Numba Data parallel Python

Data Parallel Essentials for Python: Bringing oneAPI to python –Part 2

Praveen Kundurthy

What is Data parallel Python?

intel®

Numba-Dpex

- Agenda
  - Overview of oneAPI
  - Overview of Intel® oneAPI AI Analytics Toolkit
  - Introduction to Numba-Data parallel extension (numba-dpex) and data parallel control (dpctl)
  - Pairwise distance using @njit and @Kernel decorator
  - Intel® Extension for Scikit-learn
  - Pairwise distance using scikit learn
  - Compute follows data approach
  - Black Scholes using @njit and @Kernel decorator
  - Profiling using Intel® VTune™ Profiler and Intel® Advisor
- Hands On Intel® DevCloud / JLSE
  - Pairwise Distance and Blackscholes

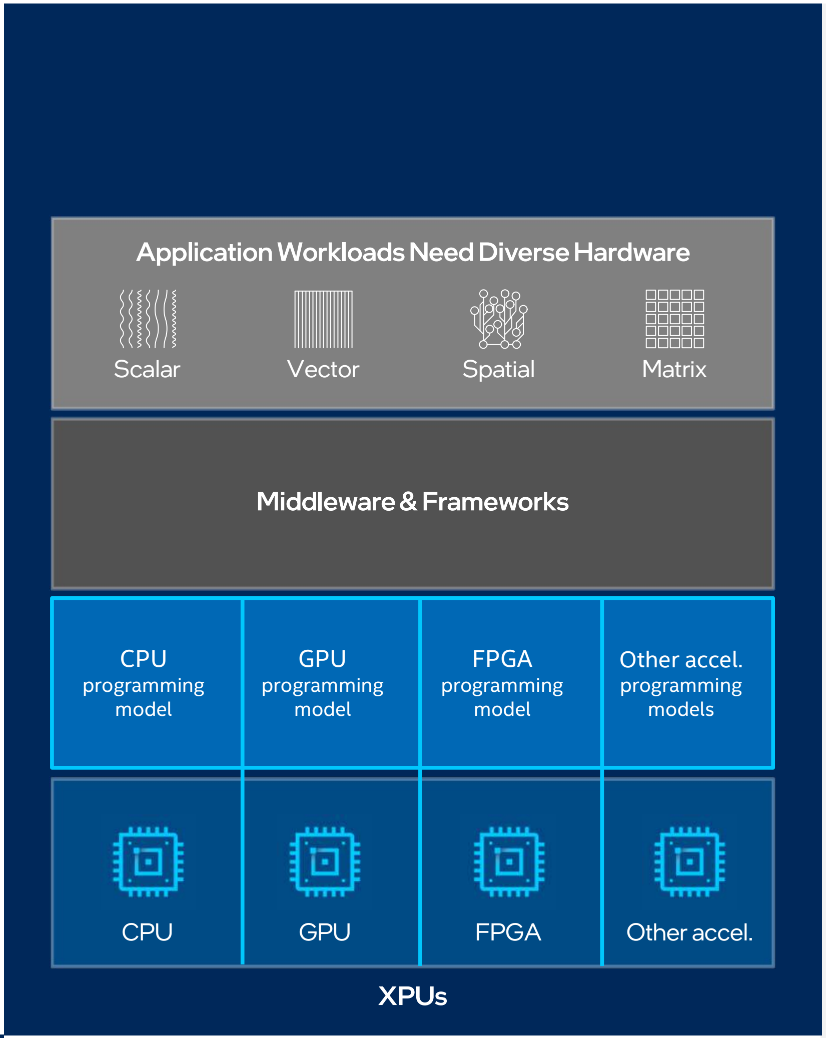
Programming Challenges
for Multiple Architectures

Growth in specialized workloads

Variety of data-centric hardware required

Separate programming models and toolchains for each architecture are required today

Software development complexity limits freedom of architectural choice



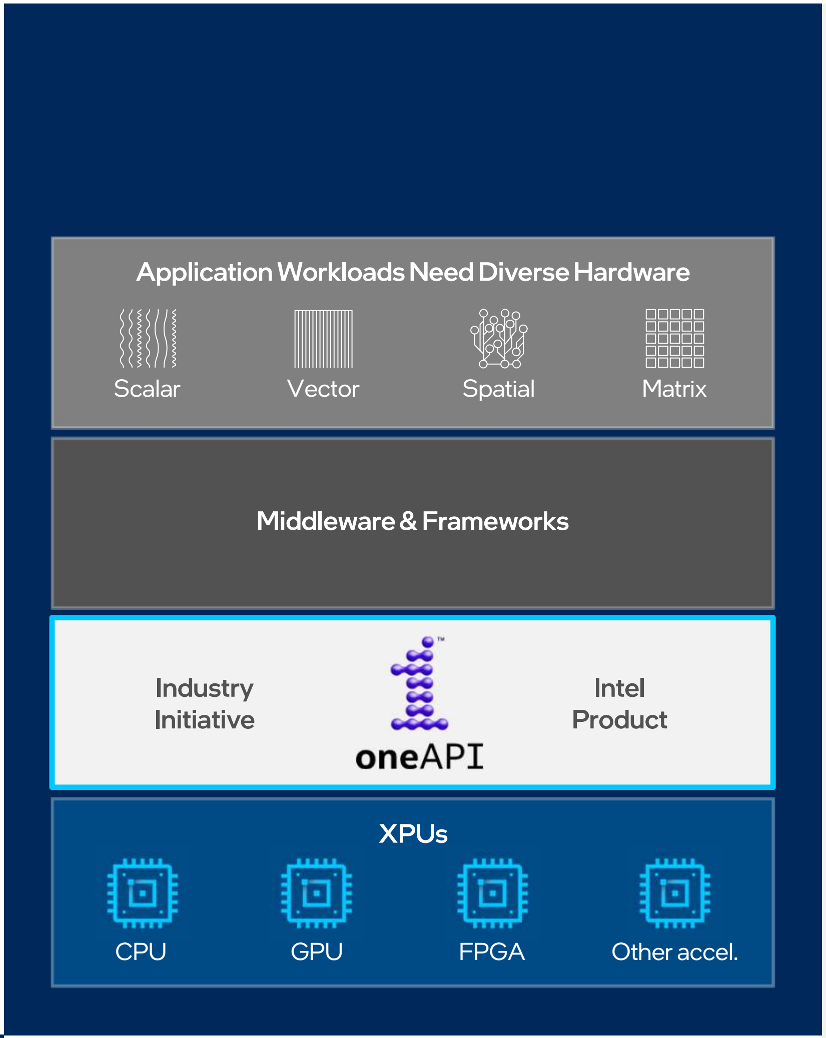
Introducing
oneAPI

Cross-architecture programming that delivers freedom to choose the best hardware

Based on industry standards and open specifications

Exposes cutting-edge performance features of latest hardware

Compatible with existing high-performance languages and programming models including C++, OpenMP, Fortran, and MPI



Optimization Notice

Intel® oneAPI AI Analytics Toolkit

Accelerate end-to-end AI and data analytics pipelines with libraries optimized for Intel® architectures

Who Uses It?

Data scientists, AI researchers, ML and DL developers, AI application developers

Top Features/Benefits

- Deep learning performance for training and inference with Intel optimized DL frameworks and tools
- Drop-in acceleration for data analytics and machine learning workflows with compute-intensive Python packages

Intel® oneAPI AI Analytics Toolkit

Deep Learning
- Intel® Optimization for TensorFlow
- Intel® Optimization for PyTorch
- Intel® Neural Compressor
- Model Zoo for Intel® Architecture

Machine Learning
- Intel® Extension for Scikit-learn
- Intel-optimized XGBoost

Data Analytics
- Intel® Distribution of Modin
- OmniSci Backend

Intel-optimized Python
- NumPy
- SciPy
- Pandas
- Numba
- Python

CPU GPU

Hardware support varies by individual tool. Architecture support will be expanded over time.

Get the Toolkit HERE or via these locations

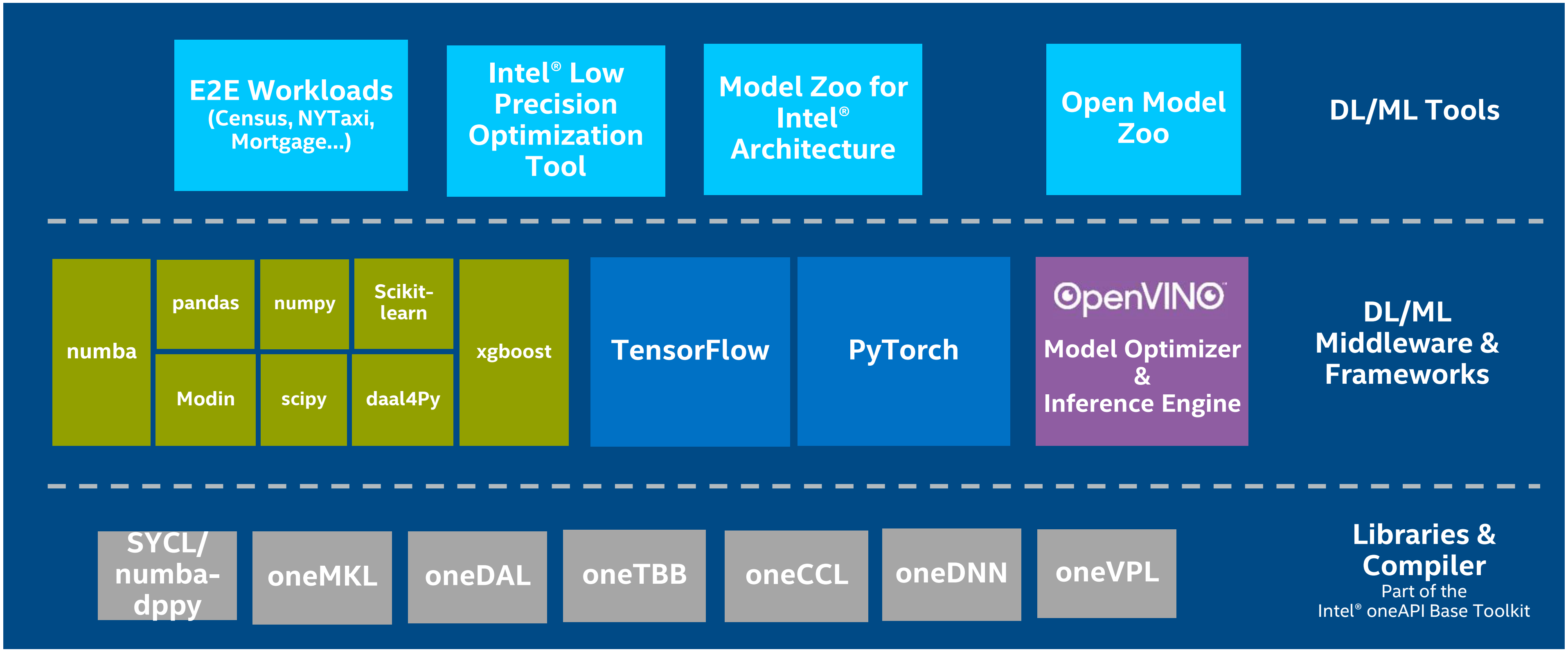
Intel Installer | Docker | Apt, Yum | Conda | Intel® DevCloud

Learn More: software.intel.com/oneapi/ai-kit

Back to Domain-specific Toolkits for Specialized Workloads

AI Software Stack for Intel XPUs

Intel offers a Robust Software Stack to Maximize Performance of Diverse Workloads



Intel® VTune™ Profiler

SYCL Profiling-Tune for CPU, GPU & FPGA

Analyze SYCL

See the lines of SYCL that consume the most time

Tune for Intel CPUs, GPUs & FPGAs

Optimize for any supported hardware accelerator

Optimize Offload

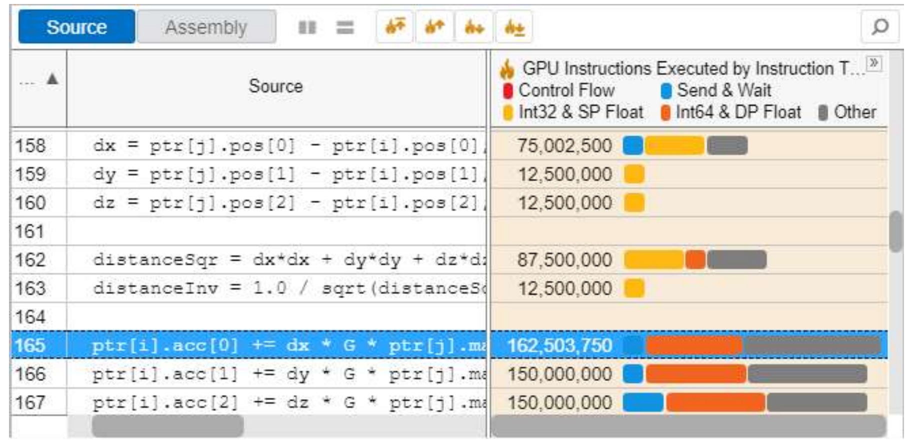
Tune OpenMP offload performance

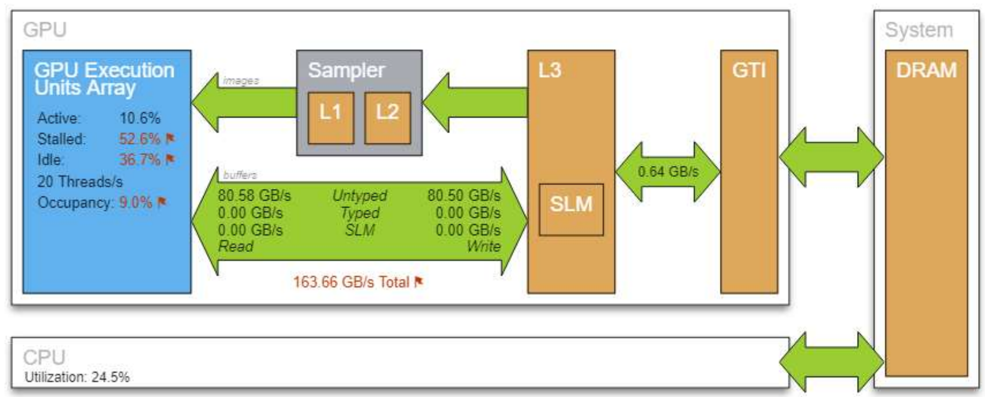
Wide Range of Performance Profiles

CPU, GPU, FPGA, threading, memory, cache, storage…

Supports Popular Languages

SYCL, C, C++, Fortran, Python, Go, Java, or a mix





There will still be a need to tune for each architecture.

Intel® Advisor

Design Assistant - Design for Modern Hardware

Offload Advisor

Estimate performance of offloading to an accelerator

Roofline Analysis

Optimize CPU/GPU code for memory and compute

Vectorization Advisor

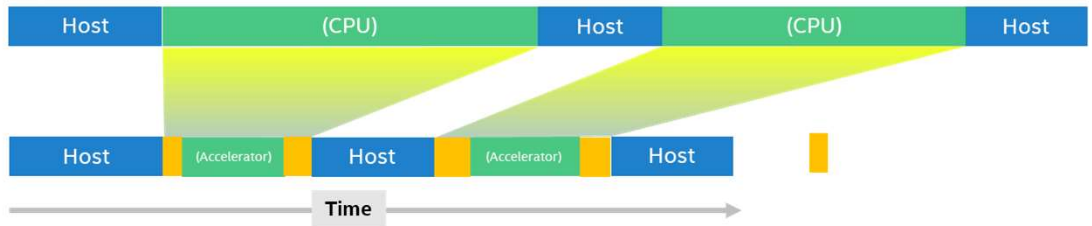
Add and optimize vectorization

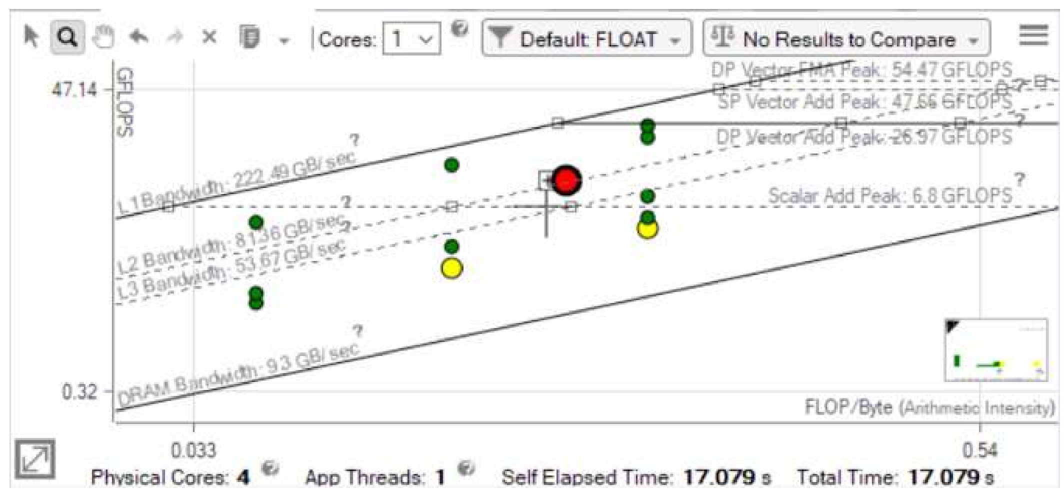
Threading Advisor

Add effective threading to unthreaded applications

Flow Graph Analyzer

Create and analyze efficient flow graphs





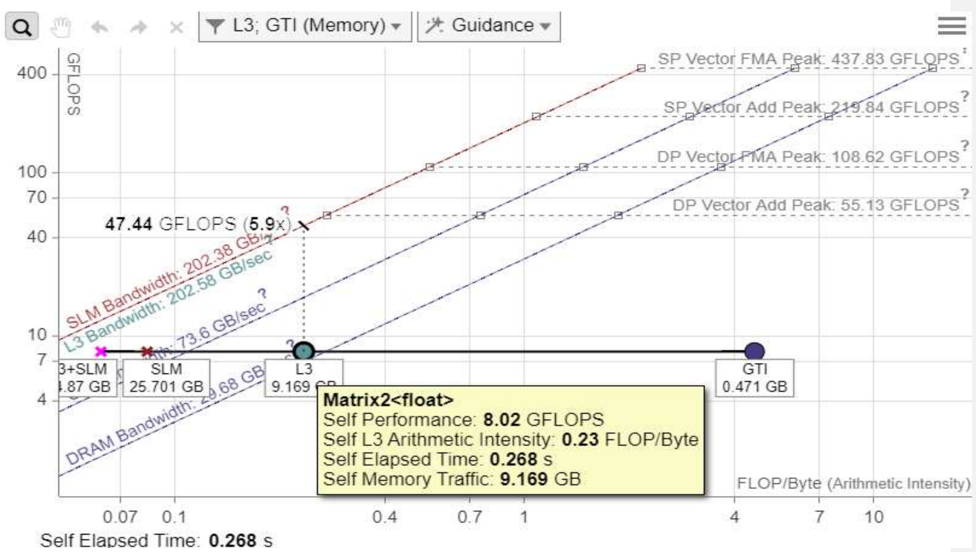
There will still be a need to tune for each architecture.

Find Effective Optimization Strategies

Intel® Advisor - GPU Roofline

GPU Roofline Performance Insights

- Highlights poor performing loops
- Shows performance 'headroom' for each loop
  - Which can be improved
  - Which are worth improving
- Shows likely causes of bottlenecks
  - Memory bound vs. compute bound
- Suggests next optimization steps



Learn More at the Intel® DevCloud for oneAPI

Free Access, A Fast Way to Start Coding

A development sandbox to develop, test and run workloads across a range of Intel® CPUs, GPUs, and FPGAs using Intel's oneAPI software

For customers focused on data-centric workloads on a variety of Intel® architecture

- Learn Data Parallel C++
- Use Intel® oneAPI Toolkits
- Evaluate Workloads
- Prototype Your Project
- Build Cross-architecture Applications

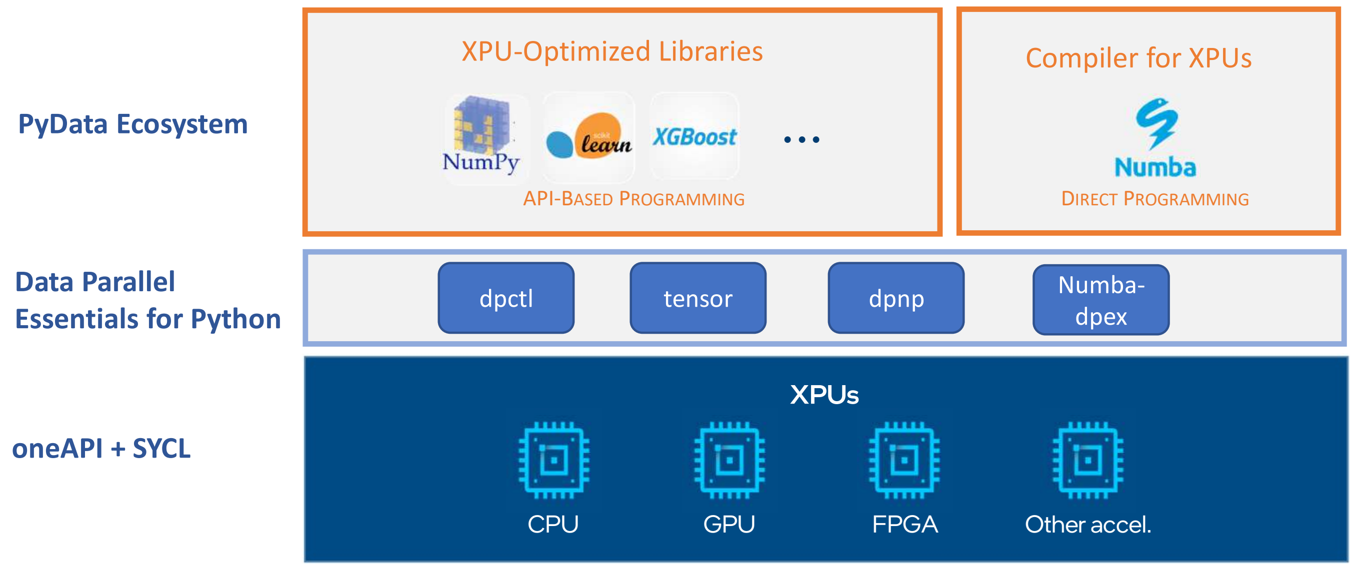
No Downloads | No Hardware Acquisition | No Installation | No Set-up & Configuration

Get Up & Running in Seconds!

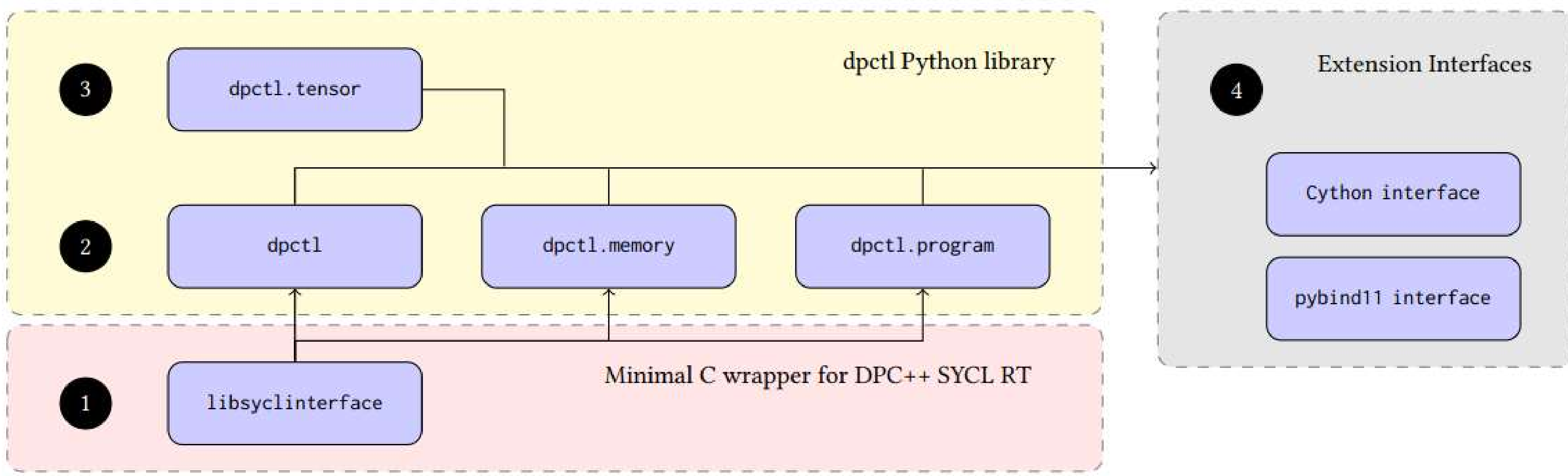
https://devcloud.intel.com/oneapi/get_started/

Data Parallel Essentials for Python

Fostering a oneAPI/SYCL-based ecosystem for PyDATA



dpctl – Data parallel control



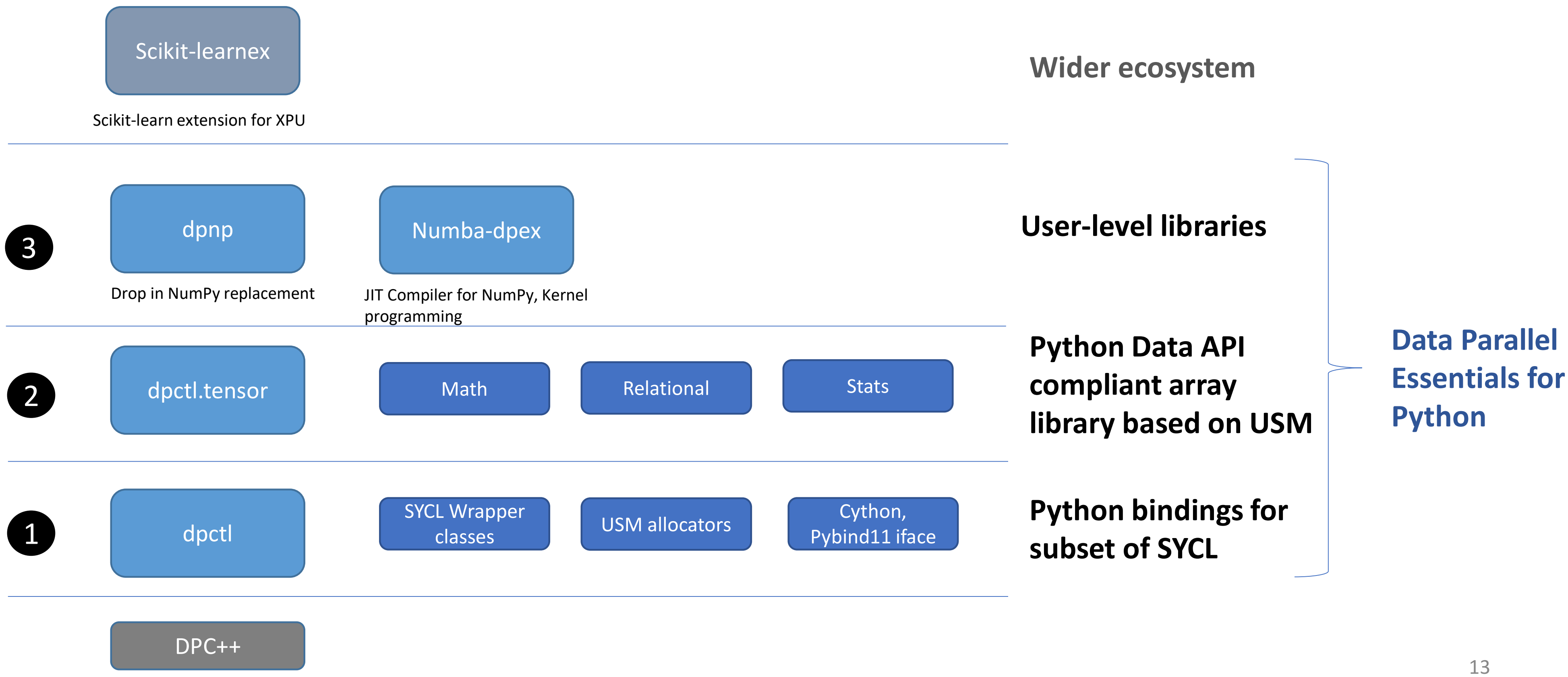
1 Library providing a minimal C API for the main DPC++ SYCL runtime classes

2 Python modules exposing SYCL runtime classes, USM allocators, and kernel bundle

3 A data API standard complaint array library supporting USM allocated memory

4 Native API to use dpctl objects in Cython and pybind11 extensions modules

Current Ecosystem



Compute Follows data

Offload Model

- Pythonic offload model following array API spec (https://data-apis.org/array-api/latest/)
- Offload happens where data currently resides ("compute follows data")

```
X = dp.array([1,2,3])
Y = X * 4
```

executed on default device

```
X = dp.array([1,2,3], device="gpu:0")
Y = X * 4
```

executed on "gpu:0" device

```
X = dp.array([1,2,3], device="gpu:0")
Y = dp.array([1,2,3], device="gpu:1")
Z = X + Y
```

Error! Arrays are on different devices

Programming Model

Compute Follows Data

- Pythonic offload model following array API spec
- Explicit control over execution based on data placement

```
import dpnp as dp
    # Case 1
    #  Allocate X on the default device
    X = dp.array([1,2,3])
    #  scaling of X executed on device of X, result
         placed into Y
    Y = X * 4
    # Case 2
    #  Allocate X on "gpu:1"
    X = dp.array([1,2,3], device="gpu:1")
    # Executed on "gpu:1"
    Y = X * 4
    # Case 3
    X1 = dp.array([1,2,3], device="gpu:1")
    X2 = dp.array([1,2,3], device="gpu:0")
    #  error!
    Y = X1 + X2

    # Arrays can be associated with another device
    # (copy is performed if needed)
    X1a = X1.to_divice(device=dev)
```

Numba-dpex

oneAPI

Array-style programming

```
@njit(parallel=True)
def l2_distance(a, b, c)
    return np.sum((a-b)**2)
```

NumPy (array) style programming. Requires minimum code changes to compile existing Numpy code for XPU.

Explicit prange (parfor) loops

```
@njit(parallel=True)
def l2_distance(a, b, c)
    s = 0.0
    for i in prange(len(a))
        s += (a[i]-b[i])**2
    return s
```

Parfor-style programming. Preferred by some users when iteration space requires complex indexing.
Unique for CPU. Intel extends to XPU via numba-dpex. No CUDA alternatives to date

OpenCl-style kernel programming

```
@kernel(access_type={"read_only": ["a", "b"], write_only:["c"]})
def l2_distance(a, b, c)
    i = numba_dpex.get_global_id(0)
    j = numba_dpex.get_global_id(1)
    sub = a[i,j] - b[i,j]
    sq = sub ** 2
    atomic.add(c, 0, sq)
```

Most advanced programming model. Recommended to get highest performance on XPU yet avoiding DPC++.
Nvidia @cuda.jit offers this programming model in Numba

Automatic offload using @njit Decorator

Import njit and prange from numba

Use @njit decorator to directly detect data parallel kernels using numpy expressions

Automatic offload mode for NumPy data-parallel expressions

Use dpctl.device context to offload this to a device

```
import dpctl
import numpy as np
import numba

@numba.njit(parallel=True)
def l2_distance_kernel(a, b):
    sub = a - b
    sq = np.square(sub)
    sum = np.sum(sq)
    d = np.sqrt(sum)
    return d
def main():
    R = 64
    C = 1
    X = np.random.random((R,C))
    Y = np.random.random((R,C))
    device = dpctl.select_default_device()
    print("Using device ...")
    device.print_device_info()
    with dpctl.device_context(device):
        result = l2_distance_kernel(X, Y)
    print("Result :", result)
    print("Done...")
if __name__ == "__main__":
    main()
```

Explicit parallel for loop - @njit Decorator

Import njit and prange from numba

Use @njit decorator to directly detect data parallel kernels using numpy expressions

Use prange to specify explicitly a loop to be parallelized

Use dpctl.device context to offload this to a device

```
import numpy as np
from numba import njit, prange
import dpctl

@njit
def add_two_arrays(b, c):
    a = np.empty_like(b)
    for i in prange(len(b)):
        a[i] = b[i] + c[i]
    return a

def main():
    N = 10
    b = np.ones(N)
    c = np.ones(N)
    device = dpctl.select_default_device()
    with dpctl.device_context(device):
        result = add_two_arrays(b, c)

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

@dppy.kernel Decorator

Import dpctl

Vector addition in parallel using the @ddpy.kernel decorator

Common way of Kernel invocation

Offload this to a device

```python
import dpctl
import numba_dppy as dppy
import numpy as np

@dppy.kernel
def data_parallel_sum(a, b, c):
    i = dppy.get_global_id(0)
    c[i] = a[i] + b[i]

def driver(a, b, c, global_size):
    data_parallel_sum[global_size, dppy.DEFAULT_LOCAL_SIZE
](a, b, c)
    print("C ", c)

def main():
    global_size = 10
    N = global_size
    print("N", N)
    a = np.array(np.random.random(N), dtype=np.float32)
    b = np.array(np.random.random(N), dtype=np.float32)
    c = np.ones_like(a)
    with dpctl.device_context("opencl:gpu"):
      driver(a, b, c, global_size)

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

What Categories of AI are covered?

Types of Machine Learning

Supervised	data points have known outcome
Unsupervised	data points have unknown outcome

Types of Supervised Learning

Regression	outcome is continuous (numerical)
Classification	outcome is a category

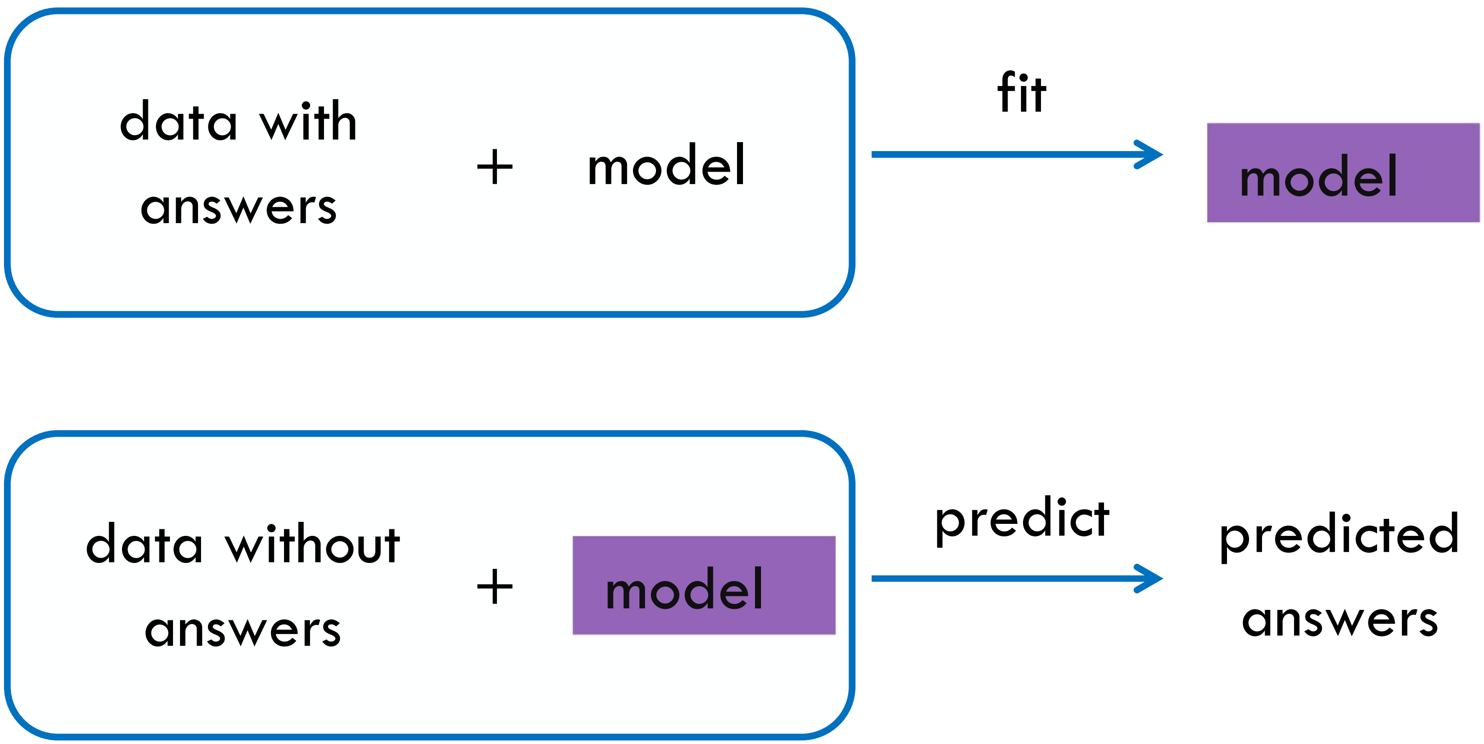
Types of Unsupervised Learning

Clustering	identify unknown structure in data
Dimensionality Reduction	use structural characteristics to simplify data

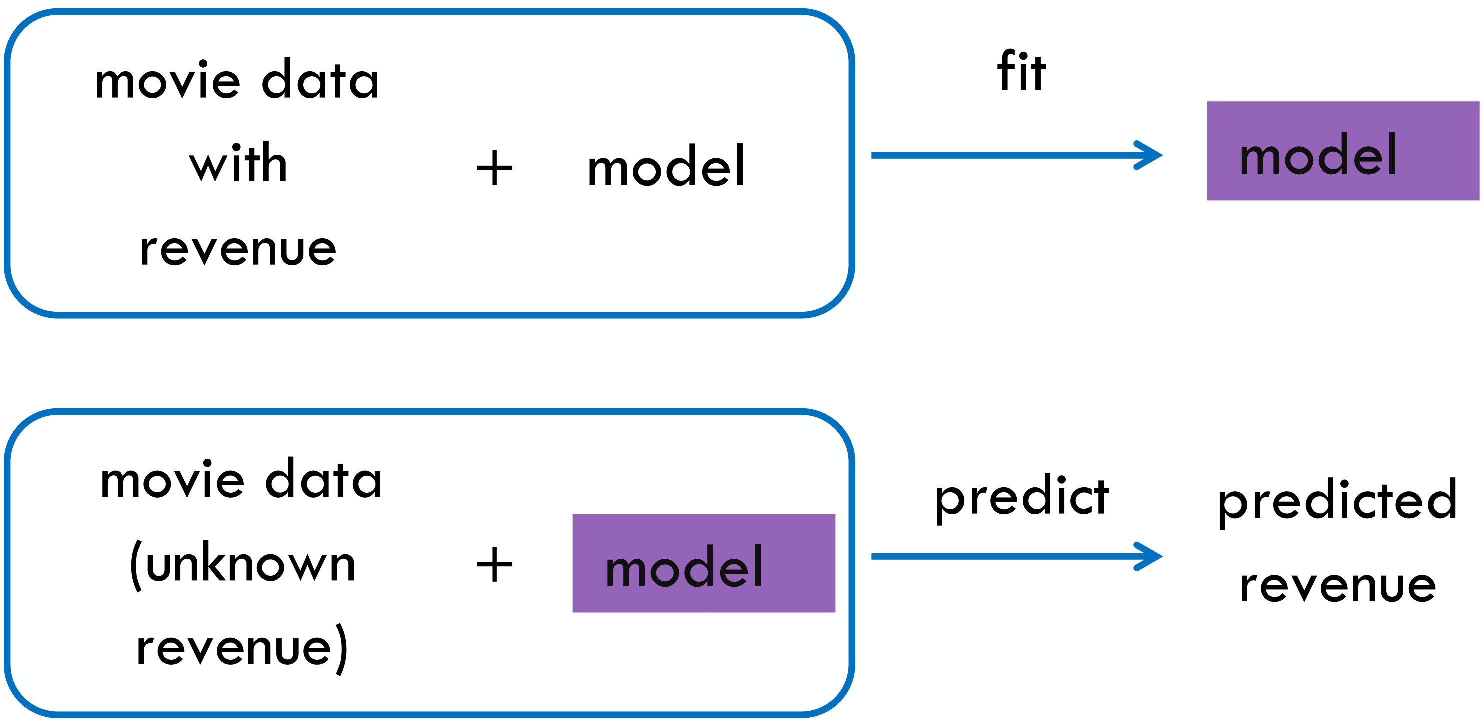
Classification & Regression

- Have features in a dataset “X”
- Have targets in a column “y”
- Goal: learn to predict “y”
- Classification: discrete targets (“cats”, “dogs”, “hair”)
- Regression: continuous targets (12.37, -15.2, 98.6)

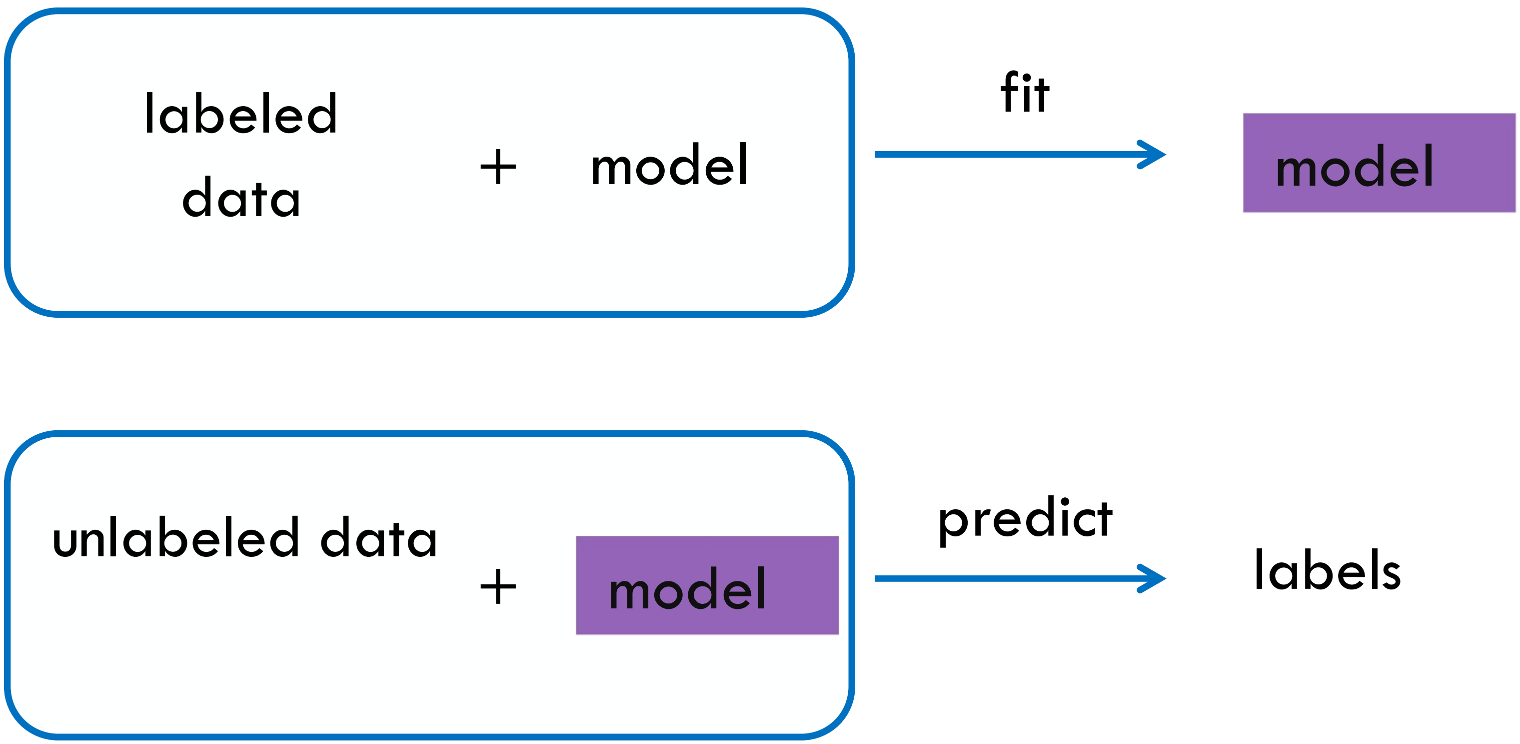
Supervised Learning Overview



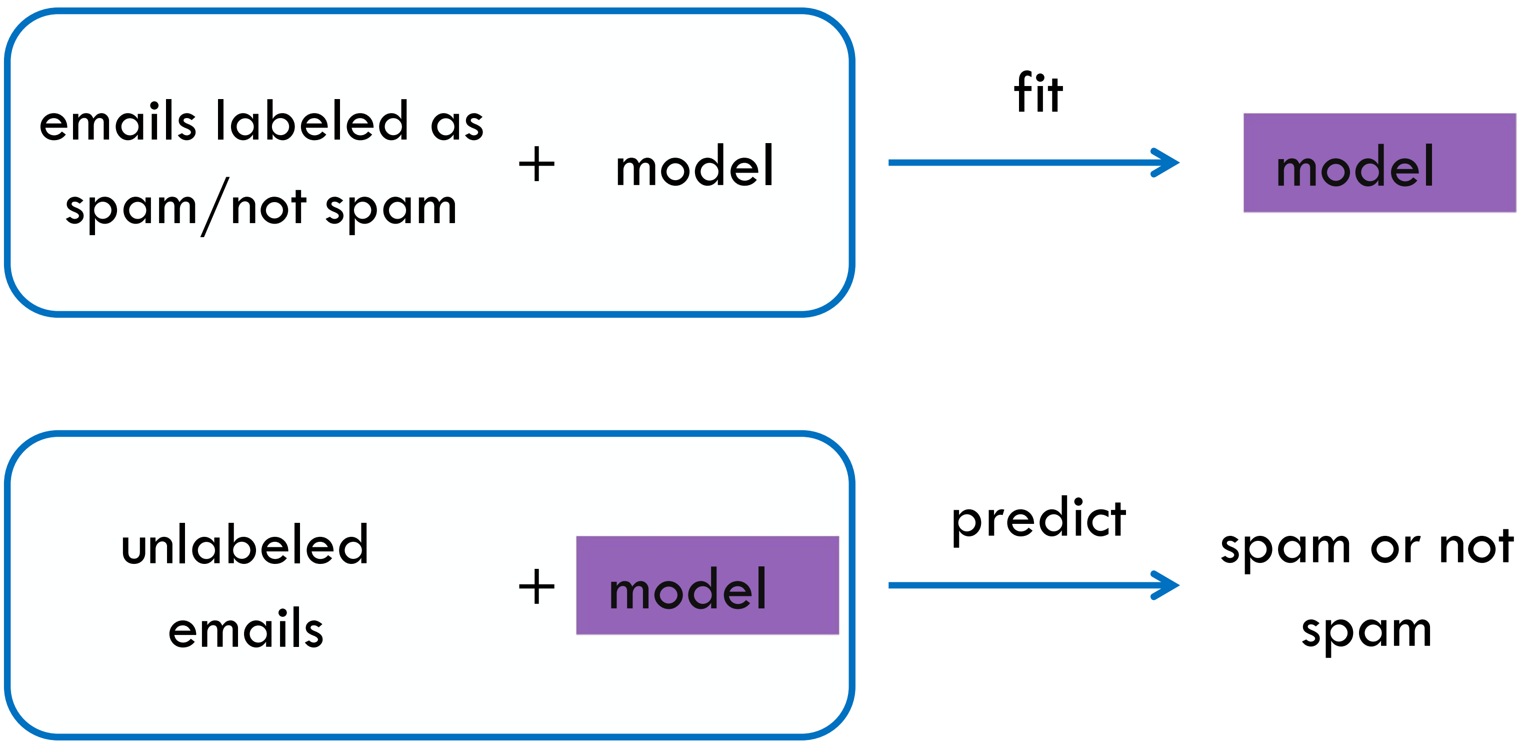
Regression: Numeric Answers



Classification: Categorical Answers

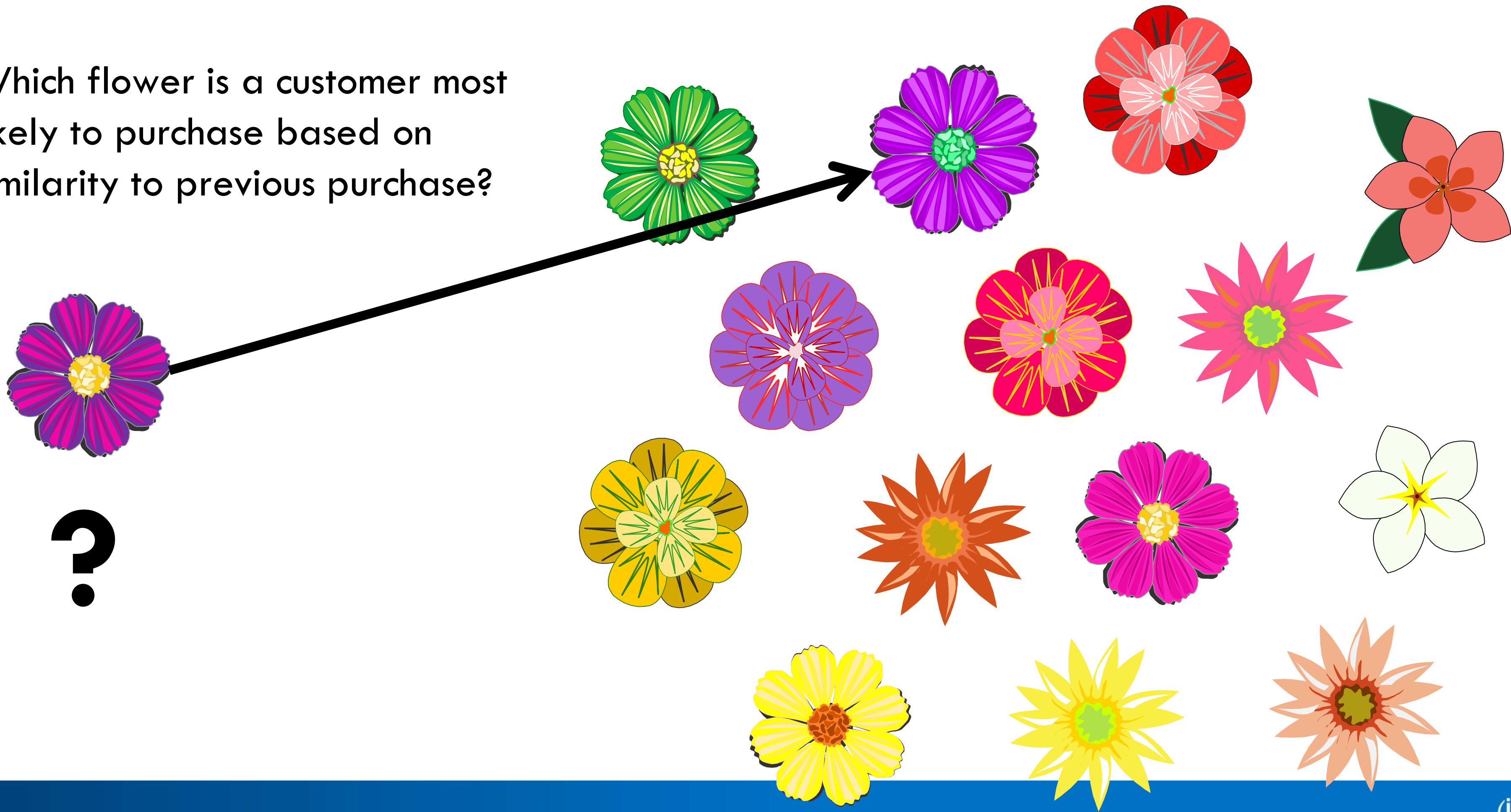


Classification: Categorical Answers



What is Classification?

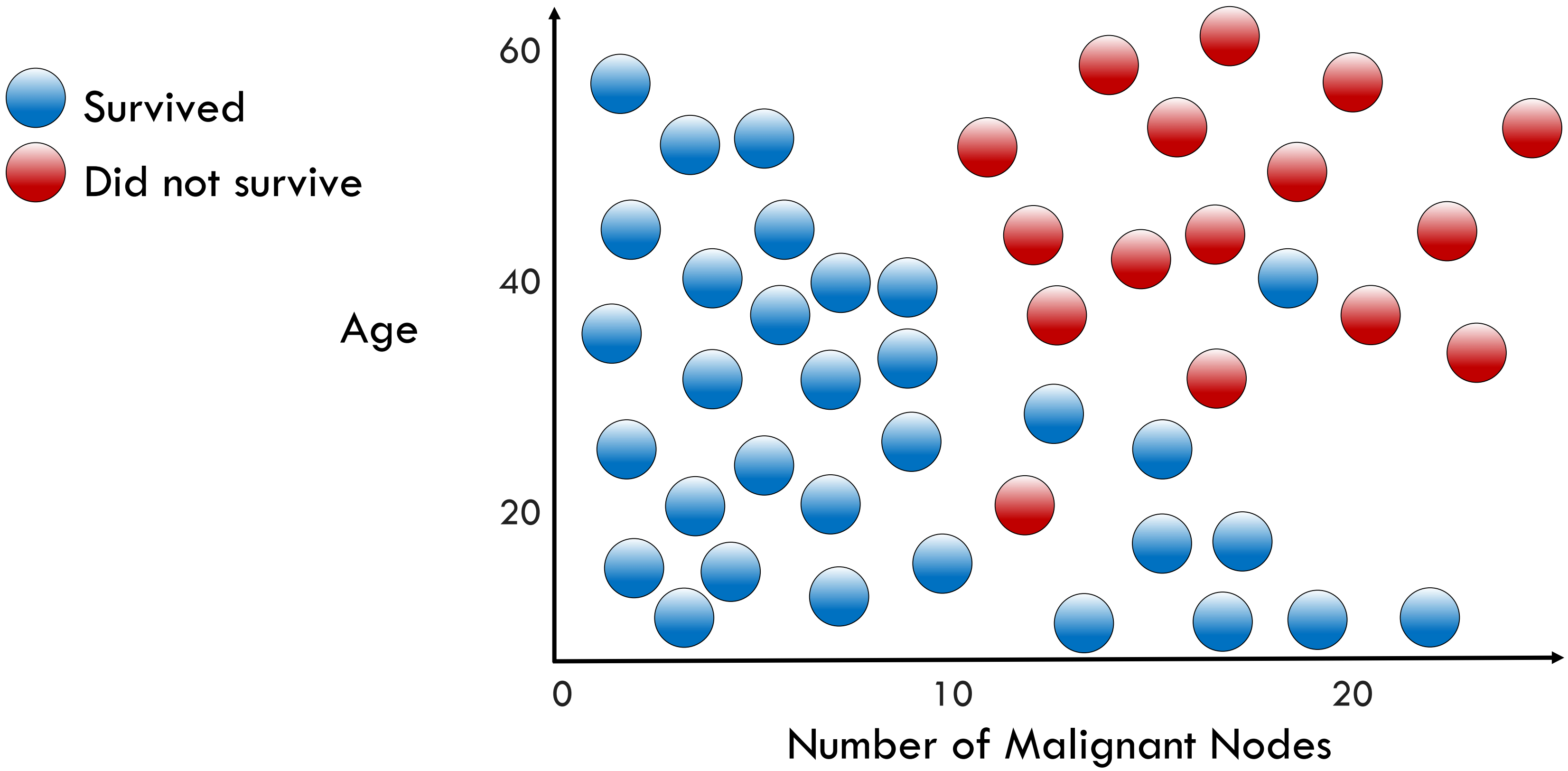
Which flower is a customer most likely to purchase based on similarity to previous purchase?



What is Needed for Classification?

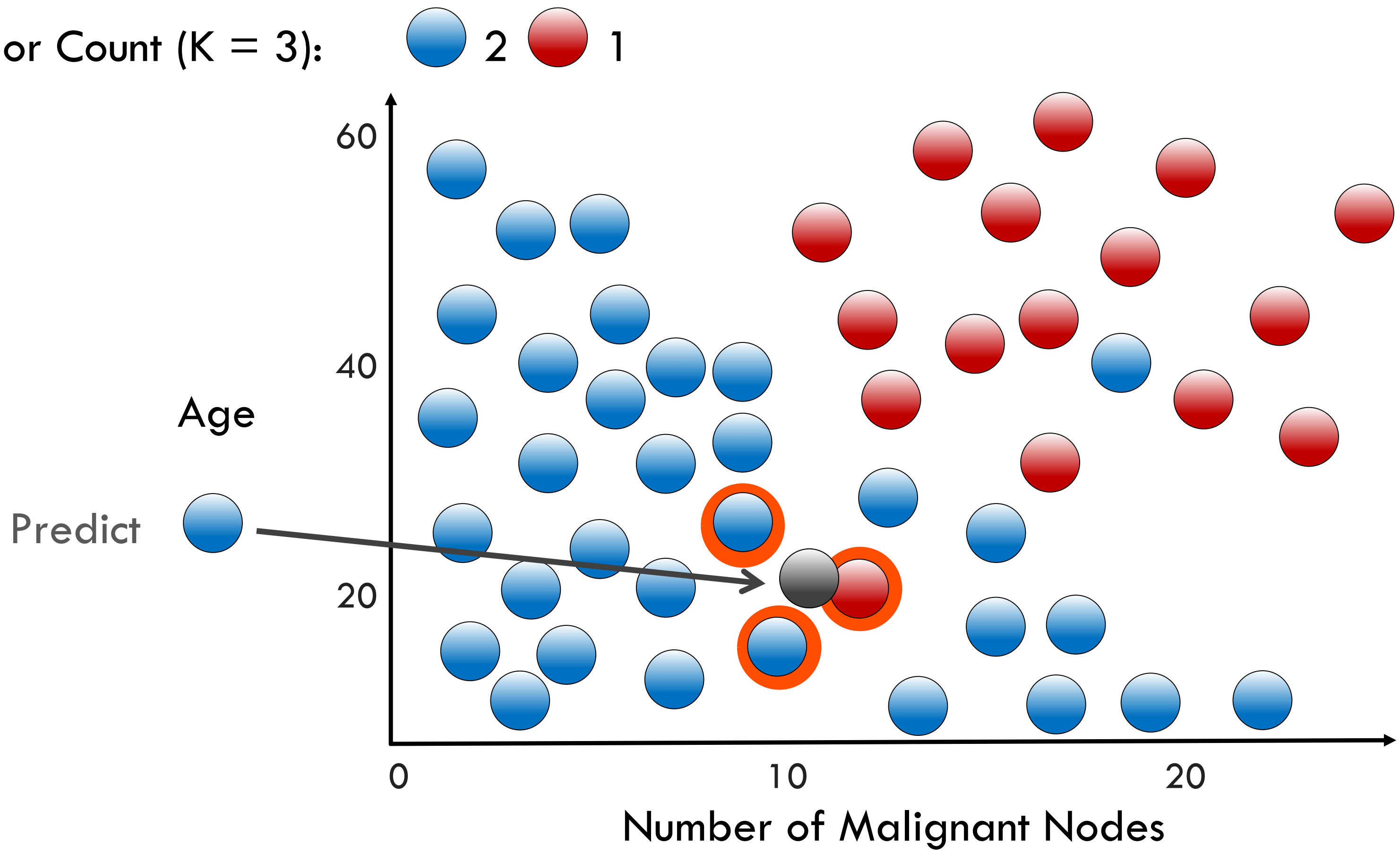
- Model data with:
  - Features that can be quantitated
  - Labels that are known
- Method to measure similarity

K Nearest Neighbors Classification



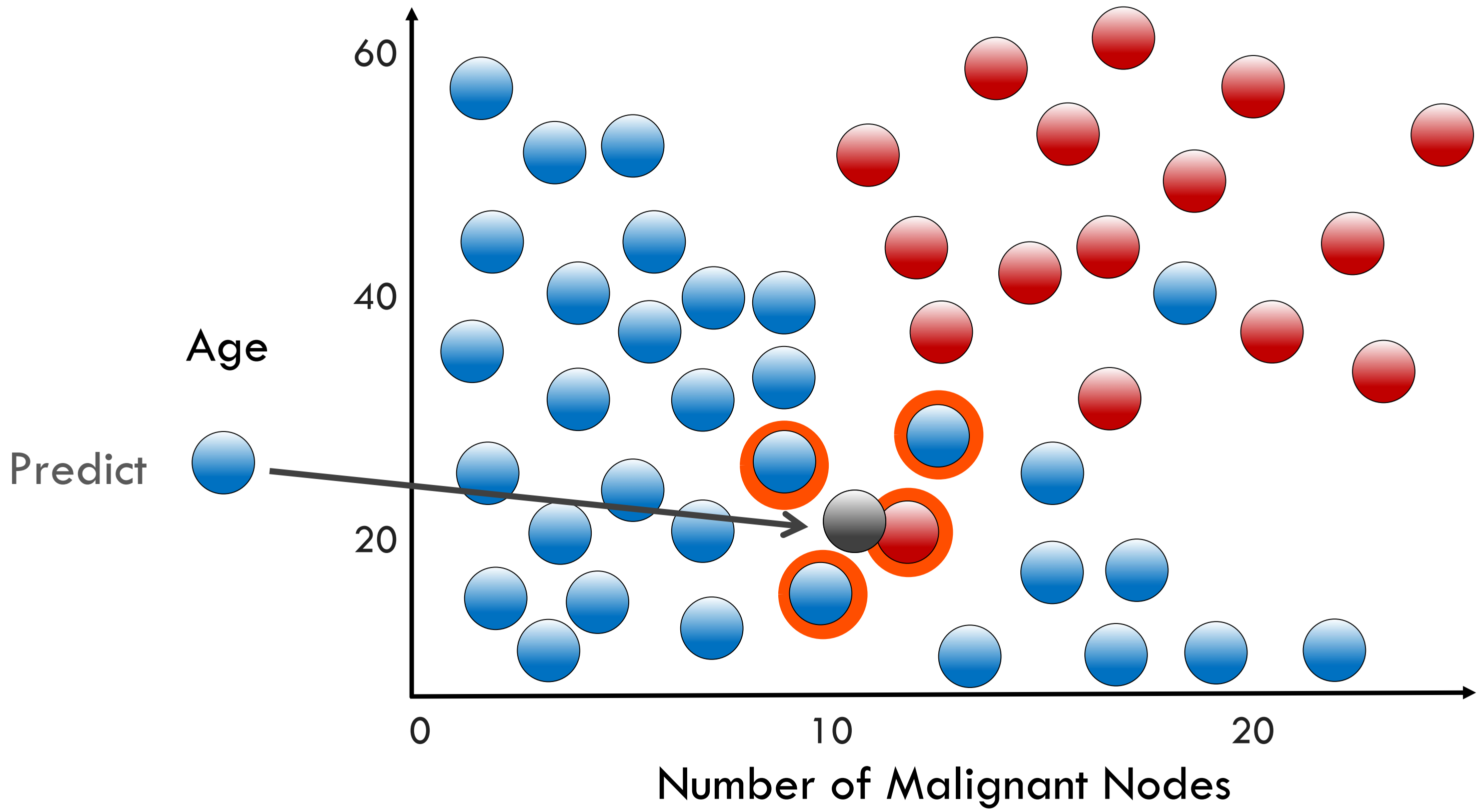
K Nearest Neighbors Classification

Neighbor Count (K = 3):



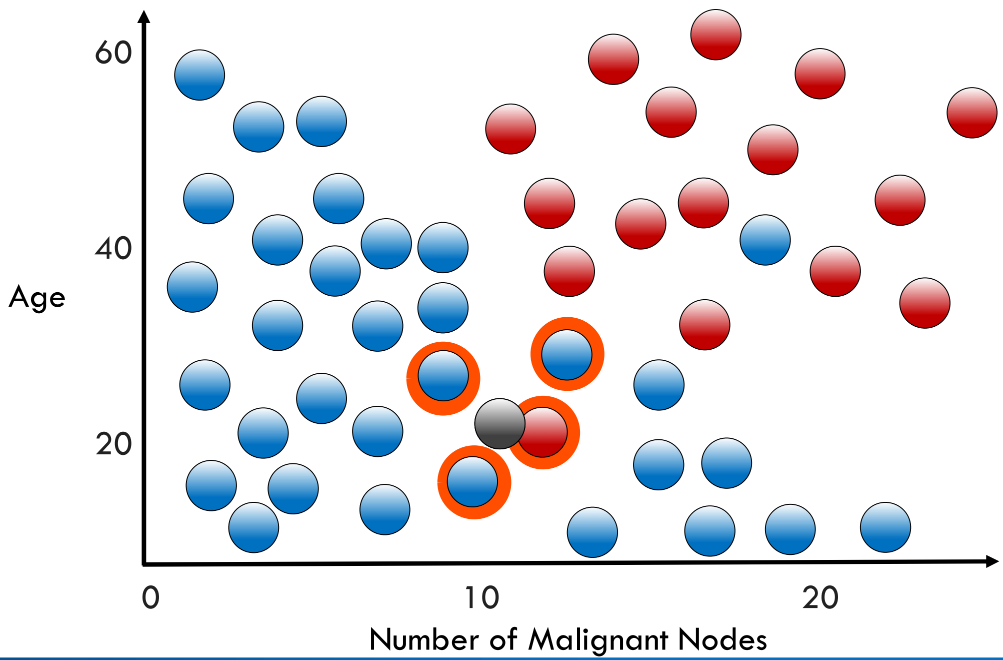
K Nearest Neighbors Classification

Neighbor Count (K = 4): 3 1

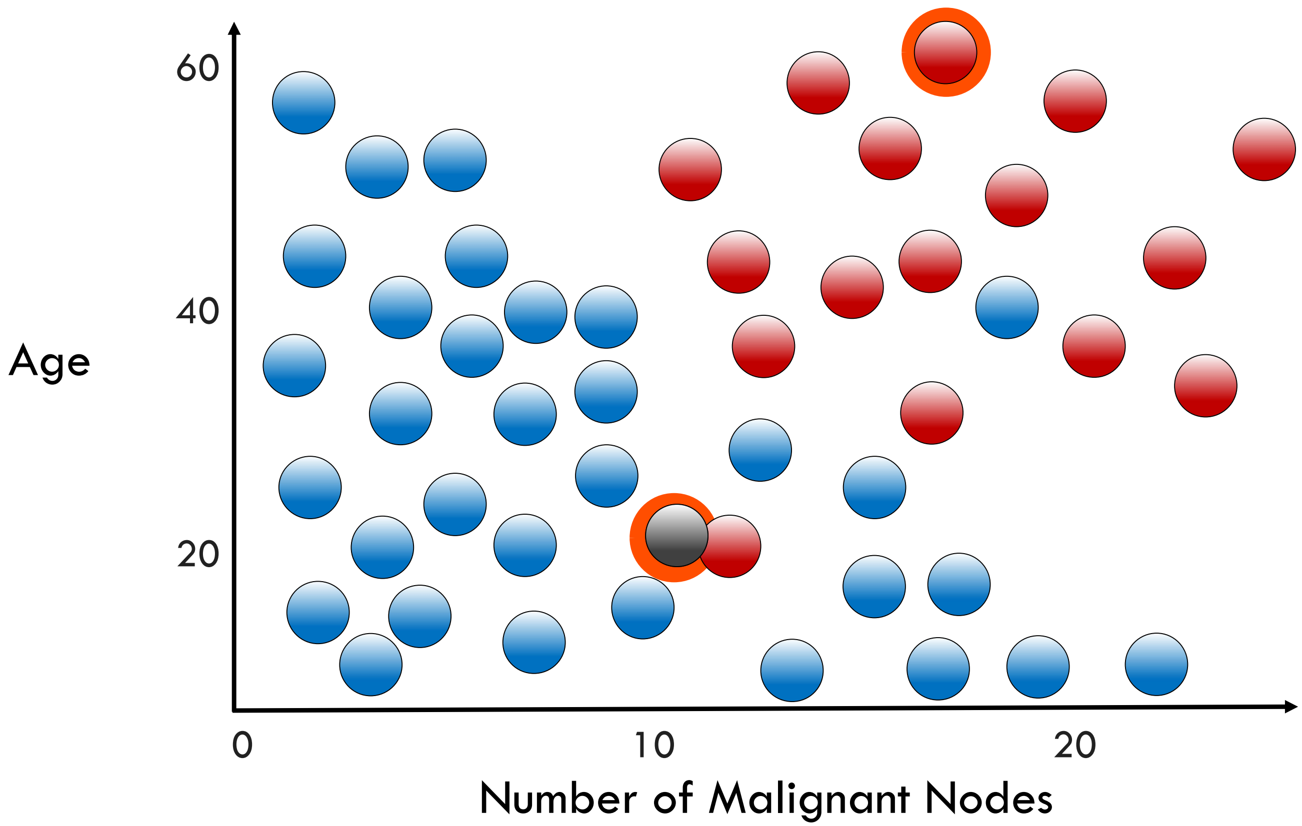


What is Needed to Select a KNN Model?

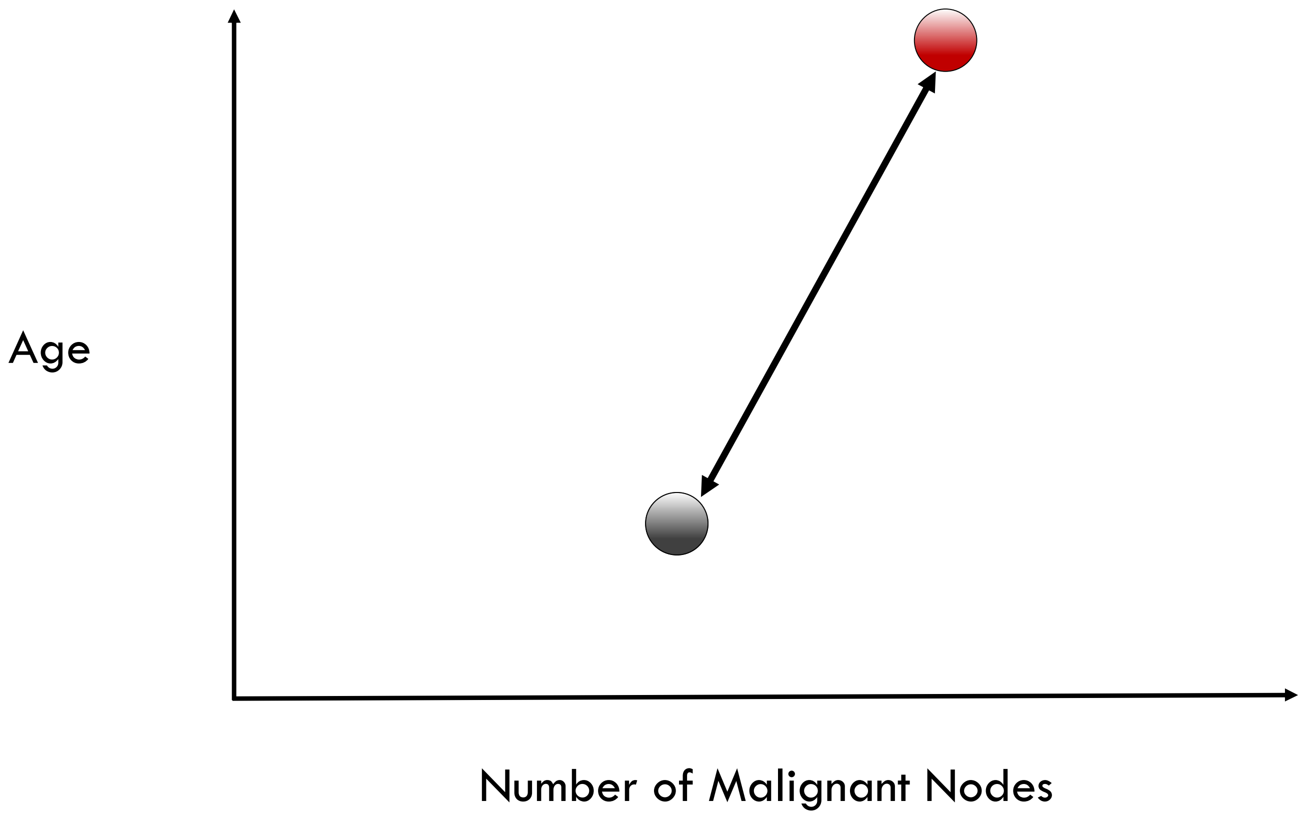
- Correct value for 'K'
- How to measure closeness of neighbors?



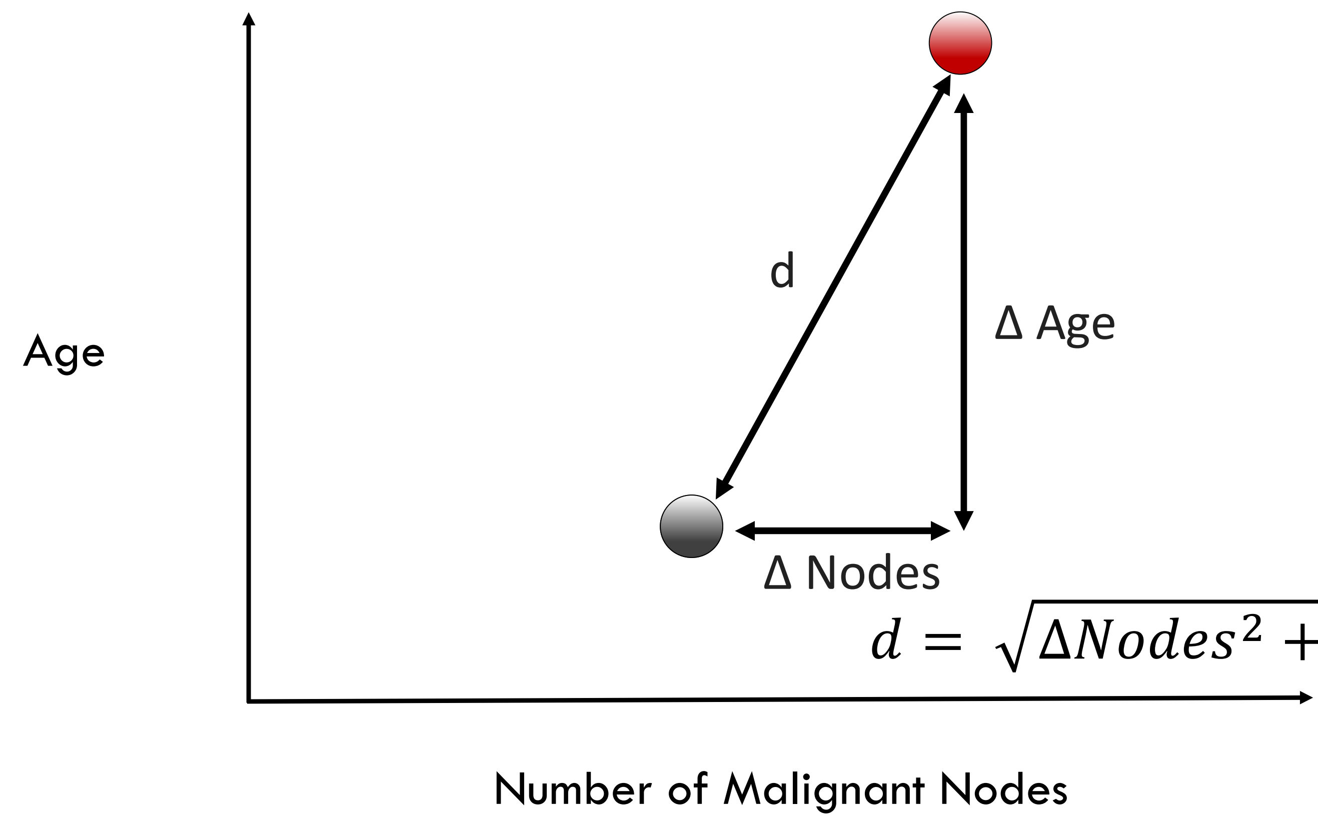
Measurement of Distance in KNN



Euclidean Distance

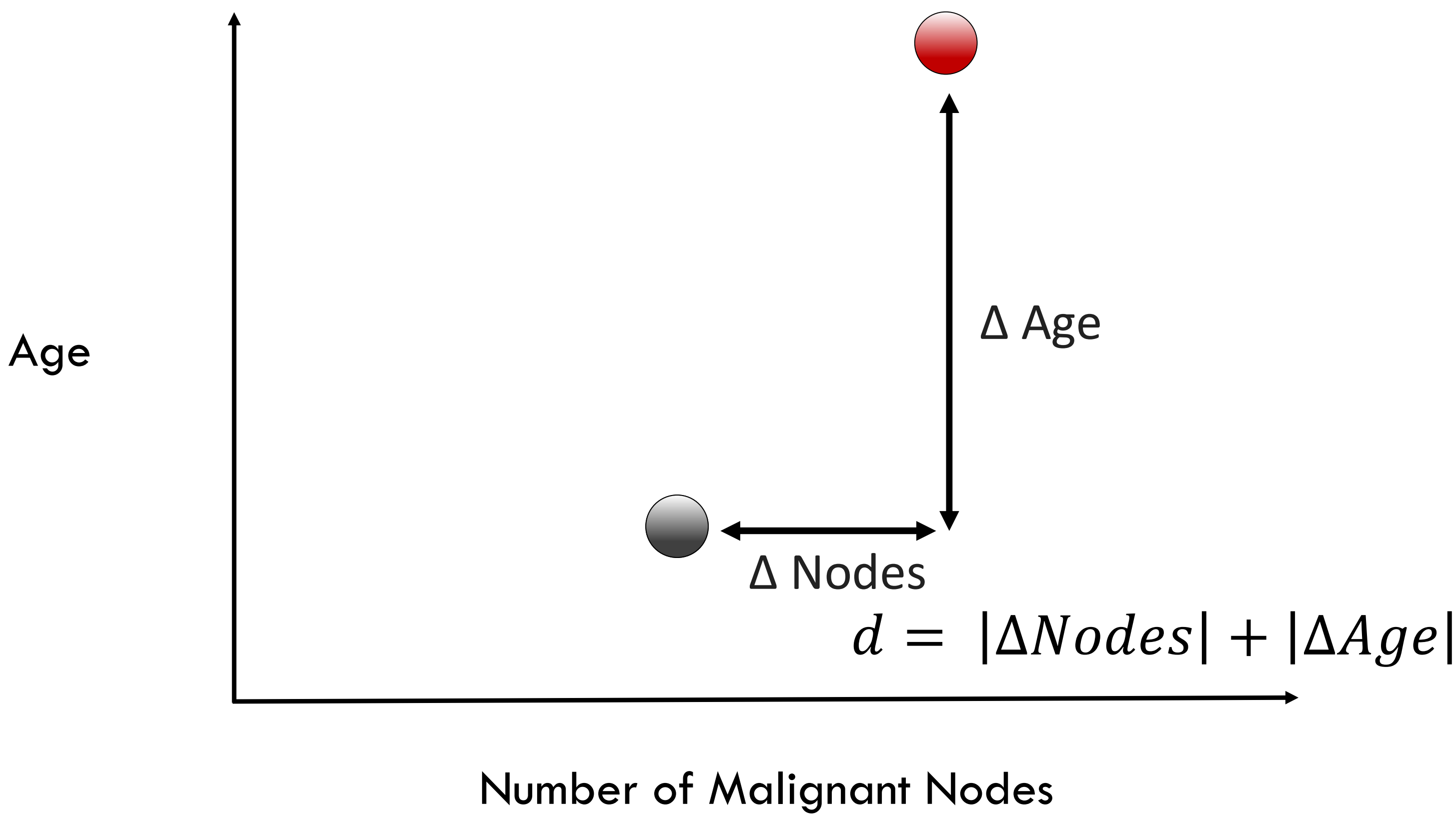


Euclidean Distance (L2 Distance)



$$d = \sqrt{\Delta Nodes^2 + \Delta Age^2}$$

Manhattan Distance (L1 or City Block Distance)



Introduction to patching

- Intel® Extension for Scikit-learn* provides a way to accelerate existing scikit-learn code.
- In code, we will **import sklearnex** – this is the python library name for Intel Extensions for Scikit-learn*
- Via patching: replacing the stock scikit-learn algorithms with their optimized versions provided by the extension.
- You may enable patching in different ways:
- Without editing the code: using a command line flag
- Within code: using an import and a function call
- Un-patching: using methods to follow

Patching Alternatives

- Command line:

```
python -m sklearnex my_application.py
```

- Inside script or Jupyter Notebook:

```
from sklearnex import patch_sklearn
patch_sklearn()
```

K Nearest Neighbor: The Syntax

Import sklearnex

Apply “monkey patch”

Import desired sklearn algorithms AFTER the patch

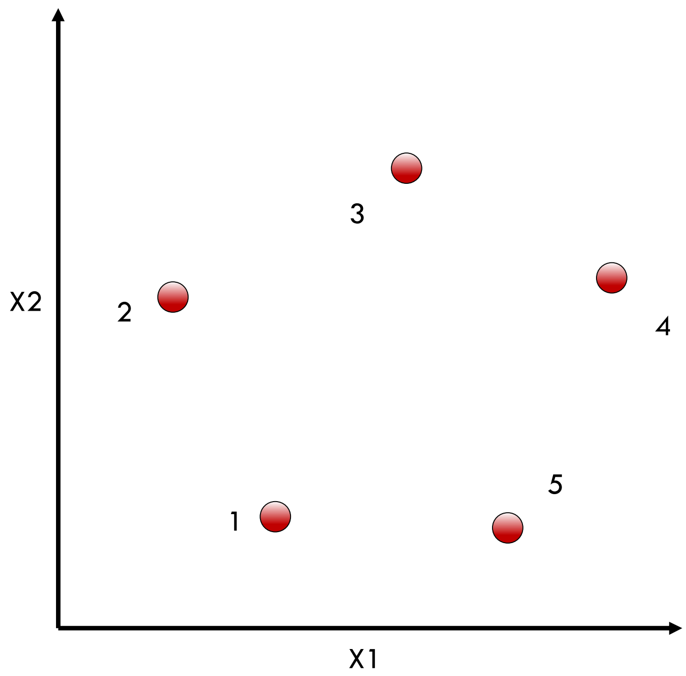
```
from sklearnex import patch_sklearn
patch_sklearn() # apply BEFORE import of targets
```

```
from sklearn.linear_model import KNeighborsClassifier
```

```
# Create an instance of the class
KNN = KNeighborsClassifier(n_neighbors=3, n_jobs=-1)
# Fit the instance on the data and then predict the
expected value
KNN = KNN.fit(X_data, y_data)
y_predict = KNN.predict(X_data)
```

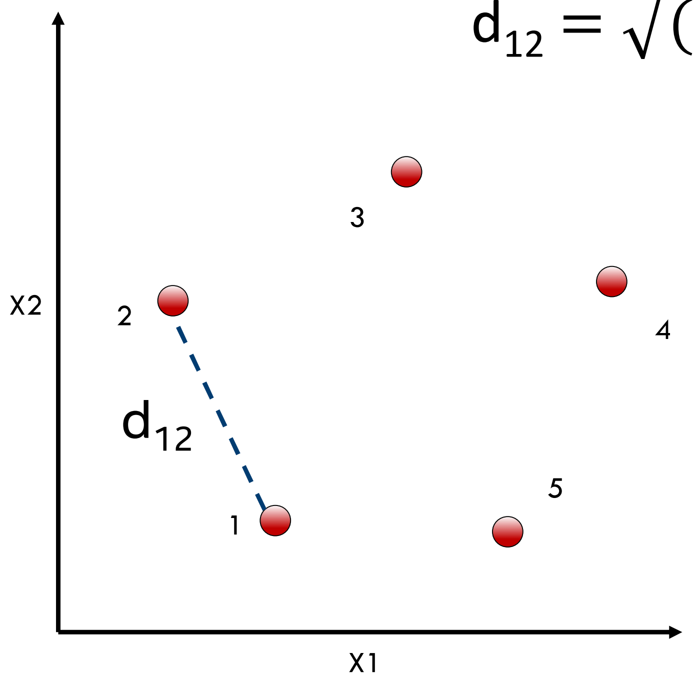
Pairwise Distance

Pairwise Distance



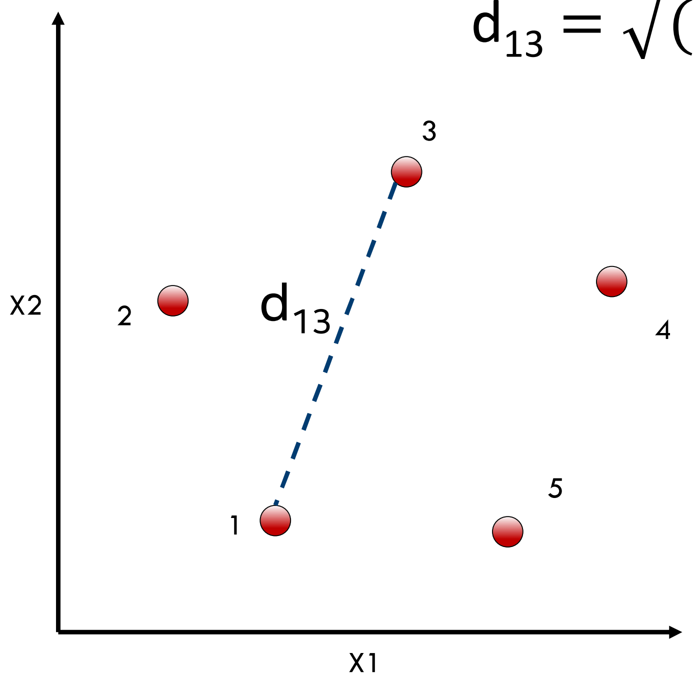
Pairwise Distance

$$d_{12} = \sqrt{(p_{x1} - p_{x2})^2 + (p_{y1} - p_{y2})^2}$$



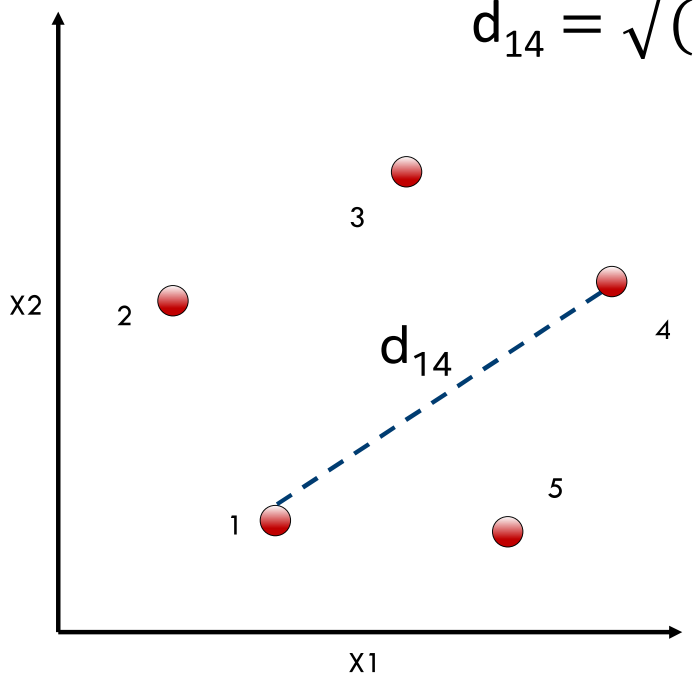
Pairwise Distance

$$d_{13} = \sqrt{(p_{x1} - p_{x3})^2 + (p_{y1} - p_{y3})^2}$$



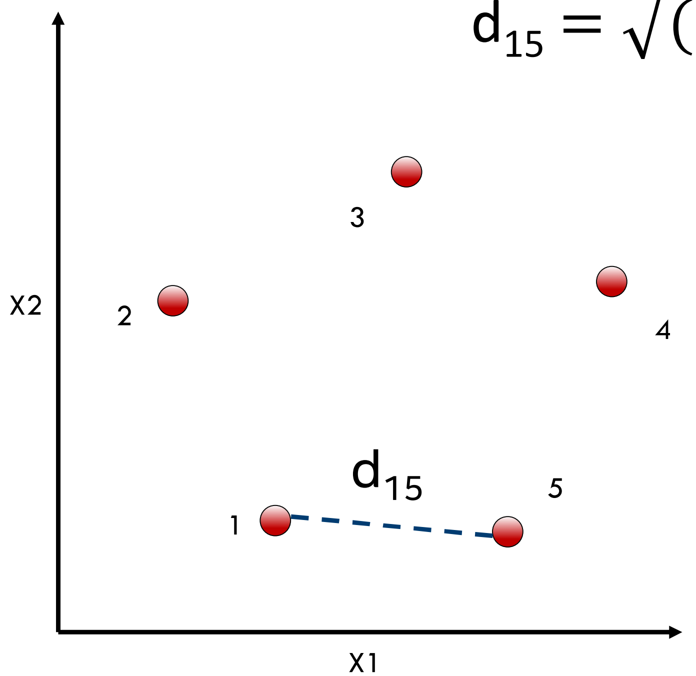
Pairwise Distance

$$d_{14} = \sqrt{(p_{x1} - p_{x4})^2 + (p_{y1} - p_{y4})^2}$$



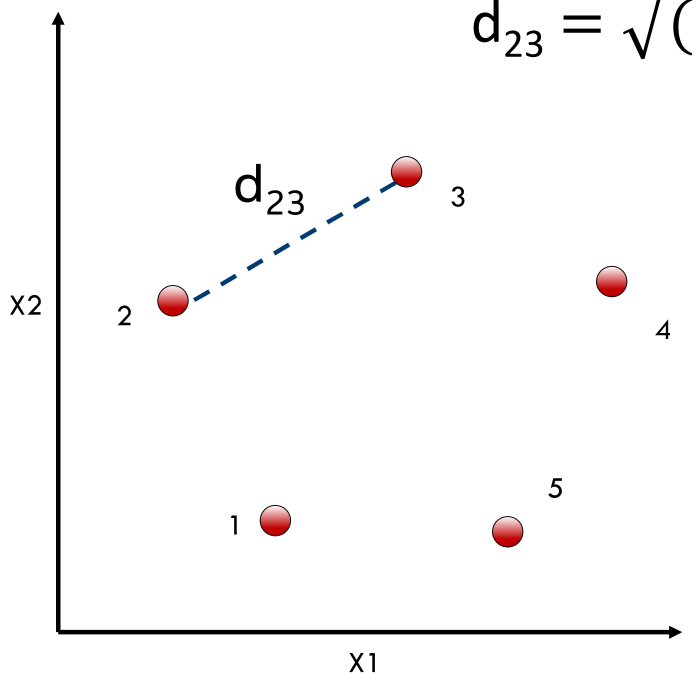
Pairwise Distance

$$d_{15} = \sqrt{(p_{x1} - p_{x5})^2 + (p_{y1} - p_{y5})^2}$$

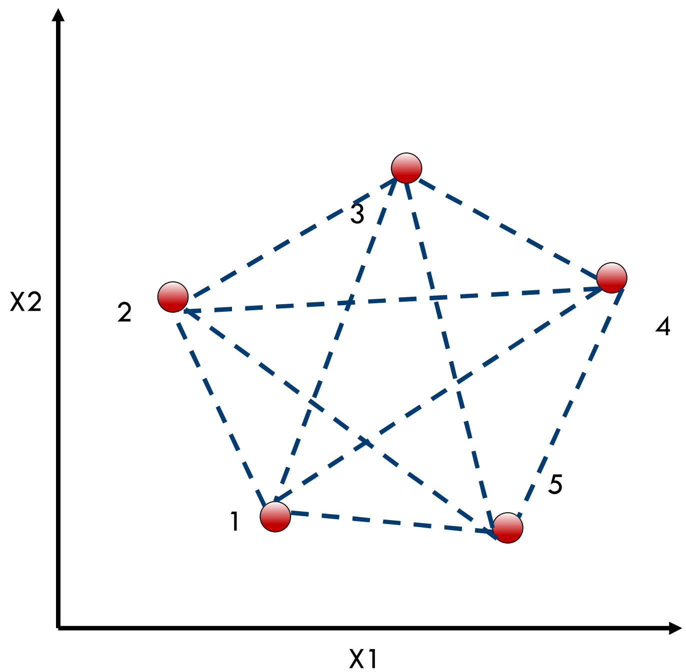


Pairwise Distance

$$d_{23} = \sqrt{(p_{x2} - p_{x3})^2 + (p_{y2} - p_{y3})^2}$$



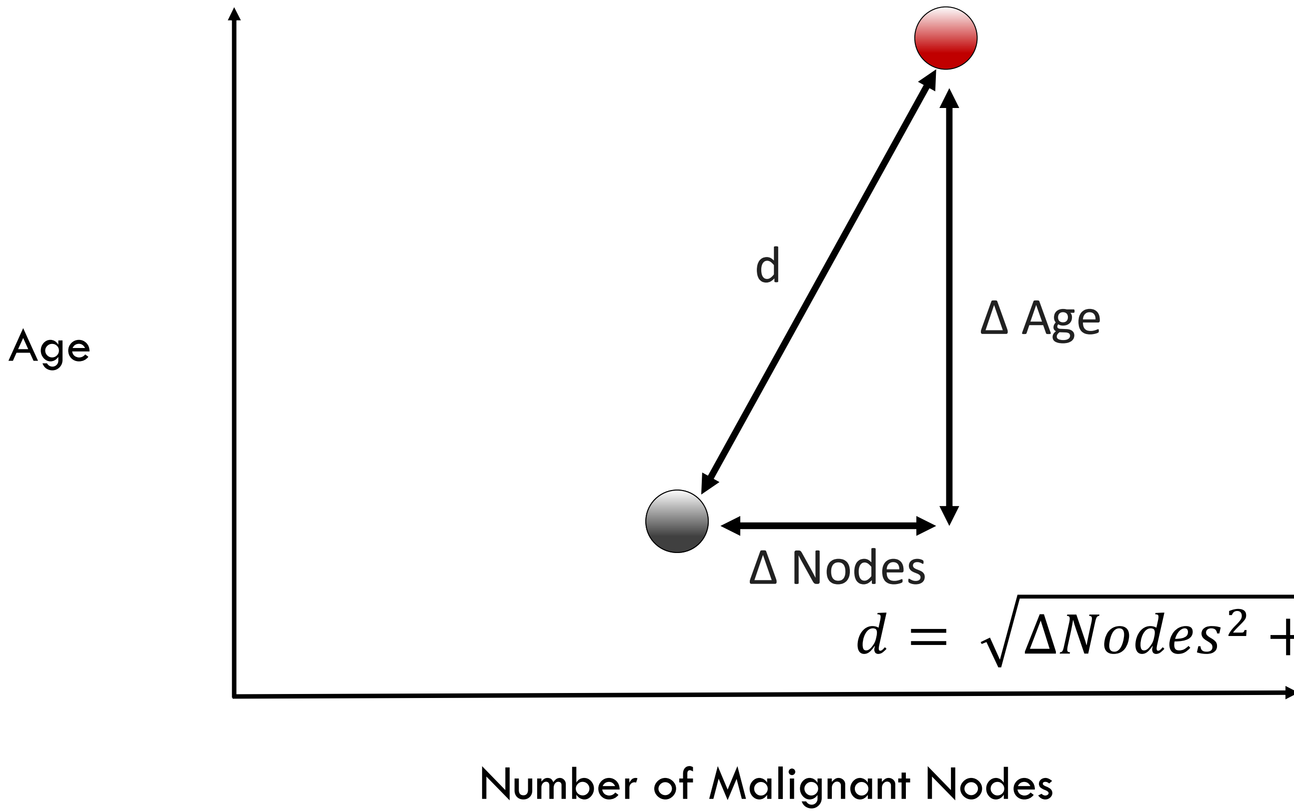
Pairwise Distance



$$\begin{vmatrix} 0 & d_{12} & d_{13} & \cdots \\ d_{21} & 0 & d_{23} & \cdots \\ d_{31} & d_{32} & 0 & \cdots \\ \vdots & \vdots & \vdots & \end{vmatrix}$$

Euclidean Distance (L2 Distance)

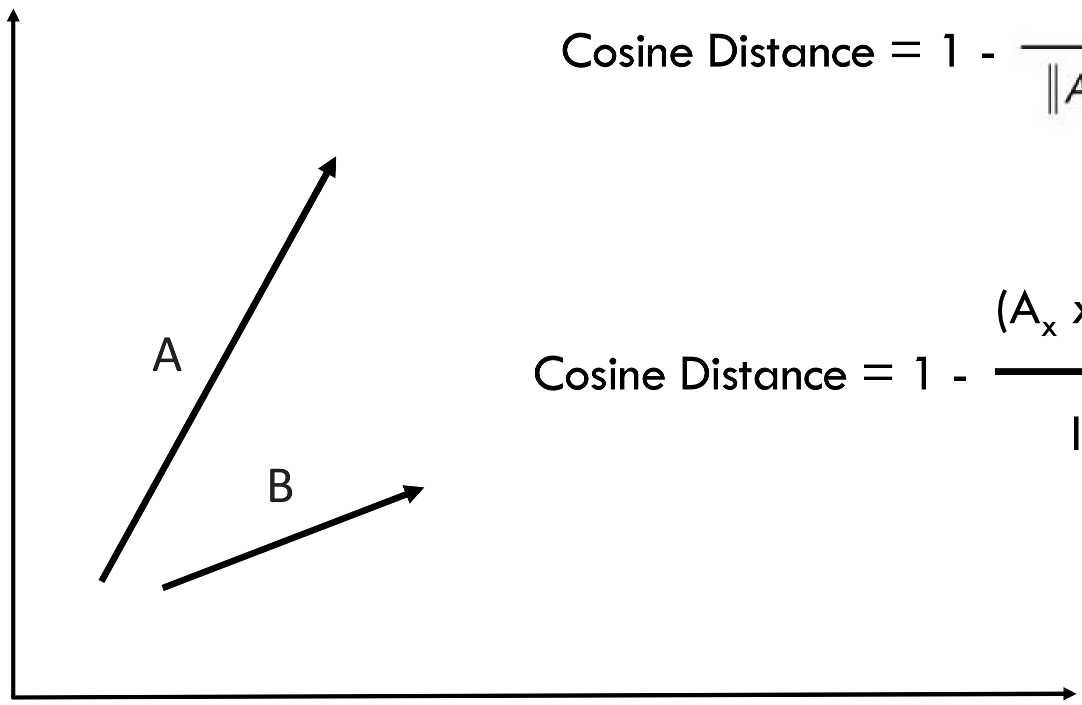
NOT currently optimized by Intel Extensions for scikit-learn



$$d = \sqrt{\Delta Nodes^2 + \Delta Age^2}$$

Cosine Distance

Optimized by Intel Extensions for scikit-learn



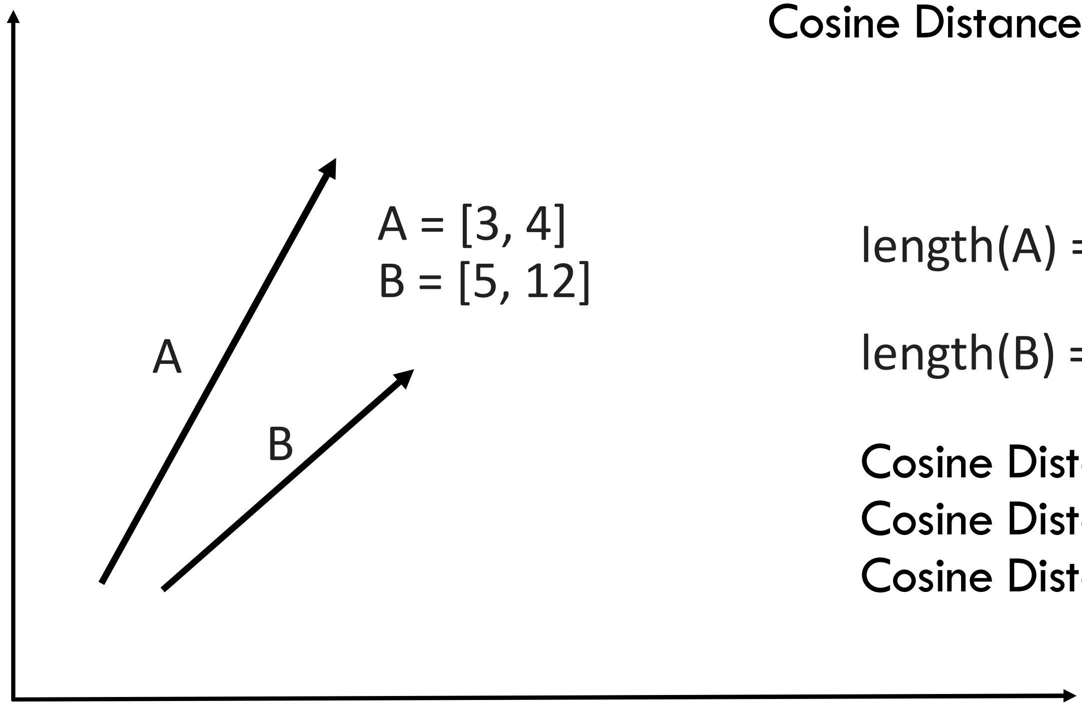
$$\text{Cosine Distance} = 1 - \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

$$\text{Cosine Distance} = 1 - \frac{(A_x \times B_x + A_y \times B_y + \ldots)}{\text{length}(A) \times \text{length}(B)}$$

Cosine Distance

Optimized by Intel Extensions for scikit-learn

$$\text{Cosine Distance} = 1 - \frac{(A_x \times B_x + A_y \times B_y + \ldots)}{\text{length}(A) \times \text{length}(B)}$$



length(A) = sqrt(3 x 3 + 4 x 4) = 5

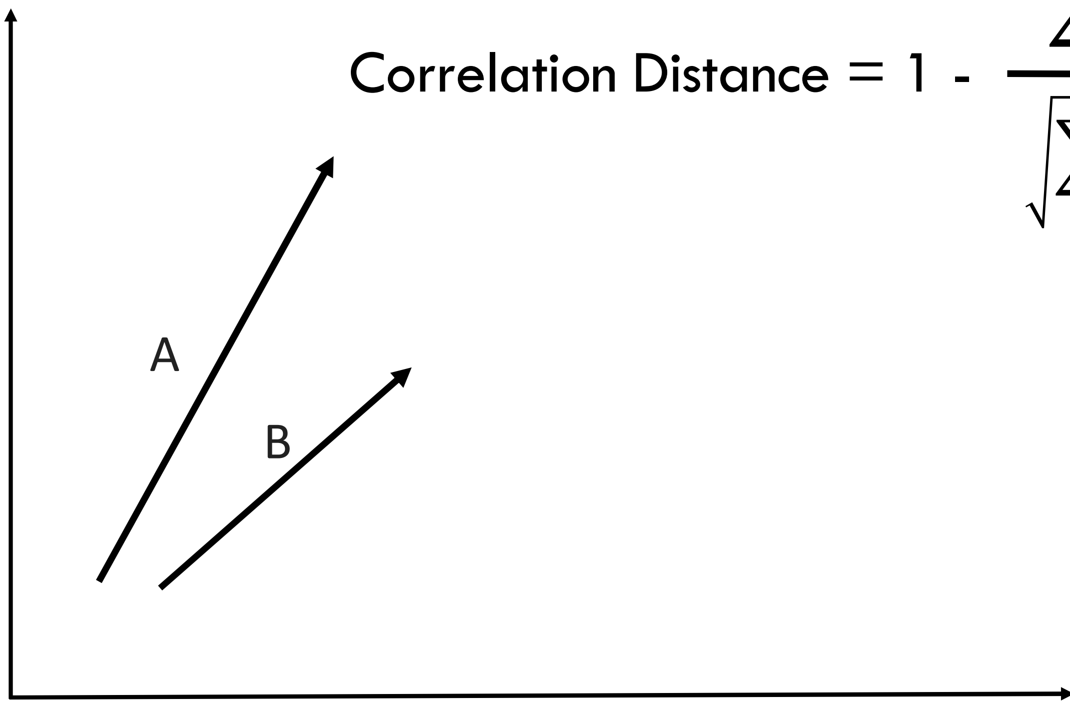
length(B) = sqrt(5 x 5 + 12 x 12) = 13

Cosine Distance = 1 - cos θ
Cosine Distance = 1 - (3 x 5 + 4 x 12) / (5 x 13)
Cosine Distance = 1 - 0.969 = .031

Correlation Distance

Optimized by Intel Extensions for scikit-learn

$$\text{Correlation Distance} = 1 - \frac{\sum (xi - xmean)(yi - ymean)}{\sqrt{\sum (xi - xmean)^2 \sum (y_i \_ y_{mean})^2}}$$

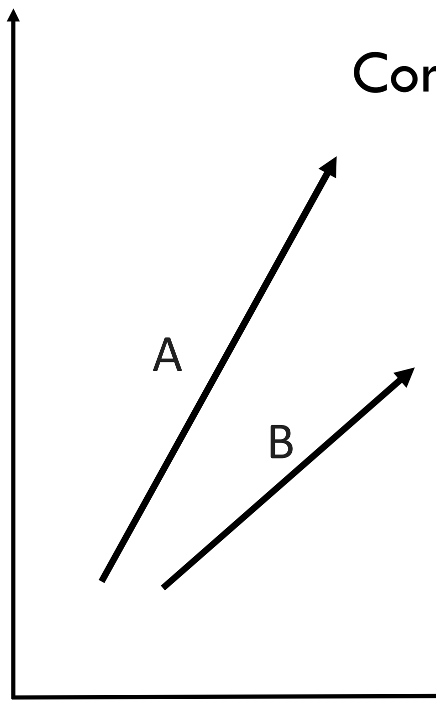


$x_1 = Ax$
$x_2 = Bx$
$y_1 = Ay$
$y_2 = By$

$x_{mean} = mean(Ax, Bx)$
$y_{mean} = mean(A_y, By)$

Correlation Distance

Optimized by Intel Extensions for scikit-learn



$$\text{Correlation Distance} = 1 - \frac{\sum(xi - xmean)(yi - ymean)}{\sqrt{\sum(xi - xmean)^2 \sum(y_i - y_{mean})^2}}$$

A = [5, 12]

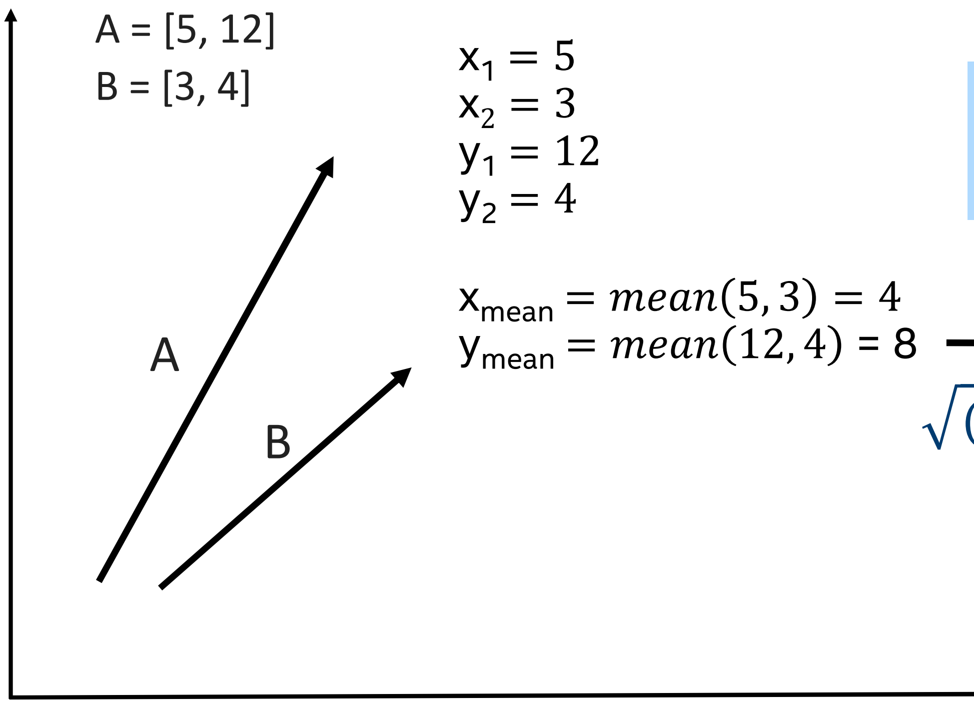
B = [3, 4]

$x_1 = 5$
$x_2 = 3$
$y_1 = 12$
$y_2 = 4$

$x_{mean} = mean(5, 3) = 4$
$y_{mean} = mean(12, 4) = 8$

Correlation Distance

Optimized by Intel Extensions for scikit-learn



A = [5, 12]

B = [3, 4]

$x_1 = 5$
$x_2 = 3$
$y_1 = 12$
$y_2 = 4$

$x_{mean} = mean(5, 3) = 4$
$y_{mean} = mean(12, 4) = 8$

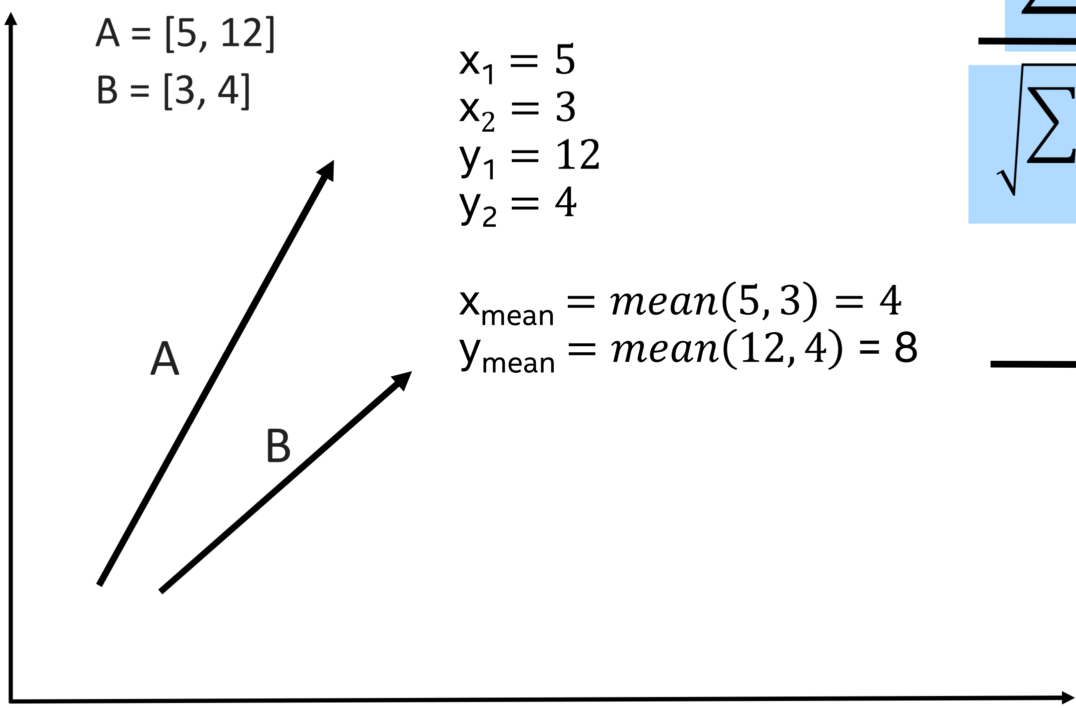
$$\frac{\sum (xi - xmean)(yi - ymean)}{\sqrt{\sum (xi - xmean)^2 \sum (y_i - y_{mean})^2}}$$

$$\frac{(5-4)(12-8) + (3-4)(4-8)}{\sqrt{((5-4)^2 + (3-4)^2)((12-8)^2 + (4-8)^2)}}$$

$$\frac{(4) + (4)}{\sqrt{(1+1)(16+16)}}$$

Correlation Distance

Optimized by Intel Extensions for scikit-learn



$x_1 = 5$
$x_2 = 3$
$y_1 = 12$
$y_2 = 4$

$x_{mean} = mean(5, 3) = 4$
$y_{mean} = mean(12, 4) = 8$

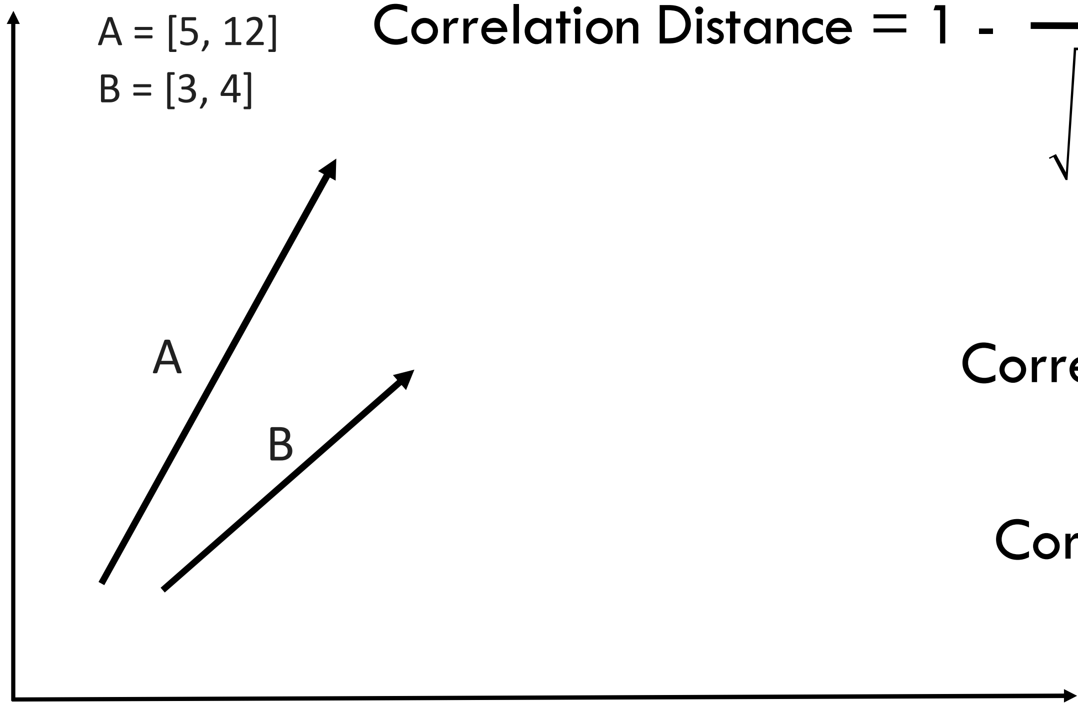
$$\frac{\sum (xi - xmean)(yi - ymean)}{\sqrt{\sum (xi - xmean)^2 \sum (y_i \_ y_{mean})^2}} =$$

$$\frac{(4) + (4)}{\sqrt{(64)}} =$$

$$\frac{8}{8} = 1$$

Correlation Distance

Optimized by Intel Extensions for scikit-learn



$$\text{Correlation Distance} = 1 - \frac{\sum (xi - xmean)(yi - ymean)}{\sqrt{\sum (xi - xmean)^2 \sum (y_i \_ y_{mean})^2}} =$$

$$\text{Correlation Distance} = 1 - 1$$

$$\text{Correlation Distance} = 0$$

Pairwise distance using @dppy.kernel

Import dpctl

Pairwise distance in parallel using the @ddpy.kernel decorator

Kernel invocation of the Pairwise distance

Offload this to opencl:gpu

```
import dpctl
import numpy as np
import numba_dppy

@numba_dppy.kernel
def pairwise_python(X1, X2, D):
    i = numba_dppy.get_global_id(0)

    N = X2.shape[0]
    O = X1.shape[1]
    for j in range(N):
        d = 0.0
        for k in range(O):
            tmp = X1[i, k] - X2[j, k]
            d += tmp * tmp
        D[i, j] = np.sqrt(d)

def pw_distance(X1, X2, D):
    with dpctl.device_context("opencl:gpu"):
        # pairwise_python[X1.shape[0],numba_dppy.DEFAULT_L
OCAL_SIZE](X1, X2, D)
        pairwise_python[X1.shape[0], 128](X1, X2, D)
```

Distance : The Syntax

Import sklearnex

Apply “monkey patch”

Import desired sklearn algorithms AFTER the patch

```
from sklearnex import patch_sklearn
patch_sklearn() # apply BEFORE import of targets
#patch_sklearn('distances’) # to be surgical
```

```
from sklearn.metrics.pairwise import pairwise_distances
```

```
#Create an instance of the class
dist = pairwise_distances (X, y, metric="correlation")
# or
dist = pairwise_distances (X, y, metric="cosine")
```


intel.

Black Scholes using @njit

Import dpctl

Black Scholes in parallel using the @njit decorator

Calculate Calls and puts with the change in the current price and the strike price

Offload this to level_zero:gpu

```
import dpctl
import numba as nb
from math import log, sqrt, exp, erf

# blackscholes implemented as a parallel loop using numba.prange
@nb.njit(parallel=True, fastmath=True)
def black_scholes_kernel(nopt, price, strike, t, rate, vol, call, put):
    mr = -rate
    sig_sig_two = vol * vol * 2
    for i in nb.prange(nopt):
        P = price[i]
        S = strike[i]
        T = t[i]
        a = log(P / S)
        b = T * mr
        z = T * sig_sig_two
        c = 0.25 * z
        y = 1.0 / sqrt(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
        r = P * d1 - Se * d2
        call[i] = r
        put[i] = r - P + Se

def black_scholes(nopt, price, strike, t, rate, vol, call, put):
    # offload blackscholes computation to GPU (toggle level0 or opencl driv
er)
    with dpctl.device_context("level_zero:gpu"):
        black_scholes_kernel(nopt, price, strike, t, rate, vol, call, put)
```


intel

Hands-on Coding on Intel® DevCloud / JLSE

Summary

- Illustrate How oneAPI Can help solve the challenges of programming in a heterogeneous world
- How to use Data Parallel Python and Data Parallel Control
- Performed 3 code walkthroughs via hands on activities demonstrating:
  - A Pairwise Algorithm using Jit and Kernel decorators on CPU and GPU
  - A Blackscholes Algorithm using Jit and Kernel decorators on CPU and GPU

Thanks for attending the session

NOTICES &

- This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.
- The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request. No product or component can be absolutely secure. Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.
- 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.
- 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.

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