WHITE PAPER

Healthcare
Artificial Intelligence
Intel AI® Builders

Intel® Distribution for Python* versus Non-Optimized Python: Breast Cancer Classification

Table of Contents

Abstract

This case study compares the performance of Intel® Distribution for Python* to that of non-optimized Python using a breast cancer classification. This comparison was done using machine learning algorithms from the scikit-learn* package in Python.

Introduction

Cancer refers to cells that grow out of control and invade other tissues. This process can also result in a tumor, where there is more cell growth than cell death. There are various types of cancer including bladder cancer, kidney cancer, lung cancer, and breast cancer. Currently, breast cancer is one of the most prevalent types of cancer, especially in women. It occurs when the cells in the breast divide and grow uncontrollably. Early detection of breast cancer can save lives. Causes of cancer include inherited genes, hormones, and an individual's lifestyle.

This article provides a comparative study between the performance of non-optimized Python* and the Intel® Distribution for Python using breast cancer classification as an example. The classifiers used for breast cancer classification were taken from the scikit-learn* package in Python. The time and accuracy of each classifier for each distribution was calculated and compared.

Dataset Description

The dataset for this study can be accessed from the Breast Cancer Wisconsin (Diagnostic) Data Set. The features of this dataset were computed from a digitized image of a fine needle aspirate of a breast mass in a CSV format and describe the characteristics of the cell nuclei present in the image. These values obtained were the features for classification. Using these features, a cancer cell can be classified into two classes: benign and malignant. Benign refers to a tumor that is not cancerous, whereas a malignant tumor has cancer in it. Observing the class distribution, there were 357 benign and 212 malignant data rows. The classification is based on the diagnosis field that has values M or B, where M denotes malignant and B denotes benign. Hence, this is a binary classification.

Hardware Configuration

The experiment used Intel® architecture with the following hardware configuration:

Feature	Specifications
Architecture	x86_64
CPU op-mode(s)	32-bit, 64-bit
Byte order	Little Endian
CPU(s)	256
On-line CPU(s) list	0-255
Thread(s) per core	4
Core(s) per socket	64
Socket(s)	1
NUMA node(s)	2
Vendor ID	GenuineIntel
CPU family	6
Mode	87
Model name	Intel® Xeon Phi™ processor 7210 @ 1.30 GHz
Stepping	1
CPU MHz	1375.917
BogoMIPS	2593.85
L1d cache	32K
L1i cache	32K
L2 cache	1024K
NUMA node0 CPU(s)	0-255

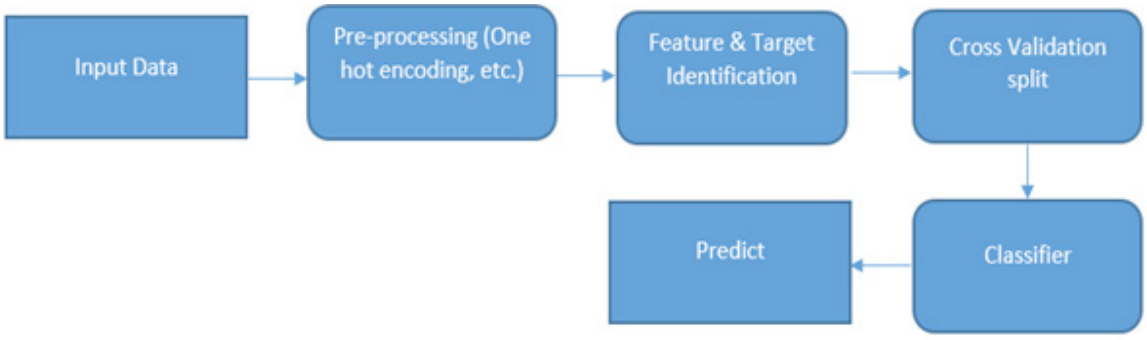
Software Configuration

The following are the software dependencies used to perform these classification:

Software	Version
Python*	2.7.13
scikit-learn*	0.18.2
Anaconda*	4.3.25

Classifier Implementation Pipeline

The goal was to identify the class (M or B) to which the tumor belonged. The following block diagram shows the classification steps, explained in the following section, for both the Intel Distribution for Python and non-optimized Python.



Implementation

The scikit-learn Python library provides a wide variety of machine learning algorithms for classification. Ten classifiers from the package were used for the study: Decision Tree Classifier, Gaussian NB, SGD Classifier, SVC, KNeighbors Classifier, OneVsRest Classifier, Quadratic Discriminant Analysis (QDA), Random Forest Classifier, MLP Classifier, and AdaBoost Classifier.

Create a Python file called classifier_ml.py. The following steps are implemented in this file:

1. The input data mentioned in **Dataset Description** section is given for preprocessing.
2. As part of the preprocessing, the given dataset is checked for categorical values (if any) and are converted to numerical data. This is performed using a technique called **One Hot Encoding**. This is important because a few classifiers in scikit-learn work only with numerical values. Here, diagnosis fields containing values "M" and "B" are converted to 1 and 0, respectively. Columns such as "id" are irrelevant for classification and hence can be dropped.
3. After preprocessing, all the columns except diagnosis field is considered as the features. Diagnosis column is taken as the target.
4. 70 percent of the data is used for training and 30 percent is used for testing. The split is done using the **StratifiedShuffleSplit** function from cross_validation module of sklearn[1].
5. Keeping the default environment intact, the accuracy of each classifier is recorded using the scikit-learn package of Python[2].
6. The file 'classifier_ml.py', is now executed. The time taken (t_nop) is measured as a 10-times average for better accuracy as follows:

```
time(cmd="python classifier_ml.py"; for i in $(seq 10); do $cmd; done)
```

Steps 1 through 6 provide the time and accuracy values for non-optimized Python. Repeat these steps for the Intel Distribution for Python. The time (t_idp) and accuracy are calculated.

To enable the Intel Distribution for Python, follow the steps given in Installing Intel® Distribution for Python* and Intel® Performance Libraries with Anaconda*.

The results are shown in Table 1.

The accuracy values for each classifier are the same for both non-optimized Python and the Intel Distribution for Python. Therefore, the accuracy values listed in Table 1 are common for both distributions.

Performance Gain percentage with respect to time is calculated by the given formula:

```
Performance Gain % = (t_nop - t_idp) / t_nop * 100
```

From the formula, it is clear that a positive value of **Performance Gain** percentage indicated better performance of the Intel Distribution for Python. The higher the value, the better the performance for the Intel Distribution for Python compared to non-optimized Python.

Results

Table 1 shows the percentage performance gain for Intel Distribution of Python* over non-optimized Python.

Table 1: Percentage performance gain for Intel Distribution of Python* over non-optimized Python

Classifier	Accuracy (Percent)	Performance Gain Percentage
DecisionTreeClassifier	90.64	34.69
GaussianNB	94.74	35.01
SGDClassifier	88.89	33.04
SVC	94.74	32.29
KNeighborsClassifier	92.98	34.35
OneVsRestClassifier	92.40	33.00
QuadraticDiscriminantAnalysis	94.15	33.65
RandomForestClassifier	93.57	30.36
MLPClassifer	65.50	32.09
AdaBoostClassifier	94.74	27.09

Conclusion

The performance gain clearly shows that the Intel Distribution for Python had better performance in terms of the time taken for execution as compared to non-optimized Python. The accuracy remained the same as expected and did not change whether non-optimized Python or the Intel Distribution for Python was used.

References

1. Cross Validation - Stratified Shuffle Split: http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html
2. An introduction to machine learning with scikit-learn: http://scikit-learn.org/stable/tutorial/basic/tutorial.html