pybda

Aug 16, 2019

Contents

A commandline tool for analysis of big biological data sets for distributed HPC clusters.

CHAPTER 1

Getting started

Getting started walks you through the installation of koios and to submit your first program. See the *examples* to get a better overview that koios has to offer.

1.1 Installation

Installing PyBDA is easy.

1) Make sure to have `python3` installed. PyBDA does not support previous versions. The best way to do that is to download and install anaconda and create a virtual environment like this:

```
conda create -y -n pybda python=3.6
source activate pybda
```

Note: Make sure to use Python version 3.6 since some dependencies PyBDA uses so far don't work with versions 3.7 or 3.8.

2) I recommend installing PyBDA from Bioconda:

```
conda install -c bioconda pybda
```

You can however also directly install using PyPI:

```
pip install pybda
```

or by downloading the _latest_ release

```
tar -zxvf pybda*.tar.gz
pip install pybda
```

I obviously recommend installation using the first option.

3) Download Spark from Apache Spark (use the *prebuilt for Apache Hadoop* package type) and install the unpacked folder into a custom path like `/opt/local/spark`. Put an alias into your `.bashrc` (or whatever shell you are using)

```
echo "alias spark-submit='opt/local/spark/bin/spark-submit'" >> .bashrc
```

4) That is it.

1.2 Usage

Using PyBDA requires providing two things:

- a config file that specifies the methods you want to use, paths to files, and parameters,
- and the `IP` to a running spark-cluster which runs the algorithms and methods to be executed. If no cluster environment is available you can also run PyBDA locally. This, of course, somehow limits what PyBDA can do for you, since it's real strength lies in distributed computation.

1.2.1 Config

Running PyBDA requires a `yaml` configuration file that specifies several key-value pairs. The config file consists of

- general arguments, such as file names,
- method specific arguments,
- arguments for Apache Spark.

General arguments

The following table shows the arguments that are **mandatory** and need to be set in every application.

Parameter	*Explanation*
`spark`	path to Apache spark `spark-submit` executable
`infile`	tab-separated input file to use for any of the methods
`outfolder`	folder where all results are written to.
`meta`	names of the columns that represent meta information ("n"-separated)
`features`	names of the columns that represent numerical features, i.e. columns that are used for analysis ("n"-separated).
`sparkparams`	specifies parameters that are handed over to Apache Spark (which we cover in the section below)

Method specific arguments

The following tables show the arguments required for the single methods, i.e. dimension reduction, clustering and regression.

Parameter	*Argument*	*Explanation*
Dimension reduction		
`dimension_reduction`	`factor_analysis`/`pca`/`lda`/`ica`	Specifies which method to use for dimension reduction
`n_components`	e.g `2,3,4` or `2`	Comma-separated list of integers specifying the number of variables in the lower dimensional space to use per reduction
`response`	(only for `lda`)	Name of column in `infile` that is the response. Only required for linear discriminant analysis.
Clustering		
`clustering`	`kmeans`/`gmm`	Specifies which method to use for clustering
`n_centers`	e.g `2,3,4` or `2`	Comma-separated list of integers specifying the number of clusters to use per cluystering
Regression		
`regression`	`glm`/`forest`/`gbm`	Specifies which method to use for regression
`response`		Name of column in `infile` that is the response
`family`	`gaussian`/`binomial`	Distribution family of the response variable

The abbreveations of the methods are explained in the following list.

- `factor_analysis` for factor analysis,
- `forest` for random forests,
- `gbm` for stochastic gradient boosting,
- `glm` for generalized linear regression models,
- `gmm` for Gaussian mixture models,
- `ica` for independent component analysis,
- `lda` for linear discriminant analysis,
- `kmeans` for K-means,
- `kpca` for kernel principal component analysis using Fourier features to approximate the kernel map,
- `pca` for principal component analysis.

Example

For instance, consider the config file below:

Listing 1: Contents of `data/pybda-usecase.config` file

```
spark: spark-submit
infile: data/single_cell_imaging_data.tsv
predict: data/single_cell_imaging_data.tsv
outfolder: data/results
meta: data/meta_columns.tsv
features: data/feature_columns.tsv
dimension_reduction: pca
n_components: 5
clustering: kmeans
n_centers: 50, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200
regression: forest
family: binomial
response: is_infected
sparkparams:
```

(continues on next page)

(continued from previous page)

```
  - "--driver-memory=3G"
  - "--executor-memory=6G"
debug: true
```

It would execute the following jobs:

1) dimension reduction (PCA) on the input file with 5 components,
2) clustering on the result of the dimensionality reduction with multiple cluster centers (`k`-means),
3) binomial-family random forest (i.e. logistic) on the input file with response *is_infected* and features from `data/feature_columns.tsv`

In addition we would allow Spark to use *3G* driver memory, *6G* executor memory and set the configuration variable `spark.driver.maxResultSize` to *3G* (all configurations can be found here).

Note: PyBDA first parses through the config file and builds a DAG of the methods that should be executed. If it finds dimensionality reduction *and* clustering, it will first embed the data in a lower dimensional space und use the result of this for clustering (i.e. in order to remove correlated features). The same does *not* happen with regression.

Spark parameters

The Spark documentation for submitting applications provides details which arguments are valid here. You provide them as list in the yaml file as key `sparkparams`: Below, the most important two are listed:

`"--driver-memory=xG"` Amount of memory to use for the driver process in gigabyte, i.e. where SparkContext is initialized.

`"--executor-memory=xG"` Amount of memory to use per executor process in giga byte.

`"--conf spark.driver.maxResultSize=3G"` Limit of total size of serialized results of all partitions for each Spark action.

1.2.2 Spark

In order for PyBDA to work you need to have a working *standalone spark environment* set up, running and listening to some `IP`. You can find a good introduction here on how to start the standalone Spark cluster. Alternatively, as mentioned above, a desktop PC suffices as well, but will limit what PyBDA can do for you.

We assume that you know how to use Apache Spark and start a cluster. However, for the sake of demonstration the next two sections give a short introduction how Spark clusters are set up.

Local Spark context

On a local resource, such as a laptop or desktop computer, there is no need to start a Spark cluster. In such a scenario the `IP` PyBDA requires for submitting jobs is just called `local`.

Alternatively, you can always *simulate* a cluster environment. You start the Spark environment using:

```
$SPARK_HOME/sbin/start-master.sh
$SPARK_HOME/sbin/start-slave.sh <IP>
```

where `$SPARK_HOME` is the installation path of Spark and `IP` the IP to which we will submit jobs. When calling `start-master.sh` Spark will log the `IP` it uses. Thus you need to have a look there to find it. Usually the line looks something like:

```
2019-01-23 21:57:29 INFO  Master:54 - Starting Spark master at spark://<COMPUTERNAME>
↪:7077
```

In the above case the `IP` is `spark://<COMPUTERNAME>:7077`. Thus you start the slave using

```
$SPARK_HOME/sbin/start-slave.sh spark://<COMPUTERNAME>:7077
```

That is it.

Cluster environment

If you are working on a cluster, you can use `sparkhpc` to set up a Spark instance (find the documenation here).

Note: If you want to use `sparkhpc` , please read its documentation to understand how Spark clusters are started.

Sparkhpc can be used to start a standalone cluster on an LSF/SGE high-performance computing environment. In order for them to work make sure to have **openmpi and Java installed**. Sparkhpc installs with PyBDA, but in case it didn't just reinstall it:

```
pip install sparkhpc
```

Sparkhpc helps you setting up spark clusters for LSF and Slurm cluster environments. If you have one of those start a Spark cluster, for instance, using:

```
sparkcluster start --memory-per-executor 50000 --memory-per-core 10000 --walltime␣
↪4:00 --cores-per-executor 5 2 &
sparkcluster launch &
```

Warning: For your own cluster, you should modify the number of workers, nodes, cores and memory.

In the above call we would request *2* nodes with *5* cores each. Every core would receive 10G of memory, while the entuire executor would receive *50G* of memory.

After the job has started, you need to call

```
sparkcluster info
```

in order to receive the Spark `IP`.

1.2.3 Calling

If you made it thus far, you successfully

1) modified the config file,
2) started a Spark standalone cluster and have the `IP` to which the Spark cluster listens.

Now we can finally start our application.

For dimension reduction:

```
pybda dimension-reduction data/pybda-usecase.config IP
```

For clustering:

```
pybda clustering data/pybda-usecase.config IP
```

For regression:

```
pybda regression data/pybda-usecase.config IP
```

And, finally, if you want to execute all methods (i.e., regression/clustering/dimension reduction/...) you would call PyBDA with a `run` argument:

```
pybda run data/pybda-usecase.config IP
```

In all cases, the methods create `tsv` files, plots and statistics.

1.3 References

Murphy, Kevin P. Machine learning: a probabilistic perspective. *MIT press* (2012).

Breiman, Leo. "Random forests." *Machine learning* 45.1 (2001): 5-32.

Friedman, Jerome H. "Stochastic gradient boosting." *Computational Statistics & Data Analysis* 38.4 (2002): 367-378.

Trevor, Hastie, Tibshirani Robert, and Friedman JH. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Hyvärinen, Aapo, Juha Karhunen, and Erkki Oja. Independent component analysis. Vol. 46. *John Wiley & Sons* (2004).

Köster, Johannes, and Sven Rahmann. "Snakemake—a scalable bioinformatics workflow engine." *Bioinformatics* 28.19 (2012): 2520-2522.

Meng, Xiangrui, et al. "MLlib: Machine Learning in Apache Spark." *The Journal of Machine Learning Research* 17.1 (2016): 1235-1241

Rahimi, Ali, and Benjamin Recht. "Random features for large-scale kernel machines." *Advances in Neural Information Processing Systems* (2008).

Zaharia, Matei, et al. "Apache Spark: a unified engine for big data processing." *Communications of the ACM* 59.11 (2016): 56-65.

CHAPTER 2

Examples

PyBDA provides a wide range of ML algorithms which can easily be used. Below we provide some examples to help getting started.

2.1 On models

- Applying dimension-reduction on massive data sets.
- Examples for clustering and mixture models.
- Fitting regression models to data.
- How to combine multiple algorithms with each others in one config file.

CHAPTER 3

FAQ

3.1 What's the best way to start using Spark?

Apache Spark has a steep learning curve. If you want to use one of the methods it's recommended to first go through the documentation here. Then test some applications with small data sizes such that you can figure out memory requirements or how many compute nodes you will need to run an algorithm in a specified amount of time. If everything works out, try increasing the data size until you either encounter errors or everything works fine.

3.2 How can I check Apache Spark is executing correctly?

Sometimes jobs might fail, because you launched Spark's compute nodes with too little memory, or a node lost its connection to the main worker, etc. When starting a cluster and running a method, it's thus recommended to monitor what Spark is actually doing. You can do so by first starting a Spark cluster. We do that here using spark_hpc:

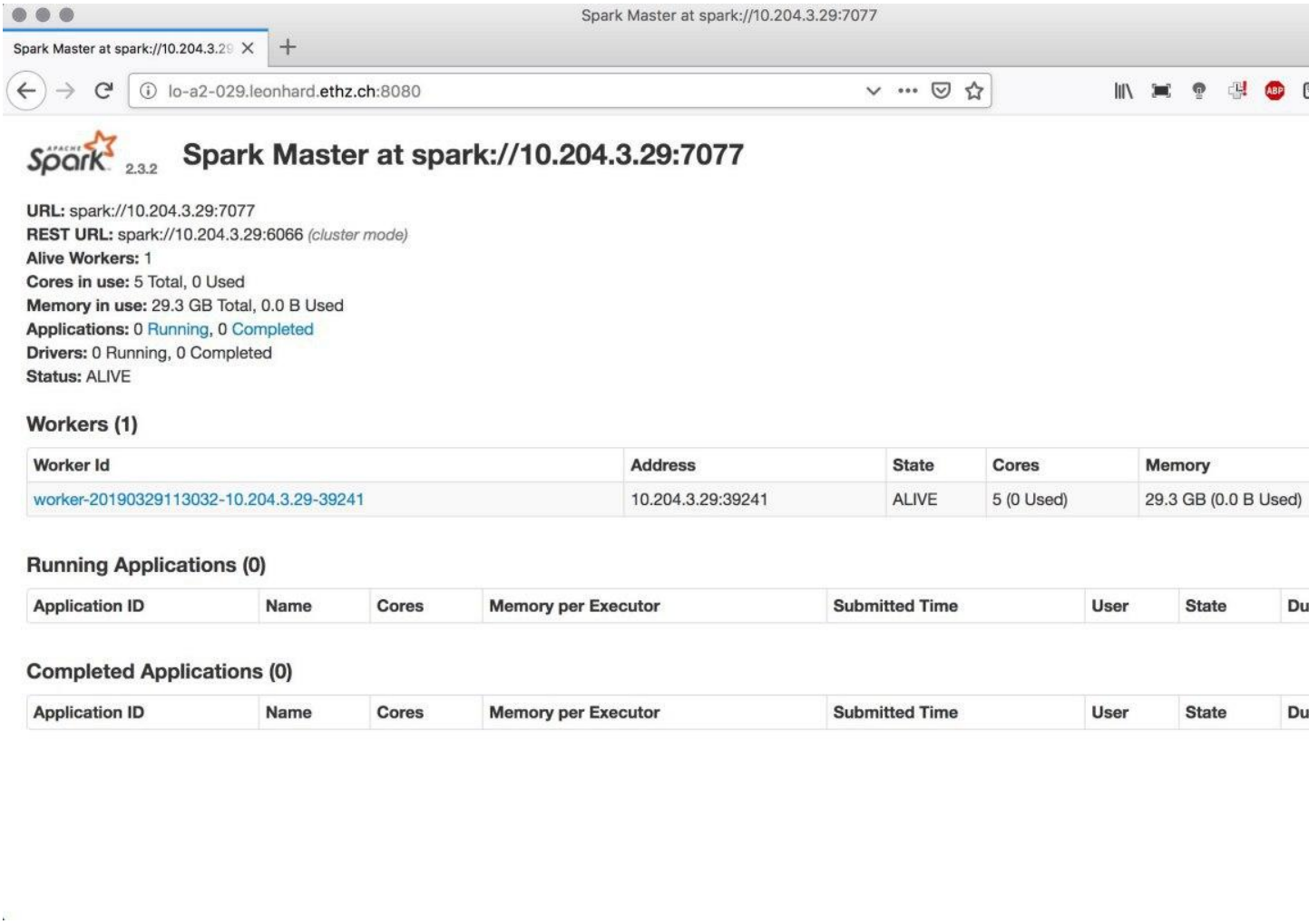
```
sparkcluster start --memory-per-executor 30000 \
                   --memory-per-core 5000 \
                   --walltime 4:00 \
                   --cores-per-executor 5 1
```

Having the cluster started, we can get information to which URL the *Spark UI* is listening to:

```
spark-cluster info

> ----- Cluster 0 -----
> Job id: 1756002
> Number of cores: 1
> Status: submitted
> Spark UI: http://lo-a2-029.leonhard.ethz.ch:8080
> Spark URL: spark://10.204.3.29:7077
```

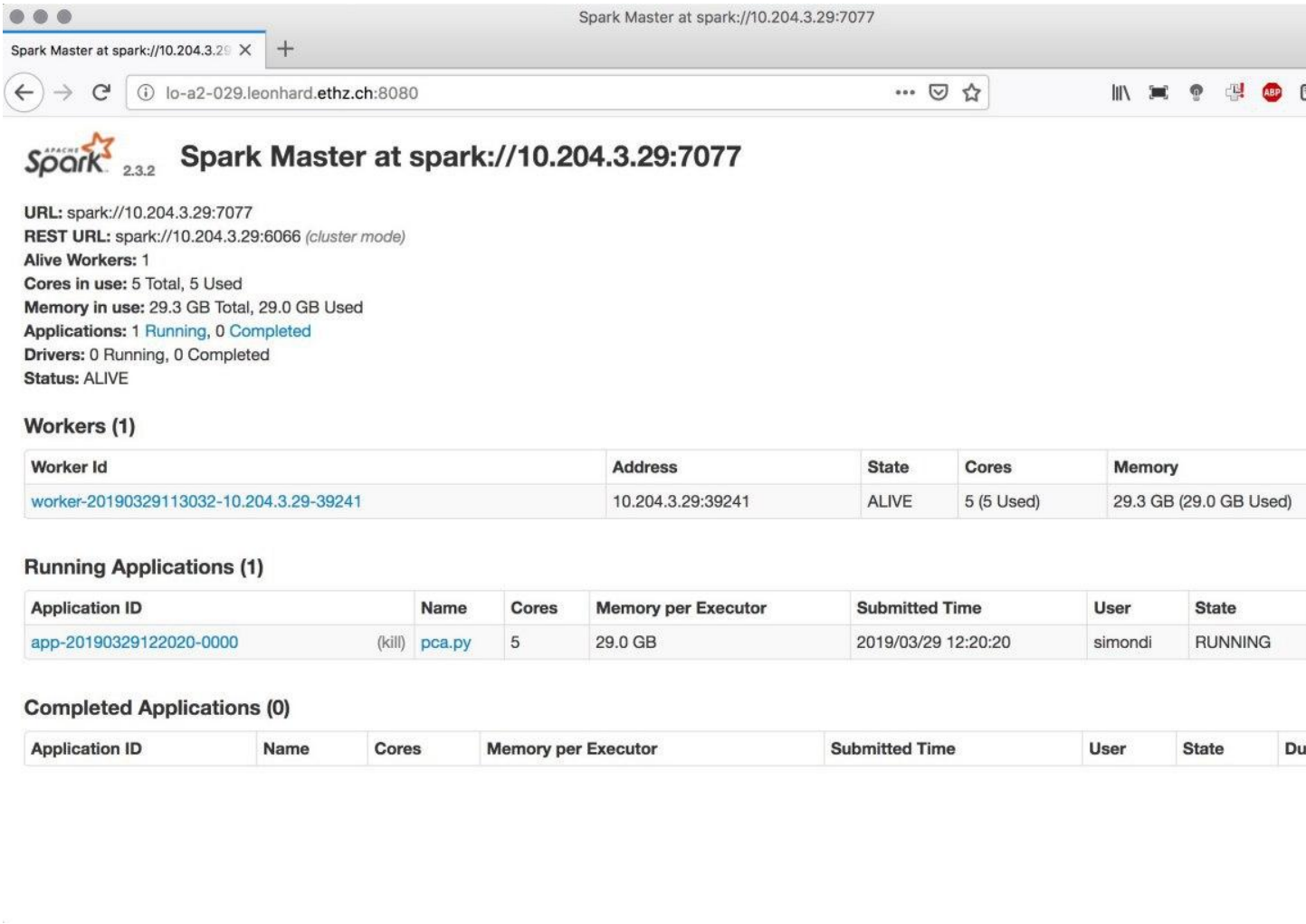
In this case it's `http://lo-a2-029.leonhard.ethz.ch:8080`. The Spark UI can then be accessed from your desktop computer (by ssh port forwarding):



We then start a PyBDA application and can monitor what Spark is doing:

```
pybda dimension-reduction pca.yml spark://10.204.3.29:7077
```

We see that Spark started the application and runs it on 5 cores with 29Gb of memory:


Spark Master at spark://10.204.3.29:7077
Spark Master at spark://10.204.3.29
lo-a2-029.leonhard.ethz.ch:8080
Spark 2.3.2
Spark Master at spark://10.204.3.29:7077
URL: spark://10.204.3.29:7077
REST URL: spark://10.204.3.29:6066 (cluster mode)
Alive Workers: 1
Cores in use: 5 Total, 5 Used
Memory in use: 29.3 GB Total, 29.0 GB Used
Applications: 1 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE
Workers (1)
Worker Id
Address
State
Cores
Memory
worker-20190329113032-10.204.3.29-39241
10.204.3.29:39241
ALIVE
5 (5 Used)
29.3 GB (29.0 GB Used)
Running Applications (1)
Application ID
Name
Cores
Memory per Executor
Submitted Time
User
State
app-20190329122020-0000
(kill)
pca.py
5
29.0 GB
2019/03/29 12:20:20
simondi
RUNNING
Completed Applications (0)
Application ID
Name
Cores
Memory per Executor
Submitted Time
User
State

3.3 How can I debug my config file?

If PyBDA exits for unkown reasons, it is often due to misspecified file paths, wrong parameterization, etc. To see how PyBDA starts applications you can add `debug: true` to your config file. This will print the Spark commands to stdout. For instance, we use the following config file:

Listing 1: Example of a configuration file.

```
spark: spark-submit
infile: data/single_cell_imaging_data.tsv
predict: data/single_cell_imaging_data.tsv
outfolder: data/results
meta: data/meta_columns.tsv
features: data/feature_columns.tsv
dimension_reduction: pca
n_components: 5
clustering: kmeans
n_centers: 50, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200
regression: forest
family: binomial
response: is_infected
sparkparams:
  - "--driver-memory=3G"
  - "--executor-memory=6G"
debug: true
```

We then call PyBDA using the `clustering` subcommand and pipe the output to a file. Grepping `spark-submit` gives us the calls Spark does.

```
pybda clustering data/pybda-config.config local > job
grep -i spark-submit job

> Submitting job spark-submit --master local --driver-memory=3G --executor-memory=5G
>                             pybda/factor_analysis.py 5
>                             single_cell_imaging_data.tsv feature_columns.tsv
>                             results/factor_analysis
> Submitting job spark-submit --master local --driver-memory=3G --executor-memory=5G
>                             pybda/kmeans.py 50,100,110,120,130,140,150,160,170,180,
↪190,200
>                             results/factor_analysis.tsv feature_columns.tsv results/
↪kmeans
```

The output shows that our application consists of two calls. One being the dimension reduction, the other being the clustering.

3.4 How can I find out what went wrong with the algorithm?

Every method or algorithm creates a log file suffixed with `*.log`. Having a look at the log should make clear if errors and what kind of errors happened.

3.5 How can I find out if snakemake ran properly?

Snakemake produces a hidden folder called `.snakemake/log` within the directory from which you call an application. The log files keep track what Snakemake is executing.

CHAPTER 4

Contributing

We welcome pull requests and contributions to make PyBDA a community driven tool for big data analytics.

You can contribute in the following ways:

- improve the documentation,
- add custom methods or algorithms,
- report bugs,
- improve general usability or speed up code,
- write unit tests.

4.1 How to contribute

In order to make a contribution best follow these steps:

- Create a fork of the repository on GitHub.
- Checkout the `develop` branch and create your own branch using

```
git checkout develop
git checkout -b myfeature
```

- Install all dependencies using `pip install '.[dev]'`.
- Add your feature and submit a pull request.

4.2 Coding standards

Please format your code using `yapf` and `flake8`:

```
cd pybda
yapf --style ../.styles.yapf -i my_file.py
tox -e lint
```

Furthermore, please don't duplicate code and try to use type annotations where you find them necessary or useful.

CHAPTER 5

About

Welcome to PyBDA.

PyBDA is a Python library and command line tool for big data analytics and machine learning.

In order to make PyBDA scale to big data sets, we use Apache [Spark]'s DataFrame API which, if developed against, automatically distributes data to the nodes of a high-performance cluster and does the computation of expensive machine learning tasks in parallel. For scheduling, PyBDA uses [Snakemake] to automatically execute pipelines of jobs. In particular, PyBDA will first build a DAG of methods/jobs you want to execute in succession (e.g. dimensionality reduction into clustering) and then compute every method by traversing the DAG. In the case of a successful computation of a job, PyBDA will write results and plots, and create some statistics. If one of the jobs fails PyBDA will report where and which method failed (owing to Snakemake's scheduling) such that the same pipeline can effortlessly be continued from where it failed the last time.

PyBDA supports multiple machine learning methods that scale to big data sets which we either implemented from scratch entirely or interface the methodology from [MLLib]:

- dimensionality reduction using PCA, factor analysis, kPCA, linear discriminant analysis and ICA,
- clustering using k-means and Gaussian mixture models,
- supervised learning using generalized linear regression models, random forests and gradient boosting.

The package is actively developed. If you want to you can also contribute, for instance by adding new features or methods: fork us on GitHub.

CHAPTER 6

Dependencies

- Apache Spark == 2.4.0
- Python == 3.6
- Linux or MacOS

CHAPTER 7

Example

To run PyBDA you only need to provide a config-file and, if possible, the IP of a spark-cluster (otherwise you can just call PyBDA locally using `local`). The config file for several machine learning tasks might look like this:

Listing 1: Example of a configuration file.

```
spark: spark-submit
infile: data/single_cell_imaging_data.tsv
predict: data/single_cell_imaging_data.tsv
outfolder: data/results
meta: data/meta_columns.tsv
features: data/feature_columns.tsv
dimension_reduction: pca
n_components: 5
clustering: kmeans
n_centers: 50, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200
regression: forest
family: binomial
response: is_infected
sparkparams:
  - "--driver-memory=3G"
  - "--executor-memory=6G"
debug: true
```

The above configuration would tell PyBDA to execute multiple things:

- first use an PCA to embed the data into a 5-dimensional latent space,
- do a `k`-means clustering with different numbers of clusters centers on that space,
- fit a random forest to the response called `is_infected` and use a `binomial` family,
- give the Spark driver 3Gb of memory and the executor 6Gb,
- print debug information.

You call PyBDA like that:

```
pybda run data/pybda-usecase.config local
```

where `local` tells PyBDA to just use your desktop as Spark cluster.

The result of any PyBDA call creates several files and figures. For instance, we should check the performance of the forest:

Listing 2: Performance statistics of the random forest.

```
family         response         accuracy         f1         precision         recall
binomial         is_infected         0.8236         0.8231143143597965         0.
↪8271935801788475         0.8236
```

For the PCA, we for instance create a biplot. It's always informative to look at these:

Fig. 1: PCA biplot of the single-cell imaging data.

And for the consecutive clustering, two of the plots generated from the clustering are shown below:

Fig. 2: Number of clusters vs explained variance and BIC.

Fig. 3: Distribution of the number of cells per cluster (component).

CHAPTER 8

References

Bibliography

[Snakemake] Köster, Johannes, and Sven Rahmann. "Snakemake—a scalable bioinformatics workflow engine." Bioinformatics 28.19 (2012): 2520-2522.

[Spark] Zaharia, Matei, et al. "Apache Sspark: a unified engine for big data processing." Communications of the ACM 59.11 (2016): 56-65.

[MLLib] Meng, Xiangrui, et al. "MLlib: Machine Learning in Apache Spark." The Journal of Machine Learning Research 17.1 (2016): 1235-1241.