Problems Which arose with the research:

1. Getting the tweet information from the exact Geographical location – Sri Lanka. Therefore, complete information set is taken as the dataset for the project. That will be taken to the further process.

2. Classifying and clustering the tweets according the keyword collection which gives the exact descriptive analysis is impossible with the tweet messages with certain way of typing.

3. More tweets provide a neutral ideas and decisions regarding the concept and it follows to some major issues like understanding to which cluster whether that tweet falls.

4. Visualization can be done only as Economic vs Lockdown due to constraints of the tweets.

KNN Classifying Model Algorithm

```
# the actual classifier script for predicting a sentiment using KNN
from __future__ import division
from sklearn import neighbors
from sklearn import cross_validation
from sklearn import preprocessing as pr
from sklearn import metrics

import numpy as np
from itertools import product

import features
import polarity
import ngramGenerator
import preprocessing

#WEUGHTS_VECTOR=[1.0,1.0,0.6]

print "Initializing dictionnaries"
stopWords = preprocessing.getStopWordList('../resources/stopWords.txt')
slangs = preprocessing.loadSlangs('../resources/internetSlangs.txt')
afinn=polarity.loadAfinn('../resources/afinn.txt')
#sentiWordnet=polarity.loadSentiWordnet('../resources/sentiWordnetBig.csv')
emoticonDict=features.createEmoticonDictionary("../resources/emoticon.txt")

print "Bulding unigram vector"
positive=ngramGenerator.mostFreqList('../data/used/positive1.csv',3000)
negative=ngramGenerator.mostFreqList('../data/used/negative1.csv',3000)
neutral=ngramGenerator.mostFreqList('../data/used/neutral1.csv',3000)


total=positive+negative+neutral # total unigram vector
for w in total:
   count=total.count(w)
   if (count > 1):
      while (count>0):
         count=count-1
         total.remove(w)
# equalize unigrams sizes
m=min([len(positive),len(negative),len(neutral)])
```

```
positive=positive[0:m-1]
negative=negative[0:m-1]
neutral=neutral[0:m-1]


def mapTweet(tweet,afinn,emoDict,positive,negative,neutral,slangs):
    out=[]
    line=preprocessing.processTweet(tweet,stopWords,slangs)
    p=polarity.afinnPolarity(line,afinn)
    out.append(p)
    out.append(float(features.emoticonScore(line,emoDict))) # emo aggregate score be careful to modify weights
    out.append(float(len(features.hashtagWords(line))/40)) # number of hashtagged words
    out.append(float(len(line)/140)) # for the length
    out.append(float(features.upperCase(line))) # uppercase existence : 0 or 1
    out.append(float(features.exclamationTest(line)))
    out.append(float(line.count("!")/140))
    out.append(float((features.questionTest(line))))
    out.append(float(line.count('?')/140))
    out.append(float(features.freqCapital(line)))
    u=features.scoreUnigram(line,positive,negative,neutral)
    out.extend(u)
    return out

# load matrix
def loadMatrix(posfilename,neufilename,negfilename,poslabel,neulabel,neglabel):
    vectors=[]
    labels=[]
    print "Loading training dataset..."
    f=open(posfilename,'r')
    kpos=0
    kneg=0
    kneu=0
    line=f.readline()
    while line:
        kpos=kpos+1
        z=mapTweet(line,afinn,emoticonDict,positive,negative,neutral,slangs)
        vectors.append(z)
        labels.append(float(poslabel))
        line=f.readline()
#       print str(kpos)+"positive line loaded"+str(len(vectors))+" "+str(len(labels))
    f.close()

    f=open(neufilename,'r')
    line=f.readline()
    while line:
        kneu=kneu+1
        z=mapTweet(line,afinn,emoticonDict,positive,negative,neutral,slangs)
        vectors.append(z)
        labels.append(float(neulabel))
        line=f.readline()
#       print str(kneu)+"neutral lines loaded"
    f.close()

    f=open(negfilename,'r')
    line=f.readline()
    while line:
```

```
        kneg=kneg+1
        z=mapTweet(line,afinn,emoticonDict,positive,negative,neutral,slangs)
        vectors.append(z)
        labels.append(float(neglabel))
        line=f.readline()
#       print str(kneg)+"negative lines loaded"
    f.close()
    print "Loading done."
    return vectors,labels

# WEIGHTING LIST of VECTORS
def weight(X,w): # n startig for unigram weights
    result=[]
    def simple(x,w):
        r=[]
        n=len(w)
        for i in range(0,n):
            r.append(x[i]*w[i])
        for i in range(n,len(x)):
            r.append(x[i]*w[n-1])
        return r

    for f in X:
        result.append(simple(f,w))
    return result
# OPTIMIZE PRECISION
def optimizeKNN(X,Y,n): # n number of features : unigram is one fature
    print "Optimizing features weights..."
    best_weights=[]
    best_accuracy=0.0
    w=[0.2*i for i in range (1,6)] # wifferent possible weights for a single feature
    W=[]
    for x in product(w,repeat=n):
        W.append(list(x))

    for ww in W: # searching for best weights combination
        X=weight(X,ww)
        x=np.array(X)
        y=np.array(Y)
        clf = neighbors.KNeighborsClassifier(N_NEIGHBORS)
        scores = cross_validation.cross_val_score(clf, x, y, cv=5)
        if (scores.mean() > best_accuracy):
            best_accuracy=scores.mean()
            best_weights=ww

    print "best accuracy is :"+str(best_accuracy)
    print "best weight vector is :"
    print best_weights
    return best_accuracy,best_weights

# training model
def trainModel(X,Y,n): # number of neighbors
    clf = neighbors.KNeighborsClassifier(n)
    clf.fit(X,Y)
    return clf

# predict tweet class
```

```
def predict(tweet,model): # test a tweet against a built model
   z=mapTweet(tweet,afinn,emoticonDict,positive,negative,neutral,slangs) # mapping
   z_scaled=scaler.transform(z)
   z=normalizer.transform([z_scaled])
   z=z[0].tolist()
   return model.predict([z]).tolist() # transform nympy array to list

# load test data set
def loadTest(filename): # function to load test file in the csv format : sentiment,tweet
   f=open(filename,'r')
   line=f.readline()
   labels=[]
   vectors=[]
   while line:
      l=line[:-1].split(r'"',"'")
      s=float(l[0][1:])
      tweet=l[5][:-1]

      z=mapTweet(tweet,afinn,emoticonDict,positive,negative,neutral,slangs)
      z_scaled=scaler.transform(z)
      z=normalizer.transform([z_scaled])
      z=z[0].tolist()
      vectors.append(z)
      labels.append(s)
      line=f.readline()
#       print str(kneg)+"negative lines loaded"
   f.close()
   return vectors,labels

# load test data set
def writeTest(filename,knn_model): # function to load test file in the csv format : sentiment,tweet
   f=open(filename,'r')
   line=f.readline()
   fo=open(filename+".knn_result","w")
   fo.write("old,tweet,new\n")
   while line:
      l=line[:-1].split(r'"',"'")
      s=float(l[0][1:])
      tweet=l[5][:-1]
      nl=predict(tweet,knn_model)
      fo.write(r'"'+str(s)+r'",'"'+tweet+r'",'"'+str(nl[0])+r'"'+"\n")
      line=f.readline()
#       print str(kneg)+"negative lines loaded"
   f.close()
   fo.close()
   print "labelled test dataset is stores in : "+str(filename)+".knn_result"

#def predictFile
def predictFile(filename,knn_model): # function to load test file in the csv format
   f=open(filename,'r')
   fo=open(filename+".result",'w')
   line=f.readline()
   while line:
      tweet=line[:-1]

      nl=predict(tweet,knn_model)
```

```
        fo.write(r'"'+str(nl)+r'","'+tweet+'"\n')
        line=f.readline()

    f.close()
    fo.close()
    print "Tweets are classified . The result is in "+filename+".result"

def testModel(vectors,labels,model): # for a given set of labelled vectors calculate model labels and give accuract
    a=0 # wrong classified vectors
    newLabels=model.predict(vectors).tolist()
    acc=metrics.accuracy_score(labels,newLabels)
    pre=metrics.precision_score(labels,newLabels)
    print "average accuracy over test dataset : %.2f" %(acc)
    print "average precision over test dataset : %.2f" %(pre)

# loading training data
X,Y=loadMatrix('../data/used/positive1.csv','../data/used/neutral1.csv','../data/used/negative1.csv','4','2','0')
#X,Y=loadMatrix('../data/small_positive_processed.csv','../data/small_neutral_processed.csv','../data/small_negativ
e_processed.csv','4','2','0')

# features standardization
X_scaled=pr.scale(np.array(X))
scaler = pr.StandardScaler().fit(X) # to use later for testing data scaler.transform(X)

# features Normalization
X_normalized = pr.normalize(X_scaled, norm='l2') # l2 norm
normalizer = pr.Normalizer().fit(X_scaled)  # as before normalizer.transform([[-1.,  1., 0.]]) for test

X=X_normalized
X=X.tolist()

# features selection

# 5 fold cross validation
x=np.array(X)
y=np.array(Y)

N_NEIGHBORS=1
ACC=0.0
PR=0.0
iter=0
for k in range(10,11):
    iter=iter+1
    clf = neighbors.KNeighborsClassifier(k)
    scores = cross_validation.cross_val_score(clf, x, y, cv=5,scoring='accuracy')
    precisions=cross_validation.cross_val_score(clf, x, y, cv=5,scoring='precision')
    print "Iter : "+str(iter)+" :"
    print("Accuracy of the model using 5 fold cross validation : %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))#
Actual testing
    print("Precision of the model using 5 fold cross validation : %0.2f (+/- %0.2f)" % (precisions.mean(), precisions.std()
* 2))# Actual testing
    if (scores.mean()>ACC and precisions.mean() > PR):
        ACC=scores.mean()
        PR=precisions.mean()
        N_NEIGHBORS=k

# cross validation
```

```
#N_NEIGHBORS=10
clf = neighbors.KNeighborsClassifier(N_NEIGHBORS)
scores = cross_validation.cross_val_score(clf, x, y, cv=5,scoring='accuracy')
precisions=cross_validation.cross_val_score(clf, x, y, cv=5,scoring='precision')

print "Optimal number of neighbors : "+str(N_NEIGHBORS)
print("Accuracy of the model using 5 fold cross validation : %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))# Actual testing
print("Precision of the model using 5 fold cross validation : %0.2f (+/- %0.2f)" % (precisions.mean(), precisions.std() * 2))# Actual testing


# Actual testing
print "Building model"
MODEL=trainModel(X,Y,N_NEIGHBORS) # 3nn

print "Model Built . Testing ..."
# uncomment to see performance over test data set
V,L=loadTest('../data/test_dataset.csv')
#V,L=loadTest('../data/small_test_dataset.csv')

print "Classification done : Performance over test dataset : "
testModel(V,L,MODEL)

# write new labelled test dataset
writeTest('../data/test_dataset.csv',MODEL)

user_input=raw_input("Write a tweet to test or a file path for bulk classification with knn model. press q to quit\n")
while user_input!='q':
  try:
    predictFile(user_input,MODEL)
    print "labels are : 4.0 for positive, 2.0 for neutral and 0.0 for negative tweets"
    user_input=raw_input("Write a tweet to test or a file path for bulk classification . press q to quit\n")
  except:
    print "sentiment : "+str(predict(user_input,MODEL))
    print "labels are : 4.0 for positive, 2.0 for neutral and 0.0 for negative tweets"
    user_input=raw_input("Write a tweet to test or a file path for bulk classification . press q to quit\n")

# the end !
```

Actual classifier script for predicting a sentiment using SVM

```
from __future__ import division
from sklearn import svm
from sklearn import cross_validation
from sklearn import preprocessing as pr
from sklearn import metrics

import numpy as np
import nltk # for pos tags

import features
import polarity
import ngramGenerator
import preprocessing
```

```
KERNEL_FUNCTION='linear'
C_PARAMETER=0.6

print "Initializing dictionnaries"
stopWords = preprocessing.getStopWordList('../resources/stopWords.txt')
slangs = preprocessing.loadSlangs('../resources/internetSlangs.txt')
afinn=polarity.loadAfinn('../resources/afinn.txt')
#sentiWordnet=polarity.loadSentiWordnet('../resources/sentiWordnetBig.csv')
emoticonDict=features.createEmoticonDictionary("../resources/emoticon.txt")

print "Bulding Bag of words ..."
positive=ngramGenerator.mostFreqList('../data/used/positive1.csv',3000)
negative=ngramGenerator.mostFreqList('../data/used/negative1.csv',3000)
neutral=ngramGenerator.mostFreqList('../data/used/neutral1.csv',3000)


for w in positive:
   if w in negative+neutral :
      positive.remove(w)

for w in negative:
   if w in positive+neutral :
      negative.remove(w)

for w in neutral:
   if w in negative+positive :
      neutral.remove(w)

# equalize unigrams sizes
m=min([len(positive),len(negative),len(neutral)])

positive=positive[0:m-1]
negative=negative[0:m-1]
neutral=neutral[0:m-1]


#print len(total)

def mapTweet(tweet,afinn,emoDict,positive,negative,neutral,slangs):
   out=[]
   line=preprocessing.processTweet(tweet,stopWords,slangs)
   p=polarity.afinnPolarity(line,afinn)
   out.append(p)
   out.append(float(features.emoticonScore(line,emoDict))) # emo aggregate score be careful to modify weights
   out.append(float(len(features.hashtagWords(line))/40)) # number of hashtagged words
   out.append(float(len(line)/140)) # for the length
   out.append(float(features.upperCase(line))) # uppercase existence : 0 or 1
   out.append(float(features.exclamationTest(line)))
   out.append(float(line.count("!")/140))
   out.append(float((features.questionTest(line))))
   out.append(float(line.count('?')/140))
   out.append(float(features.freqCapital(line)))
   u=features.scoreUnigram(line,positive,negative,neutral)
   out.extend(u)
   return out
```

```
# load matrix
def loadMatrix(posfilename,neufilename,negfilename,poslabel,neulabel,neglabel):
    vectors=[]
    labels=[]
    print "Loading training dataset..."
    f=open(posfilename,'r')
    kpos=0
    kneg=0
    kneu=0
    line=f.readline()
    while line:

        try:
            kpos+=1
            z=mapTweet(line,afinn,emoticonDict,positive,negative,neutral,slangs)
            vectors.append(z)
            labels.append(float(poslabel))
        except:
            None
        line=f.readline()
#       print str(kpos)+"positive lines loaded : "+str(z)
    f.close()

    f=open(neufilename,'r')
    line=f.readline()
    while line:
        try:
            kneu=kneu+1
            z=mapTweet(line,afinn,emoticonDict,positive,negative,neutral,slangs)
            vectors.append(z)
            labels.append(float(neulabel))
        except:
            None
        line=f.readline()
#       print str(kneu)+"neutral lines loaded : "+str(z)
    f.close()

    f=open(negfilename,'r')
    line=f.readline()
    while line:
        try:
            kneg=kneg+1
            z=mapTweet(line,afinn,emoticonDict,positive,negative,neutral,slangs)
            vectors.append(z)
            labels.append(float(neglabel))
        except:
            None
        line=f.readline()
#       print str(kneg)+"negative lines loaded : "+str(z)
    f.close()
    print "Loading done."
    return vectors,labels



# map tweet into a vector
def trainModel(X,Y,knel,c): # relaxation parameter
```

```
    clf=svm.SVC(kernel=knel) # linear, poly, rbf, sigmoid, precomputed , see doc
    clf.fit(X,Y)
    return clf

def predict(tweet,model): # test a tweet against a built model
    z=mapTweet(tweet,afinn,emoticonDict,positive,negative,neutral,slangs) # mapping
    z_scaled=scaler.transform(z)
    z=normalizer.transform([z_scaled])
    z=z[0].tolist()
    return model.predict([z]).tolist()[0] # transform nympy array to list

def predictFile(filename,svm_model): # function to load test file in the csv format : sentiment,tweet
    f=open(filename,'r')
    fo=open(filename+".result",'w')
    fo.write('"auto label","tweet","anouar label","ziany label"') # header
    line=f.readline()
    while line:
        tweet=line[:-1]

        nl=predict(tweet,svm_model)


        fo.write(r'"'+str(nl)+r'","'+tweet+'"\n')
        line=f.readline()

    f.close()
    fo.close()
    print "Tweets are classified . The result is in "+filename+".result"

def loadTest(filename): # function to load test file in the csv format : sentiment,tweet
    f=open(filename,'r')
    line=f.readline()
    labels=[]
    vectors=[]
    while line:
        l=line[:-1].split(r'","')
        s=float(l[0][1:])
        tweet=l[5][:-1]

        z=mapTweet(tweet,afinn,emoticonDict,positive,negative,neutral,slangs)
        z_scaled=scaler.transform(z)
        z=normalizer.transform([z_scaled])
        z=z[0].tolist()
        vectors.append(z)
        labels.append(s)
        line=f.readline()
#       print str(kneg)+"negative lines loaded"
    f.close()
    return vectors,labels

# write labelled  test dataset
def writeTest(filename,model): # function to load test file in the csv format : sentiment,tweet
    f=open(filename,'r')
    line=f.readline()
    fo=open(filename+".svm_result","w")
    fo.write("old,tweet,new\n")
    while line:
```

```
        l=line[:-1].split(r'","')
        s=float(l[0][1:])
        tweet=l[5][:-1]
        nl=predict(tweet,model)
        fo.write(r'"'+str(s)+r'","'+tweet+r'","'+str(nl)+r'"'+"\n")
        line=f.readline()
#       print str(kneg)+"negative lines loaded"
    f.close()
    fo.close()
    print "labelled test dataset is stores in : "+str(filename)+".svm_result"


def testModel(vectors,labels,model): # for a given set of labelled vectors calculate model labels and give accuract
    a=0 # wrong classified vectors
    newLabels=model.predict(vectors).tolist()
    acc=metrics.accuracy_score(labels,newLabels)
    pre=metrics.precision_score(labels,newLabels)
    print "average accuracy over test dataset : %.2f" %(acc)
    print "average precision over test dataset : %.2f" %(pre)


# loading training data
X,Y=loadMatrix('../data/used/positive1.csv','../data/used/neutral1.csv','../data/used/negative1.csv','4','2','0')
#X,Y=loadMatrix('../data/small_positive_processed.csv','../data/small_neutral_processed.csv','../data/small_negativ
e_processed.csv','4','2','0')


# features standardization
X_scaled=pr.scale(np.array(X))
scaler = pr.StandardScaler().fit(X) # to use later for testing data scaler.transform(X)

# features Normalization
X_normalized = pr.normalize(X_scaled, norm='l2') # l2 norm
normalizer = pr.Normalizer().fit(X_scaled)  # as before normalizer.transform([[-1.,  1., 0.]]) for test

X=X_normalized
X=X.tolist()

# 5 fold cross validation
x=np.array(X)
y=np.array(Y)
KERNEL_FUNCTIONS=['linear']
C=[0.01*i for i in range(1,2)]
ACC=0.0
PRE=0.0
iter=0

for knel in KERNEL_FUNCTIONS:
    for c in C:

        clf = svm.SVC(kernel=KERNEL_FUNCTION, C=c)
        scores = cross_validation.cross_val_score(clf, x, y, cv=5,scoring='accuracy')
        precisions=cross_validation.cross_val_score(clf, x, y, cv=5,scoring='precision')
        if (scores.mean() > ACC and precisions.mean() > PRE):
            ACC=scores.mean()
            PRE=precisions.mean()
            KERNEL_FUNCTION=knel
```

```
            C_PARAMETER=c
        iter=iter+1
        print "iteration "+str(iter)+" : c parameter : "+str(c)+", kernel : "+str(knel)
        print("Accuracy of the model using 5 fold cross validation : %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() *
2))# Actual testing
        print("Precision of the model using 5 fold cross validation : %0.2f (+/- %0.2f)" % (precisions.mean(),
precisions.std() * 2))# Actual testing

print "Optimal C : "+str(C_PARAMETER)
print "Optimal kernel function  : "+KERNEL_FUNCTION
print "Accurracy : "+str(ACC)
print "Precision : "+str(PRE)


print "Training model with optimized parameters"
MODEL=trainModel(X,Y,KERNEL_FUNCTION,C_PARAMETER)
print "Training done !"


# uncomment to classify test dataset
print "Loading test data..."
V,L=loadTest('../data/test_dataset.csv')
#V,L=loadTest('../data/small_test_dataset.csv')

# writ labelled test dataset
writeTest('../data/test_dataset.csv',MODEL)

print "Classification done : Performance over test dataset : "
testModel(V,L,MODEL)

user_input=raw_input("Write a tweet to test or a file path for bulk classification with svm model. press q to quit\n")
while user_input!='q':
    try:
        predictFile(user_input,MODEL)
        print "labels are : 4.0 for positive, 2.0 for neutral and 0.0 for negative tweets"
        user_input=raw_input("Write a tweet to test or a file path for bulk classification . press q to quit\n")
    except:
        print "sentiment : "+str(predict(user_input,MODEL))
        print "labels are : 4.0 for positive, 2.0 for neutral and 0.0 for negative tweets"
        user_input=raw_input("Write a tweet to test or a file path for bulk classification . press q to quit\n")

# the end !
```