

ATME COLLEGE OF ENGINEERING
13th KM Stone, Bannur Road, Mysore – 560028



A T M E
College of Engineering

Department of Master of Computer Applications
(ACADEMIC YEAR 2024-25)

Lesson Notes

SUBJECT: Machine Learning and
Data Analytics Using Python

CODE: MMC204

SEMESTER: II

SCHEME: 2024

Module 1: Introduction to Machine Learning and Python

1. Introduction to Machine Learning

1.1 Definition

Machine Learning (ML) is a method of data analysis that automates analytical model building. It is a branch of **artificial intelligence (AI)** based on the idea that systems can learn from data, identify patterns, and make decisions with minimal human intervention.

1.2 Importance of ML

- Allows systems to **adapt** automatically.
- Enhances **decision-making**.
- Reduces the need for **explicit programming**.
- Powers many technologies today like Google Search, Netflix recommendations, and self-driving cars.

2. Types of Machine Learning

Type	Description	Example
Supervised	Learns from labeled data.	Email spam detection
Unsupervised	Learns patterns from unlabeled data.	Market segmentation
Reinforcement	Learns via trial and error using rewards and punishments.	Game AI like AlphaGo

3. Applications of Machine Learning

- **Healthcare:** Cancer detection, diagnosis.
- **Finance:** Fraud detection, credit risk scoring.
- **Retail:** Customer segmentation, recommendation systems.
- **Transportation:** Self-driving cars.
- **Manufacturing:** Predictive maintenance.

4. Python for Data Analysis

4.1 Why Python?

- Easy syntax.
- Extensive libraries.
- Great for prototyping.

4.2 Libraries in Python

Library	Use
NumPy	Numerical computing
Pandas	Data manipulation & analysis
Matplotlib	Data visualization
Seaborn	Statistical plotting

4.3 Sample Code: Pandas & Visualization

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
df = pd.read_csv('data.csv')
print(df.head())
```

```
# Plotting a histogram of Age
df['Age'].hist()
plt.title("Age Distribution")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.show()
```

5. Data Preprocessing

5.1 Cleaning

- Remove null values.
- Fix inconsistent data types.
- Remove duplicates.

```
df.dropna(inplace=True)
df.drop_duplicates(inplace=True)
```

5.2 Handling Missing Values

```
# Fill with mean
df['Age'].fillna(df['Age'].mean(), inplace=True)

# Fill with a fixed value
df['Gender'].fillna('Unknown', inplace=True)
```

5.3 Feature Scaling and Normalization

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Standardization
scaler = StandardScaler()
scaled = scaler.fit_transform(df[['Age', 'Income']])

# Normalization
minmax = MinMaxScaler()
normalized = minmax.fit_transform(df[['Age', 'Income']])
```

Module 2: Supervised Learning

1. Regression

1.1 Linear Regression

- Predicts a continuous value.
- Assumes linear relationship.

```
from sklearn.linear_model import LinearRegression
```

```
X = df[['Experience']]  
y = df['Salary']
```

```
model = LinearRegression()  
model.fit(X, y)  
predicted = model.predict(X)
```

1.2 Polynomial Regression

- Fits non-linear data using polynomial terms.

```
from sklearn.preprocessing import PolynomialFeatures  
from sklearn.linear_model import LinearRegression
```

```
poly = PolynomialFeatures(degree=2)  
X_poly = poly.fit_transform(X)
```

```
model = LinearRegression()  
model.fit(X_poly, y)
```

1.3 Evaluation Metrics (Regression)

Metric	Formula
MAE	Mean Absolute Error
MSE	Mean Squared Error
RMSE	Root Mean Squared Error

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

mae = mean_absolute_error(y, predicted)
mse = mean_squared_error(y, predicted)
rmse = np.sqrt(mse)
```

2. Classification

2.1 Logistic Regression

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

2.2 K-Nearest Neighbors (KNN)

```
from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_neighbors=5)
model.fit(X_train, y_train)
```

2.3 Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier()
model.fit(X_train, y_train)
```

2.4 Random Forest

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, y_train)
```

2.5 Evaluation Metrics (Classification)

Metric	Description
Accuracy	Correct predictions / Total predictions
Precision	$TP / (TP + FP)$
Recall	$TP / (TP + FN)$
F1 Score	Harmonic mean of precision and recall
ROC-AUC	Area under the ROC curve

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
```

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, model.predict_proba(X_test)[: ,1])
```

3. Model Training and Evaluation

3.1 Train-Test Split

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

3.2 Cross Validation

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(model, X, y, cv=5)
print("Cross-Validation Score:", scores.mean())
```

3.3 Hyperparameter Tuning with GridSearchCV

```
from sklearn.model_selection import GridSearchCV

params = {'n_neighbors': [3, 5, 7]}
grid = GridSearchCV(KNeighborsClassifier(), params, cv=3)
grid.fit(X_train, y_train)

print(grid.best_params_)
```

4. Overfitting and Underfitting

Term	Description
Overfitting	Model performs well on training but poorly on new data
Underfitting	Model performs poorly on both training and test data

Solutions:

- Use **cross-validation**
- Use **regularization techniques**
- Apply **pruning** in trees
- Reduce complexity or gather more data

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.

Background

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).

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).

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 .

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.

```

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)

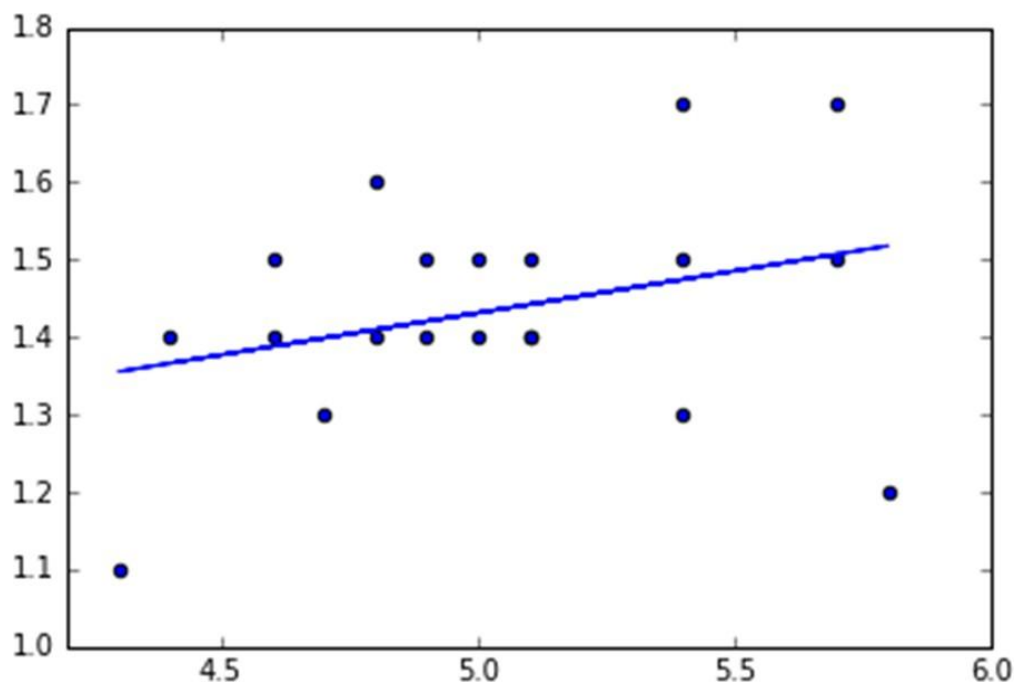


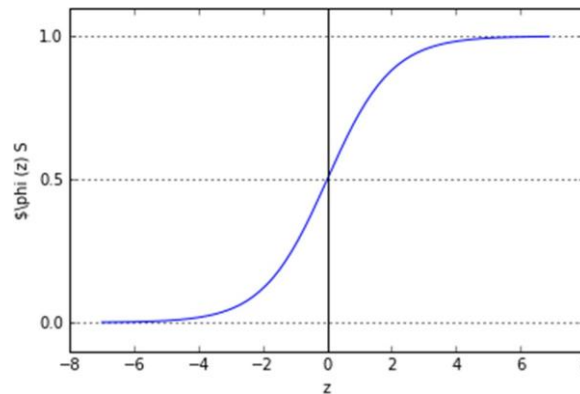
Figure 2 describes the linear model and the line describes the

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.

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).

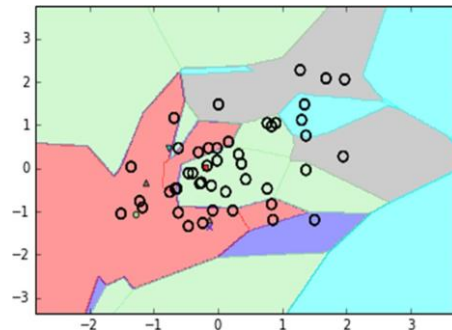
In Snippet 1, decision regions are classified as training and test through which a signed cost function and the classification are shown by a plot for decision regions.

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)



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).

$$\frac{1}{n} \sum_{i=1}^n \ell_i$$

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.

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.

2.1 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.

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)

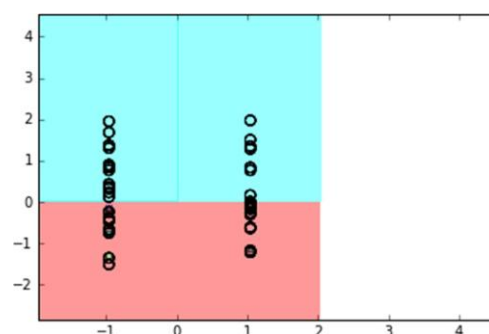
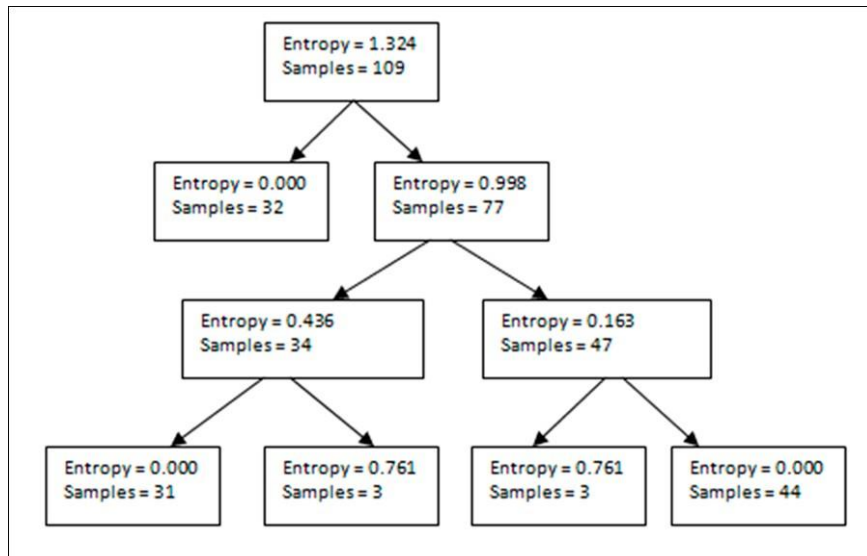


Figure 7 Decision tree for a diabetes dataset giving entropy (see online version

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.

2.2 Random forest

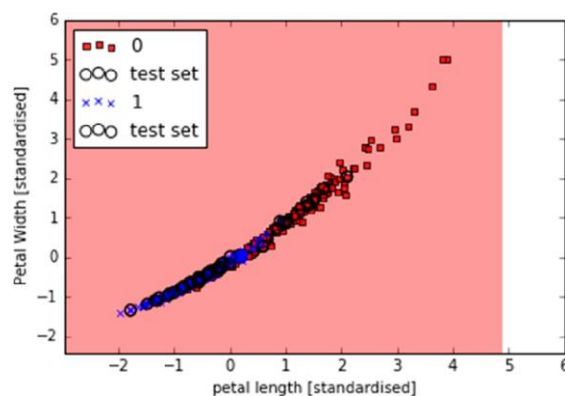
Random forest typically prunes the hyperparameter values since the ensemble model is quite robust to noise from the individual decision trees. The parameter is the number of trees chosen for better performance of the random forest classifier at the expense of an increased computational cost.

- 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()
```

Unsupervised techniques

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

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.

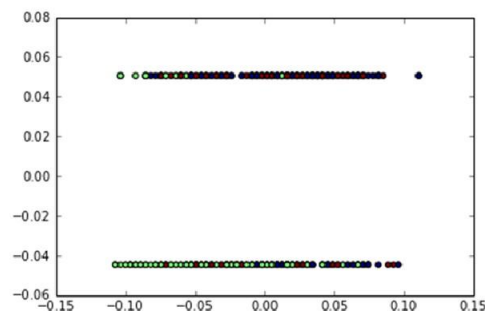
Snippet 6 Clustering with K-means (see online version for colours)

```
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 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]
```



3.1 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”.

he 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}

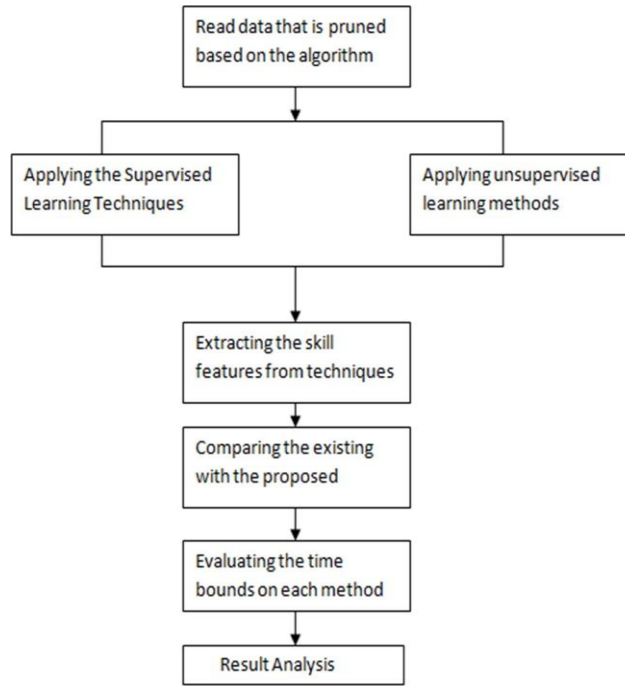
```

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



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.

4 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)

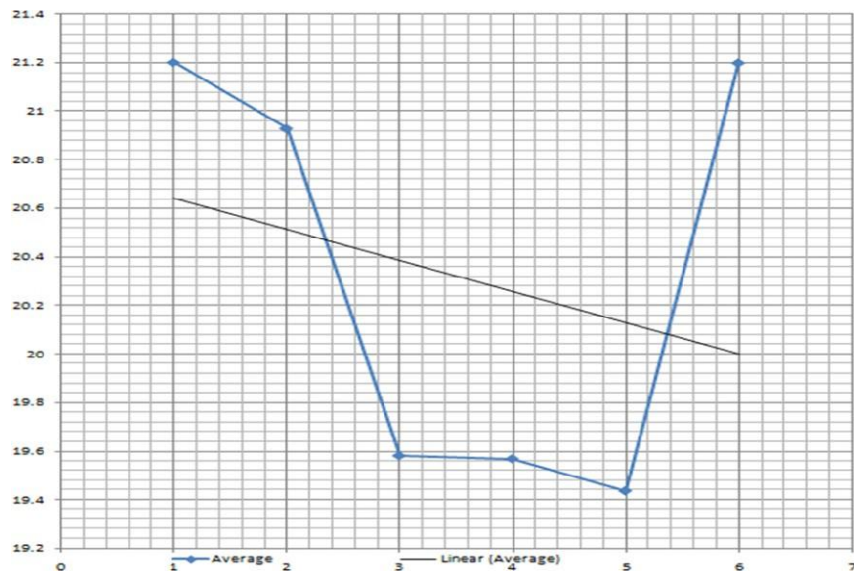


Figure 12 Variation in learning techniques (see online version for colours)

