Intel® Math Kernel Library
( Intel® MKL )
Intel® Math Kernel Library (Intel® MKL) Introduction

Highly optimized threaded math routines
- Performance, Performance, Performance!

Industry’s leading math library
- Widely used in science, engineering, data processing

Tuned for Intel® processors – current and next generation

Be multiprocessor aware
- Cross-Platform Support
- Be vectorised, threaded, and distributed multiprocessor aware
Intel MKL unleashes the performance benefits of Intel architectures

DGEMMM Performance Boost by using Intel® MKL vs. ATLAS*

Configuration Info - Versions: Intel® Math Kernel Library (Intel® MKL) 11.3, ATLAS* 3.10.2; Hardware: Intel® Xeon® Processor E5-2699v3, 2 Eighteen-core CPUs (45MB LLC, 2.3GHz), 64GB of RAM; Intel® Core™ Processor i7-4770K, Quad-core CPU (8MB LLC, 3.5GHz), 8GB of RAM; Operating System: RHEL 6.4 GA x86_64.

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. *Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation

Optimization Notice: Intel’s compilers may or may not optimize to the same degree as 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.
Intel MKL 2017 Performance

DGEMM Performance
On Intel® Xeon® Processor E5-2699 v4

DGEMM Performance
On Intel® Xeon Phi™ Processor 7250

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Intel MKL 2017 Performance
# Optimized Mathematical Building Blocks

<table>
<thead>
<tr>
<th>Linear Algebra</th>
<th>Fast Fourier Transforms</th>
<th>Vector Math</th>
<th>Vector RNGs</th>
<th>Summary Statistics</th>
<th>Deep Neural Networks (DNN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• BLAS</td>
<td>• Multidimensional</td>
<td>• Trigonometric</td>
<td>• Congruential</td>
<td>• Kurtosis</td>
<td>• Convolution</td>
</tr>
<tr>
<td>• LAPACK</td>
<td>• FFTW interfaces</td>
<td>• Hyperbolic</td>
<td>• Variation coefficient</td>
<td>• Pooling</td>
<td></td>
</tr>
<tr>
<td>• ScaLAPACK</td>
<td>• Cluster FFT</td>
<td>• Exponential</td>
<td>• Order statistics</td>
<td>• Normalization</td>
<td></td>
</tr>
<tr>
<td>• Sparse BLAS</td>
<td>• Log</td>
<td>• Log</td>
<td>• Min/max</td>
<td>• ReLU</td>
<td></td>
</tr>
<tr>
<td>• Sparse Solvers</td>
<td>• Power</td>
<td>• Power</td>
<td>• Variance-covariance</td>
<td>• Softmax</td>
<td></td>
</tr>
<tr>
<td>• Iterative</td>
<td>• Root</td>
<td>• Root</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• PARDISO* SMP &amp; Cluster</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Other names and brands may be claimed as the property of others.*
**BLAS – Basic Linear Algebra Subprograms**

Defacto-standard APIs since the 1980s (Fortran 77)

- Level 1 – vector-vector operations
- Level 2 – matrix-vector operations
- Level 3 – matrix-matrix operations
- Precisions: single, double, single complex, double complex

Original BLAS available at http://netlib.org/blas/

<table>
<thead>
<tr>
<th>Operation</th>
<th>MKL Routine “D is for double”</th>
<th>Example</th>
<th>Computational complexity (work)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector Vector</td>
<td>D AXPY</td>
<td>y = y + α x</td>
<td>O(N)</td>
</tr>
<tr>
<td>Matrix Vector</td>
<td>D GEMV</td>
<td>y = αAx + βy</td>
<td>O(N^2)</td>
</tr>
<tr>
<td>Matrix Matrix</td>
<td>D GEMM</td>
<td>C = αA * B + βC</td>
<td>O(N^3)</td>
</tr>
</tbody>
</table>
Defacto-standard APIs since early 1990s
1000s of linear algebra functions
4 floating point precisions supported

Breadth of coverage:
- Matrix factorizations: the 3 Amigos – LU, Cholesky, QR
- Solving systems of linear equations
- Condition number estimates
- Singular value decomposition
- Symmetric and non-symmetric eigenvalue problems
- And much, much more

Original LAPACK is available at:
http://netlib.org/lapack/
Fast Fourier Transform (FFT)

Support multidimensional transforms

Multiple transforms on single call

Input/output strides supported

   Allow FFT of a part of image, padding for better performance, transform combined with transposition, facilitates development of mixed-language applications.

Integrated FFTW interfaces

   Source code of FFTW3 and FFTW2 wrappers in C/C++ and Fortran are provided.
   FFTW3 wrappers are also built into the library.
Vector Math Functions

Example: \( y(i) = e^{x(i)} \) for \( i = 1 \) to \( n \)

- Arithmetic
  - add/sub/sqrt/ ...

- Exponential and log
  - exp/pow/log/log10

- Trigonometric and hyperbolic
  - sin/cos/sincos/tan(h)
  - asin/acos/atan(h)

- Rounding
  - ceil, floor, round ...

- And many more ...

- Real and complex
- Single/double precision
- 3 accuracy modes
  - High accuracy
    - (Almost correctly rounded)
  - Low accuracy
    - (2 lowest bits in error)
  - Enhanced performance
    - (1/2 the bits correct)

*Vector-based elementary functions allow developers to balance accuracy with performance*
## Vector Statistics

<table>
<thead>
<tr>
<th><strong>Random Number Generators (RNGs)</strong></th>
<th>Psuedo-random, quasi-random, and non-deterministic generators</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Continuous and discrete distributions of various common distribution types</td>
</tr>
<tr>
<td><strong>Summary Statistics (SS)</strong></td>
<td>Parallelized algorithms for computation of statistical estimates for raw multi-dimensional datasets.</td>
</tr>
<tr>
<td><strong>Convolution/ correlation</strong></td>
<td>A set of routines intended to perform linear convolution and correlation transformations for single and double precision real and complex data.</td>
</tr>
</tbody>
</table>
## Intel® MKL Sparse Solvers

<table>
<thead>
<tr>
<th>Solver Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| **PARDISO – Parallel Direct Sparse Solver** | Support a wide range of matrix types.  
Based on BLAS level 3 update and pipelining parallelism.  
Supports out-of-core execution for huge problem sizes.  
New: Cluster support. |
| **DSS – Direct Sparse Solver Interface for PARDISO** | An alternative, simplified interface to PARDISO. |
| **ISS – Iterative Sparse Solver** | Symmetric positive definite: CG solver.  
Non-symmetric indefinite: Flexible generalized minimal residual solver.  
Based on Reverse Communication Interface (RCI). |
More Intel® MKL Components

Data Fitting
- 1D linear, quadratic, cubic, step-wise const, and user-defined splines
- Spline based interpolation/extrapolation

PDEs (Partial Differential Equations)
- Solving Helmholtz, Poisson, and Laplace problems.

Optimization Solvers
- Solvers for nonlinear least square problems with/without constraints

Support Functions
- Memory management
- Threading control
  …
What are Intel MKL DNN Primitives?

A set of performance primitives to speed up image recognition topologies on existing or custom NN frameworks

- **Topologies**: AlexNet, VGG, GoogleNet, ResNet
- **Frameworks**: Caffe*, TensorFlow*, CNTK*, Torch*, MXNet*, ......

<table>
<thead>
<tr>
<th>Operations (forward/backward)</th>
<th>Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activation</td>
<td>ReLU</td>
</tr>
<tr>
<td>Normalization</td>
<td>batch, local response</td>
</tr>
<tr>
<td>Pooling</td>
<td>max, min, average</td>
</tr>
<tr>
<td>Convolutional</td>
<td>fully connected, direct batched convolution</td>
</tr>
<tr>
<td>Inner product</td>
<td>forward/backward propagation of inner product computation</td>
</tr>
<tr>
<td>Data manipulation</td>
<td>layout conversion, split, concat, sum, scale</td>
</tr>
</tbody>
</table>
## NN Primitives API Examples

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dnnConvolutionCreateForwardBias_F32(&amp;primitive, attributes, dnnAlgorithmConvolutionDirect, dimension, inputSize, outputSize, filterSize, stride, inputOffset, dnnBorderZeros);</td>
<td>Create a convolution primitive for forward pass. This only creates a descriptor of the operation. Input and output data is not specified yet.</td>
</tr>
<tr>
<td>dnnExecute(primitive, inputs, outputs)</td>
<td>Execute the primitive. Input and output data is specified at the execution time.</td>
</tr>
<tr>
<td>dnnLayoutCreate(&amp;layout, params)</td>
<td>Create a user defined data layout by specifying number of dimensions, and size and stride for each dimension.</td>
</tr>
<tr>
<td>dnnLayoutCreateFromPrimitive(&amp;layout, primitive, type)</td>
<td>Query the layout required by a primitive.</td>
</tr>
<tr>
<td>dnnAllocateBuffer(&amp;ptr, layout)</td>
<td>Allocate memory buffer for converted layout.</td>
</tr>
<tr>
<td>if (!dnnLayoutCompare(l1, l2)) dnnConversionCreate(&amp;conversion_prim, l1, l2)</td>
<td>Compare different layout types. Create a conversion operation if necessary.</td>
</tr>
</tbody>
</table>
DNN Primitives in Intel® MKL Highlights

A plain C API to be used in the existing DNN frameworks

Brings IA-optimized performance to popular image recognition topologies:

– AlexNet, Visual Geometry Group (VGG), GoogleNet, and ResNet

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 information go to http://www.intel.com/performance. *Other names and brands may be property of others.
What’s New: Intel® MKL 2017

- Optimized math functions to enable neural networks (CNN and DNN) for deep learning
- Improved ScaLAPACK performance for symmetric eigensolvers on HPC clusters
- New data fitting functions based on B-splines and monotonic splines
- Improved optimizations for newer Intel processors, especially Knight’s Landing Xeon Phi
- Extended TBB threading layer support for all BLAS level-1 functions
Intel® MKL Resources

Intel® MKL website

Intel MKL forum

Intel® MKL benchmarks

Intel® MKL link line advisor
Intel® Data Analytics Acceleration Library (Intel® DAAL)
Intel® Data Analytics Acceleration Library (Intel® DAAL)

An industry leading Intel® Architecture based data analytics acceleration library of fundamental algorithms covering all machine learning stages.

Pre-processing
- (De-)Compression
- Outlier Detection
- Normalization

Transformation
- PCA
- Statistical moments
- Variance matrix
- Pp-QR, SVD, Cholesky
- Apriori
- Sorting

Analysis
- Ridge linear regression
- Naïve Bayes
- SVM
- Classifier boosting
- Kmeans
- EM GMM

Modeling
- Collaborative filtering
- Neural Networks

Validation

Decision Making

Scientific/Engineering
Web/Social
Business
Validation
Intel DAAL Main Features

- Building end-to-end data applications
- Optimized for Intel architectures, from Intel® Atom™, Intel® Core™, Intel® Xeon®, to Intel® Xeon Phi™
- A rich set of widely applicable algorithms for data mining and machine learning
- Batch, online, and distributed processing
- Data connectors to a variety of data sources and formats: KDB*, MySQL*, HDFS, CSV, and user-defined sources/formats
- C++, Java, and Python APIs
PyDAAL (Python API for Intel® DAAL)

Turbocharged machine learning tool for Python developers

Interoperability and composability with the SciPy ecosystem:

- Work directly with NumPy
- Faster than scikit-learn
Processing modes

Batch Processing

$$R = F(D_1, \ldots, D_k)$$

Online Processing

$$S_{i+1} = T(S_i, D_i)$$
$$R_{i+1} = F(S_{i+1})$$

Distributed Processing

$$R = F(R_1, \ldots, R_k)$$
<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Batch</th>
<th>Distributed</th>
<th>Online</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Descriptive statistics</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low order moments</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Quantiles/sorting</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Statistical relationships</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Correlation / Variance-Covariance</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(Cosine, Correlation) distance matrices</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Matrix decomposition</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVD</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Cholesky</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QR</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Regression</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linear/ridge regression</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Classification</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multinomial Naïve Bayes</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SVM (two-class and multi-class)</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boosting (Ada, Brown, Logit)</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Unsupervised learning</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Association rules mining (Apriori)</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anomaly detection (uni-/multi-variate)</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KMeans</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EM for GMM</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Recommender systems</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALS</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td><strong>Deep learning</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fully connected, convolution, normalization, activation layers, model, NN, optimization solvers,</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Intel® DAAL Neural Networks Support

**General purpose API for building typical NN topologies**

**Tensors**
- Multi-dimensional data structures to represent complex data

**Layers**
- Forward and backward computation

**Topology**
- Predefined structure of a neural network

**Optimization solver**
- Computing weights and biases to minimize the objective function

**Model**
- A network fleshed out with weights and biases of each layer fully defined

**Driver**
- Engine that drives training and scoring

## Compare NN Features in Intel MKL and Intel DAAL

<table>
<thead>
<tr>
<th></th>
<th>Intel MKL</th>
<th>Intel DAAL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DNN primitives</strong></td>
<td>Performance critical</td>
<td>Easy integration and high performance</td>
</tr>
<tr>
<td><strong>DNN layers</strong></td>
<td>No</td>
<td>All building blocks for NN topology</td>
</tr>
<tr>
<td><strong>Optimization solvers</strong></td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Performance</strong></td>
<td>Top in the class, full control from user side</td>
<td>On-par with Intel MKL</td>
</tr>
<tr>
<td><strong>Distributed memory</strong></td>
<td>Not easy, yet</td>
<td>Can be integrated with Spark, MPI cluster, …</td>
</tr>
<tr>
<td><strong>Language support</strong></td>
<td>C</td>
<td>C++, Java, Python</td>
</tr>
<tr>
<td><strong>Target audience</strong></td>
<td>Users who want to speed up existing frameworks</td>
<td>Users who want to build from scratch or prototype</td>
</tr>
</tbody>
</table>
Intel® DAAL vs. Spark® Mllib

K-means Performance Comparison on Eight-node Cluster

![Graph showing speedup comparison between Intel DAAL and Spark Mllib.](Image)

Configuration Info - Versions: Intel® Data Analytics Acceleration Library 2017, Spark 1.2; Hardware: Intel® Xeon® Processor E5-2699 v3, 2 Eighteen-core CPUs (40MB LLC, 2.3GHz), 128GB of RAM per node; Operating System: CentOS 6.6 x86_64.

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. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation

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.
Demo: Handwritten Digit Recognition
Handwritten Digit Recognition

Training multi-class SVM for 10 digits recognition.

3,823 pre-processed training data.

- available at

99.6% accuracy with 1,797 test data from the same data provider.

Confusion matrix:

<table>
<thead>
<tr>
<th></th>
<th>0.000</th>
<th>0.000</th>
<th>0.000</th>
<th>0.000</th>
<th>0.000</th>
<th>0.000</th>
<th>0.000</th>
<th>0.000</th>
<th>0.000</th>
<th>0.000</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>177.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.000</td>
<td>181.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>0.000</td>
<td>2.000</td>
<td>173.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>176.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>3.000</td>
<td>3.000</td>
</tr>
<tr>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>179.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>180.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>180.000</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>170.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.000</td>
<td>3.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>166.000</td>
<td>5.000</td>
</tr>
<tr>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>2.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>2.000</td>
<td>175.000</td>
</tr>
</tbody>
</table>

Average accuracy: 0.996
Error rate: 0.004
Micro precision: 0.978
Micro recall: 0.978
Micro F-score: 0.978
Macro precision: 0.978
Macro recall: 0.978
Macro F-score: 0.978
Training Handwritten Digits

```cpp
void trainModel()
{
    /* Initialize FileDataSource<CSVFeatureManager> to retrieve input data from .csv file */
    FileDataSource<CSVFeatureManager> trainDataSource(trainDatasetFileName,
    DataSource::doAllocateNumericTable, DataSource::doDictionaryFromContext);

    /* Load data from the data files */
    trainDataSource.loadDataBlock(nTrainObservations);

    /* Create algorithm object for multi-class SVM training */
    multi_class_classifier::training::Batch< > algorithm;

    algorithm.parameter.nClasses = nClasses;
    algorithm.parameter.training = training;

    /* Pass training dataset and dependent values to the algorithm */
    algorithm.input.set(classifier::training::data, trainDataSource.getNumericTable());

    /* Build multi-class SVM model */
    algorithm.compute();

    /* Retrieve algorithm results */
    trainingResult = algorithm.getResult();

    /* Serialize the learned model into a disk file */
    ModelFileWriter writer("./model");
    writer.serializeToFile(trainingResult->get(classifier::training::model));
}
```
```cpp
void testDigit()
{
    /* Initialize FileDataSource<CSVFeatureManager> to retrieve the test data from .csv file */
    FileDataSource<CSVFeatureManager> testDataSource(testDatasetFileName,
        DataSource::doAllocateNumericTable, DataSource::doDictionaryFromContext);
    testDataSource.loadDataBlock(1);

    /* Create algorithm object for prediction of multi-class SVM values */
    multi_class_classifier::prediction::Batch<> algorithm;
    algorithm.setParameter(prediction = prediction);

    /* Deserialize a model from a disk file */
    ModelFileReader reader("./model");
    services::SharedPtr<
        multi_class_classifier::Model> pModel(new multi_class_classifier::Model());
    reader.deserializeFromFile(pModel);

    /* Pass testing dataset and trained model to the algorithm */
    algorithm.setInput(classifier::prediction::data,
        testDataSource.getNumericTable());
    algorithm.setInput(classifier::prediction::model, pModel);

    /* Predict multi-class SVM values */
    algorithm.compute();

    /* Retrieve algorithm results */
    predictionResult = algorithm.getResult();

    /* Retrieve predicted labels */
    predictedLabels = predictionResult->get(classifier::prediction::prediction);
}
```

Deserialized learned model
SVM Performance Boosts Using Intel® DAAL vs. scikit-learn on Intel® CPU

Configuration Info - Versions: Intel® Data Analytics Acceleration Library 2016 U2, scikit-learn 0.16.1; Hardware: Intel Xeon E5-2680 v3 @ 2.50GHz, 24 cores, 30 MB L3 cache per CPU, 256 GB RAM; Operating System: Red Hat Enterprise Linux Server release 6.6, 64-bit.

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. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation

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.
## Intel® DAAL + Intel® MKL = Complementary Big Data Libraries Solution

<table>
<thead>
<tr>
<th>Intel MKL</th>
<th>Intel DAAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>C and Fortran API</td>
<td>Python, Java &amp; C++ API</td>
</tr>
<tr>
<td>Primitive level</td>
<td>High-level</td>
</tr>
<tr>
<td>Processing of homogeneous data in single or</td>
<td>Processing heterogeneous data (mix of integers and floating point),</td>
</tr>
<tr>
<td>double precision</td>
<td>internal conversions are hidden in the library</td>
</tr>
<tr>
<td>Type of intermediate computations is defined</td>
<td>Type of intermediate computations can be configured independently of the</td>
</tr>
<tr>
<td>by type of input data (in some library domains</td>
<td>type of input data</td>
</tr>
<tr>
<td>higher precision can be used)</td>
<td></td>
</tr>
<tr>
<td>Most of MKL supports batch computation mode</td>
<td>3 computation modes: Batch, streaming and distributed</td>
</tr>
<tr>
<td>only</td>
<td></td>
</tr>
<tr>
<td>Cluster functionality uses MPI internally</td>
<td>Developer chooses communication method for distributed computation (e.g.</td>
</tr>
<tr>
<td></td>
<td>Spark, MPI, etc.) Code samples provided.</td>
</tr>
</tbody>
</table>
Summary

Intel® DAAL is the only data analytics library optimized for current and future Intel® Architectures.

Product page:


Forum:

Legal Disclaimer & Optimization Notice

**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