A Review of Clustering and Classification Techniques on Big Data

Prasanna Kolar\(^1\) and Mohammad Jamshidi\(^2\)
Department of Electrical and Computer Engineering
The University of Texas at San Antonio
One UTSA Circle, San Antonio, TX 78249, USA
Email: prasanna.kolar@utsa.edu, mojamshidi@wacong.org

Abstract—The Electroencephalogram (EEG) signal is an electric signal, particularly voltage arising from synchronized neural activity. EEG is produced due to coordinated activity of millions of neurons in the brain. EEG can be measured in two ways. 1. By placing an electrode on the scalp. 2. Implanting an electrode inside the skull. EEG can be used to classify different mental states and to find abnormalities in neural activity. This is because, synchronized neural activities vary according to mental state, and cognitive activity, stress levels etc and these cause variations in EEG signal. These variations, or abnormalities in neural activity force us to use classifiers, in order to classify the EEG signals. As part of this homework, we discuss 2 classifiers namely k-means and Fuzzy-C mean classifiers, to classify EEG. There are 2 categories of classifiers namely: Supervised, Un-supervised. Supervised classifiers: Linear discriminant analysis, Neural networks and Naive Bayes classifiers are supervised classifiers. Unsupervised classifiers (clustering): K-means clustering and Fuzzy c means (FCM) classifiers are some of the unsupervised classifiers or clustering methods. Supervised classifiers require prior knowledge about the data to be classified and are trained with data input and when the unknown data point is input to the classifiers, it classifies the data point into a particular class based on training. Un-supervised or (clustering) do not need prior knowledge about the data. So clustering is generally used prior to classifiers to prepare training data set for classifiers. Neural networks are widely used as supervised classifier. Here an input data set is used to train the network. In this work, k-means clustering and FCM are used to cluster the input BCI signal data set.

I. INTRODUCTION

A. K-Means Clustering

The k-means clustering is an un-supervised method of classification. Although it was initially designed for signal processing, it is also used extensively in cluster analysis, pattern recognition and data mining applications [1].

The main benefit of k-means clustering is, its ability to separate \( n \) input observations or variables into \( k \) clusters where each observation belongs to the cluster with the nearest distance from cluster center called centroid. The clustering is done by minimizing the Euclidean distance between data and its corresponding cluster centroid. Continuing this strategy, k-mean clustering classifies the data into \( k \) clusters.

1) *Algorithm:* The classification is quite simple. The technique initially uses a fixed \( k \) number of clusters and \( n \) observations. First, \( k \) centroids are defined one for each cluster. Centroids, in the inception are assumed to be at a distance far from each other. Each data point in the given dataset is associated with the nearest centroid and the classification is performed by assigning each data point to its nearest centroid. Then, k new centroids are calculated. They are the barycenter of the clusters resulting from previous step. After these \( k \) new centroids are calculated, the system iterates between the same data points and the nearest new centroid. This looping the \( k \) centroids change their location step by step until no more changes in centroid location occur.

\(^1\)P Kolar is a PhD Candidate at the Electrical and Computer Engineering Dept., The University of Texas at San Antonio, USA email: prasanna.kolar@utsa.edu
\(^2\)M. Jamshidi is a Professor at the Electrical and Computer Engineering Dept., The University of Texas at San Antonio, USA. email: mojamshidi@wacong.org
1) Determine \( k \) initial centroids
2) Using Euclidean distance method, calculate the distance between objects to the centroids.
3) Find the closest centroid for each data point.
4) Group the data point based on minimum Euclidean distance.
5) Iterate through steps 2 and 4 until no object moves between the clusters.
6) If number of clusters is more than objects then each object coordinates are cluster centroids.

B. Fuzzy-C Mean Clustering

Using the principles of Fuzzy logic, which concentrate on fluidity of data validation, Fuzzy clustering is a classifier where data elements can belong to more than one cluster, and each element is associated with a set of membership levels. The membership values indicate the strength of the association between that data element and a particular cluster. Membership values are always between zero and one. Fuzzy clustering is a process of membership level allotment and data elements are assigned to one or more clusters. The Fuzzy C-Means (FCM) Algorithm is one of the widely used fuzzy clustering algorithms. It was developed by Dunn in 1973 and improved by Bezdek in 1981. It is frequently used in pattern recognition and image classifications [2]. As part of this homework the Fuzzy c-means clustering is accomplished using cmeans from skfuzzy, and the output from this function is fed into cmeans_predict from skfuzzy in order to classify new data according to the calculated clusters, which is also known as prediction.

1) Algorithm: Using the concept of the distance between the data point and the calculated cluster, the algorithm works by assigning membership to each data point corresponding to each cluster center. If the data is near to the cluster center its membership value is large. Summation of membership value of each data point should be equal to one. The algorithm is based on minimization of the following objective function:

\[
J_m = \sum_{i=1}^{N} \sum_{j=1}^{C} u_{ij}^m \|X_i - C_j\|^2
\]

- \( m \): Any real number greater than 1
- \( u_{ij} \): The degree of membership of \( x_i \) in the cluster \( j \)

Fuzzy partitioning is carried out through an iterative optimization of the objective function shown above, with the update of membership \( u_{ij} \) and the cluster center \( c_j \) by:

\[
u_{ij} = \frac{1}{\sum_{k=1}^{C} \left( \frac{\|x_i - c_j\|^{2/m}}{\|x_i - c_k\|^{2/m}} \right)}
\]

\[
\max_{ij} \{|u_{ij}^{k+1} - u_{ij}^k|\} < \varepsilon
\]

where \( \varepsilon \) is a termination criterion between 0 and 1, whereas \( k \) are the iteration steps. This procedure converges to a local minimum or a saddle point of \( J_m \).

Fuzzy c-means has been a very important tool for image processing in clustering objects in an image. It is also used for cluster analysis of Plant and animal ecology, Human genetics. It is also used in the field of robotics and climatology.

C. Convolutional Neural Networks

One of the most efficient neural networks to classify image based data is the CNN. CNN ConvNet) [3] is a class of deep, feed-forward artificial neural networks that are applied to analyzing visual imagery. An example of CNN is used in detecting pedestrians [4].

\[
(f * g)(t) \overset{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)g(t - \tau) \, d\tau = \int_{-\infty}^{\infty} f(t - \tau)g(\tau) \, d\tau.
\]

Fig. 1: CNN Algorithm
D. Linear Regression

Regression analysis is a statistical technique for investigating and modeling the relationship between variables [5]. We explain Linear Regression as follows: Consider 2 variables $X$ and $Y$: The linear regression model posits an exact linear relationship between the expected or average value of $Y$, the dependent variable $Y$, and $X$, the independent or predictor variable: $\mu_{y|x} = \alpha + \beta x$ Actual observed values of $Y(y)$ differ from the expected value ($\mu_{y|x}$) by an unexplained or random error($\epsilon$):

$Y = \mu_{y|x} + \epsilon$

$Y = \alpha + \beta (x) + \epsilon$

The relationship between $X$ and $Y$ is a straight-Line (linear) relationship. The values of the independent variable $X$ are assumed fixed (not random); the only randomness in the values of $Y$ comes from the error term $\epsilon$. The errors $\epsilon$ are uncorrelated (i.e. Independent) in successive observations. The errors $\epsilon$ are Normally distributed with mean 0 and variance $\sigma^2$ (Equal variance).

E. Principal Component Analysis

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components [6], [7]. If there are $n$ observations with $p$ variables, then the number of distinct principal components is $\min(n - 1, p)$. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors are an uncorrelated orthogonal basis set. PCA is sensitive to the relative scaling of the original variables. When large multivariate datasets are analyzed, it is often required to reduce their dimensionality. Principal component analysis is one technique for doing this. It replaces the $p$ original variables by a smaller number, $q$, of derived variables, the principal components, which are linear combinations of the original variables. Often, it is possible to retain most of the variability in the original variables with $q$ very much smaller than $p$. Despite its apparent simplicity, principal component analysis has a number of subtleties, and it has many uses and extensions. PCA is mostly used as a tool in exploratory data analysis and for making predictive models. It’s often used to visualize genetic distance and relatedness between populations. PCA can be done by eigenvalue decomposition of a data covariance (or correlation) matrix or singular value decomposition of a data matrix, usually after mean centering (and
normalizing or using Z-scores) the data matrix for each attribute.[4] The results of a PCA are usually discussed in terms of component scores, sometimes called factor scores (the transformed variable values corresponding to a particular data point), and loadings (the weight by which each standardized original variable should be multiplied to get the component score).[5]

II. Motivation

Data Clustering and Classification is an essential part of any Big Data processing and is associated with finding solution to a lot of problems in many areas eg: Neural sciences, Agriculture, Horticulture, Autonomous vehicles to name a few. This review is meant to serve the community by providing a simple comparison of the aforementioned techniques of big data processing. For further research, it is recommended to continue this study with bigger and more complex data like EEG, MEG etc.

III. Implementation

A. K-means

```python
# importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

from sklearn import decomposition
from sklearn import datasets

np.random.seed(5)

centers = [[1, 1], [-1, -1], [1, -1]]
dataset = datasets.load_iris()
x = dataset.data

# Find the optimum number of clusters for k-means classification
from sklearn.cluster import KMeans
wcss = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
    kmeans.fit(x)
    wcss.append(kmeans.inertia_)

# Plotting the results onto a line graph, allowing us to observe 'The elbow'
plt.plot(range(1, 11), wcss)
plt.title('Big Data - HW1 K-Means elbow graph using iris Dataset')
plt.xlabel('Plant Data')
plt.ylabel('SOS within cluster')
plt.show()

# Apply kmeans to the dataset
# Create the kmeans classifier
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
y_kmeans = kmeans.fit_predict(x)
```

Fig. 4: Principal Component Graph
# Visualising the clusters

```python
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 100, c = 'blue', label = 'setosa')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 100, c = 'red', label = 'versicolor')
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1], s = 100, c = 'green', label = 'virginica')
```

# Place the centroids of the clusters

```python
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], marker='*', s = 300, c = 'brown', label = 'Centroid')
```

```python
plt.title("Big Data HW 1 K- Means Clustering")
plt.xlabel('Plant Data')
plt.ylabel('SOS within cluster')
plt.legend()
plt.show()
```

---

**B. Fuzzy C-means**

```python
# Extraction code before the Fuzzy C application
# Fundamental python imports
import numpy as np
```

```python
# Plotting imports
import matplotlib.pyplot as plt
import itertools
```

```python
# Data extraction imports
import pandas as pd, os
import statsmodels.formula.api as smf
```

```python
# Data processing imports
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler, Normalizer
from sklearn.decomposition import TruncatedSVD
```

```python
# Load the Iris Dataset
iris = datasets.load_iris()
```
32 x = pd.DataFrame(iris.data,
    columns=['Sepal Length', 'Sepal Width', 'Petal Length', 'Petal Width'])
33 y = pd.DataFrame(iris.target,
    columns=['Target'])
34 scaler = StandardScaler()
35 X_std = scaler.fit_transform(x)
36 lsa = TruncatedSVD(2, algorithm='arpack')
37
dtm_lsa = lsa.fit_transform(X_std)
dtm_lsa = Normalizer(copy=False).fit_transform(dtm_lsa)
a = pd.DataFrame(dtm_lsa, columns=['component_1', 'component_2'])
a['targets'] = y

46 alldata = np.vstack((a['component_1'],
    a['component_2']))
fpcs = []
49 colors = ['b', 'orange', 'g', 'r', 'c',
    'm', 'y', 'k', 'Brown', 'ForestGreen']
50 for ncenters, ax in enumerate(axes1.reshape(-1), 2):
    cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
        alldata, ncenters, 2, error=0.005,
        maxiter=1000, init=None)
    # Store fpc values for later plots
    fpcs.append(fpc)
    # Plot assigned clusters, for each data point in training set
    cluster_membership = np.argmax(u,
        axis=0)
    for j in range(ncenters):
        ax.plot(a['component_1'][cluster_membership == j],
            a['component_2'][cluster_membership == j], '.', color=colors[j])
    # Mark the center of each fuzzy cluster
    for pt in cntr:
        ax.plot(pt[0], pt[1], 'rs')
    ax.set_title('Centers = {0}; FPC = {1:.2f}'.format(ncenters, fpc))
    ax.axis('off')
59 fig1.tight_layout()
62 fig1.savefig('HW1_iris_Fuzzycmeans.png')

C. Neural Networks

import pandas as pd
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import normalize

from subprocess import check_output

# Open the file for reading...
df = pd.read_csv('../input/Iris.csv')
df.head()

# checking drop/null values check
def = df.drop(['Id'],axis=1)
def.isnull().sum()
df['Species'] = df['Species'].map({'Iris-setosa': 0,
                                     'Iris-versicolor': 1,
                                     'Iris-virginica': 2}).astype(int)

x_train = df[['SepalLengthCm',
               'SepalWidthCm', 'PetalLengthCm',
               'PetalWidthCm']]  

y_train = df['Species']

new_y = []
for i in y_train:
a = [0,0,0]
a[i] = 1
new_y.append(a)

columns = list(x_train)
X = pd.DataFrame.as_matrix(x_train,columns=columns)
Y = np.array(new_y)

#flatten the features for feeding into network base layer
X_train_flatten = X.reshape(X.shape[0],-1).T
Y_train_flatten = Y.reshape(Y.shape[0],-1).T

print("No of training (X):"+str(X_train_flatten.shape))
print("No of training (X):"+str(Y_train_flatten.shape))

#Normalize
XX_train_flatten = normalize(X_train_flatten)
YY_train_flatten = normalize(Y_train_flatten)

# creating the placeholders for X & Y
def create_placeholders(n_x, n_y):
  X = tf.placeholder(shape=[n_x,None],dtype=tf.float32)
  Y = tf.placeholder(shape=[n_y,None],dtype=tf.float32)
  return X,Y

def forward_propagation(X, parameters):
  W1 = parameters['W1']
  b1 = parameters['b1']
  Z1 = tf.add(tf.matmul(W1,X),b1)
  return Z1

def compute_cost(Z1,Y):
  logits = tf.transpose(Z1)
  labels = tf.transpose(Y)
  cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits,labels=labels))
  return cost

# Running the NN !!
tf.reset_default_graph()
(n_x, m) = X_train_flatten.shape
  # shape of X
n_y = Y_train_flatten.shape[0]  # shape of Y
X, Y = create_placeholders(n_x, n_y)  # creating placeholder
tf.set_random_seed(42)
p = initialize_parameters()  # initialize parameter
Z6 = forward_propagation(X,p)  # forward prop
y_softmax = tf.nn.softmax(Z6)  # softmax function
cost = compute_cost(Z6,Y)  # cost function
optimizer =
    tf.train.GradientDescentOptimizer(learning_rate=.01).minimize(cost)
sess = tf.Session()
par = sess.run(p)
Y_pred = sess.run(Z6, feed_dict={X:X_train_flatten})  # forward prop test
cost_value = sess.run(cost, feed_dict={Z6:Y_pred, Y:Y_train_flatten})  # cost function test - First cost
for i in range(0,2000):
    _,new_cost_value = sess.run([optimizer, cost], feed_dict={X: X_train_flatten, Y: Y_train_flatten})
costs.append(new_cost_value)
for i in range(0,2000):
    # 2000 epoch !!
    _,new_cost_value = sess.run([optimizer, cost], feed_dict={X: X_train_flatten, Y: Y_train_flatten})
costs.append(new_cost_value)
p = sess.run(p)  # parameter saving
y_softmax = sess.run(y_softmax, feed_dict={X: X_train_flatten, Y: Y_train_flatten})  # running softmax

d = ScalerPlot of the Iris dataset:

CNN Classification Accuracy : 1.0

\[ \text{Fig. 8: Iris data CNN Classification} \]

D. Linear Regression

#!/usr/bin/python
# -*- coding: utf-8 -*-

""
Linear Regression Example  Big Data Course""
This code shows how linear regression features by drawing a straight line that minimizes the residual sum of squares between the observed responses in the dataset, and the responses as predicted by the linear approximation. The coefficients, the residual sum of squares and the variance score are also calculated.

```
print(__doc__)

import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Loads from the sklearn diabetes data set
diabetes = datasets.load_diabetes()

# Use only two features;
# this can be changed by altering the parameters
diabetes_X = diabetes.data[:, np.newaxis, 3]

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-70]
diabetes_X_test = diabetes_X[-30:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes.target[:-70]
diabetes_y_test = diabetes.target[-30:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# The coefficients
print('Linear Regression Coefficients: 
', regr.coef_)

# The mean squared error
print('Mean squared error: %.2f ' % mean_squared_error(diabetes_y_test, diabetes_y_pred))

# Explained variance score: 1 is perfect prediction
print('Linear Regression Variance score: %.2f ' % r2_score(diabetes_y_test, diabetes_y_pred))

# Plot outputs
plt.title("Linear Regression Plot - HW 4:
Coefficients:{0}
Mean Squared Error:{1}
Variance score: {2}"
.format(regr.coef_,
mean_squared_error(diabetes_y_test, diabetes_y_pred),
r2_score(diabetes_y_test, diabetes_y_pred)))
plt.scatter(diabetes_X_test, diabetes_y_test, color='Green')
plt.plot(diabetes_X_test, diabetes_y_pred, color='Red', linewidth=3)
plt.xlabel("Test")
plt.ylabel("Prediction")
plt.xticks(())
plt.yticks(())
plt.show()
```

E. Linear Regression

F. PCA
Principal component analysis (PCA) is a mainstay of modern data analysis – a black box that is widely used but poorly understood. The goal of this paper is to dispel the magic behind this black box. This tutorial focuses on building a solid intuition for how and why principal component analysis works; furthermore, it crystallizes this knowledge by deriving from simple intuitions, the mathematics behind PCA. This tutorial does not shy away from explaining the ideas informally, nor does it shy away from the mathematics. The hope is that by addressing both aspects, readers of all levels will be able to gain a better understanding of PCA as well as the when, the how and the why of applying this technique.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import decomposition
from sklearn import datasets
np.random.seed(5)
centers = [[1, 1], [-1, -1], [1, -1]]
iris = datasets.load_iris()
X = iris.data
y = iris.target
fig = plt.figure(1, figsize=(4, 3))
plt.clf()
ax = Axes3D(fig, rect=[0, 0, .95, 1],
            elev=48, azim=134)
plt.cla()
pca = decomposition.PCA(n_components=3)
```python
c.fit(X)
X = c.transform(X)

for name, label in [('Setosa', 0), ('Versicolour', 1), ('Virginica', 2)]:
    ax.text3D(X[y == label, 0].mean(),
              X[y == label, 1].mean() + 1.5,
              X[y == label, 2].mean(), name,
              horizontalalignment='center',
              bbox=dict(alpha=.5, edgecolor='r',
                        facecolor='w'))

# Reorder the labels to have colors
y = np.choose(y, [1, 2, 0]).astype(np.float)

ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y,
            cmap=plt.cm.spectral,
            edgecolor='r')

ax.w_xaxis.set_ticklabels([1,2,3,4,5,6,7])
ax.w_yaxis.set_ticklabels([1,2,3,4,5,6])
ax.w_zaxis.set_ticklabels([1,2,3,4,5,6,7])

ax.set_xlabel('Setosa')
ax.set_ylabel('Versicolour')
ax.set_zlabel('Virginica')
plt.title("Big Data HW 2 Principle Component Analysis")
plt.show()
```

**IV. RESULTS**

We find that all the classification and clustering techniques discussed in this study have varying efficiencies, but are most useful in their particular area of use. For example, image classification is best done by CNN. Un-supervised classifications of dataset is efficiently accomplished by K-means and it is simple to implement as well. We find the study complete with respect to the class assignments and project. However, a deeper study can be done on various other datasets, which will bring out and document the efficiencies of the aforementioned and similar techniques.

**V. DATA SET**

As part of this report the following test data were used.
1) EEG signals at 8Hz, 11Hz, 13Hz
2) Diabetes Data records
3) Agricultural flower petal records

**ACKNOWLEDGMENT**

The author likes to thank Dr. Mo Jamshidi, ECE Department, University of Texas, San Antonio. the members of ACE Lab.

**VI. CONCLUSIONS**

The Goal of providing a simple review of 5 Data clustering and classifying techniques has been accomplished. We find that the techniques are simple and a further investigation is recommended to improve this research.

**REFERENCES**


