An In-depth analysis Of Logistic Regression and Feature Selection

- **Sridhar Natarajan**

Introduction

Binary classification is the most basic task in machine learning, and yet the most frequent. Binary classi_ers often serve as the foundation for many high tech ML applications such as ad placement, feed ranking, spam _ltering, and recommendation systems. Logistic regression predicts a confidence probability which when combined with a decision rule can be used to classify observations into classes. This paper is primarily divided into three parts:

1) Theoretical foundations of logistic regression gradient rule, a motivated implementation of gradient descent
2) Emperical analysis of performance using plotting learning curves, loss functions on two different implementations
3) Improved generalization discussions using feature reduction methods. The generalization methods discussed here include comparison of feature transformation methods PCA (Principal component analysis) and ICA (Independent component Analysis.We also understand the role of unsupervised learning methods in pre-assessing effectiveness of Feature transformation techniques.

1.1 Derivation of Logistic Regression Gradient Descent

Step1: Terminology:

x (i) *refers to the ith row in the given training data set (with the exception of class label).*
y(i) *refers to -1 or +1 label in the ith row in the given data set*
theta *-refers to the parameter vector of the classifier which has same dimensions as x.*
Computing

Step 2: Formulate an optimization problem:

Exploring theta variations is the equivalent to modeling the decision boundary of the classifier since they are orthogonal vectors. The idea behind logistic regression gradient descent is to use the Minimize parameter **Theta** in the maximum likelihood expression (MLE):

Sigma (1 to n) {log (1+ exp(y(i)*dotproduct(theta*x(i))))}

We explore the value of theta using gradient descent:

1) Initialize the value of theta to random values
2) For each dimension of theta use the below rule to performa an adjustment

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial \sum_{i=1}^{n} \log\left(1+\exp(y^{(i)}\langle\theta, x^{(i)}\rangle)\right)}{\partial \theta_j}$$

Alpha in the above expression is a learning rate that indicates the degree to which inertia of previous values of theta are respected for controlled exploration

Step 3 Derivation of Gradient Descent Rule

As one can observe in the above equation, we need to take the partial derivative with respect to theta (j). Below let's perform the below substitution to simplify

Exp (y(i)*dotproduct(theta*x(i))) can be rewritten as *exp(K*thetaj+b)* --------(1)

- ***K is the coefficient of jth component of theta.***
- ***thetaj is the jth dimension of the theta vector.***
- ***This coefficient is* (jth component of x(i))*y(i)**

Let's represent the ith component in the summation to understand obtaining the differential.

Derivative (Log (1+exp (K*thetaj+b)))

- ➔ ***(1/(1+exp(K*thetaj+b)))*Derivative(1+exp(K*theta(j+b))***
- ➔ ***(1/(1+exp(K*thetaj+b)))*Derivative(exp(K*thetaj+b))***
- ➔ ***(1/(1+exp(K*thetaj+b)))*exp(K*thetaj+b)*Derivative(K*thetaj+b)***
- ➔ ***(1/(1+exp(K*thetaj+b)))*exp(K*thetaj+b)*K***
- ➔ ***(1/(1+* exp(y(i)*dotproduct(theta*x(i))))* exp(y(i)*dotproduct(theta*x(i)))*K**
- ➔ ***(1/(1+* exp(y(i)*dotproduct(theta*x(i))))* exp(y(i)*dotproduct(theta*x(i)))* (jth component of x(i))*y(i)**
- ➔ ***Divided the numerator and denominator in the above equation by* exp(y(i)*dotproduct(theta*x(i)))) and used this result in my gradient descent implementations----------(2)**
- ➔ ***Let's call the above valuethe ith term in the summation series that we call thetaj-delta***

While Theta converges:

For each dimension j in theta

Thetaj<-thetaj-alpha*thetaj-delta

The idea behind alpha (learning rate) is to weighting inertia while performing an adjustment to theta component. The algorithm converges to optimal values of theta. The above pseudo code is further reinforced in the comments section for Q2 where the above expression is leveraged in the code implementation.

Performance of Gradient Descent

Time Complexity

There are a total of d dimensions. For each dimension the theta update will involve (nxd) operations due to the dot product. Hence the number of operations expressed in terms of n and d per gradient descent iteration will be **nxdxd.** The computational cost of optimization problems are typically expressed as cost per iteration along with a hypothesis convergence. Since we are using the MLE to formulate our gradient descent rule, the algorithm is guaranteed to converge without requiring random restarts

Space complexity:

Each iteration of logistic regression is expensive in space owing to requiring the entire dataset in vectorized data structures. This can be impossible or slow performance down when there are millions of training records. Hence, to scale we use stochastic gradient descent which computes gradient using reduced points that are representative of the distribution being attempted to learn which makes up for a scalable implementation of gradient descent.

1.2. Implementation

In this section we implement logistic regression gradient descent using R by formalizing the pseudo code discussed above. We also write R code to implement logistic regression using the in-built GLM package. Now, we compare the performance of both implementations and discuss performance parity

Part1: Custom Implementation of Gradient Descent Code

The below code computes classification accuracy on my own implementation of gradient descent using the rule derived in the previous section

```
# The gradient descent code takes in as input a preprocessed dataset and # #outputs theta value
gradient_desc<-function(bc)
{
#Convert To Numeric
xvec<-scale(as.matrix(sapply(bc[,1:9],as.numeric)))
yvec<-bc$Class
#Initialize Random Theta
thetavec<-runif(9)
newtheta<-cbind(11:19)
oldtheta<-cbind(1:9)
thetavec<-runif(9)
i1<-0
# Initialize Learning Rate
alpha<-0.1
# that's our convergence criteria to check if thetas no longer changing
while((sum((oldtheta-newtheta)^2)^0.5)>0.000001)
{
#Decay alpha- somewhat unsure of its impact though
alpha<-alpha*0.999
i1<-i1+1
oldtheta<-thetavec
j<-1
{
#Equation(2) from Q1 in following 3 lines
numer<-(xvec[,j]*yvec)
denom<-(exp((xvec%*%thetavec)*(yvec)*(-1))+1)
thetavec[j]<-thetavec[j]-(alpha*sum(numer/denom))
j<-j+1
# Move on to next dimension for the next 8 times to avoid looping .This repetition actually saves ton of performance though awakward looking
numer<-(xvec[,j]*yvec)
denom<-(exp((xvec%*%thetavec)*(yvec)*(-1))+1)
thetavec[j]<-thetavec[j]-(alpha*sum(numer/denom))
j<-j+1
numer<-(xvec[,j]*yvec)
```

```
denom<-(exp((xvec%*%thetavec)*(yvec)*(-1))+1)
thetavec[j]<-thetavec[j]-(alpha*sum(numer/denom))
j<-j+1
numer<-(xvec[,j]*yvec)
denom<-(exp((xvec%*%thetavec)*(yvec)*(-1))+1)
thetavec[j]<-thetavec[j]-(alpha*sum(numer/denom))
j<-j+1
numer<-(xvec[,j]*yvec)
denom<-(exp((xvec%*%thetavec)*(yvec)*(-1))+1)
thetavec[j]<-thetavec[j]-(alpha*sum(numer/denom))
j<-j+1
numer<-(xvec[,j]*yvec)
denom<-(exp((xvec%*%thetavec)*(yvec)*(-1))+1)
thetavec[j]<-thetavec[j]-(alpha*sum(numer/denom))
j<-j+1

numer<-(xvec[,j]*yvec)
denom<-(exp((xvec%*%thetavec)*(yvec)*(-1))+1)
thetavec[j]<-thetavec[j]-(alpha*sum(numer/denom))
j<-j+1

numer<-(xvec[,j]*yvec)
denom<-(exp((xvec%*%thetavec)*(yvec)*(-1))+1)
thetavec[j]<-thetavec[j]-(alpha*sum(numer/denom))
j<-j+1

numer<-(xvec[,j]*yvec)
denom<-(exp((xvec%*%thetavec)*(yvec)*(-1))+1)
thetavec[j]<-thetavec[j]-(alpha*sum(numer/denom))
j<-j+1

}
newtheta<-thetavec
}
return(newtheta)
}

#The method invokes gradientDescent code by taking as input the BreastCancer dataset and a random start point
cv_TrainControl<-function(BreastCancer,startx)
{
bc<-BreastCancer
#Remove Id column
bc<-bc[,-1]
#remove na
bc<-na.omit(bc)
bc$Class<-gsub("malignant",1,bc$Class)
bc$Class<-gsub("benign",-1,bc$Class)
bc$Class<-as.numeric(bc$Class)
#partition into 70% train and 30% test
cv_train1<-bc[1:startx,]
cv_test<-bc[(startx+1):(startx+205),]
if((startx+205)<683)
{
cv_train2<-bc[(startx+206):683,]
cv_train<-rbind(cv_train1,cv_train2)
}
new_theta<-gradient_desc(cv_train)
test_cv<-(sapply(bc[(startx+1):(startx+205),],as.numeric))
test_cv<-scale(test_cv[,-10])
total_right<-( sum( (test_cv%*%new_theta>0)==(cv_test$Class<0) ))
```

```
#return classification accuracy percentage
return ((total_right/205)*100)
}
```

Testing custom gradient descent implementation:

```
startpos<-sample(1:400,10)
accuracy_vec<-c()
for (i in 1:8)
 {
accuracy_vec<-append(accuracy_vec,(cv_TrainControl(BreastCancer,startpos[i])))
 }
Print (mean(accuracy_vec))
```

Custom Gradient Descent Results

94.81707 --- Mean accuracy of 70,30 split on test data using my own implementation

Change of accuracy with change of starting positions

```
> accuracy_vec
[1] 94.14634 95.12195 96.58537 95.12195 94.14634 94.63415 93.65854 95.12195
> mean(accuracy_vec)
[1] 94.81707
> startpos
 [1]   5 185 370 225 151 137 156 200 266 127
>
```

Part 2 : A GLM Implementation OF Gradient Descent

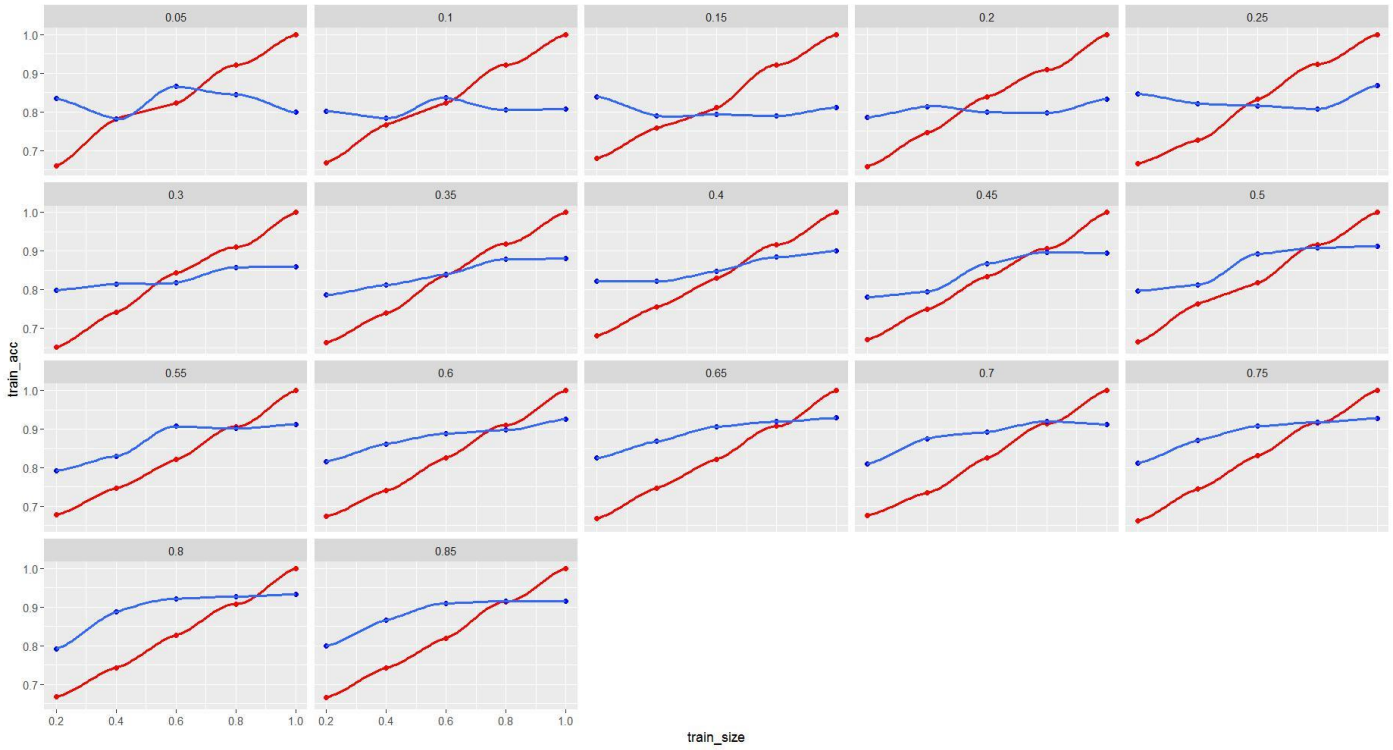
```
#PreProcess BreastCancer Data
preprocessBC<-function(BreastCancer)
{
bc<-BreastCancer
bc<-na.omit(bc)
bc$Class<-as.numeric(factor(bc$Class))
bc$Class<-as.numeric(gsub(2,0,bc$Class))
bc<-(bc[,-1])
return(bc)
}

scvTest<-function(bc)
{
#we will return this data frame for plotting
df <- data.frame(train_acc=double(),
            test_acc=double(),
            partn=double(),
            train_size=double(),
            stringsAsFactors=FALSE)
p1<-0.05
j<-1
```

```
#p1 indicates the train-test split fromm 5%,95%..... 90%,10%
while(p1<0.9)
{
print(p1)
#setsize indicates training set size. we need this to generate learning curve x axis
setsize<-0.2
while(setsize<=1)
{
i<-1
train_all<-c()
test_all<-c()
while(i<10)
{
bc1<-bc[sample.int(NROW(bc)),]
#Perform Partitions for train-test split
inTrain <- createDataPartition(y = bc1$Class,p = p1,list = FALSE)
inTrain1<-inTrain[1:round((setsize*NROW(inTrain))),]
training<-bc1[inTrain1,]
testing <-bc1[-inTrain,]
train_rows<-NROW(training)
test_rows<-NROW(testing)
#Train on Intended amount of training split for iteration
trained_model<-train(Class ~ .,data = training,method="glm",family="binomial",maxit=10000)
#Predict on Train and Test data
train_results<-predict(trained_model,bc1[inTrain,1:9])
test_results<-predict(trained_model,testing[,1:9])

#Data Collected for averaging to reduce noise
train_all<-append(train_all,sum((train_results>0.5)==(training$Class>0)))
test_all<-append(test_all,sum((test_results>0.5)==(testing$Class>0)))
i<-i+1
}
df[j,"train_acc"]<-mean(train_all)/NROW(bc1[inTrain,])
df[j,"test_acc"]<-mean(test_all)/test_rows
df[j,"partn"]<-p1
df[j,"train_size"]<-setsize
j<-j+1
setsize<-setsize+0.2
}
p1<-p1+0.05

}
return(df)
}

#Invoking Q4/Q5 code and plotting. The Learning Curves generated are displayed below in the next page
q5df<-scvTest(preprocessBC(BreastCancer))
ggplot(data=q5df)+geom_point(aes(x=train_size,y=train_acc),color="RED")+geom_smooth(aes(x=train_size,y=train_acc),method="loess")+f
acet_wrap(~partn)+geom_point(aes(x=train_size,y=test_acc),Color="BLUE")+geom_smooth(aes(x=train_size,y=test_acc),method="loess")
```

2. Emperical Analysis of Performance & Comparison of Classification Results

My Implementation -> 94.817 vs GLM Implementation -> 93.296

The x axis is the training data size, the y axis is the train (red)/test(blue) accuracy, the graphs are faceted by the partition combinations we have chosen for the glm implementation



2.1 Performance Parity – Possible Causes

1) Choice of **Learning Rate** (Choice Of Alpha)
2) Choice of **convergence criteria**. The degree to which theta must be evaluated for convergence, the formulae used in computing error
3) The **decay rate** of alpha
4) **Stochastic Vs Batch GD**: I have implemented batch gradient descent while R most likely uses SGD. For this dataset of 700 rows, batch GD seems to perform better while only Stochastic GD would scale for larger dataset with millions of rows

2.2 Loss Functions Traces

The loss function used in (multinomial) logistic regression is defined as the negative log-likelihood of the true labels given a probabilistic classifier's predictions. The log loss is only defined for two or more labels. For a single sample with true label yt in {0,1} and estimated probability yp that yt = 1, the log loss is

$$-\log P(yt|yp) = -(yt \log(yp) + (1 - yt) \log(1 - yp))$$

The below code graphs the logistic regression loss function (negative log likelihood) over the train set and over the test set as a function of the train set size.

2.2.1 R code - Generate loss function traces on training and test data

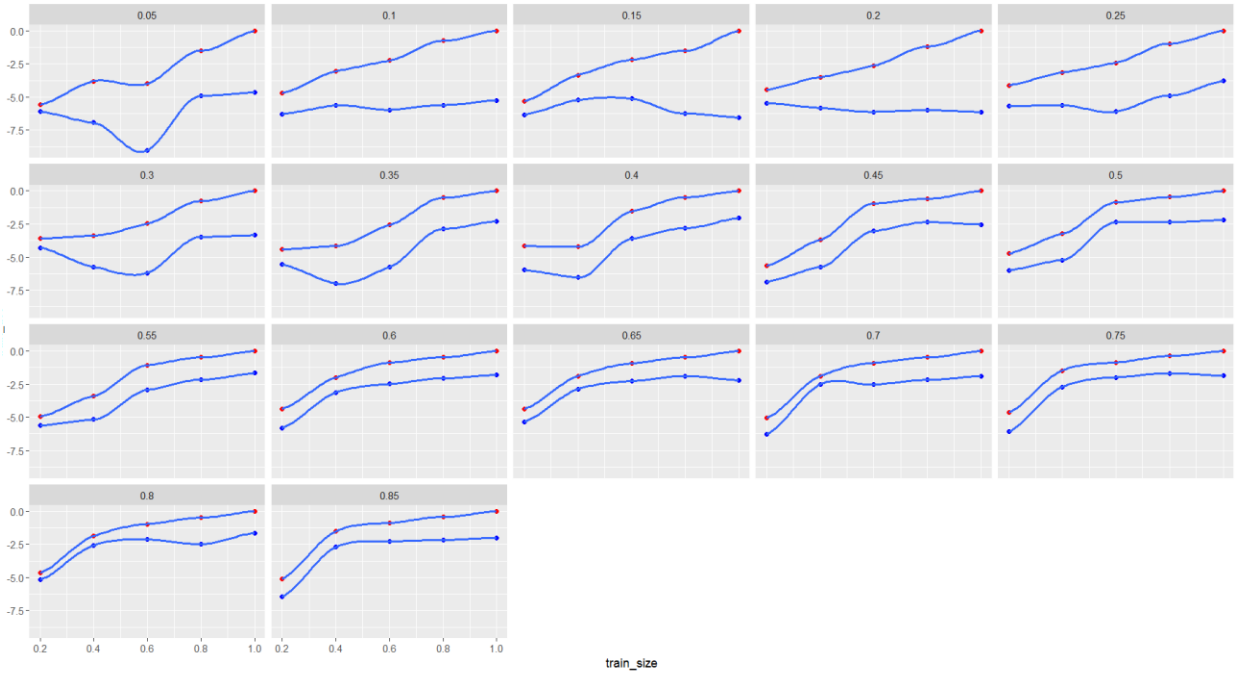
```
#PreProcess BreastCancer Data
preprocessBC<-function(BreastCancer)
{
bc<-BreastCancer
bc<-na.omit(bc)
bc$Class<-as.numeric(factor(bc$Class))
bc$Class<-as.numeric(gsub(2,0,bc$Class))
bc<-(bc[,-1])
return(bc)
}
scvTestQ6<-function(bc)
{
df <- data.frame(train_acc=double(), test_acc=double(), partn=double(), train_size=double(), stringsAsFactors=FALSE)
p1<-0.05
j<-1
while(p1<0.9)
{
print(p1)
setsize<-0.2
while(setsize<=1)
{
i<-1
train_all<-c()
test_all<-c()
while(i<10)
{
bc1<-bc[sample.int(NROW(bc)),]
inTrain <- createDataPartition(y = bc1$Class,p = p1,list = FALSE)
inTrain1<-inTrain[1:round((setsize*NROW(inTrain))),]
training<-bc1[inTrain1,]
testing <-bc1[-inTrain,]
train_rows<-NROW(training)
test_rows<-NROW(testing)
#trained_model<-glm2(Class ~ .,data = training,family="binomial",maxit=30000)
ytrain<-bc1[inTrain,10]
ytest<-bc1[-inTrain,10]
#This calculates the metric to plot as per the referenced link
trained_model<-train(Class ~ .,data = training,method="glm",family="binomial",maxit=10000)
train_results<-predict(trained_model,bc1[inTrain,1:9])
test_results<-predict(trained_model,testing[,1:9])
train_lik<-mean(ytrain * log(train_results) + (1-ytrain) * log(1-train_results))
```

```
test_lik<-mean(ytest * log(test_results) + (1-ytest) * log(1-test_results))
train_all<-append(train_all,train_lik)
test_all<-append(test_all,test_lik)
i<-i+1
}
df[j,"train_acc"]<-mean(train_all)
df[j,"test_acc"]<-mean(test_all)
df[j,"partn"]<-p1
df[j,"train_size"]<-setsize
j<-j+1
setsize<-setsize+0.2
}
p1<-p1+0.05
}
return(df)
}
```

#Generate Loss Function Plots q6df<-scvTestQ6(preprocessBC(BreastCancer))

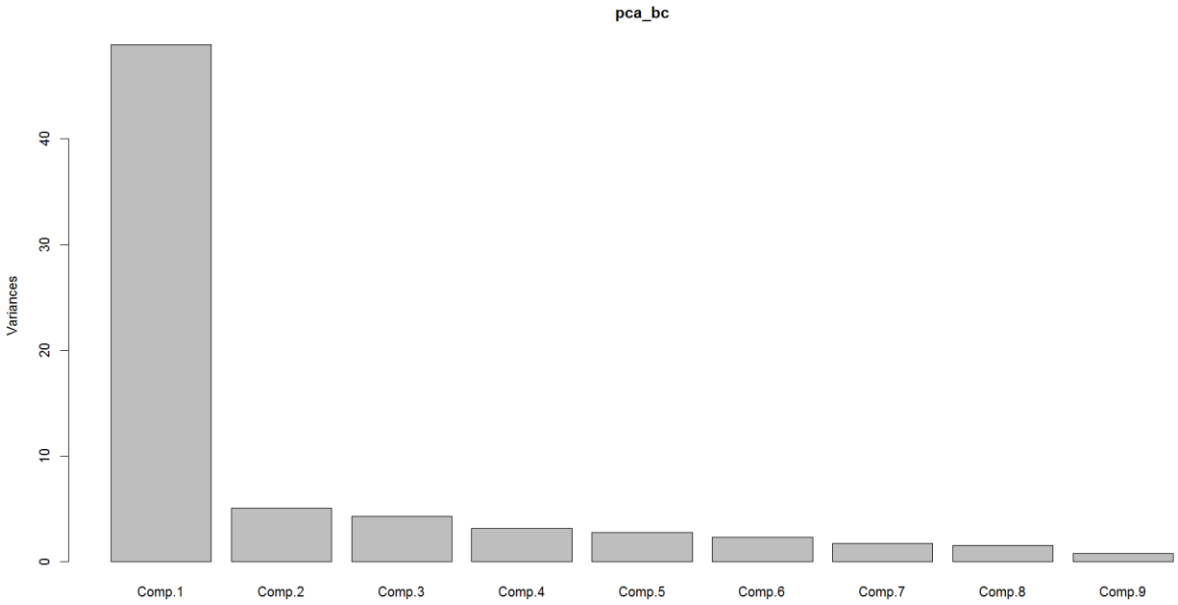
```
ggplot(data=q6df)+geom_point(aes(x=train_size,y=train_acc),color="RED")+geom_smooth(aes(x=train_size,y=train_acc),method="loess")+facet_wrap(~partn)+geom_point(aes(x=train_size,y=test_acc),Color="BLUE")+geom_smooth(aes(x=train_size,y=test_acc),method="loess")
```

The x axis is the training data size, the y axis is the log likelihood, the graphs are faceted by the partition combinations we have chosen



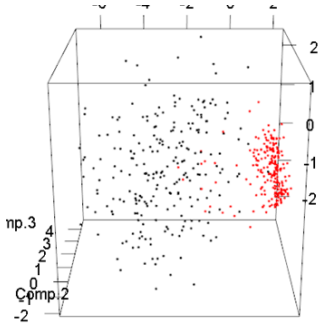
3. Principal Components Analysis for Feature Transformation

Principal components analysis is a procedure for identifying a smaller number of uncorrelated variables, called "principal components", from a large set of data. The goal of principal components analysis is to explain the maximum amount of variance with the fewest number of principal components. We seek to apply PCA to reduce dimensioanlity of the data set, so the prediction quality on unseen data improves without additional data being known to us.



The above plot shows the parity in variance across the reduced PCA components. The disproportionate amount of variance captured in the first 3 components indicate the stability of prediction in unseen data.

K-Means Clustering Of PCA Components



3.1 Inferences From Breast Cancer Data

1) The positive test(red) has a well defined cluster centroid which probably is likely to result in less false negatives for the test. However, the test does have a misdiagnosis false positive and negative around 2-3%.
2) We understand that high values of component1 is one of the causes for the positive test .

3.2 Performance Of PCA Reduced Data

```
pca_bc<-princomp(bc[,1:9])
pca_reduced<-data.frame(cbind(pca_bc$scores[,1],pca_bc$scores[,2],pca_bc$scores[,3],bc$Class))
cv.err <- cv.glm(bc,glm_nonreduced, cost, K =10)
print(cv.err$delta)
```

CustomGradient Descent	GLM Gradient Descent	PCA Reduced Gradient Descent
94 .81	93.296	97.14

Conclusion

The paper brings out the theoretical foundations of logistic regression, a hand coded implementation of gradient descent and emperical comparisons on various accuracy metrics including learning curves and loss functions. Finally, we bring out the benefits of feature reduction , the role of visualization and comparitive tabulation of results from all discussed approaches