gallery.html`

Conclusion

- We have seen various tools that should help once you get into more “niche” scenarios
- You don’t always have those massive amounts of data
- Use keyboard shortcuts
- Avoid scaling up when you don’t need it