Practical Implementation

# 1.Dataset Analysis

The initial Dataset consisted of 39,644 observations with 61 features collected over a 2 years period from Jan 2013 - Jan 2015 from mashable website.

After extensive analysis it was discovered that the data of numerous anomalies, for instance:

|  |  |
| --- | --- |
| **Data Set** | **Website** |
| 843,330 shares  12 videos  128 videos | 792 shares  0 videos  12 videos |

Fig. : Comparison of Actual vs Recorded Data for an article(leaked: More Low Cost iPhone photos)

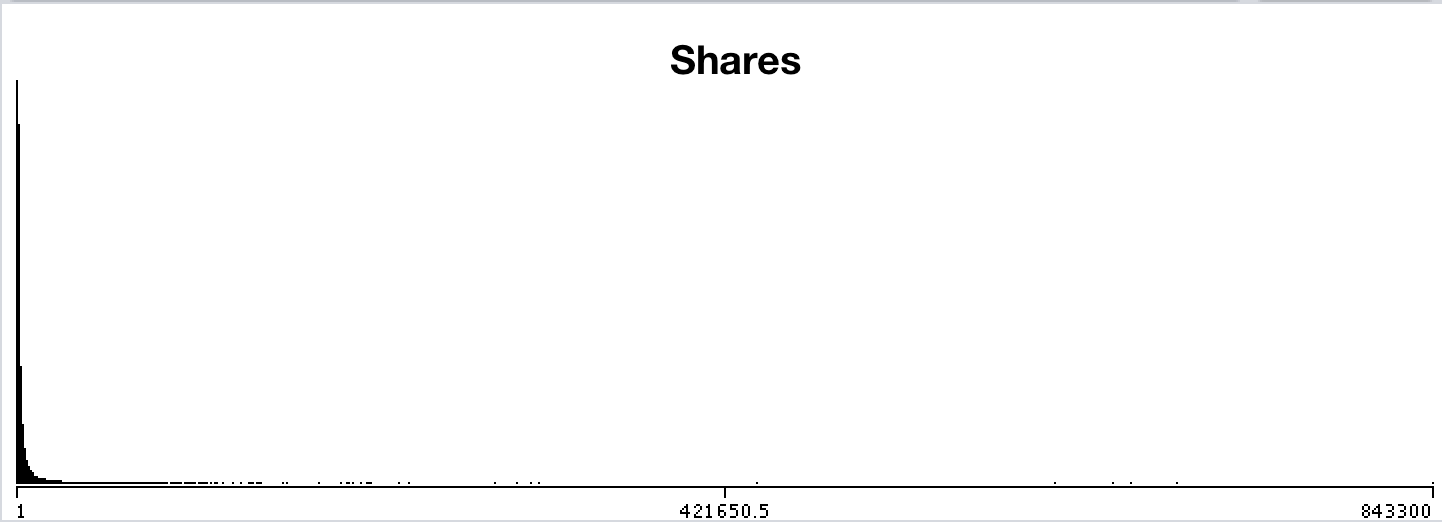
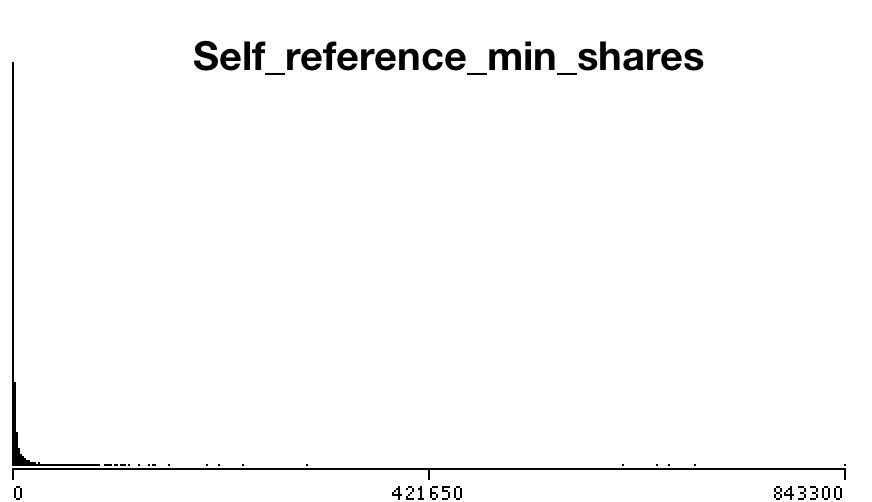
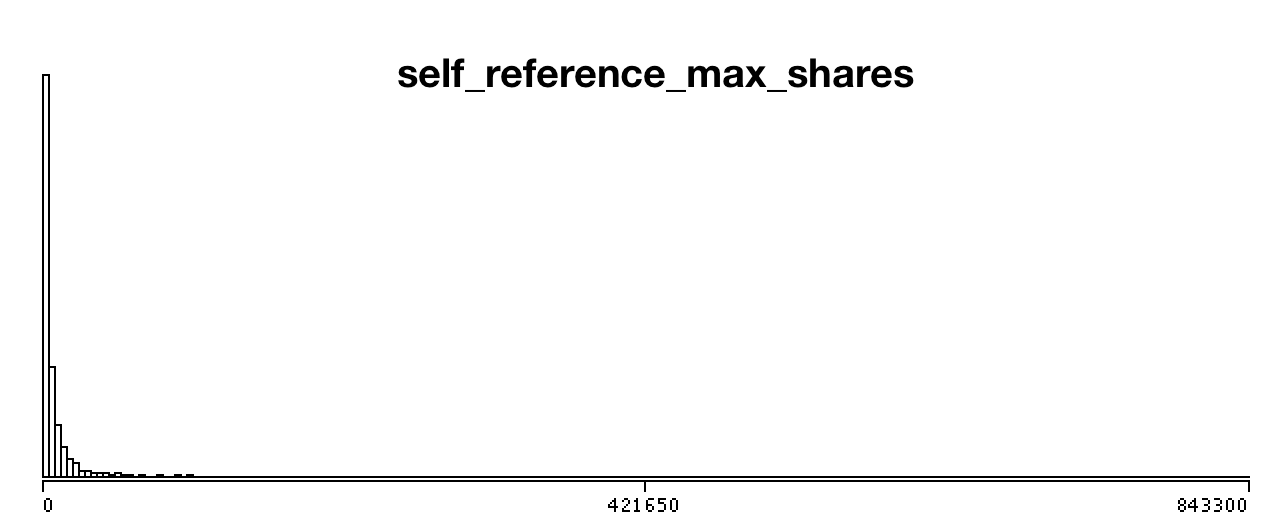
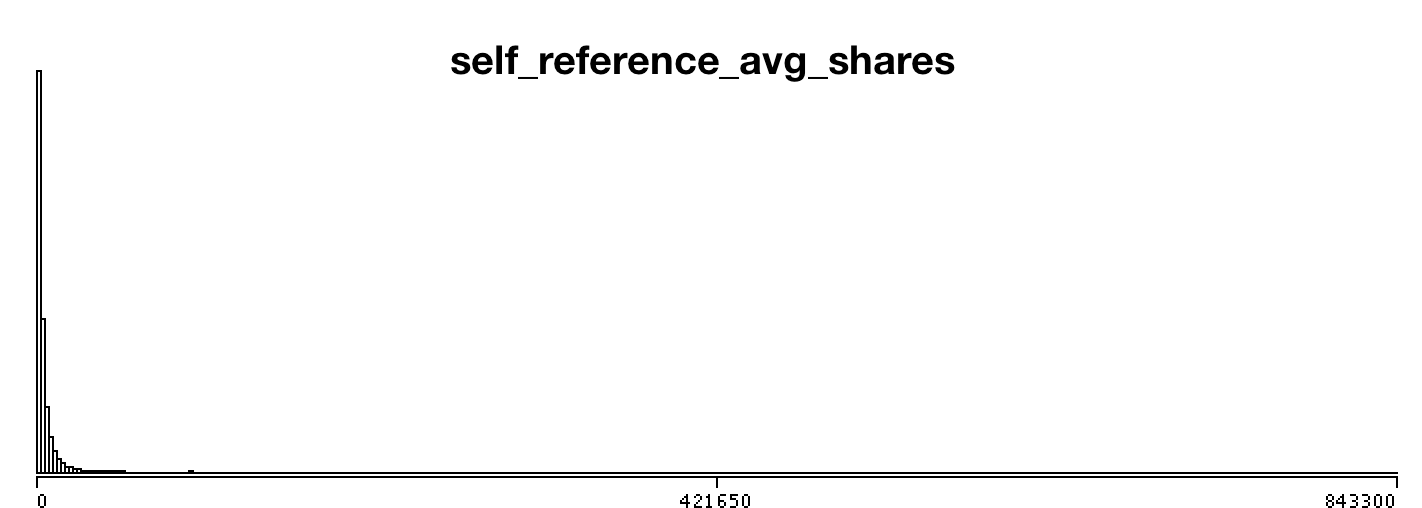
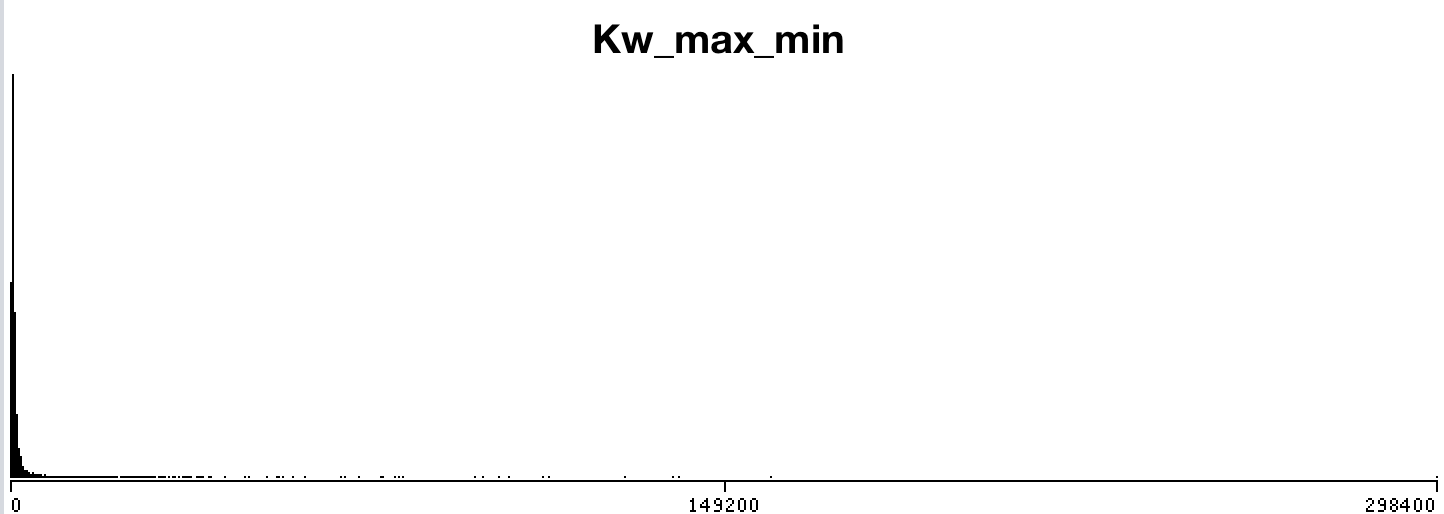
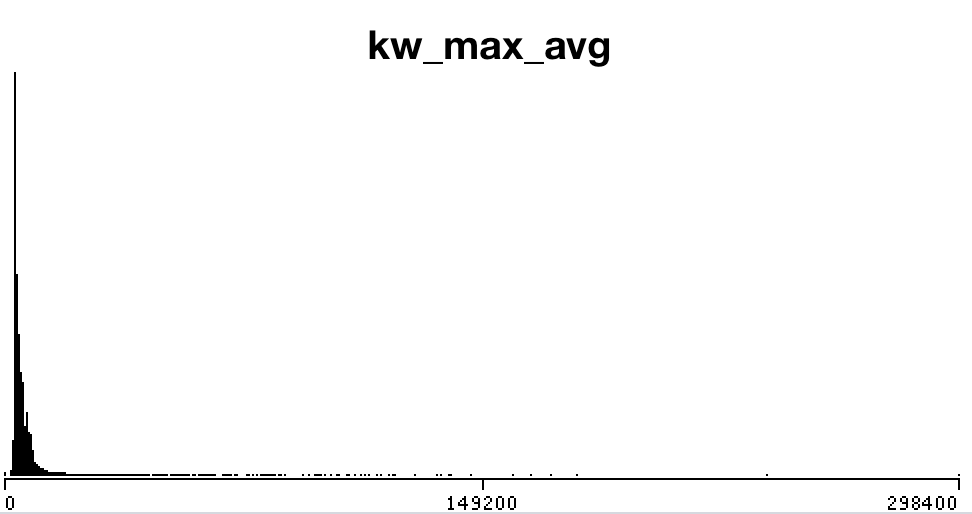


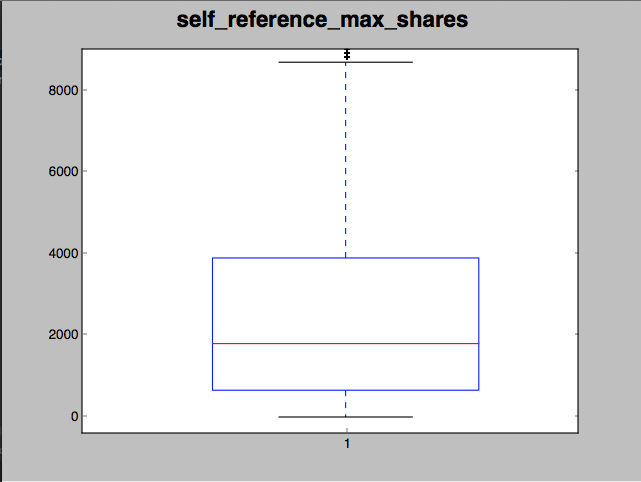
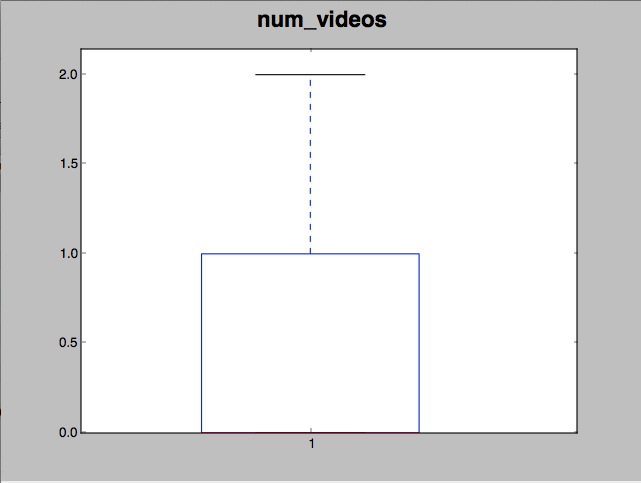
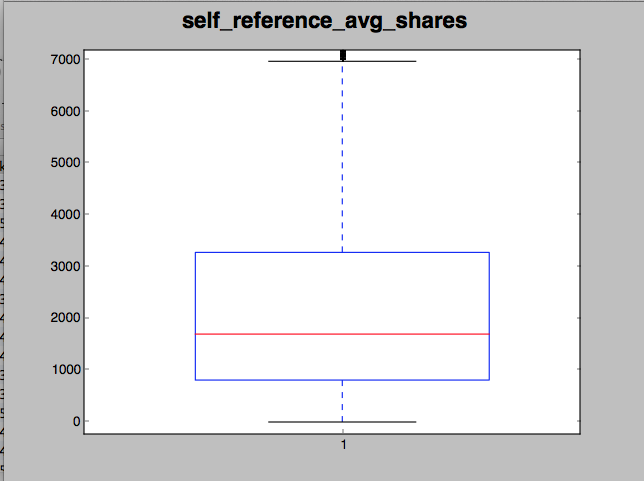
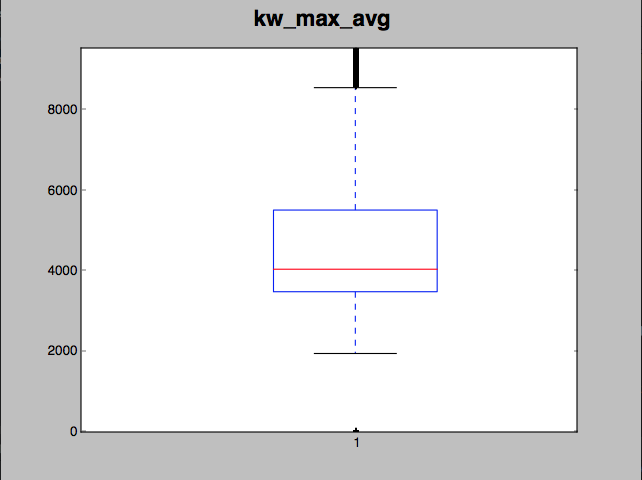
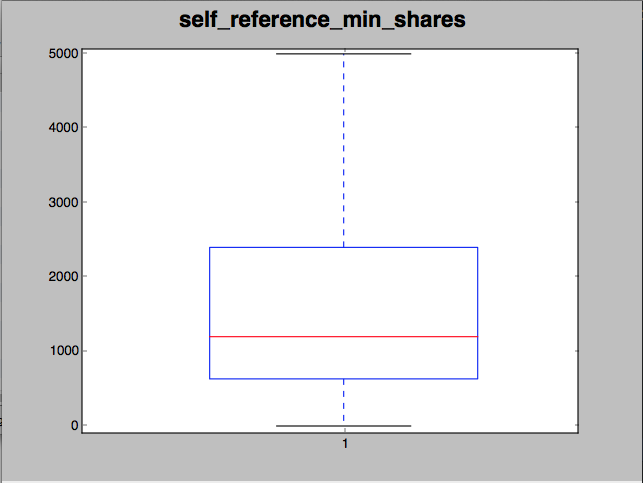
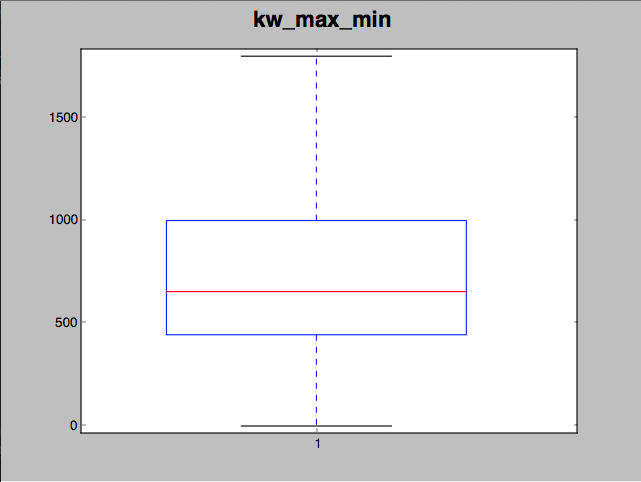
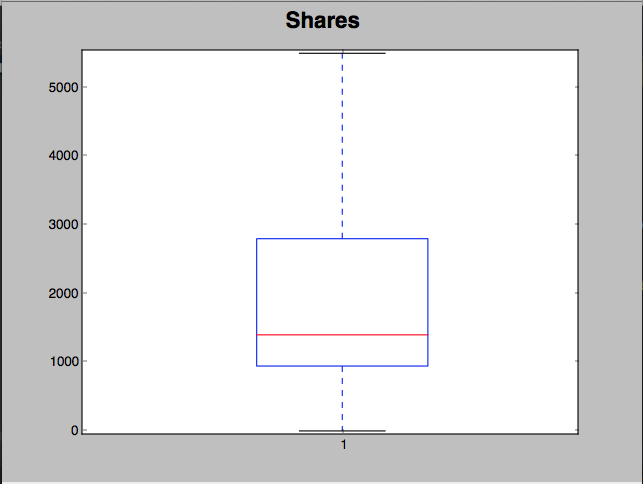
Fig: frequency Distribution histogram showing outlier observations

# 2. Data Pre-processing

2.1 Outlier Reduction

A **Boxplot** is a convenient way of graphically depicting groups of numerical data through their quartiles. Hence the aforementioned erroneous attributes were box plotted and observations containing outliers were removed.

The dataset observations were reduced from 39,644 to 21,105 after removing outliers using boxplotting. The boxplots are as follows:



After removing outliers frequency distribution histograms showed massive improvements as presented below:

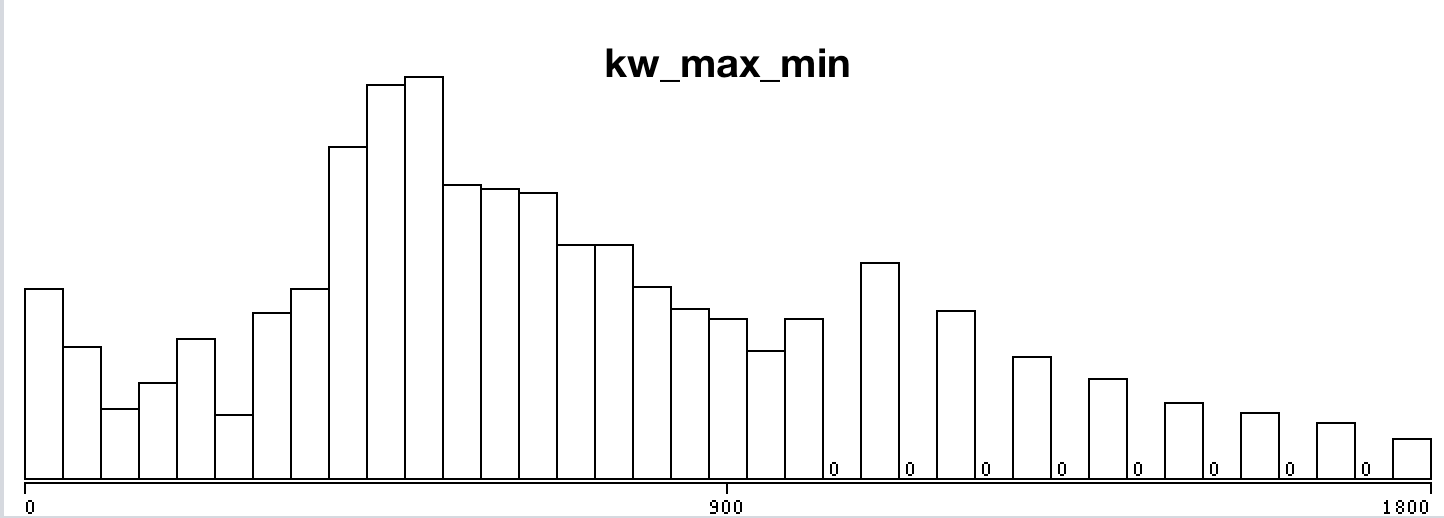
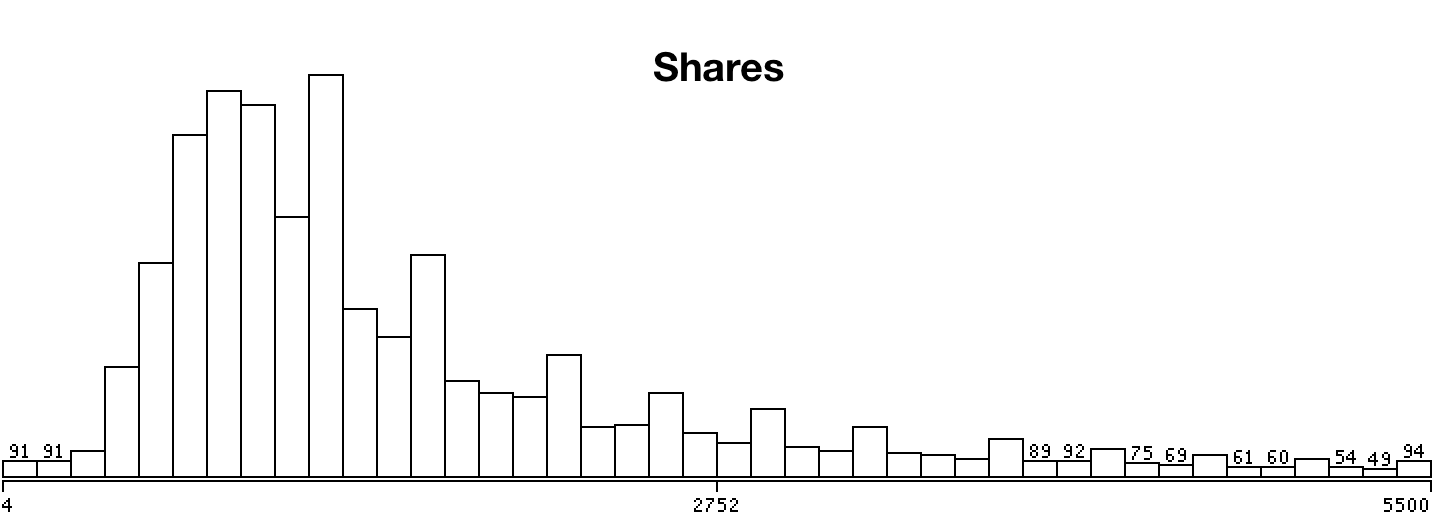
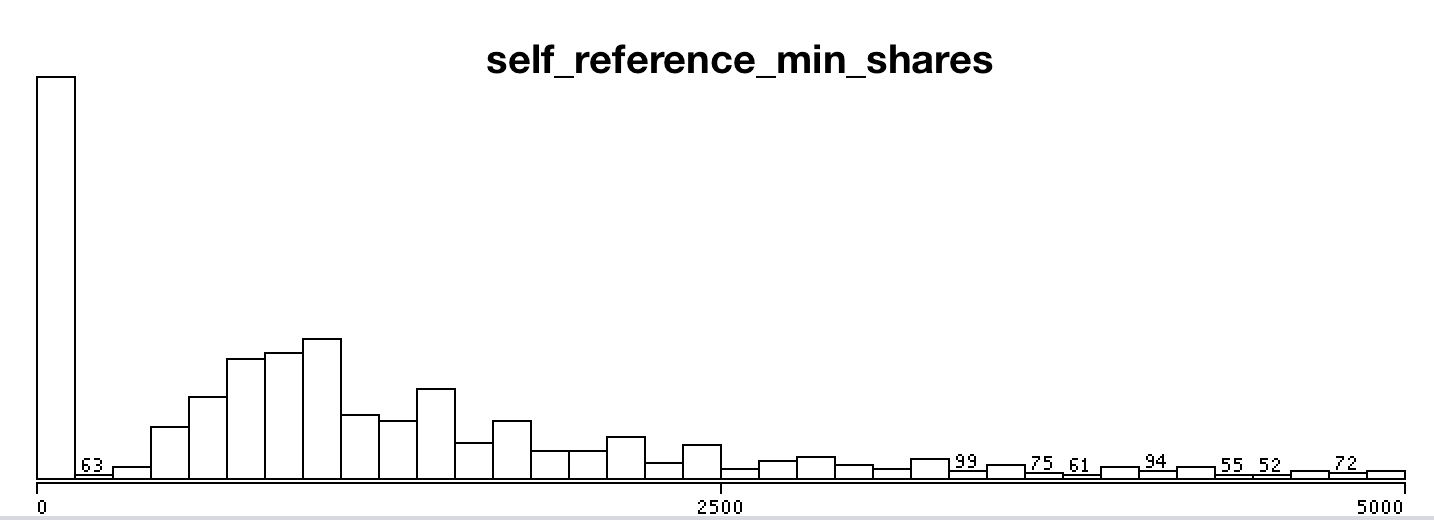
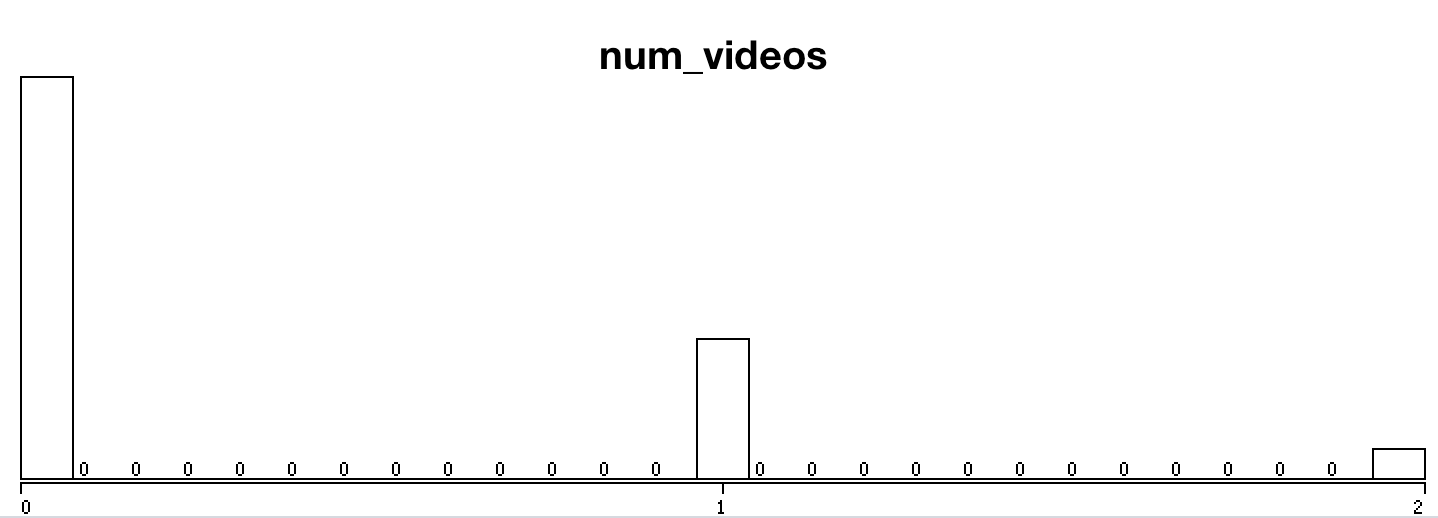
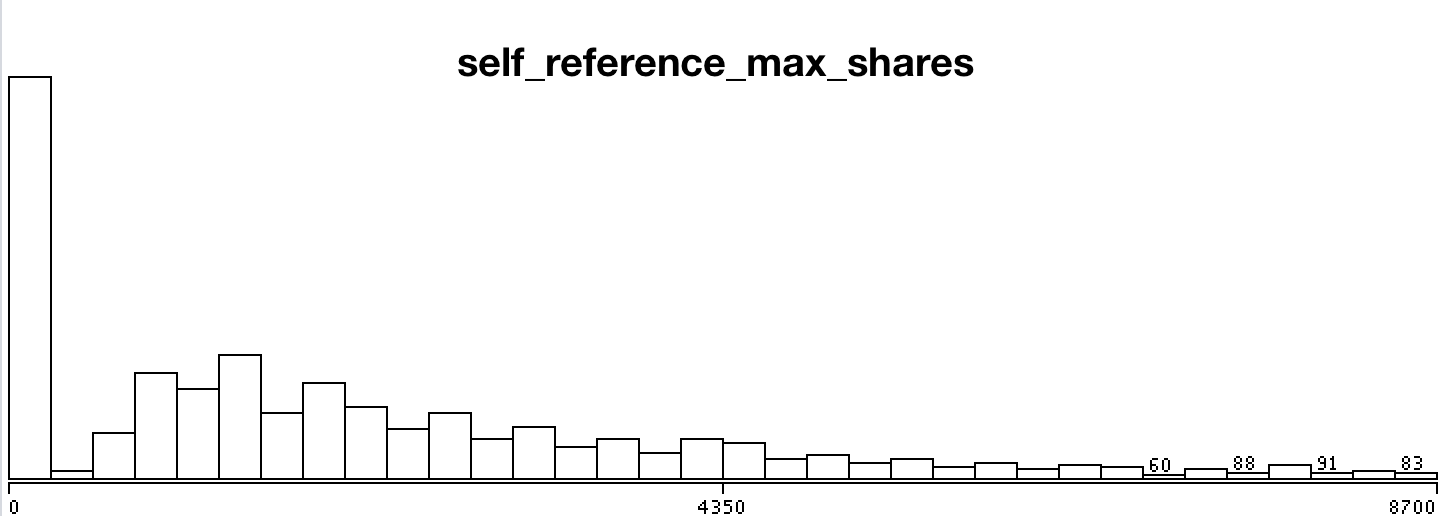
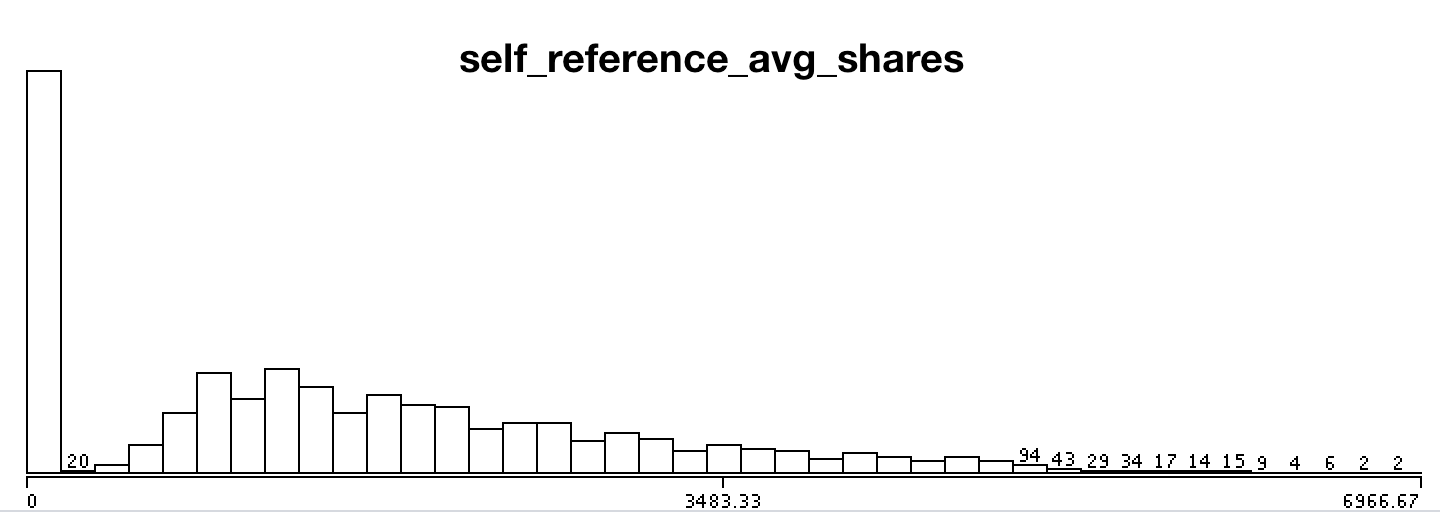
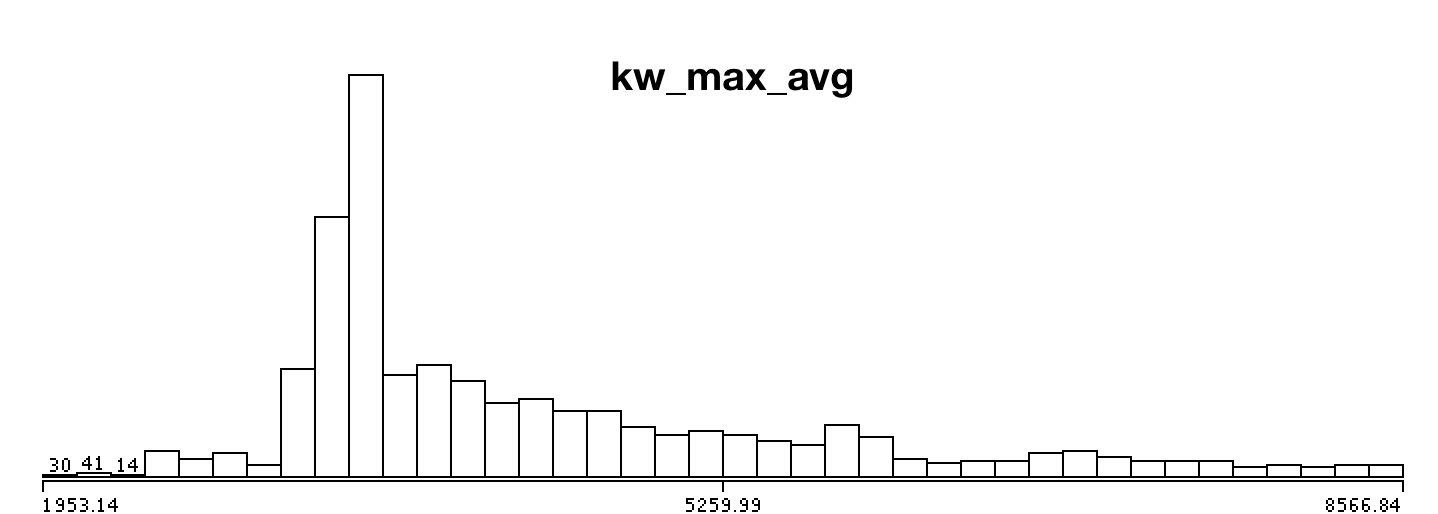


Fig: Frequency Distribution after Box Plot

2.2 Normalization

By looking at the data values, we observed that most of the attributes are greater in magnitude than the number of videos or images. When features differ by orders of magnitude, it is important to perform a feature scaling that can make gradient descent converge much more quickly.

The basic steps are:

* Subtract the mean value of each feature from the dataset.
* After subtracting the mean, additionally scale (divide) the feature values by their respective “standard deviations.”

The standard deviation is a way of measuring how much variation there is in the range of values of a particular feature (most data points will lie within ±2 standard deviations of the mean); this is an alternative to taking the range of values (max-min).

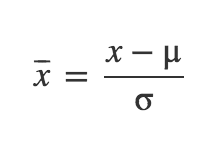
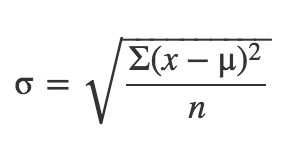
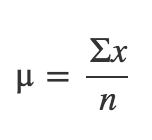


Fig: Mean, Standard Deviation and Normalization Formula

The Following python script was used to normalize and store the data values.

Normalize.py

import numpy as np

import csv

with open('dataset.csv', 'r') as f:

reader = csv.reader(f)

X = list(reader)

X = np.delete(X, (0), axis=0)

X = np.delete(X, (0), axis=1)

X = np.delete(X, (-1), axis=1)

X = np.array(X).astype(np.float)

def normalize(X):

#Normalize the values column-wise

mean\_r = []

std\_r = []

X\_norm = X

n\_c = X.shape[1]

for i in range(1, n\_c):

m = np.mean(X[:, i])

s = np.std(X[:, i])

mean\_r.append(m)

std\_r.append(s)

if s==0:

s=X[0, i]

if s == 0:

s = float("inf")

X\_norm[:, i] = (X\_norm[:, i] - m) / s

np.savetxt("normalizedData.csv", X\_norm, delimiter=",")

print "Saved"

#return X\_norm, mean\_r, std\_r

normalize(X)

## 3. Removal of Collinear Attributes

Collinearity is when attributes are highly correlated such that they represent the same predictor and is expressed in extreme high attribute correlations. These should be avoided at all costs because they make the prediction or classification unstable. Small changes may give very different predictors or classifiers more over eigenvalue and maximum likelihood techniques can't find solutions.

In this project weka classifier was used to remove collinear attributes. The squared coefficient of multiple correlation (Rmc^2) far all attributes that are still in the race   
is calculated and a threshold is set. The Attribute with the largest Rmc^2   
is compared to the threshold first and deselected when larger. Subsequently   
the Rmc^2 for the remaining attributes is adapted and so forth.

The following thirteen attributes were removed:

1 time\_delta

2 n\_tokens\_title

5 n\_non\_stop\_words

6 n\_non\_stop\_unique\_tokens

9 num\_imgs

12 num\_keywords

13 data\_channel\_is\_lifestyle

15 data\_channel\_is\_bus

18 data\_channel\_is\_world

20 kw\_avg\_min

22 kw\_max\_max

23 kw\_avg\_max

29 self\_refence\_avg\_sharess

31 weekday\_is\_tuesday

36 weekday\_is\_sunday

37 is\_weekend

41 lda\_03

46 global\_rate\_negative\_words

# New Chapter

Model Selection and Evaluation

# Multivariate Linear Regression

In multivariate Linear Regression, the aim is to predict the depend variable(number\_of\_shares) by determining set of parameters (theta). To determine theta, we employed gradient descent to find the point of minima in the hyper-dimensional space.

The python implementation is provided hereunder:

Code:

import matplotlib.pyplot as plt

import numpy as np

import csv

from sklearn.model\_selection import KFold

with open('normalizedDataWithShares.csv', 'r') as f:

reader = csv.reader(f)

X = list(reader)

X = np.asarray(X)

Y = X[:, -1] #last column (shares)

X = np.delete(X, (-1), axis=1) #delete last column from x

X = np.array(X).astype(np.float)

Y = np.array(Y).astype(np.float).reshape(-1, 1)

m, n = X.shape #Number of Instances, Attributes

num\_iters = 8000

alpha = 0.2

def compute\_cost(X, Y, theta):

# No of Training Samples

m = Y.size

predictions = X.dot(theta)

errors = (predictions - Y)

J = (1.0 / (2 \* m)) \* errors.T.dot(errors)

return J

def gradient\_descent(X, Y, alpha, num\_iters):

'''

Performs gradient descent to learn theta

by taking num\_items gradient steps with learning

rate alpha

'''

#Resetting Theta

theta = np.zeros(shape=(X.shape[1], 1))

m = Y.size

J\_history = np.zeros(shape=(num\_iters, 1))

for i in range(num\_iters):

predictions = X.dot(theta)

theta\_size = theta.size

for it in range(theta\_size):

temp = X[:, it]

temp.shape = (m, 1)

errors\_x1 = (predictions - Y) \* temp

theta[it][0] -= alpha \* (1.0 / m) \* errors\_x1.sum()

J\_history[i, 0] = compute\_cost(X, Y, theta)

# print i, J\_history[i, 0]

return theta, J\_history

def LinearCrossValidation(X,Y, k=10):

kf = KFold(n\_splits=k)

k=0

error\_mae = 0

error\_mrae =0

error\_pred =0

for train\_index, test\_index in kf.split(X):

k+=1

print "Fold", k

X\_train, X\_test = X[train\_index], X[test\_index]

Y\_train, Y\_test = Y[train\_index], Y[test\_index]

theta, J\_history = gradient\_descent(X\_train, Y\_train, alpha, num\_iters)

error\_mae\_fold = calculateMeanAbsoluteError(X\_test, Y\_test, theta)

error\_mrae\_fold = calculateMeanRelativeAbsoluteError(X\_test, Y\_test, theta)

error\_pred\_fold = calculatePred(X\_test, Y\_test, theta, 0.25)

error\_mae+=error\_mae\_fold

error\_mrae+=error\_mrae\_fold

error\_pred+=error\_pred\_fold

print "Average Mean Absolute Error %f" % (error\_mae / k)

print "Average Mean Relative Absolute Error%f" % (error\_mrae / k)

print "Average PRED 0.25 %f" % (error\_pred / k)

def calculateMeanAbsoluteError(X\_test,Y\_test, theta):

dot = X\_test.dot(theta)

error = abs(Y\_test - dot)

# plt.scatter(Y\_test, dot)

totalErr = error.sum()

# plt.show()

print "\nMAE%f" %(totalErr/error.shape[0])

return totalErr/error.shape[0]

def calculateMeanRelativeAbsoluteError(X\_test, Y\_test, theta):

dot = X\_test.dot(theta)

error = abs(Y\_test - dot)/Y\_test

totalErr = error.sum()

print "\nMRAE:%f" %(totalErr/error.shape[0])

return totalErr/error.shape[0]

def calculatePred(X\_test,Y\_test,theta,q=0.25):

dot = X\_test.dot(theta)

error = abs((Y\_test - dot)/Y\_test)

k=0

for i in range(0,error.shape[0]):

if(error[i][0]<=q):

k=k+1

predErr = float(k)/float(error.shape[0])

print "PRED ",q, " ", predErr

return predErr

LinearCrossValidation(X, Y)

Results:

For gradient descent the parameters learning rate(α) and the number of iterations were chosen as follows:

* α = 0.2
* number of iterations = 8000

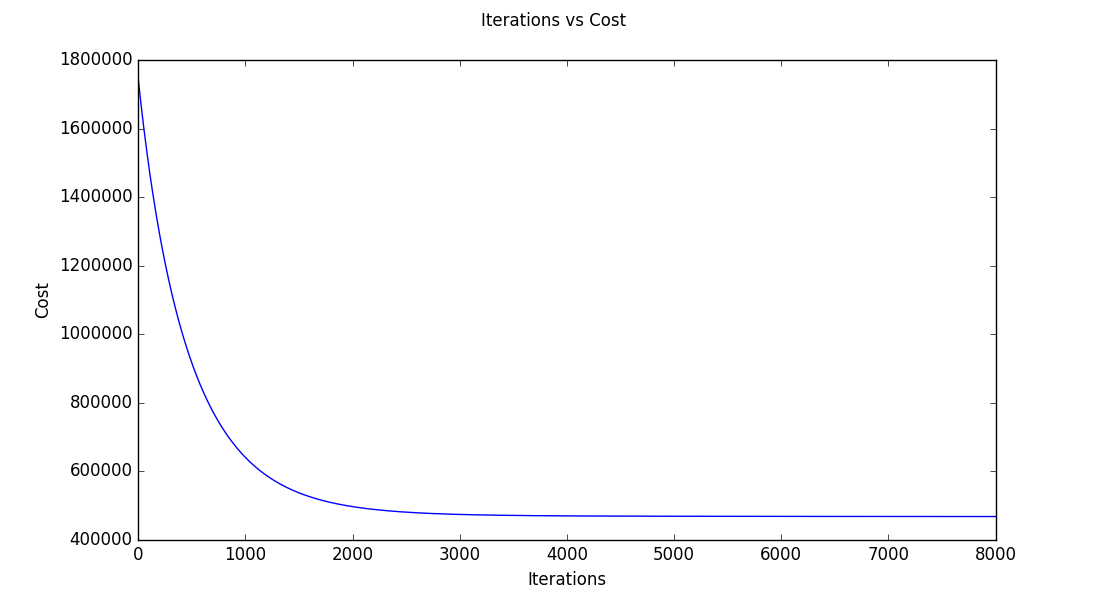


Fig: Minimization of Cost Function in Gradient Descent with Feature Selection

1. Without Feature Selection

Mean Absolute Error 712.830325

Mean Relative Absolute Error 0.681046

PRED(0.25) 0.311377

1. With feature Selection

Mean Absolute Error 722.328057

Mean Relative Absolute Error 0.704558

PRED(0.25) 0.306595

The model achieved an accuracy of:

* 31.89% - without feature selection
* 29.54% - with feature selection

Hence the performance of model degraded slightly after application of feature selection. Hence feature selection was not suitable for linear regression on the given dataset.

# Logistic Regression

We used logistic regression (classification model) to improve our accuracy further and classified the instances into two categories “popular” and “unpopular” based on the median of dependent variable (number of shares).

The python implementation is provided below:

Code:

import numpy as np

import csv

from sklearn.model\_selection import KFold

import matplotlib.pyplot as plt

with open('normalizedDataWithShares.csv', 'r') as f:

reader = csv.reader(f)

X = list(reader)

X = np.asarray(X)

Y = X[:, -1] #last column (shares)

X = np.delete(X, (-1), axis=1) #delete last column from x

X = np.array(X).astype(np.float)

Y = np.array(Y).astype(np.float).reshape(-1, 1)

m, n = X.shape #Number of Instances, Attributes

num\_iters = 4000

alpha = 0.003

def hypothesis(X, theta):

predictions = X.dot(theta)

return 1/(1+np.e\*\*(-1\*predictions))

def cost\_function(X, Y, theta):

predictions = hypothesis(X, theta)

# No of Training Samples

m = Y.size

cost = (Y).T.dot(np.log(predictions)) + (1-Y).T.dot(np.log(1-predictions))

J = (1.0 / m) \* cost

return J

def gradient\_descent(X, Y, alpha, num\_iters):

'''

Performs gradient descent to learn theta

by taking num\_items gradient steps with learning

rate alpha

'''

#Resetting Theta

theta = np.zeros(shape=(X.shape[1], 1))

m = Y.size

J\_history = np.zeros(shape=(num\_iters, 1))

for i in range(num\_iters):

predictions = hypothesis(X, theta)

theta\_size = theta.size

for it in range(theta\_size):

temp = X[:, it]

temp.shape = (m, 1)

errors\_x1 = (predictions - Y) \* temp

theta[it][0] -= alpha \* (1.0 / m) \* errors\_x1.sum()

J\_history[i, 0] = cost\_function(X, Y, theta)

plt.xlabel("Iterations")

plt.ylabel("Cost")

plt.suptitle("Iterations vs Cost")

plt.plot(J\_history)

plt.show()

return theta, J\_history

def LogisticCrossValidation(X,Y, k=10):

# Cross Validation Coeff

# Constants for confusion MAtrix

true\_p = 0

true\_n = 0

false\_p = 0

false\_n = 0

kf = KFold(n\_splits=k)

mae = 0

fold = 1

for train\_index, test\_index in kf.split(X):

print("\n\nFold %f" %fold)

fold +=1

X\_train, X\_test = X[train\_index], X[test\_index]

Y\_train, Y\_test = Y[train\_index], Y[test\_index]

theta, J\_history = gradient\_descent(X\_train, Y\_train, alpha, num\_iters)

tp, fp, fn, tn = calculateConfusionMatrix(X\_test, Y\_test, theta)

mae\_fold = calculateMeanAbsoluteError(X\_test.dot(theta), Y\_test)

true\_p+=tp

true\_n+=tn

false\_p+=fp

false\_n+=fn

mae +=mae\_fold

print "Average Confusion Matrix"

print true\_p, false\_p

print false\_n, true\_n

displayResultSummary(true\_p, false\_p, true\_n, false\_n, mae/k)

def displayResultSummary(tp, fp, tn, fn, mae):

total = tp+fp+tn+fn

print "Correctly Classified Instances:", (tp+tn),"\t",((float(tp+tn)/total)\*100), "%"

print "Incorrectly Classified Instances:", (fp + fn), "\t", ((float(fp+fn) / total) \* 100), "%"

print "Mean Absolute Error:\t",mae

print "TP Rate/Recall", tp/float(tp+fn)

print "FP Rate", fp/float(fp+tn)

print "Precision", tp/float(tp+fp)

def calculateMeanAbsoluteError(Y\_proba,Y\_test):

error = abs(Y\_test - Y\_proba)

# plt.scatter(Y\_test, dot)

totalErr = error.sum()

# plt.show()

print "\nMean Absolute Error%f" %(totalErr/error.shape[0])

return totalErr/error.shape[0]

def calculateConfusionMatrix(X\_test,Y\_test, theta):

true\_positive = 0

true\_negative = 0

false\_positive = 0

false\_negative = 0

dot = X\_test.dot(theta)

for i in range(0, dot.shape[0]):

if(dot[i][0]>=0.5):

if(Y\_test[i][0]>=0.5):

true\_positive+=1

else:

false\_positive+=1

else:

if(Y\_test[i][0]>=0.5):

false\_negative +=1

else:

true\_negative+=1

print "Confusion Matrix"

print true\_positive, false\_positive

print false\_negative, true\_negative

print "Correct: ", (float)(true\_negative + true\_positive) / (float)(false\_negative + true\_negative + true\_positive + false\_positive)

return true\_positive, false\_positive, false\_negative, true\_negative

LogisticCrossValidation(X, Y)

Results:

For gradient descent the parameters learning rate(α) and the number of iterations were chosen as follows:

* α = 0.003
* number of iterations = 4000

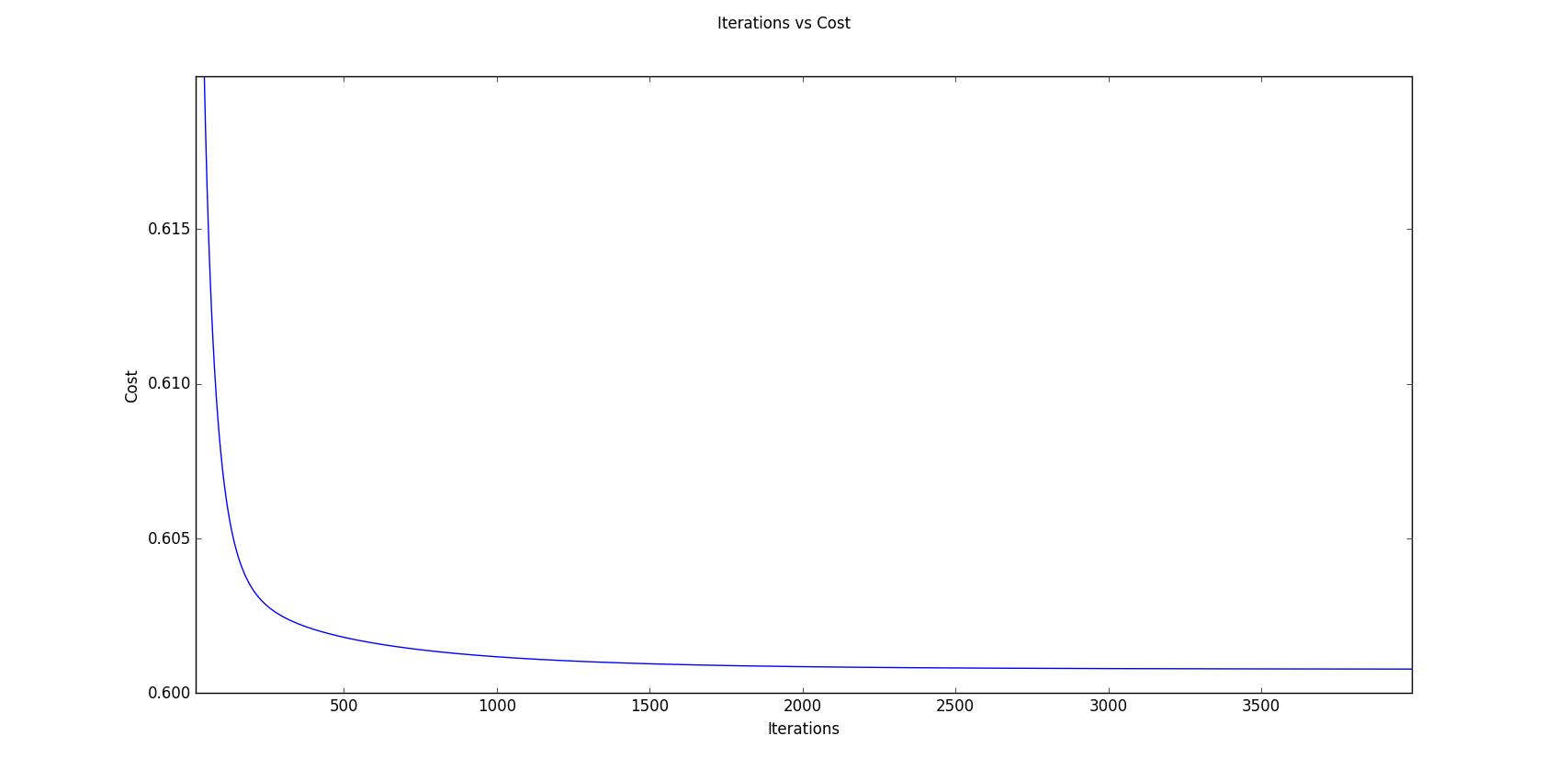


Fig: Minimization of Cost Function in Gradient Descent without Feature Selection

The **receiver operating characteristic** (**ROC**), or **ROC curve**  graphical plot illustrates the performance of the  logistic binary classifier system as its discrimination threshold is varied. The ROC curves for data with and without feature selection is shown below:

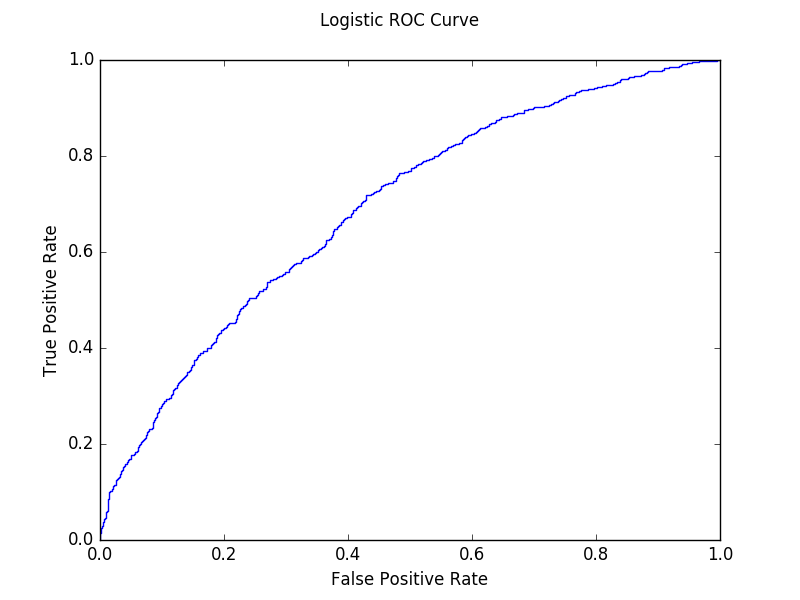


Fig: ROC Curve for Logistic Classifier without Feature Selection

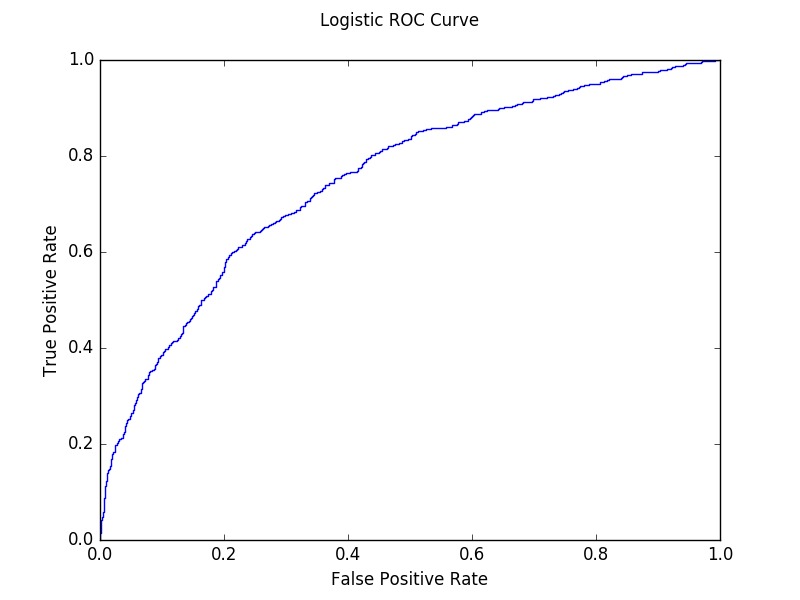


Fig: ROC Curve for Logistic Classifier with Feature Selection

1. Without Feature Selection

Mean Absolute Error 0.615069

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Actual Values | |
| 1 | 0 |
| Classified  as | 1 | 1616 | 732 |
| 0 | 6579 | 12176 |

Confusion Matrix

Correctly Classified Instances: 13792 **65.3556366393 %**

Incorrectly Classified Instances: 7311 34.6443633607 %

Mean Absolute Error: 0.837596492743

TP Rate/Recall: 0.197193410616

FP Rate: 0.0567090176635

Precision: 0.688245315162

AUC: 0.697801976868

1. With feature Selection:

Mean Absolute Error 0.909651

Average Confusion Matrix

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Actual Values | |
| 1 | 0 |
| Classified  as | 1 | 1139 | 708 |
| 0 | 6274 | 12982 |

Correctly Classified Instances: 14121 **66.9146566839** %

Incorrectly Classified Instances: 6982 33.0853433161 %

Mean Absolute Error: 0.873365774302

TP Rate/Recall 0.153648995009

FP Rate 0.0517165814463

Precision 0.616675690309

AUC 0.753054188435

The model achieved an accuracy of:

* 65.35% - without feature selection
* 66.91% - with feature selection

The accuracy of this model displayed significant improvement as compared to the Linear Model. Also the performance of the model increased slightly after application of feature selection. Hence feature selection was suitable for binary logistic regression on the given dataset.

# Support Vector Classification

We used Support vector machine (**SVM**), a supervised learning model with associated learning algorithms to analyze data and classify the instances into two categories “popular” and “unpopular” based on the median of dependent variable (number of shares).

The python implementation is provided below:

Code:

import matplotlib.pyplot as plt

import numpy as np

import csv

from sklearn import svm

from sklearn.model\_selection import KFold

# open the csv file for reading data

# X: stores the whole csv file

with open('normalizedDataWithSharesReducedAttributes.csv', 'r') as f:

reader = csv.reader(f)

X = list(reader)

# dividing X into X and Y where:

# X: all the attributes in normailized form

# Y: predicted shares in 0 and 1 form

X = np.asarray(X)

Y = X[:, -1] # last column (shares)

X = np.delete(X, (-1), axis=1) # delete last column from x

X = np.array(X).astype(np.float) # convert all values to float

Y = np.array(Y).astype(np.float).reshape(-1, 1)

# m: Number of Instances

# n: Number of Attributes

m, n = X.shape

# Cross Validaton code that contains

# InputParameters:

# X: attribute normalized variables

# Y: shares

# k: folds in cross validation by default 10

# Cost: Cost for SVC classifier

# Return: None

def SVCCrossValidation(X, Y,Cost=1.0, k=10):

# Constants for confusion MAtrix

true\_p = 0

true\_n = 0

false\_p = 0

false\_n = 0

mae = 0

# KFold library function(Sklearn) for calculating train and test indices

kf = KFold(n\_splits=k)

fold = 1

for train\_index, test\_index in kf.split(X):

print("\n\nFold %f" % fold)

fold += 1

# X\_train: train attributes data matrix

# Y\_train: train shares matrx

# X\_test: test attributes data matrix

# Y\_test: test shares data matrix

X\_train, X\_test = X[train\_index], X[test\_index]

Y\_train, Y\_test = Y[train\_index], Y[test\_index]

Y\_train = np.array(Y\_train)

print Y\_train.shape

# SVM classifier with paramters

# Kernel:Rbf

clf = svm.SVC(probability=True, C=Cost)

clf.fit(X\_train, Y\_train)

# Y\_calc as the predicted Y for the test Data by the SVC Classifier

Y\_calc = clf.predict(X\_test)

Y\_proba = clf.predict\_proba(X\_test)

Y\_proba = np.delete(Y\_proba, (-1), axis=1)

print Y\_proba.shape

# Confusion Matrix Calculation

tp, fp, fn, tn = calculateConfusionMatrix(Y\_calc, Y\_test)

mae\_fold=calculateMeanAbsoluteError(Y\_proba, Y\_test)

mae+=mae\_fold

true\_p += tp

true\_n += tn

false\_p += fp

false\_n += fn

print "Average Confusion Matrix"

print true\_p, false\_p

print false\_n, true\_n

displayResultSummary(true\_p, false\_p, true\_n, false\_n, mae/k)

def displayResultSummary(tp, fp, tn, fn, mae):

total = tp+fp+tn+fn

print "Correctly Classified Instances:", (tp+tn),"\t",((float(tp+tn)/total)\*100), "%"

print "Incorrectly Classified Instances:", (fp + fn), "\t", ((float(fp+fn) / total) \* 100), "%"

print "Mean Absolute Error:\t",mae

print "TP Rate/Recall", tp/float(tp+fn)

print "FP Rate", fp/float(fp+tn)

print "Precision", tp/float(tp+fp)

print "Recall"

def calculateMeanAbsoluteError(Y\_proba,Y\_test):

error = abs(Y\_test - Y\_proba)

# plt.scatter(Y\_test, dot)

totalErr = error.sum()

# plt.show()

print "\nMean Absolute Error%f" %(totalErr/error.shape[0])

return totalErr/error.shape[0]

# Calculation of confusion matrix

# Input paramters:

# Y\_calc: Calculated value of share group(0/1)

# Y\_test: Actual Value of test group(0/1)

# Output Parameters:

# confusion matrix Paramters

def calculateConfusionMatrix(Y\_calc, Y\_test):

true\_positive = 0

true\_negative = 0

false\_positive = 0

false\_negative = 0

for i in range(0, Y\_calc.shape[0]):

if (Y\_calc[i] >= 0.5):

if (Y\_test[i][0] >= 0.5):

true\_positive += 1

else:

false\_positive += 1

else:

if (Y\_test[i][0] >= 0.5):

false\_negative += 1

else:

true\_negative += 1

print "Confusion Matrix"

print true\_positive, false\_positive

print false\_negative, true\_negative

print "Correct: ", (float)(true\_negative + true\_positive) / (float)(

false\_negative + true\_negative + true\_positive + false\_positive)

return true\_positive, false\_positive, false\_negative, true\_negative

SVCCrossValidation(X, Y)

Results:

The **receiver operating characteristic** (**ROC**), or **ROC curve** graphical plot illustrates the performance of the  SVM binary classifier system as its discrimination threshold is varied. The ROC curves for data with and without feature selection is shown below:

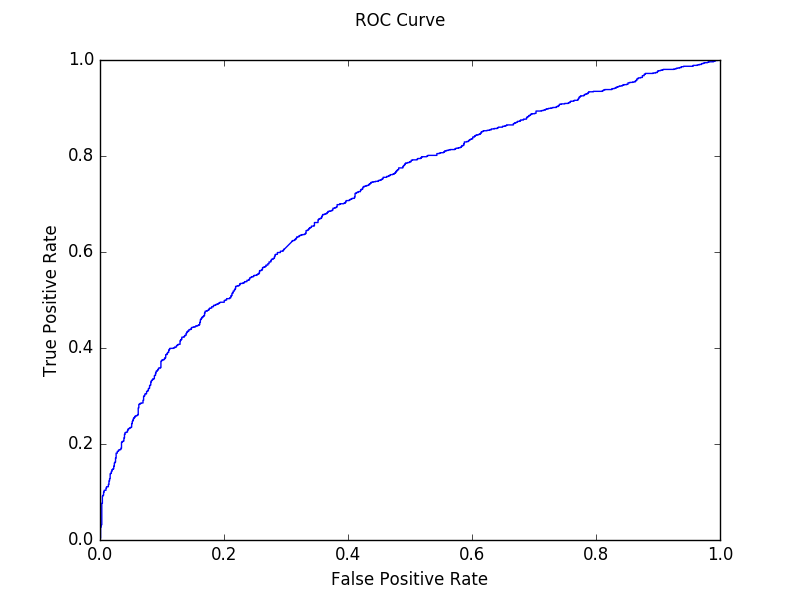


Fig: ROC Curve for SVM Classifier with Feature Selection

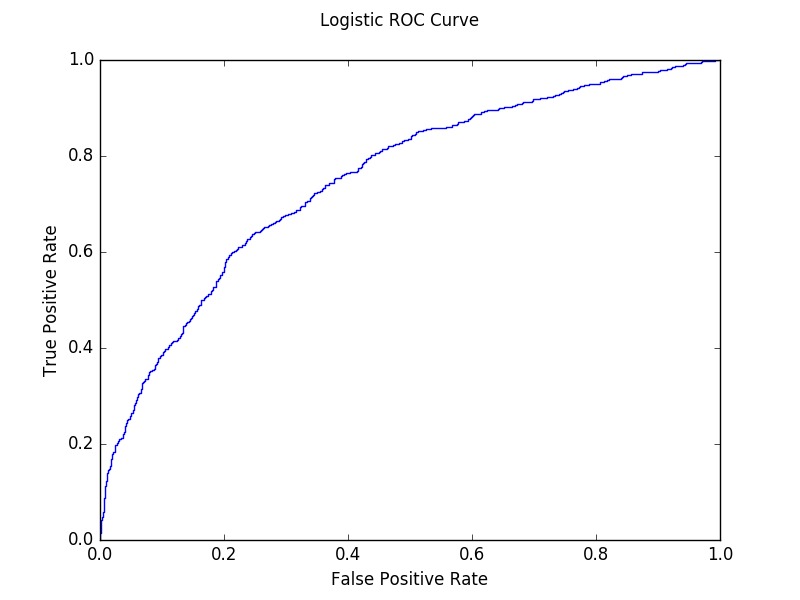


Fig: ROC Curve for SVM Classifier without Feature Selection

1. Without Reduced Attributes

Mean Absolute Error 0.568637

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Actual Values | |
| 1 | 0 |
| Classified  as | 1 | 3071 | 1751 |
| 0 | 5124 | 11157 |

Average Confusion Matrix

Correctly Classified Instances 14228 **67.4216935981 %**

Incorrectly Classified Instances 6875 32.5783064019 %

Mean Absolute Error 0.583096599995

TP Rate/Recall 0.374740695546

FP Rate 0.135652308646

Precision 0.636872666943

AUC 0.694588372063

1. With Reduced Attributes

Mean Absolute Error 0.601971

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Actual Values | |
| 1 | 0 |
| Classified  as | 1 | 2133 | 1198 |
| 0 | 5280 | 12492 |

Average Confusion Matrix

Correctly Classified Instances: 14625 **69.3029427096 %**

Incorrectly Classified Instances: 6478 30.6970572904 %

Mean Absolute Error: 0.593544224221

TP Rate/Recall 0.287737757993

FP Rate 0.0875091307524

Precision 0.640348243771

AUC 0.716094383573

The model achieved an accuracy of:

* 67.42% - without feature selection
* 69.30% - with feature selection

The accuracy of this model displayed improvement as compared to the other models. Also the performance of the model increased slightly after application of feature selection. Hence feature selection was suitable for Support Vector Classifier on the given dataset.