Machine learning techniques using python for data analysis in performance evaluation

Article *in* International Journal of Intelligent Systems Technologies and Applications · January 2018

DOI: 10.1504/IJISTA.2018.10012853

Lakshmi Jupudi
Jain University

Machine learning techniques using python for data analysis in performance evaluation

J.V.N. Lakshmi

Acharya Institute of Management and Sciences,
MCA Department,
Bangalore, 560 058, India
Email: jlakshmi.research@gmail.com

Abstract: Machine learning algorithms are used to parallelise the workloads. To achieve paramount performance required parameters are tuned in the algorithms. The jobs are implemented using machine learning techniques using various parameters. The performance is examined by executing various features and verifying time constraints depending on the assignments for a cluster. The attempt is made to obtain minimum execution time using python language for implementing machine learning algorithms. Supervised and unsupervised techniques of machine learning algorithms are used differentiating the performance evaluation and time efficiency. Linear regression, logistic regression and K-means clustering techniques are used to evaluate the data analytic jobs. This implementation reveals the best performance of supervised algorithms over unsupervised for data analysis. This paper is an attempt made to analyse the machine learning techniques and evaluates the timer feature on various methods irrespective of supervised or unsupervised.

Keywords: machine learning; linear regression; logistic regression; K-means; EM.

Reference to this paper should be made as follows: Lakshmi, J.V.N. (2018) 'Machine learning techniques using python for data analysis in performance evaluation', *Int. J. Intelligent Systems Technologies and Applications*, Vol. 17, Nos. 1/2, pp.3–18.

Biographical notes: J.V.N. Lakshmi is an Assistant Professor in Acharya Institution of Management and Sciences in MCA Department, Bangalore - India. She received her Master's degree in Statistics, Master of Computer Applications (MCA) and Bachelor of Science degree in Statistics and Computer Science from Osmania University, Hyderabad, India. Her research interest is on machine learning, big data, statistics and data analytics. Her papers are published on machine learning using big data for analytics in reputed journals like IEEE, ACM, and Scopus Indexed Journals. She is pursuing PhD from SCSVMV university, Kanchipuram, India.

This paper is a revised and expanded version of a paper entitled 'Machine learning techniques using python for data analysis in performance evaluation' presented at *International Conference on Innovative Systems*, Bangalore, 14 December, 2016.

1 Introduction

There is a massive growth in data analysis a model is widely used to parallelise by incorporating machine learning algorithms. Machine learning techniques are implemented on large, complex data clusters for analysing data. For achieving high accuracy while processing data parallel on huge complex clusters, one of the concerns is machine learning. A parallel programming model is required for efficient data processing in cluster environment (Walisa and Wichan, 2013a).

Machine learning is one of the subfields of computer science emerging from artificial intelligence in the study of pattern recognition and computational learning theory. Forecasting from data, study of observations, learning patterns and constructing the algorithms can be explored using machine learning. These algorithms can be operated by developing a framework for trained input dataset to make data-driven predictions and decisions rather than following static way of programming implementing dynamically. Machine learning is a discipline of computational statistics and mathematical optimisation by conveying methods, theory and application in various fields (Rich et al., 2008).

Machine learning uses hyper parameters for tuning parameters of an algorithm to verify the best learning method (Brownlee, 2016). Multi-label classifier learning and K-fold cross-validation tasks are used for evaluating results in adapting algorithms.

When a learning process uses various parameters, a range of patterns can be conveyed for different jobs resulting total execution time. The time for total execution depends on the pattern assigned (Pavlo, (2009). To execute jobs efficiently best pattern should be recognised from various machine learning techniques.

A method is proposed by implementing the machine learning algorithms using the python language to minimise execution time (Haroshi et al., 2011). In this paper, an attempt is made to execute supervised and unsupervised machine learning techniques using python tool to formulate the optimum efficiency (Asha and Shravanthi, 2013).

The rest of the paper is organised as follows. Section 2 introduces background in machine learning. Section 3 describes supervised techniques by python language and Section 4 about unsupervised learning algorithms. In this paper, Section 5 describes the proposed model for data analytics. Section 6 discusses the performance evaluation. Section 7 discusses the result analysis and Section 8 concludes the paper.

2 Background

2.1 *Machine learning*

Machine learning algorithms are categorised into two basically supervised and unsupervised techniques:

Supervised learning: Applications in which the training data comprise example of the input vectors along with their corresponding target vectors are known as supervised learning methods (Lakshmi, 2016).

Unsupervised learning: In other pattern-recognition problems, the training data consist of a set of input vectors x without any corresponding target values. The goal in such unsupervised learning problems may be to discover groups of similar examples within the data (Manar and Stephane, 2015).

Supervised techniques in machine learning areas can be further grouped as below:

Classification: This assigns a category to each object such as OCR, text classification, speech recognition.

Regression: This is used to predict a real value for each object such as stock prices, values, economic variables and ratings.

Clustering: This is based on partition data into homogeneous groups (analysis of very large datasets).

Ranking: The order objects according to some criterion (relevant web pages returned by a search engine).

Dimensional reduction: To find lower-dimensional manifold preserving ties of the data (computer vision).

Density estimation: This is used for learning probability distribution according to which data have been sampled.

A variety of learning algorithms including linear regression, logistic regression from supervised and expectation–maximisation (EM) in a Gaussian mixture model, and K-means of unsupervised learning satisfies these characteristics (Michael, 2015).

2.2 Python programming language

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms (Python Documentation: https://docs.python.org/2/tutorial/).

A language excels at string processing – that is the manipulation of string lists a few languages with good string processing capabilities and compares them in terms of the degree to which they are still being actively developed by a community of developers testing whether or not they are object oriented (Stuart and Harald, 2007).

3 Supervised learning algorithms in python

Supervised learning – class labels/target variable are known in learning algorithms by train sets using various training techniques such as linear regression, logistic regression, support vector machines, K-nearest neighbour, expectation and maximisation, K-means, decision tree, random forests and many more. A train set is set to learn a function $h : x \rightarrow y$ so $h(x)$ is a best predictor of y.

3.1 Linear regression

A parameterised linear function is defined with weights mapping x and y giving in equation (1):

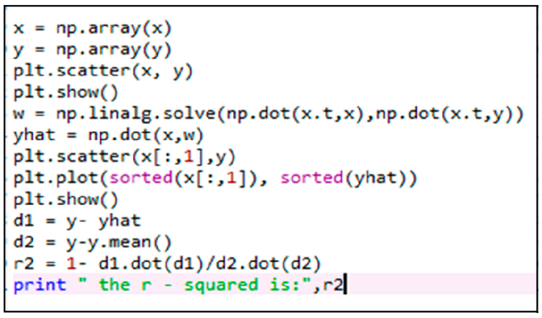
$$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x. \tag{1}$$

Equation (1) is formalised as a linear function is defined considering x as vectors. The cost function is given as shown in equation (2).

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right)^2 . \tag{2}$$

This cost function repeatedly changes and converges to predict the best fit forming gradient descent. A snippet illustrating the linear regressing is shown in Figure 1.

Figure 1 Linear regression (see online version for colours)



```
x = np.array(x)
y = np.array(y)
plt.scatter(x, y)
plt.show()
w = np.linalg.solve(np.dot(x.t,x),np.dot(x.t,y))
yhat = np.dot(x,w)
plt.scatter(x[:,1],y)
plt.plot(sorted(x[:,1]), sorted(yhat))
plt.show()
d1 = y- yhat
d2 = y-y.mean()
r2 = 1- d1.dot(d1)/d2.dot(d2)
print " the r - squared is:",r2
```

Figure 1 illustrates the r^2 calculated for a sum of mean squared errors and the covariance giving a correlation between x and y. Figure 2 shows the plot representing the predictions.

Figure 2 Linear model (see online version for colours)

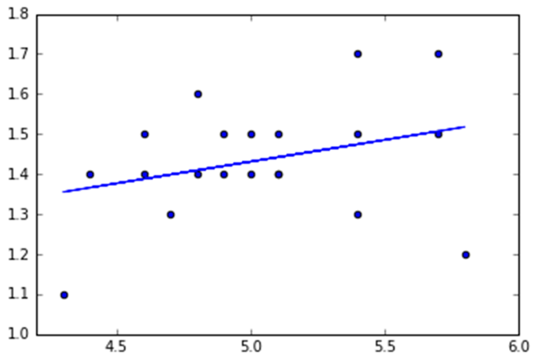


Figure 2 describes the linear model and the line describes the deviation of values from the averages taken from existing for the use of predictions.

The classification problem with the data of binary variables in which you can take values 0 and 1. 0 represents the positive class, whereas 1 represents the positive class.

This approach of classification of the discrete values can be constructed with the logistic regression. This algorithm trains you for the corresponding *x* by classification.

3.2 Logistic regression

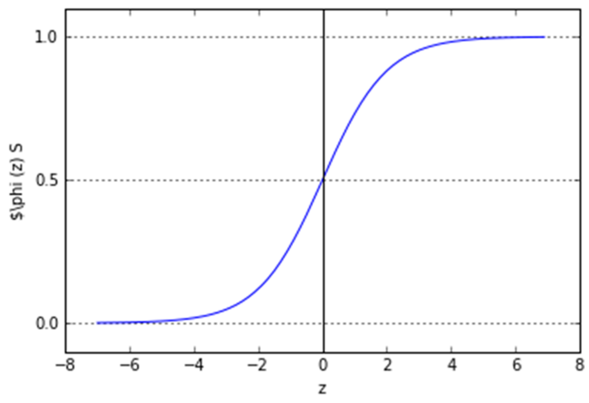
The classification problem uses binary values 0 and 1 calling positive and negative classes where $y \in \{0,1\}$. To fix the change a logistic function or a sigmoid function is described as $g(z)$:

$$h_\theta(x) = g\left(\theta^T x\right) = \frac{1}{1+e^{-\theta^T x}} \tag{3}$$

$$g(z) = \frac{1}{1+e^{-z}}. \tag{4}$$

The sigmoid curve represents the 0 and 1 classes for giving x_i, and corresponding Y_i labels in Figure 3.

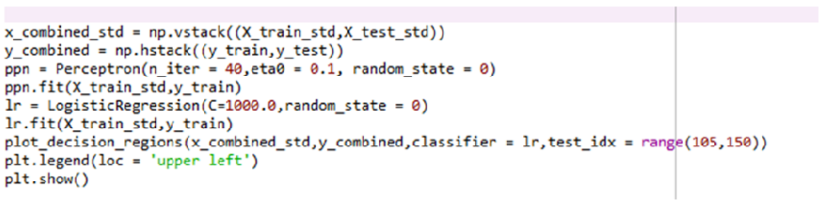
Figure 3 Sigmoid function (see online version for colours)



Logistic regression model derivatives the sigmoid function which follows the least square regression. The least square regression could be derived as the maximum likelihood estimator under a set of suppositions. These assumptions endow the stated classification model with a set of probabilistic assumptions and then fit the parameters via maximum likelihood (Walisa and Wichian, 2013b).

In Snippet 1, decision regions are classified as training and test through which a signed cost function and the classification are shown by a plot for decision regions.

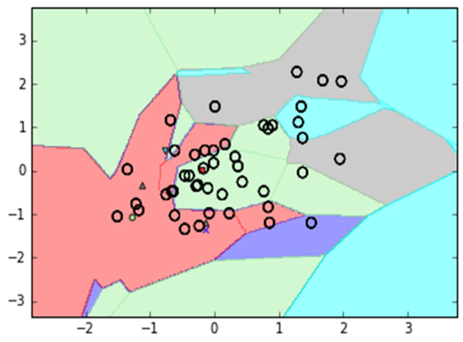
Snippet 1 Logistic regression using python (see online version for colours)



```
x_combined_std = np.vstack((X_train_std,X_test_std))
y_combined = np.hstack((y_train,y_test))
ppn = Perceptron(n_iter = 40,eta0 = 0.1, random_state = 0)
ppn.fit(X_train_std,y_train)
lr = LogisticRegression(C=1000.0,random_state = 0)
lr.fit(X_train_std,y_train)
plot_decision_regions(x_combined_std,y_combined,classifier = lr,test_idx = range(105,150))
plt.legend(loc = 'upper left')
plt.show()
```

Figures 3 and 4 describe the logistic regression, classification for the temperature dataset. It shows data points plotted in the decision regions for the test dataset.

Figure 4 Logistic regression (see online version for colours)



3.3 Support vector machines

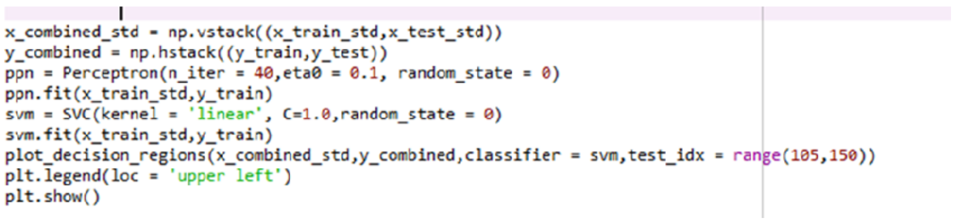
Support vector machines represent an extension to nonlinear models of the generalised portrait algorithm developed by Vladimir Vapnik (Schwarz, 1978). The SVM algorithm is based on the statistical learning theory. It will be useful computationally if only a small fraction of the data points is supported vectors (Rich et al., 2008).

$$r = \frac{y_s(\mathbf{w}^T\mathbf{x}_s + b)}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|} \tag{5}$$

$$\begin{aligned} \mathbf{w}^T\mathbf{x}_i + b &\le -\rho/2 \quad \text{if } y_i = -1 \\ \mathbf{w}^T\mathbf{x}_i + b &\ge \rho/2 \quad \text{if } y_i = 1 \end{aligned} \Leftrightarrow y_i(\mathbf{w}^T\mathbf{x}_i + b) \ge \rho/2.$$

Snippet 2 shows the regularised regression where increasing the value of *c* increases the bias and lowers the variance of the model.

Snippet 2 Support vector machines using python (see online version for colours)



```
x_combined_std = np.vstack((x_train_std,x_test_std))
y_combined = np.hstack((y_train,y_test))
ppn = Perceptron(n_iter = 40,eta0 = 0.1, random_state = 0)
ppn.fit(x_train_std,y_train)
svm = SVC(kernel = 'linear', C=1.0,random_state = 0)
svm.fit(x_train_std,y_train)
plot_decision_regions(x_combined_std,y_combined,classifier = svm,test_idx = range(105,150))
plt.legend(loc = 'upper left')
plt.show()
```

The variable *C* can control, the penalty for misclassification. Large values if *C* corresponds to large error penalties were as misclassification errors can be occurred when *C* value is small. Therefore, the bias variance is tuned to adjust the trade offs. The concept of regularised regression is shown in Snippet 2 where increasing the values of *C* increases the bias and lowers the variance of the model.

3.4 K nearest neighbour

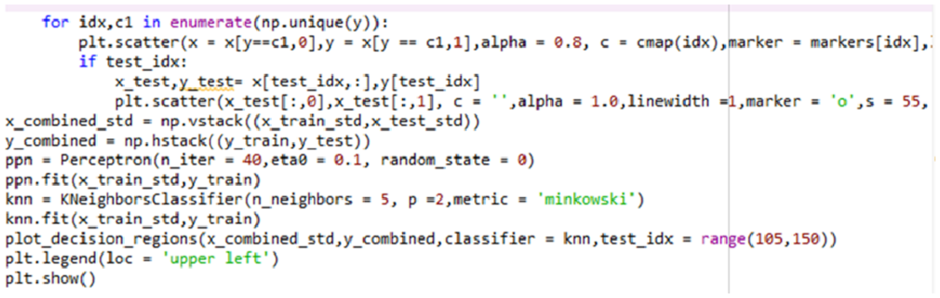
KNN is a lazy learner algorithm which is a non-parameterised model that is instance based. Models which are instance-based characterised by memorising the training dataset and choose the best with the special case of instance-based learning with zero cost during the learning process.

KNN is summarised in the following steps:

- choose the number of *k* and a distance metric
- find the K nearest neighbours of the sample that we want to classify
- assign the class label by majority vote.

The right choice of *k* is crucial to find a good balance between over and under the fitting. A simple Euclidean distance measure is used for real valued samples of temperature dataset. Snippet 3 measures the Euclidean distance to standardise the data so that each feature contributes equally to the distance (Asha and Shravanthi, 2013).

Snippet 3 K nearest neighbour using python (see online version for colours)

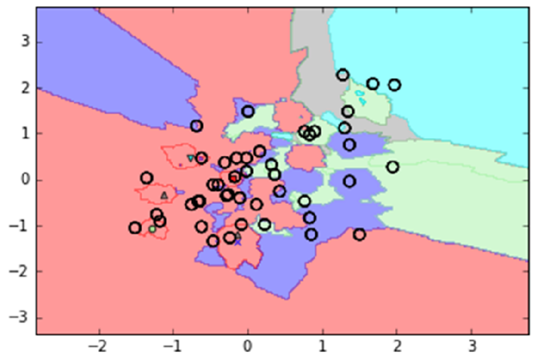


```
    for idx,c1 in enumerate(np.unique(y)):
        plt.scatter(x = x[y==c1,0],y = x[y == c1,1],alpha = 0.8, c = cmap(idx),marker = markers[idx],
        if test_idx:
            x_test,y_test= x[test_idx,:],y[test_idx]
            plt.scatter(x_test[:,0],x_test[:,1], c = '',alpha = 1.0,linewidth =1,marker = 'o',s = 55,
x_combined_std = np.vstack((x_train_std,x_test_std))
y_combined = np.hstack((y_train,y_test))
ppn = Perceptron(n_iter = 40,eta0 = 0.1, random_state = 0)
ppn.fit(x_train_std,y_train)
knn = KNeighborsClassifier(n_neighbors = 5, p =2,metric = 'minkowski')
knn.fit(x_train_std,y_train)
plot_decision_regions(x_combined_std,y_combined,classifier = knn,test_idx = range(105,150))
plt.legend(loc = 'upper left')
plt.show()
```

In this snippet, a generalisation of Euclidean and Manhattan are used which set the parameter $p = 2$ or the Manhattan distance $p = 1$ provided for the metric parameter.

By implementing the above described snippet, Figure 5 shows the balancing for choosing the right K-value of the data.

Figure 5 K nearest neighbour (see online version for colours)



This memory-based approach adapts to the classification rapidly for the training data collected. The training data in context are the diabetics' dataset was chosen very few features is shown in Figure 5.

3.5 Decision tree

Decision tree is a classifier which concentrates on interpretability. This suggests a breaking down for making decisions on posing queries. Using the decision tree in the process begins at the root and follows by splitting the data by information gain. It is an iterative process for reaching the maximum depth of the tree pruning is the process which is implemented.

Snippet 4 illustrates the fit to the temperature data which classifies basing on the entropy generated while training the data.

$$IG(D_P, f) = I(D_P) - \sum_{j=1}^{m} \frac{N_j}{N_p} I(D_j), \tag{6}$$

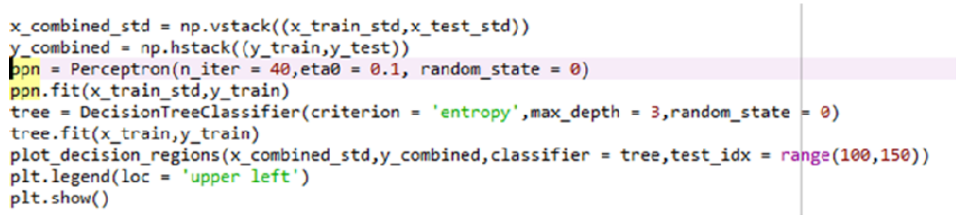
where IG is Gini index, IH is entropy and IE is the classification error defined for all non-empty classes.

$$I_H(t) = -\sum_{i=1}^{c} p(i|t)\log_2 p(i|t) \tag{7}$$

$$I_E = 1 - \max\{p(i|t)\}. \tag{8}$$

Figure 6 shows the parallel boundaries of the decision tree with the maximum depth of three using entropy as a criterion of impurity and tree is given in Figure 7.

Snippet 4 Decision region for using (see online version for colours)



```
x_combined_std = np.vstack((x_train_std,x_test_std))
y_combined = np.hstack((y_train,y_test))
ppn = Perceptron(n_iter = 40,eta0 = 0.1, random_state = 0)
ppn.fit(x_train_std,y_train)
tree = DecisionTreeClassifier(criterion = 'entropy',max_depth = 3,random_state = 0)
tree.fit(x_train,y_train)
plot_decision_regions(x_combined_std,y_combined,classifier = tree,test_idx = range(100,150))
plt.legend(loc = 'upper left')
plt.show()
```

Figure 6 Decision tree representation of the dataset (see online version for colours)

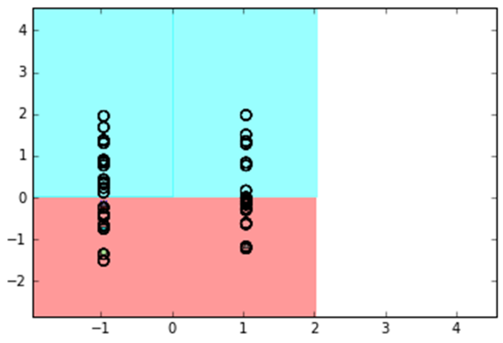
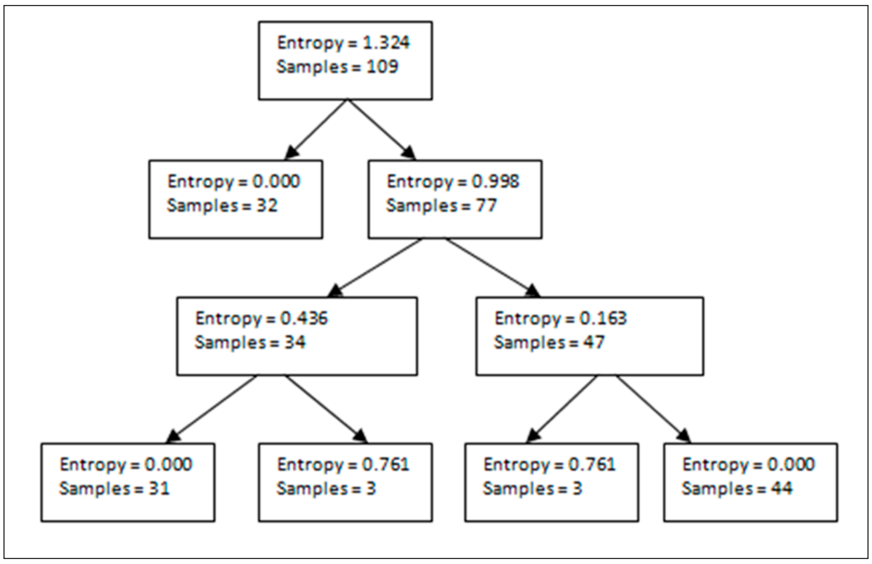


Figure 7 Decision tree for a diabetes dataset giving entropy (see online version for colours)



The decision algorithm splits the tree root and data on the feature resulting the information gain. In an iterative process, the split can be possible at each child node until the leaves remain to be pure. This shows the samples at each node belongs to the same class. This result to prune the tree by setting a limit for the maximal depth of the tree.

3.6 Random forest

Random forest typically prunes the hyperparameter values since the ensemble model is quite robust to noise from the individual decision trees. The parameter is the number of trees chosen for better performance of the random forest classifier at the expense of an increased computational cost.

1 Draw a random bootstrap sample of size *n*.

2 Grow a decision tree from the bootstrap sample. At each node:

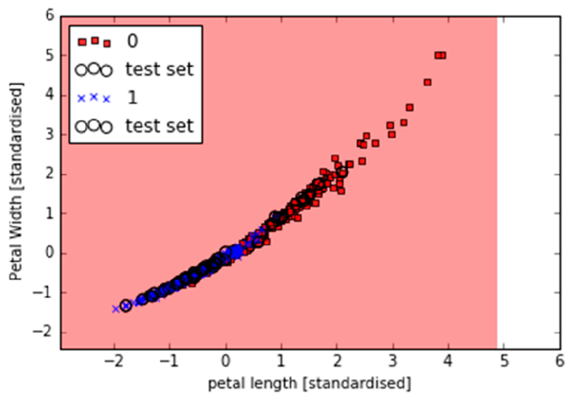
   a randomly select d features without replacement and

   b split the objective function for instance by maximising the information gain.

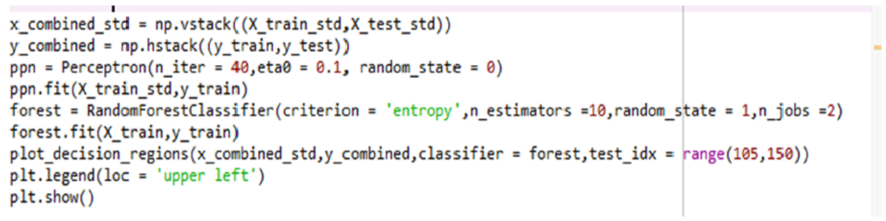
3 Aggregate the predictions by each tree to assign the class label by majority vote.

Using the preceding Snippet 5 a random forest from ten decision trees via the n_estimators parameters and used the entropy criterion as an impurity measure to split the nodes. The tiny random forest dataset uses a n_jobs parameter for parallelising the model training multiple cores. In Figure 8 random forest is used implemented using diabetes data set.

Figure 8 Diabetes dataset plotted using random forest (see online version for colours)



Snippet 5 Random forest uses python (see online version for colours)



```
x_combined_std = np.vstack((X_train_std,X_test_std))
y_combined = np.hstack((y_train,y_test))
ppn = Perceptron(n_iter = 40,eta0 = 0.1, random_state = 0)
ppn.fit(X_train_std,y_train)
forest = RandomForestClassifier(criterion = 'entropy',n_estimators =10,random_state = 1,n_jobs =2)
forest.fit(X_train,y_train)
plot_decision_regions(x_combined_std,y_combined,classifier = forest,test_idx = range(105,150))
plt.legend(loc = 'upper left')
plt.show()
```

4 Unsupervised techniques

In the cluster problem a training set $\{x(1), x(2), \ldots, x(m)\}$ are grouped under a few cohesive 'clusters'. This technique does not involve a labels $y(I)$. Hence, they are referred as unsupervised learning.

4.1 K-means

This is a clustering algorithm repeatedly carries out two steps:

- assigning each training example $x(I)$ to the closest cluster centroid μ_j
- moving each centroid μ_j with the mean of the points assigned to it. For each I set

$$c^{(i)} := \operatorname{argmin}_j \left\| x^{(i)} - \mu_j \right\|^2 . \tag{9}$$

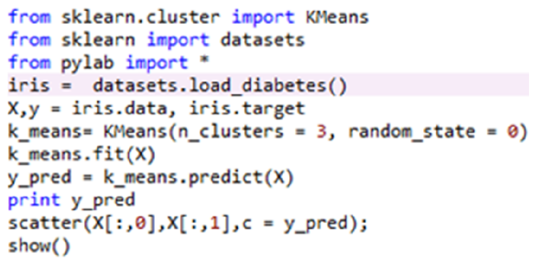
And for each j set

$$\mu_j := \frac{\sum_{i=1}^{m} 1\left\{c^{(i)} = j\right\} x^{(i)}}{\sum_{i=1}^{m} 1\left\{c^{(i)} = j\right\}}. \tag{10}$$

To initialise the cluster centroids by choosing k training sets randomly and set the cluster centroids to be equal to the values of the trained examples.

Snippet 6 illustrates the distortion function with K-means function measures the sum of squared distances between each training examples and cluster centroid.

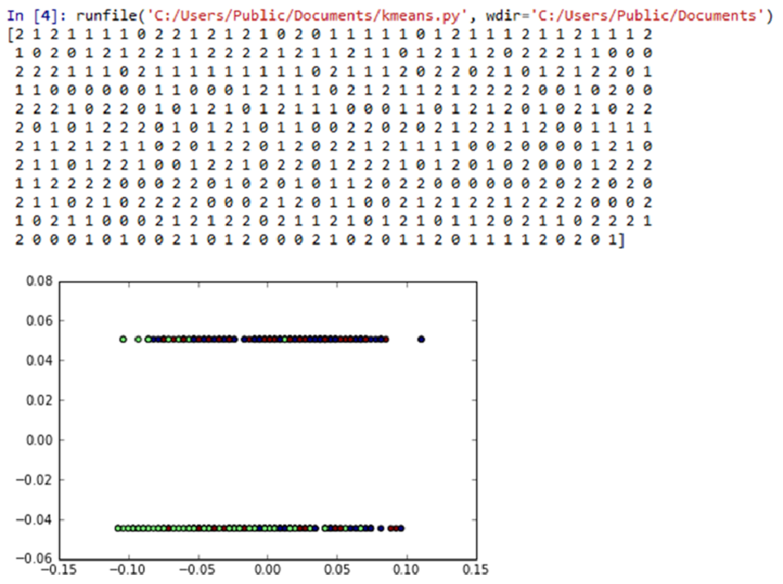
Snippet 6 Clustering with K-means (see online version for colours)



```
from sklearn.cluster import KMeans
from sklearn import datasets
from pylab import *
iris =  datasets.load_diabetes()
X,y = iris.data, iris.target
k_means= KMeans(n_clusters = 3, random_state = 0)
k_means.fit(X)
y_pred = k_means.predict(X)
print y_pred
scatter(X[:,0],X[:,1],c = y_pred);
show()
```

The distortion function c in equation (9) is a convex function implemented in iris dataset and so coordinates descent on J for converge the global minimum. The different clustering's found the lowest distortion for a cluster is chosen. Figure 9 uses the diabetes data set for executing K-Means clustering technique.

Figure 9 Results of K means clustering (see online version for colours)



```
In [4]: runfile('C:/Users/Public/Documents/kmeans.py', wdir='C:/Users/Public/Documents')
[2 1 2 1 1 1 1 0 2 2 1 2 1 2 1 0 2 0 1 1 1 1 1 0 1 2 1 1 1 2 1 1 2 1 1 1 2
 1 0 2 0 1 2 1 2 2 1 1 2 2 2 2 1 2 1 1 2 1 1 0 1 2 1 1 2 0 2 2 2 1 1 0 0 0
 2 2 2 1 1 1 0 2 1 1 1 1 1 1 1 1 1 0 2 1 1 1 2 0 2 2 0 2 1 0 1 2 1 2 2 0 1
 1 1 0 0 0 0 0 0 1 1 0 0 0 1 2 1 1 1 0 2 1 2 1 1 2 1 2 2 2 2 0 0 1 0 2 0 0
 2 2 2 1 0 2 2 0 1 0 1 2 1 0 1 2 1 1 1 0 0 0 1 1 0 1 2 1 2 0 1 0 2 1 0 2 2
 2 0 1 0 1 2 2 2 0 1 0 1 2 1 0 1 1 0 0 2 2 0 2 0 2 1 2 2 1 1 2 0 0 1 1 1 1
 2 1 1 2 1 2 1 1 0 2 0 1 2 2 0 1 2 0 2 2 1 2 1 1 1 1 1 0 0 2 0 0 0 0 1 2 1 0
 2 1 1 0 1 2 2 1 0 0 1 2 2 1 0 2 2 0 1 2 2 2 1 0 1 2 0 1 0 2 0 0 0 1 2 2 2
 1 1 2 2 2 2 0 0 0 2 2 0 1 0 2 0 1 0 1 1 2 0 2 2 0 0 0 0 0 0 2 0 2 2 0 2 0
 2 1 1 0 2 1 0 2 2 2 2 0 0 0 2 1 2 0 1 1 0 0 2 1 2 1 2 2 1 2 2 2 2 0 0 0 2
 1 0 2 1 1 0 0 0 2 1 2 1 2 2 0 2 1 1 2 1 0 1 2 1 0 1 1 2 0 2 1 1 0 2 2 2 1
 2 0 0 0 1 0 1 0 0 2 1 0 1 2 0 0 0 2 1 0 2 0 1 1 2 0 1 1 1 1 2 0 2 0 1]
```

4.2 *Expectation and maximisation*

In EM, you randomly initialise your model parameters, and then you alternate between (E) assigning values of hidden variables, based on parameters and (M) computing parameters based on fully observed data (Chu et al., 2006).

E-step: Coming up with values to hidden variables, based on parameters. If you work out the math of choosing the best values for the class variable based on the features of a given piece of data in your dataset, it comes out to "for each data point, chose the centroid that it is closest to, by Euclidean distance, and assign that centroids label".

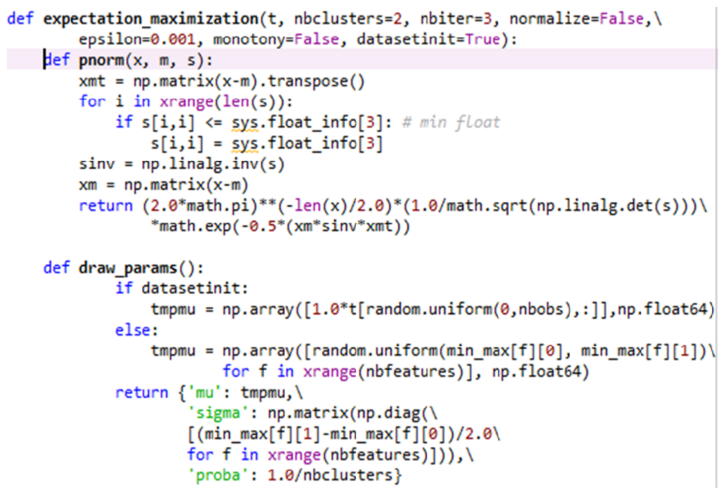
$$w_j^{(i)} := p\left(z^{(i)} = j | x^{(i)}; \varnothing, \mu, \sum\right). \quad (11)$$

M-step: Coming up with parameters, based on full assignments. If you work out the math of choosing the best parameter values based on the features of a given piece of data in your dataset, it comes out to "take the mean of all the data points that were labelled as *c*" (Schwarz, 1978).

$$\varnothing_j = \frac{\sum_{i=1}^{m} w_j^{(i)}}{-\beta}. \quad (12)$$

The EM algorithm is an iterative algorithm that uses maximum likelihood estimation becomes nearly identical to estimate the parameters of the Gaussian discriminant analysis model except that plays the role of the class labels is illustrated in the Snippet 7.

Snippet 7 Expectation and maximisation step using python (see online version for colours)



```
def expectation_maximization(t, nbclusters=2, nbiter=3, normalize=False,\
        epsilon=0.001, monotony=False, datasetinit=True):
    def pnorm(x, m, s):
        xmt = np.matrix(x-m).transpose()
        for i in xrange(len(s)):
            if s[i,i] <= sys.float_info[3]: # min float
                s[i,i] = sys.float_info[3]
        sinv = np.linalg.inv(s)
        xm = np.matrix(x-m)
        return (2.0*math.pi)**(-len(x)/2.0)*(1.0/math.sqrt(np.linalg.det(s)))\
                *math.exp(-0.5*(xm*sinv*xmt))

    def draw_params():
            if datasetinit:
                tmpmu = np.array([1.0*t[random.uniform(0,nbobs),:]],np.float64)
            else:
                tmpmu = np.array([random.uniform(min_max[f][0], min_max[f][1])\
                        for f in xrange(nbfeatures)], np.float64)
            return {'mu': tmpmu,\
                    'sigma': np.matrix(np.diag(\
                    [(min_max[f][1]-min_max[f][0])/2.0\
                    for f in xrange(nbfeatures)])),\
                    'proba': 1.0/nbclusters}
```

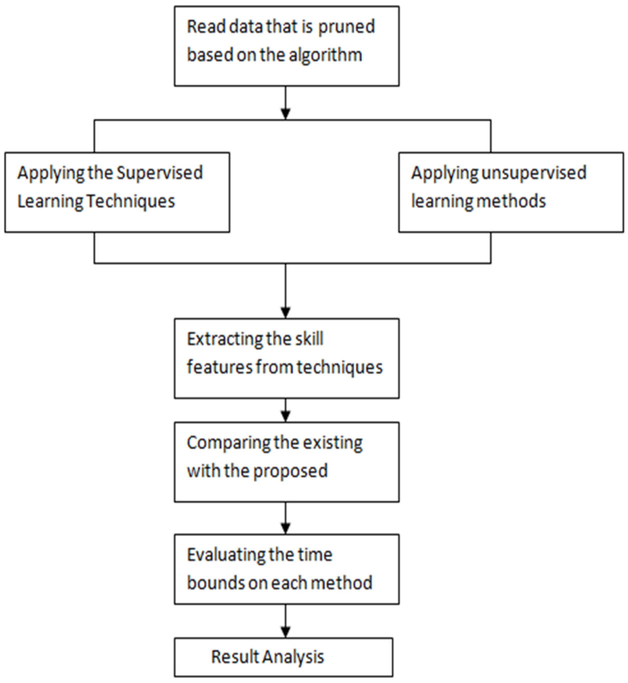
5 Proposed methodology

When analysing data, time is a vital factor which has a major impact on execution. Execution time of each node is read when datasets fit into a distributed memory. Almost all the machine learning algorithms are iterative, so execution of memory-based jobs is crucial and is less efficient.

When the dataset does not fit into distribute memory, then the data are initially stored on local disks and are used until the execution terminates. The pattern determines whether the given job is memory based or not. For improving the execution time machine learning techniques is implemented as in the following model.

This model in Figure 10 suggests the least execution time for data analytics as it follows both the techniques of supervised and unsupervised behaviours of machine learning.

Figure 10 Model


Read data that is pruned based on the algorithm
Applying the Supervised Learning Techniques
Applying unsupervised learning methods
Extracting the skill features from techniques
Comparing the existing with the proposed
Evaluating the time bounds on each method
Result Analysis

As drafted in the proposed model, prune the dataset according to the algorithm for analysing. Evaluate each technique of supervised and unsupervised on the dataset.

The model illustrated below has the following phases:

- read the dataset as input
- classification of data based on supervised and unsupervised learning methods
- extracting the featured and evaluating the techniques
- compared with the existing methods
- result analysis is done is the final phase.

Features of each algorithm are extracted for the best time, efficiency using regression, classification and clustering on the dataset. The time efficiency is evaluated by applying various ML techniques on the data. The algorithm can be improved with additional parameter for accessing the better results.

6 Evaluation of algorithms

This paper evaluates the supervised and unsupervised techniques of machine learning. The supervised methods, both linear and logistic regression are fitted into the pair of values which resulting the targeted values either sided of the line of regression. If unsupervised methods are used, the points are scattered forming clusters in K-means.

This evaluation reveals the impact on supervised techniques for data analytics. Our method targets multiple jobs using different parameters. Since the input of the jobs is the same, the job integration technique improves the performance of the jobs. Figure 11 illustrates the average of linear regression methods besides the existing average.

Figure 12, the graph is the average taken from the time complexities calculated on each machine learning technique. The straight line is the linear average and the blue line is the average of time, space, read and writes operations of each method of machine learning.

Figure 11 Average and linear average (see online version for colours)

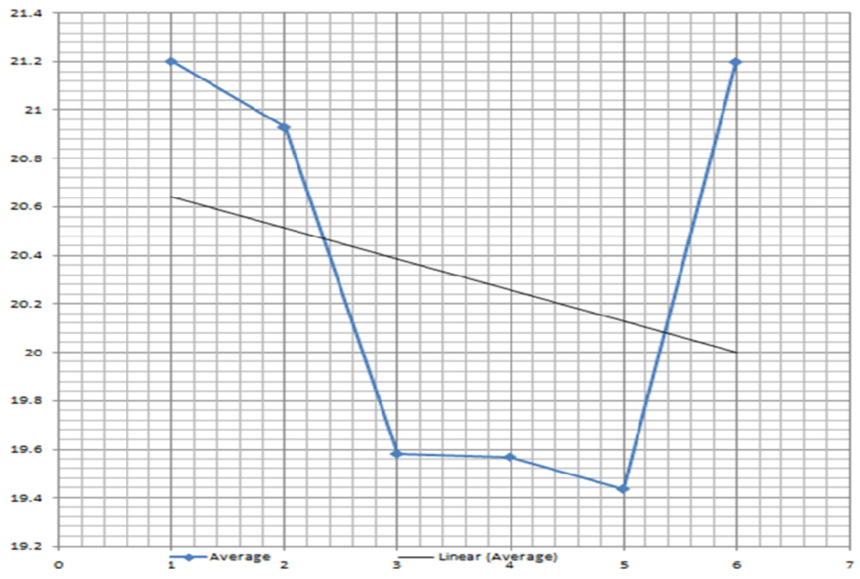
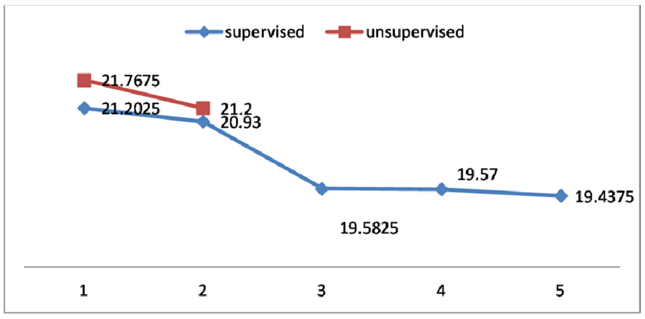


Figure 12 Variation in learning techniques (see online version for colours)



7 Result analysis

The machine learning techniques reveal the efficient use of time and space. These methods train the machine so they adapt to the dataset. Figure 13 gives the gist of the techniques used for ML.

Figure 13 Supervised learning techniques (see online version for colours)

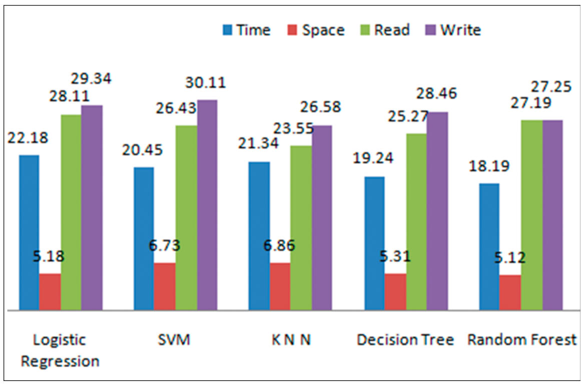
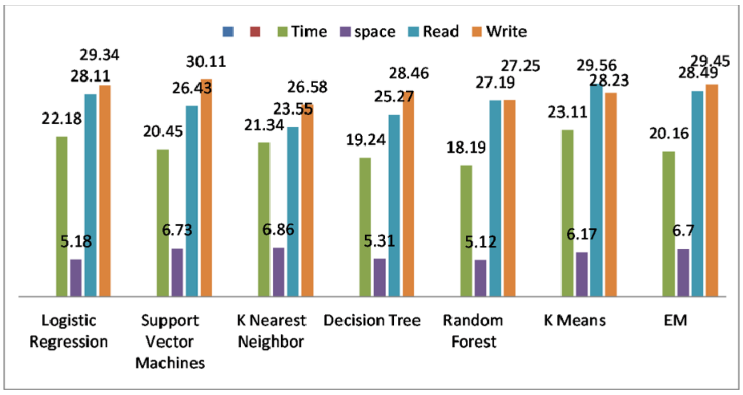


Figure 14 shows the combined measures of supervised and unsupervised learning techniques considering the time, space, read and write operations. By observation 5.12 MB and 18.19S is the minimum space and time used by random forest, respectively. But the random forest is less efficient considering the disk read and writes operations. KNN shows the disks read and write efficiently as 23.55 and 26.58 measures, respectively. From the metrics in Figure 13 it is evident that supervised learning techniques give more accurate measures in comparison with unsupervised.

Figure 14 Supervised vs. unsupervised (see online version for colours)



Our method optimises the assignments using the advantages of such jobs for developing a deadline scheduling method. Their method maximises the number of jobs that can be run in the cluster while satisfying the deadlines of all jobs. Jobs are scheduled using only the minimum number of nodes so that the cluster keeps free nodes for later jobs. In contrast, our method uses entire cluster to complete jobs as early as possible.

8 Conclusion and future work

Executing data analysis, jobs using various parameters are commonly seen in machine learning but time consuming. The proposed method for optimising the job assignment for machine learning is to minimise the total execution time. Our method can be extended for data analytics job's execution, memory-based execution and job integration, for machine learning and optimises the job assignment based on the execution. An execution of machine learning techniques is developed to predict the execution time of these jobs on the extended execution.

References

Asha, T. and Shravanthi (2013) 'Building machine learning algorithms on Hadoop for big data', *IJET UK Journal*, Vol. 3, No. 2, pp.143–147.

Chu, C-T., Lin, Y-A., Yu, Y., Bradski, G.R., Ng, A.Y. and Olukotun, K. (2006) 'Map-reduce for machine learning on multicore', *NIPS*, MIT Press, pp.281–288.

Haroshi, T., Shinji, N. and Takuyu, A. (2011) 'Optimizing multiple machine learning jobs on map reduce', *IEEE–ICCCTS Conference at Japan*, pp.59–66.

Lakshmi, J.V.N. (2016) 'Stochastic gradient descent using linear regression with python', *IJA-ERA*, Vol. 2, No. 8, December, pp.519–524.

Manar, A. and Stephane, P. (2015) 'Machine learning with Python', *SIMUREX*, October.

Pavlo, A. (2009) 'A comparison of approaches to large-scale data analysis', *Proc. ACM SIGMOD*, USA, pp.100–113.

Rich, C., Karampatziakis, N. and Yessenalina, A. (2008) 'An empirical evaluation of supervised learning in high dimensions', *Proceedings of the 25th International Conference on Machine Learning*, ACM, New York, USA, pp.96–103.

Schwarz, G. (1978) 'Estimating the dimension of a model', *The Annals of Statistics*, Vol. 6, No. 2, pp.461–464.

Stuart, R. and Harald, B. (2007) *Beginning Python for Language Research*, Vol. 2, pp.44–47.

Walisa, R. and Wichan, P. (2013a) 'An adaptive ML on map reduce for improving performance of large scale data analysis', *EC2 IEEE*, Bangkok, Thailand, pp.234–236.

Walisa, R. and Wichan, P. (2013b) 'An adaptive ML on map reduce for improving performance of large scale data analysis on EC2', *IEEE 11th Conference on ICT and Knowledge Engineering 2013*.

Michael, B. (2015) *Machine Learning in Python: Essential Techniques for Predictive Analysis*, Print ISBN:9781118961742, Online ISBN:9781119183600, DOI:10.1002/9781119183600.

Brownlee, J. (2016) *Master Machine Learning–How it Works*, pp.1–5.

Website

https://pythonhosted.org/spyder/