DATA ANALYTICS WITH INTEL® DISTRIBUTION FOR PYTHON

Moscow, March 01, 2018

ruslan.israfilov@intel.com

Most popular languages for Data Science

"**Python wins the heart of developers** across all ages, according to our Love-Hate index. Python is also the most popular language that **developers want to learn** overall, and a **significant share already knows it**"

HackerRank

2018 Developer Skills Report

- Python, Java, R are top 3 languages in job postings for data science and machine learning jobs
  - https://www.kdnuggets.com/2017/01/most-popular-language-machine-learning-data-science.html

KDnuggets

Most popular languages for Data Science

python™

SciPy

TensorFlow

scikit learn

theano

dmlc mxnet

jupyter

NumPy

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

Keras

matplotlib

intel

Python is **SLOW**

CPython provides an interpreter to run commands from Python Bytecode (.pyc)

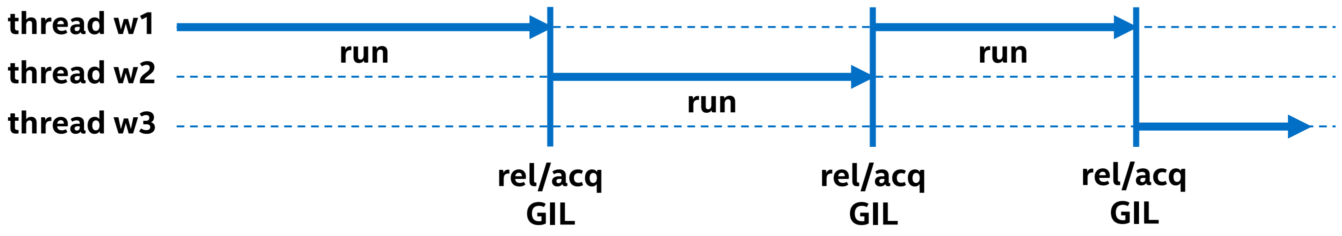
Compiling doesn't go down to CPU instructions, but instead:

Python interpreter → Compiled Bytecode → Python Virtual Machine

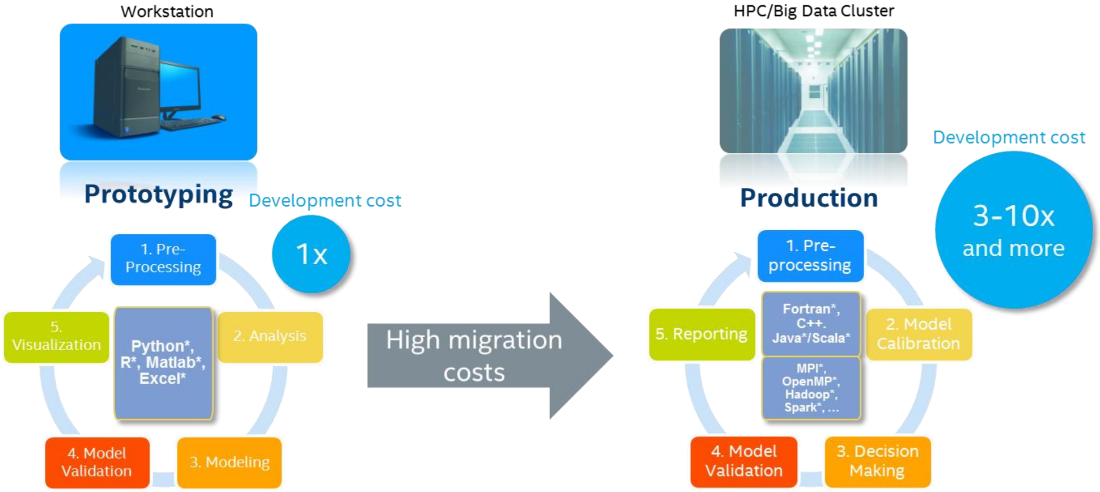
Allows for very flexible bytecode, and the Python interpreter is the main ingredient

CPython and PyPy have a Global Interpreter Lock (GIL)

CPython Global Interpreter Lock



Why scalability matters in (Data) Science



A TOAST for Next Generation CMB Experiments

Berkeley Lab Cosmology Software Scales Up to 658,784 Knights Landing Cores

newswise

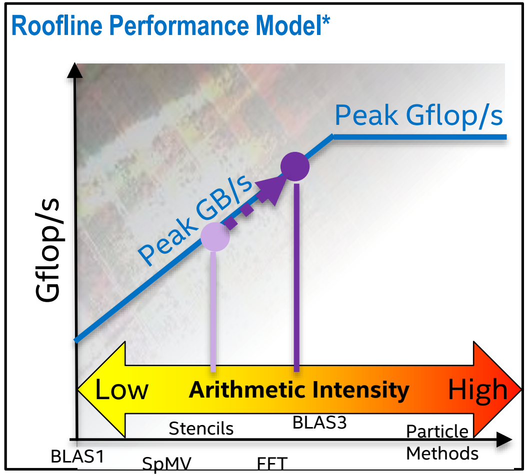
According to Kisner, the challenges to building a tool that can be used by the entire CMB community were both technical and sociological. Technically, the framework had to perform well at high concurrency on a variety of systems, including supercomputers, desktop workstations and laptops. It also had to be flexible enough to interface with different data formats and other software tools. Sociologically, parts of the framework that researchers interact with frequently had to be written in a high-level programming language that many scientists are familiar with.

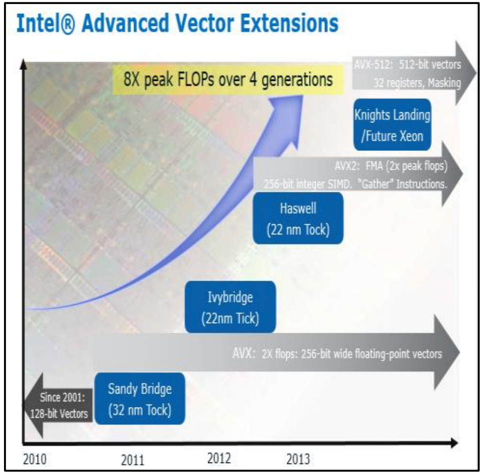
What scalability technically means

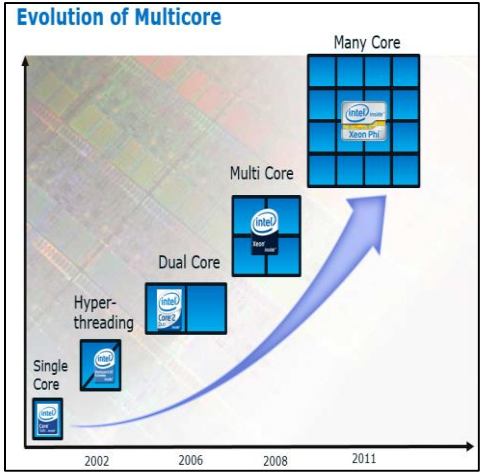
Hardware and software efficiency crucial in production (Perf/Watt, etc.)

Efficiency = Parallelism

- Instruction Level Parallelism with effective memory access patterns
- SIMD
- Multi-threading
- Multi-node



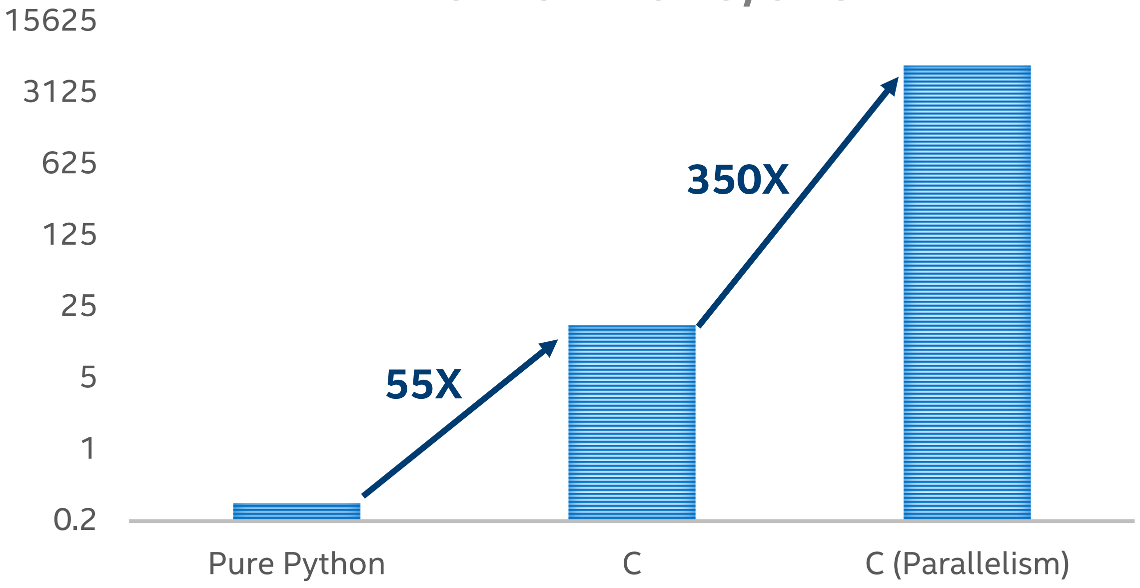


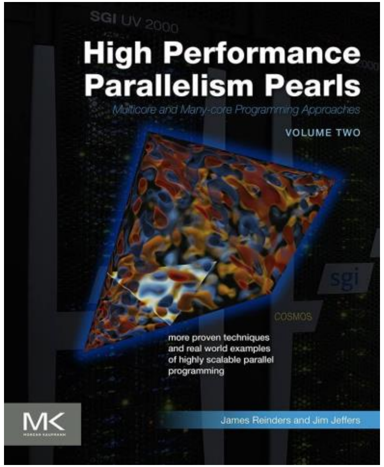


* Roofline Performance Model https://crd.lbl.gov/departments/computer-science/PAR/research/roofline/

Why parallelism matters

BLACK—SCHOLES FORMULA
MILLION OPTIONS/SEC





Chapter 19: Performance Optimization of **Black—Scholes** Pricing

$$V_{call} = S_0 \cdot \mathrm{CDF}(d_1) - e^{-rT} \cdot X \cdot \mathrm{CDF}(d_2)$$
$$V_{put} = e^{-rT} \cdot X \cdot \mathrm{CDF}(-d_2) - S_0 \cdot \mathrm{CDF}(-d_1)$$

$$d_1 = \frac{\ln\left(S_0/X\right) + \left(r + \sigma^2/2\right)T}{\sigma\sqrt{T}}$$
$$d_2 = \frac{\ln\left(S_0/X\right) + \left(r - \sigma^2/2\right)T}{\sigma\sqrt{T}}$$

Languages & Platforms

C++ – ultimate performance for (close-to-)real-time analytics

Java*/Scala* – easy integration with Big Data platforms (Hadoop*, Spark*, etc)

Python* – advanced analytics for data scientist

Is there a single solution possible?

Scikit-learn, Intel® DAAL, pyDAAL, DAAL4Py

Machine learning for in-memory homogeneous data

Scikit-learn

As simple as scikit-learn but handles non-homogeneous, streaming & distributed data

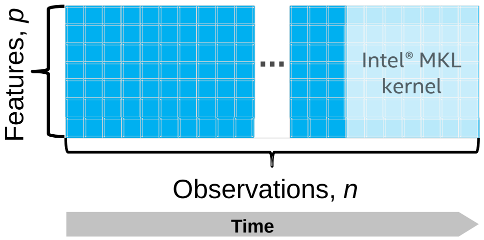
daal4py

New!

Python Binding for DAAL

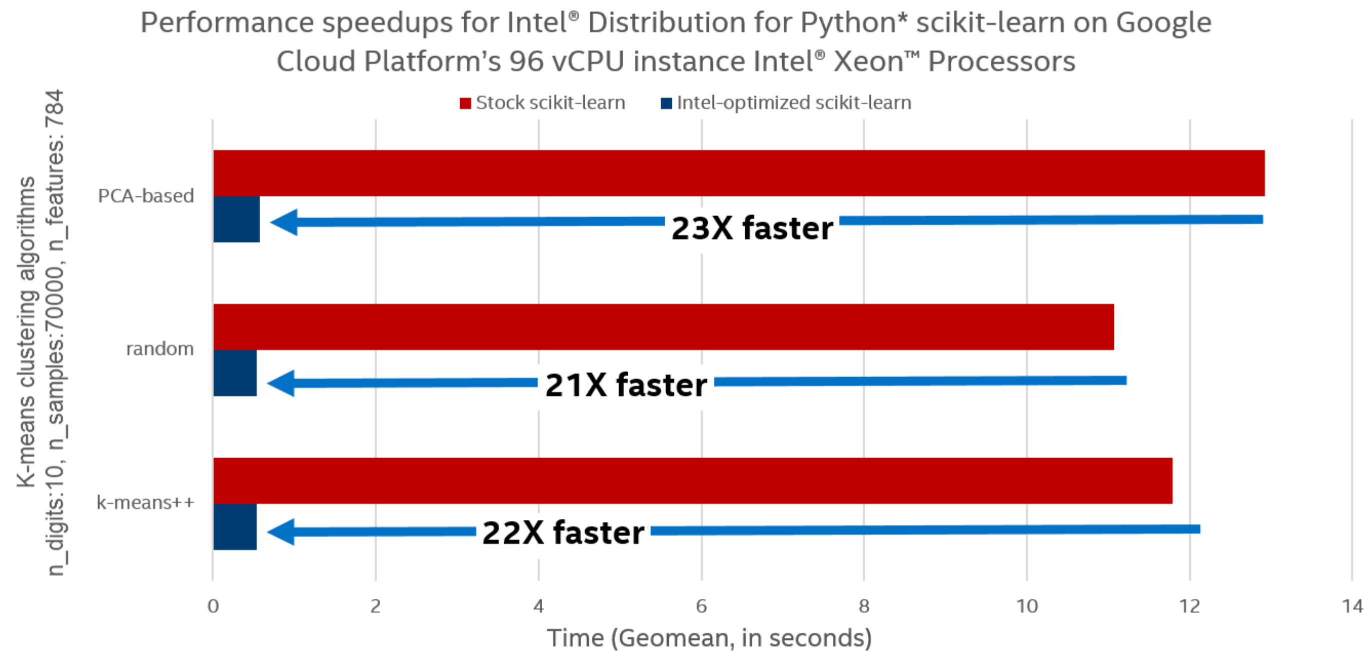
Intel® DAAL

Powerful low-level API for machine learning with non-homogeneous, streaming & distributed data



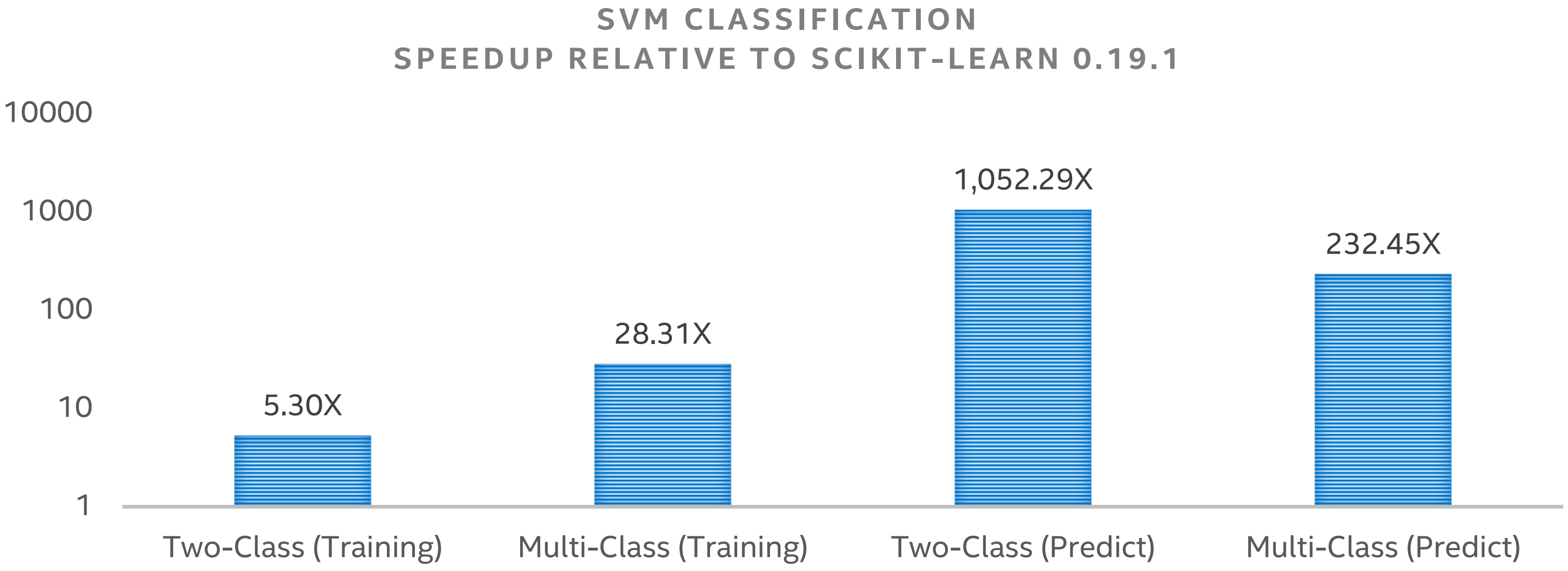
Built upon highly optimized Intel® MKL kernels

Analytics that scales within a node



System Configuration: GCP VM, zone us-central1-c; 96 vCPU, Intel Skylake; 360 GB memory. Ubuntu 16.04.3 LTS; Linux instance-1 4.10.0-38-generic #42~16.04.1-Ubuntu SMP Tue Oct 10 16:32:20 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux; Intel® Distribution for Python* from Docker image intelpython/intelpython3_full:latest (created 2017-09-12T20:10:42.862965559Z); Stock Python*: pip install scikit-learn

Analytics that scales within a node



Synthetic random data, Linear kernel SVM, 10000 rows, 1000 features, low tolerance=10^{-16}, maxiter==10^{6}. Intel® Distribution for Python* 2018 Update 2, scikit-learn 0.19.1. Intel(R) Xeon(R) Gold 6140 CPU @ 2.30GHz, 2 sockets, 18 cores/socket, HT:2. RAM: 250GB, Turbo mode and SpeedStep turned off

Distributed computing as simple as Scikit-learn*

Processing in-memory dataset loaded from CSV file

```
import daal4py as d4p
file = "kmeans_dense.csv"
dfin = loadtxt(file, delimiter=',')
centroids = d4p.kmeans_init(10, t_method="plusPlusDense")
result = d4p.kmeans(10).compute(dfin, centroids.compute(dfin))
```

```
python kmeans.py
```

Create numpy array

Parametrize algorithm object

Parametrize and execute in one line

Processing distributed dataset with MPI loaded from multiple CSV file

```
import daal4py as d4p
d4p.daalinit()
files = ["kmeans_dense.csv", …]
dfin = [loadtxt(x, delimiter=',') for x in files]
centroids = d4p.kmeans_init(10, t_method="plusPlusDense", distributed=True)
result = d4p.kmeans(10, distributed=True).compute(dfin, centroids.compute(dfin))
```

```
mpirun -n 4 -genv DIST_CNC=MPI python ./kmeans.py
```

Initialize

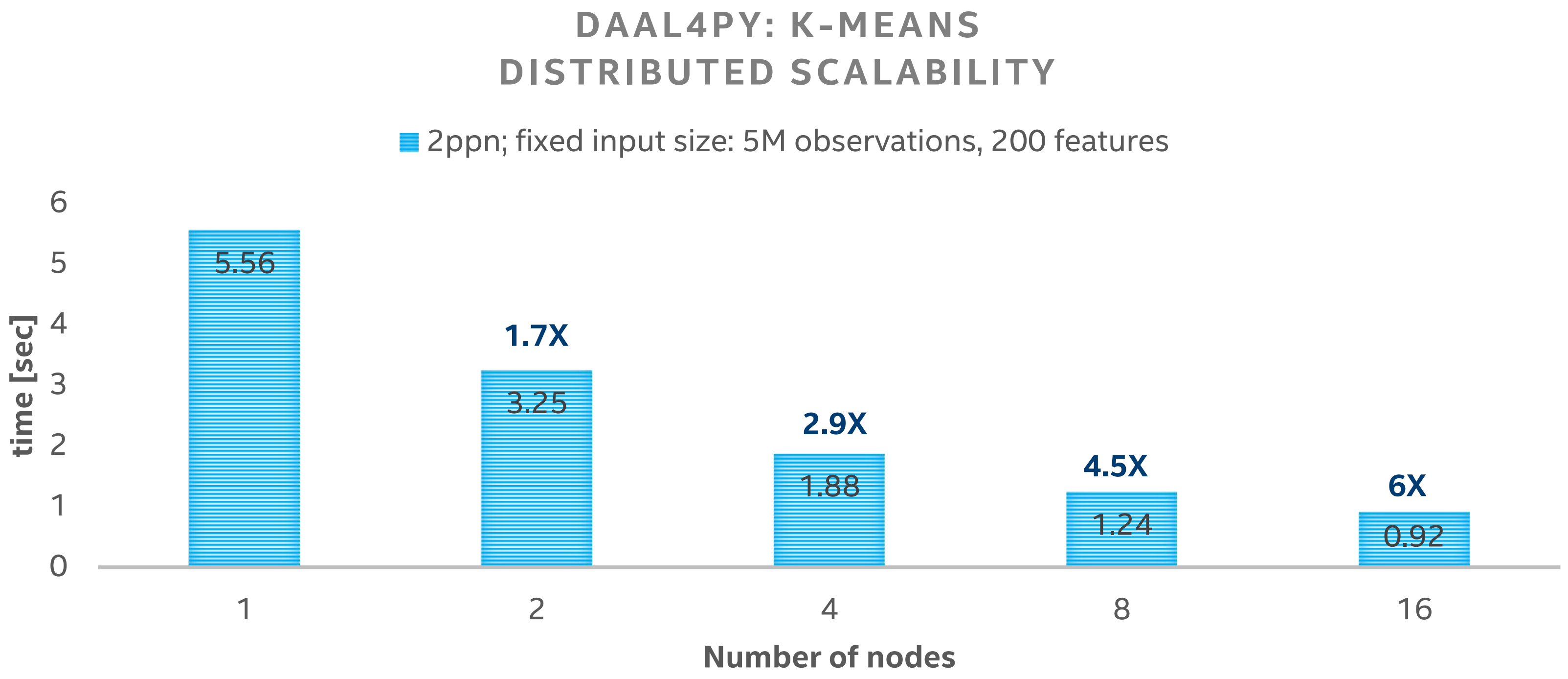
Multiple input arrays/files

Request distributed execution

Optimization Notice

Try it out `conda install -c intel/label/test dal4py`

Multi-node scaling with DAAL4PY



Configuration Info: Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz, EIST/Turbo on, 2 sockets, 20 Cores per socket, 192 GB RAM, 16 nodes connected with Infiniband, Oracle Linux Server release 7.4; Intel® Distribution for Python 2018 Update 1, DAAL4PY (Tech Preview)

Intel® Distribution for Python

FASTER PERFORMANCE	GREATER PRODUCTIVITY	ECOSYSTEM COMPATIBILITY
Performance Libraries, Parallelism, Multithreading, Language Extensions	**Prebuilt & Accelerated Packages**	**Supports Python 2.7 & 3.6, conda, pip**
Accelerated NumPy/SciPy/scikit-learn with Intel® MKL[1] & Intel® DAAL[2] Data analytics, machine learning & deep learning with scikit-learn, pyDAAL Scale with Numba* & Cython* Includes optimized mpi4py, works with Dask* & PySpark* Optimized for latest Intel® architecture	Prebuilt & optimized packages for numerical computing, machine/deep learning, HPC, & data analytics **Drop in replacement for existing Python - Usually with no code changes required** Jupyter* notebooks, Matplotlib included Conda build recipes included in packages Free download & free for all uses including commercial deployment	Compatible & powered by Anaconda*, supports conda & pip Distribution & individual optimized packages also available at conda & Anaconda.org, YUM/APT, Docker image on DockerHub Optimizations upstreamed to main Python trunk Commercial support through Intel® Parallel Studio XE 2017

Intel® Architecture Platforms

Operating System: Windows*, Linux*, MacOS[1]*

Installing Intel® Distribution for Python

Standalone Installer

Download full installer from
https://software.intel.com/en-us/intel-distribution-for-python

Anaconda.org
Anaconda.org/intel channel

```
> conda config --add channels intel
> conda install intelpython3_full
> conda install intelpython3_core
```

Docker Hub

```
docker pull intelpython/intelpython3_full
```

YUM/APT

Access for yum/apt:
https://software.intel.com/en-us/articles/installing-intel-free-libs-and-python

2.7 & 3.6

Windows*

Linux*

macOS*

Legal Disclaimer and Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

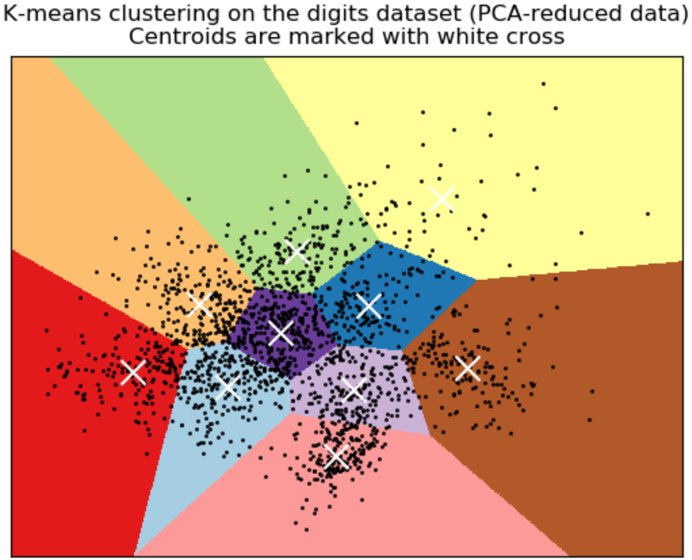
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Clustering MNIST images



http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html#sphx-glr-auto-examples-cluster-plot-kmeans-digits-py

Based on public scikit-learn demo

- Modified variant relies on Intel® Data Analytics Acceleration Library (pyDAAL)

Problem being solved:

- Unsupervised learning
- Clusterization of 70,000 MNIST images of hand-written decimal digits
- Image 28x28 pixels forms a tuple of 784 pixel values (features) that form 784-dimensional feature space
- Algorithm partitions 70,000 points into 10 clusters
- Visualization illustrates 2D projection of the original feature-space points

Benchmark: Black Scholes Formula

Problem: Evaluate fair European call- and put-option price, V_{call} and V_{put}, for underlying stock

Model Parameters:

- S_0 – present underlying stock price
- X – strike price
- σ - stock volatility
- r – risk-free rate
- T - maturity

$$V_{call} = S_0 \cdot \mathrm{CDF}(d_1) - e^{-rT} \cdot X \cdot \mathrm{CDF}(d_2)$$
$$V_{put} = e^{-rT} \cdot X \cdot \mathrm{CDF}(-d_2) - S_0 \cdot \mathrm{CDF}(-d_1)$$
$$d_1 = \frac{\ln\left(S_0/X\right) + \left(r + \sigma^2/2\right)T}{\sigma\sqrt{T}}$$
$$d_2 = \frac{\ln\left(S_0/X\right) + \left(r - \sigma^2/2\right)T}{\sigma\sqrt{T}}$$

```python
def black_scholes ( nopt, price, strike, t, rate, vol ):
    mr = -rate
    sig_sig_two = vol * vol * 2

    P = price
    S = strike
    T = t

    a = log(P / S)
    b = T * mr

    z = T * sig_sig_two
    c = 0.25 * z
    y = invsqrt(z)

    w1 = (a - b + c) * y
    w2 = (a - b - c) * y

    d1 = 0.5 + 0.5 * erf(w1)
    d2 = 0.5 + 0.5 * erf(w2)

    Se = exp(b) * S

    call = P * d1 - Se * d2
    put = call - P + Se

    return call, put
```

In practice one needs to evaluate many (*nopt*) options for different parameters

Good performance benchmark for stressing VPU and memory