Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

Programming Distributed Computing Platforms with COMPSs

Workflows & Distributed Computing Group

30-31/01/2024

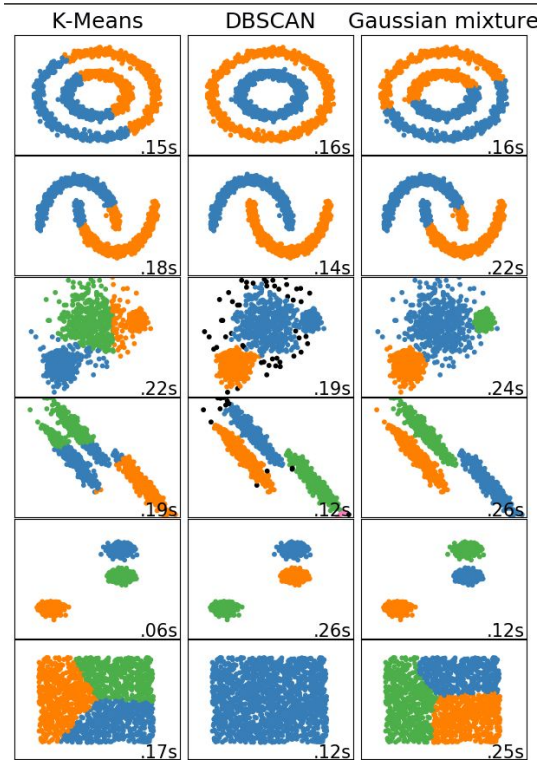
Barcelona, Spain

Outline

- Dislib overview
- The ds-array data structure
- Supported methods
- Some results
- Machine learning basics
- Typical workflow in dislib
- Sample code: C-SVM
- Browsing the dislib website

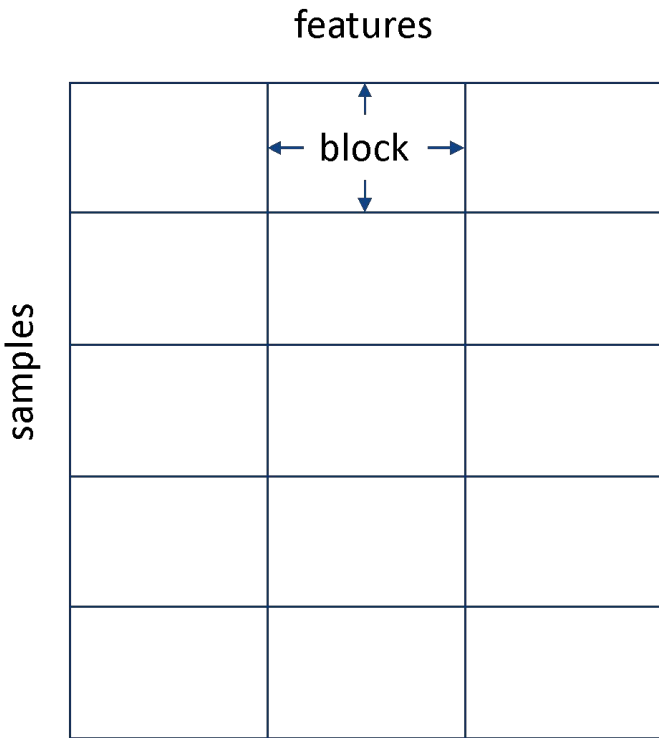
Dislib: parallel machine learning

- dislib: Collection of machine learning algorithms
  - Unified interface, inspired in scikit-learn (fit-predict)
  - Based on a distributed data structure (ds-array)
  - Unified data acquisition methods
  - Parallelism transparent to the user – PyCOMPSs parallelism hidden
  - Open source, available to the community
- Provides multiple methods:
  - data initialization
  - Clustering
  - Classification
  - Model selection, ...



Distributed array (ds-array)

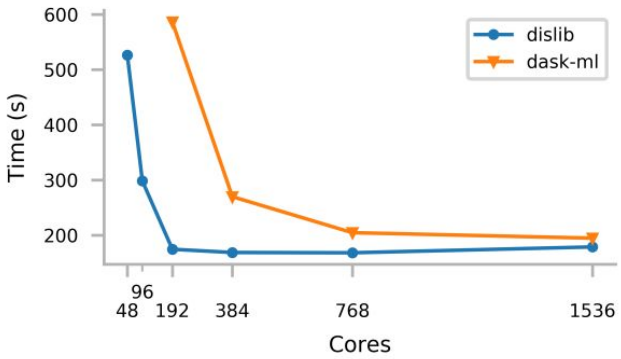
- 2-dimensional structure (i.e., matrix)
  - Divided in blocks (NumPy arrays)
- Works as a regular Python object
  - But not always stored in local memory!
- Methods for instantiation and slicing with the same syntax of numpy arrays:
  - Internally parallelized with PyCOMPSs:
  - Loading data (e.g. from a text file)
  - Indexing (e.g., x[3], x[5:10]
  - Operators (e.g., x.min(), x.transpose())
- ds-arrays can be iterated efficiently along both axes
- Samples and labels can be represented by independent distributed arrays
- Data not always in memory:
  - Inherent support for out-of-core operations, enabling large data-sets



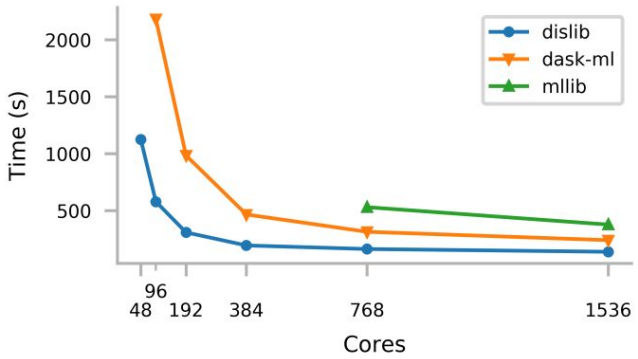
Supported methods

- Array creation routines
  - Multiple routines to create ds-arrays from random, existing data, files, ...
- Utilities to access arrays, scale, apply a function, ...
- Matrix decomposition:
  - Principal Component Analysis (PCA)
  - QR
  - TSQR
  - SVD
- Clustering:
  - DBSCAN
  - K-Means
  - Gaussian Mixture
  - Daura (Gromos)
- Classification
  - CascadeSVM
  - RandomForest classifier
  - DecisionTree classifier
- Recommendation
  - Alternating least squares (ALS)
- Regression
  - Linear regression
  - LASSO
  - RandomForest regressor
  - DecisionTree regressor
- Neighbour queries:
  - k-nearest neighbours
- Model selection:
  - GridSearch
  - RandomizedSearch
  - K-fold

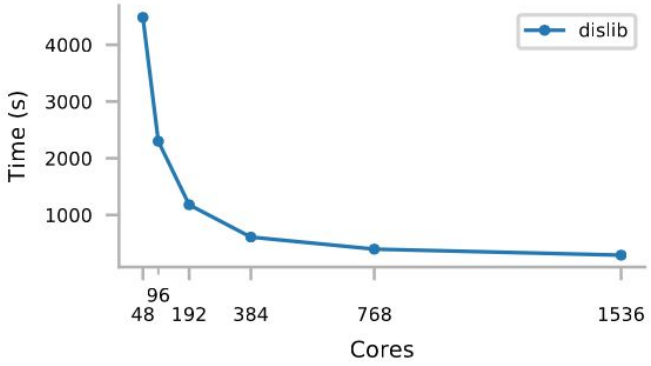
dislib sample results - K-means clustering



1 billion samples
50 features



500 million samples
100 features



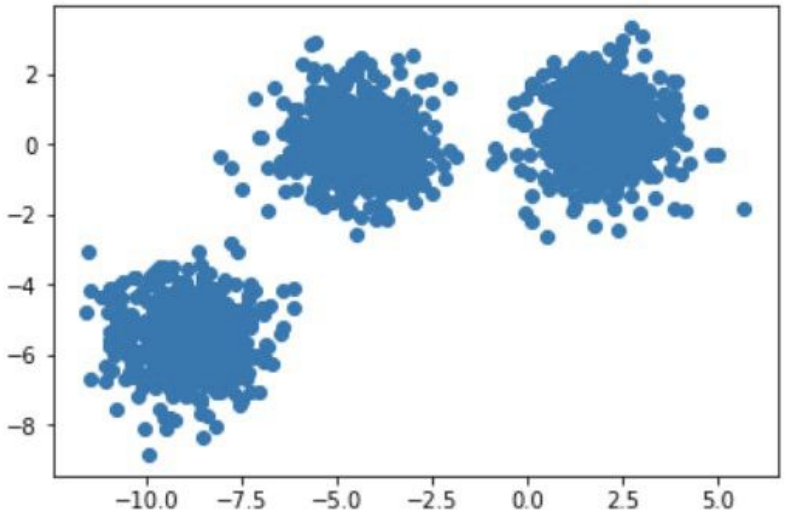
2 billion samples
100 features

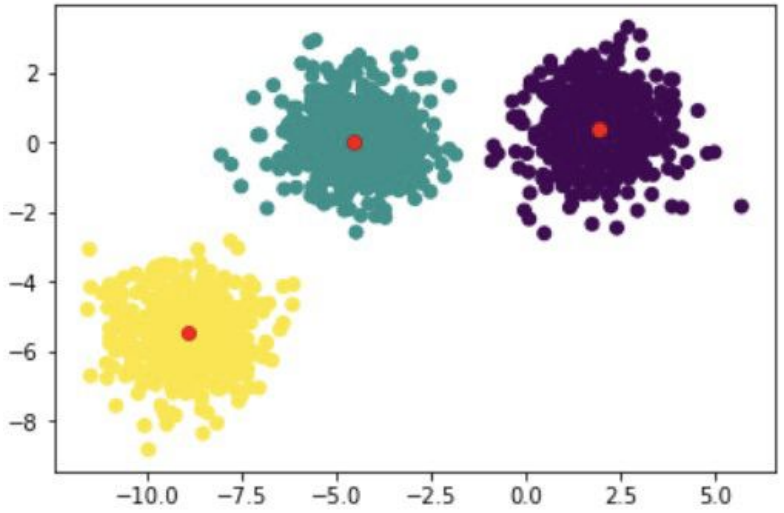
For very large sizes, dislib can obtain results while MLlib and dask fail to finish the execution

Machine learning basics

- Unsupervised
  - Find unknown patterns in (unlabelled) data
  - Example: clustering
- Supervised
  - Learn a decision function from a labelled data
  - Example: classification

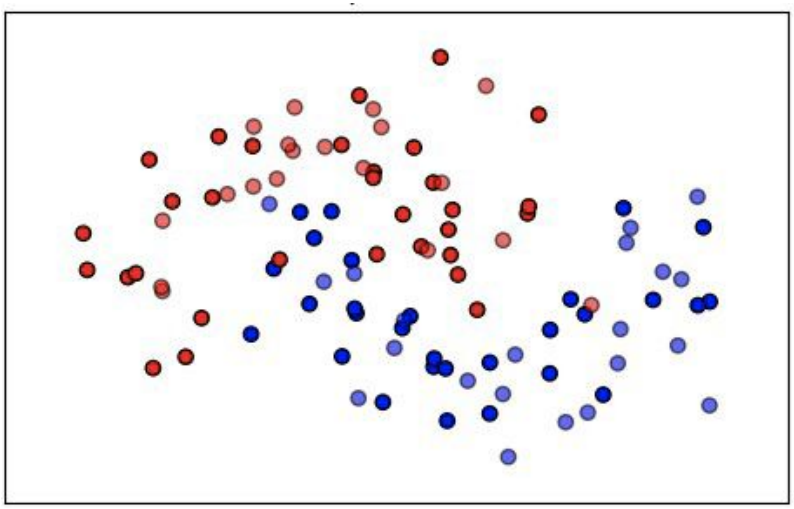
Clustering



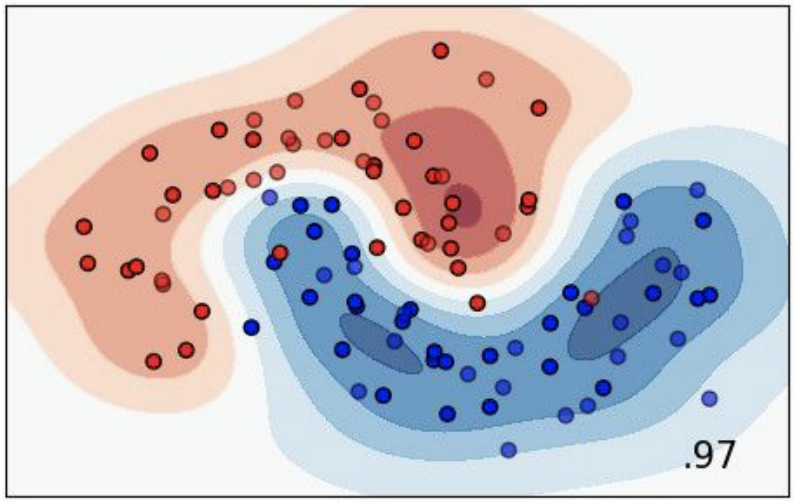


Unlabelled data

Classification



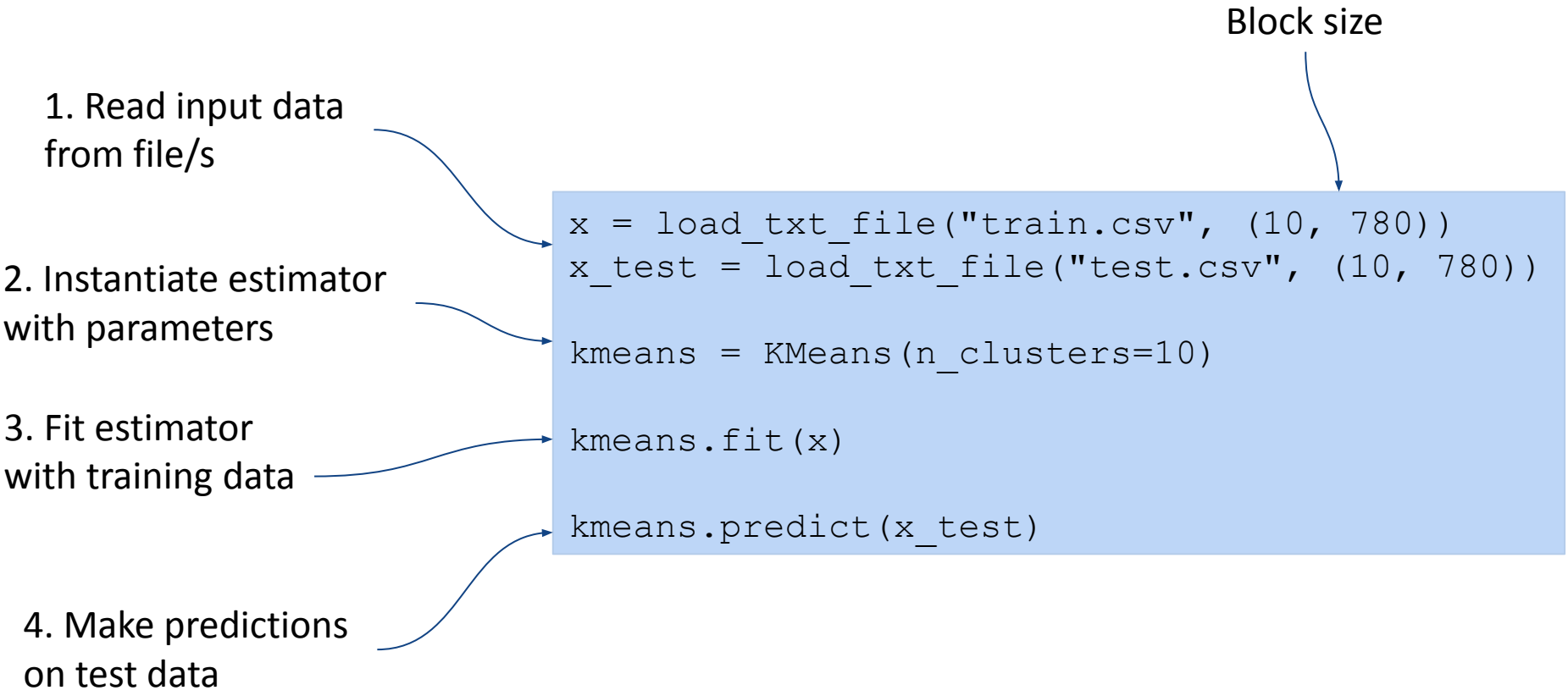
Labeled data



Estimators

- Based on scikit-learn
- Estimator = anything that learns from data (labelled or unlabelled)
- Two main methods:
  - fit → learns something from data (e.g., a decision function)
  - predict → provides new information based on a fitted model (e.g., labels data based on the computed decision function)

Typical workflow



Internals: ds-array implementation

- Implemented as an object, with main parameters:
  - Block size: shape of a regular block
  - Blocks: list of lists of NumPy ndarray (or spmatrix)
  - Sparse: whether the block is sparse or not
- Methods
  - Most of the methods for array creation or transformation are parallelized with PyCOMPSs:

```
@task(returns=np.array)
def _random_block(shape, seed):
    np.random.seed(seed)
    return np.random.random(shape)
```

```
@task(blocks={Type: COLLECTION_IN, Depth: 2}, returns=np.array)
def _block_apply_axis(func, axis, blocks, *args, **kwargs):
...
```

```
...
for block in x._iterator(axis=(not axis)):
    out = _block_apply_axis(func, axis, block._blocks, *args, **kwargs)
    out_blocks.append(out)
...
```

```
x = ds.random_array((100, 100), block_size=(25, 25))
mean = ds.apply_along_axis(np.mean, 0, x)
```

Sample code: C-SVM

```
""x : ds-array, shape=(n_samples, n_features)
            Training samples.
y : ds-array, shape=(n_samples, 1)
            Class labels of x. ""
while not self._check_finished():
    self._do_iteration(x, y, ids_list)
    if self.check_convergence:
        self._check_convergence_and_update_w()
        self._print_iteration()

return self
```

Set of tuples (x_data, y_data) that are partitions of x and y horizontally
with parts of both samples.

```
def _do_iteration(self, x, y, ids_list):
    ...
    # first level
    for partition, id_bk in zip(_paired_partition(x, y), ids_list):
        x_data = partition[0]._blocks
        y_data = partition[1]._blocks
        ...
        _tmp = _train(x_data, y_data, ids, self.random_state,
**pars)
        sv, sv_labels, sv_ids, self._clf = _tmp
        q.append((sv, sv_labels, sv_ids))

    # reduction
    while len(q) > arity:
        x_data = q[:arity]
        ...
        _tmp = _train(x_data, y_data, ids, self.random_state,
**pars)
```

Sample code: C-SVM

PyCOMPSs collections

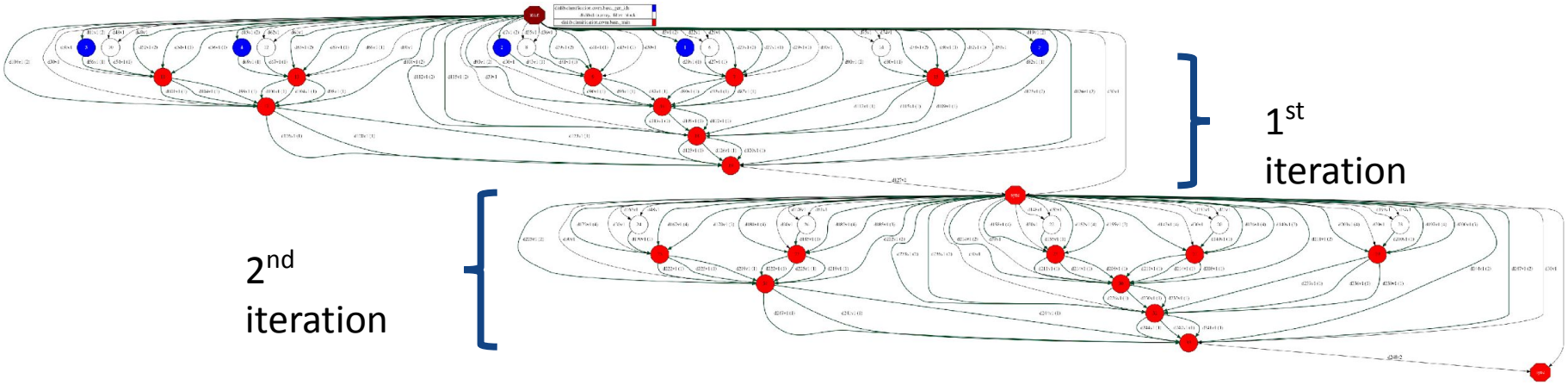
leverages Scikit-learn

```
from sklearn.svm import SVC
@task(x_list={Type: COLLECTION_IN, Depth: 2},
      y_list={Type: COLLECTION_IN, Depth: 2},
      id_list={Type: COLLECTION_IN, Depth: 2},
      returns=4)
def _train(x_list, y_list, id_list, random_state, **params):
    x, y, ids = _merge(x_list, y_list, id_list)

    clf = SVC(random_state=random_state, **params)
    clf.fit(X=x, y=y.ravel())

    sup = x[clf.support_]
    start, end = 0, 0
    sv = []

    for xi in x_list[0]:
        end += xi.shape[1]
        sv.append(sup[:, start:end])
        start = end

    sv_labels = y[clf.support_]
    sv_ids = ids[clf.support_]

    return sv, sv_labels, sv_ids, clf
```

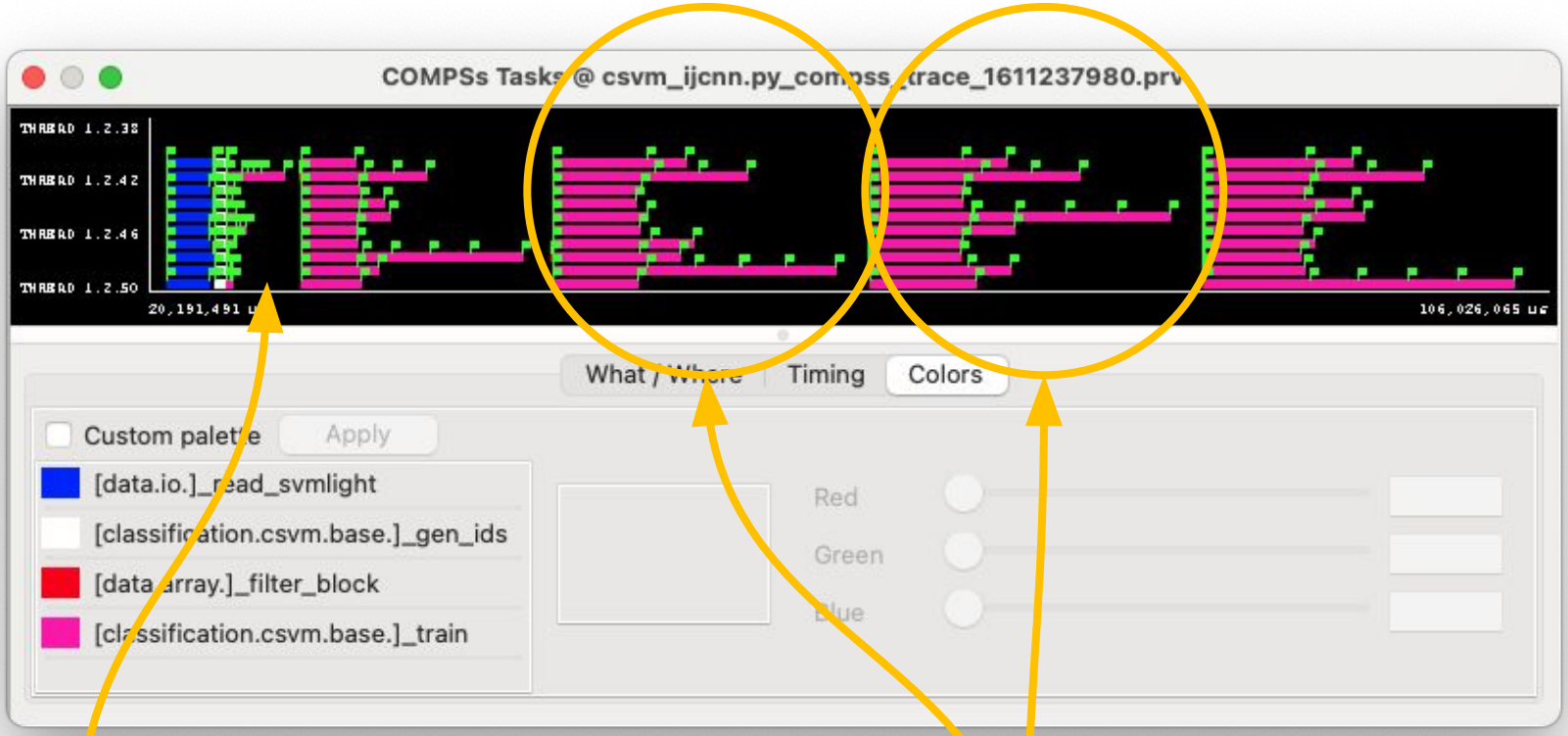
Sample user code

```
import dislib as ds
from dislib.classification import CascadeSVM
from dislib.utils import shuffle

def main():
    x_ij, y_ij = ds.load_svmlight_file("./C-SVM/datasets/train",
        block_size=(5000, 22), n_features=22, store_sparse=True)

    csvm = CascadeSVM(c=10000, gamma=0.01)

    csvm.fit(x_ij, y_ij)

if __name__ == "__main__":
    main()
```

C-SVM task graph



Sample tracefile



Data acquisition

iterations

Quick navigation in the website

- Dislib.bsc.es

Further Information

- Project page: http://www.bsc.es/compss
  - Documentation
  - Virtual Appliance for testing & sample applications
  - Tutorials
- Source Code

  https://github.com/bsc-wdc/compss
- Docker Image

  https://hub.docker.com/r/compss/compss
- Applications

  https://github.com/bsc-wdc/apps

  https://github.com/bsc-wdc/dislib

Projects where COMPSs is used/developed

MEEP | MareNostrum Experimental Exascale Platform

HPC/Exascale Centre of Excellence in Personalised Medicine

www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Thanks!