Taylor Tanita
A11216947

COGS 118A: Assignment 5
Support Vector Machine

1 ionosphere.mat

Kernel / [train, test]	[80, 20]	[60, 40]	[40, 60]
Linear SVM	C = 0.775 Error = 0.114285714286	C = 0.775 Error = 0.15	C = 0.1 Error = 0.142857142857
RBF SVM	C = 10.0 Gamma = 0.1 Error = 0.0285714285714	C = 100 Gamma = default Error = 0.05	C = 10 Gamma = default Error = 0.0761904761905

2 fisheriris.mat

Kernel / [train, test]	[80, 20]	[60, 40]	[40, 60]
Linear SVM	C = 0.1 Error = 0.0	C = 0.1 Error = 0.0	C = 0.1 Error = 0.0
RBF SVM	C = 1.0 Gamma = default Error = 0.0	C = 1.0 Gamma = default Error = 0.0	C = 1.0 Gamma = default Error = 0.0

3 arrhythmia.mat

Kernel / [train, test]	[80, 20]	[60, 40]	[40, 60]
Linear SVM	C = 10.1008 Error = 0.355555555556	C = 0.001 Error = 0.35	C = 0.83125 Error = 0.376383763838
RBF SVM	C = 0.01 Gamma = default Error = 0.488888888889	C = 0.01 Gamma = default Error = 0.461111111111	C = 0.01 Gamma = default Error = 0.568265682657

Taylor Tanita
A11216947

a5.py (methods)

```
import scipy.io as sio
import matplotlib.pyplot as plt
import numpy as np
from sklearn import svm
import copy
import math

#list conversion methods specialized based on instructions
#convert1 is for problem 1, convert2 for problem 2, and convert3 for problem 3
def convert1(elmt):
  if (elmt[0][0] == u'b'):
    return 1
  else:
    return 0

def convert2(elmt):
  if (elmt[0][0] == u'setosa'):
    return 1
  else:
    return 0

def convert3(elmt):
  if (elmt[0] == 1):
    return 1
  else:
    return 0

def convert_array(y,problem):
  yy = np.array([[-1]])
  (length,_) = y.shape

  for i in range(length):
    if (problem == 1):
      yy = np.append(yy,[[convert1(y[i])]],axis=0)
    if (problem == 2):
      yy = np.append(yy,[[convert2(y[i])]],axis=0)
    if (problem == 3):
      yy = np.append(yy,[[convert3(y[i])]],axis=0)

  return yy[1:]

# method to change a NaN to 0
def convert_0(elmt):
  if(np.isnan(elmt)):
    return 0.0
  else:
    return elmt
```

Taylor Tanita
A11216947

```
# finds NaN values and converts them to 0 in the matrix
def convert_x(x):
  (rows,col) = x.shape
  for i in range(rows):
    x[i] = [convert_0(k) for k in x[i]]
  return x

# method that selects the percentage specified for training and testing purposed
# returns a tuple with the training and testing data
def select_data(x,y,percent):
  (size,width) = x.shape
  train_size = np.ceil(size*percent)

  data = np.concatenate((x,y),axis=1)
  np.random.shuffle(data)
  x = data[:,:(width)]
  y = data[:,[(width)]]

  x_train = x[:train_size]
  x_test = x[train_size:]
  y_train = y[:train_size]
  y_test = y[train_size:]

  return (x_train,y_train,x_test,y_test)

# method to determine the error when using a specific C value and svm type
# returns error
# svm_type: 1 = linear, 0 = RBF
def test_c(x,y,x_test,y_test,c,svm_type):
# in order for split to work, needs to be a multiple of what you are splitting
# it to, so take the first partition will contain the remainder data
  (size,width) = x.shape
  mod = size % 5
  xx = x[mod:]
  yy = y[mod:]

  x_part = np.split(xx,5)
  y_part = np.split(yy,5)

  x_part[0] = np.concatenate((x_part[0],x[:mod]),axis=0)
  y_part[0] = np.concatenate((y_part[0],y[:mod]),axis=0)

# 5-fold cross validation
  error = 0
# use appropriate svm type
  clf = svm.LinearSVC(C=c)
  if (svm_type == 0):
```

```
    clf = svm.SVC(C=c)
  for i in range(5):
    X = copy.deepcopy(x_part)
    to_testx = X.pop(i)
    xx = np.concatenate((X[0],X[1]),axis=0)
    xx = np.concatenate((xx,X[2]),axis=0)
    xx = np.concatenate((xx,X[3]),axis=0)

    Y = copy.deepcopy(y_part)
    to_testy = Y.pop(i)
    yy = np.concatenate((Y[0],Y[1]),axis=0)
    yy = np.concatenate((yy,Y[2]),axis=0)
    yy = np.concatenate((yy,Y[3]),axis=0)

    clf.fit(xx,yy)
    predict = clf.predict(to_testx)
    e = test(predict,to_testy)
    error = error + e

  crossval_error = error/5
  predict_test = clf.predict(x_test)
  e = test(predict_test,y_test)
  return (crossval_error, e)


# helper method for test_c
# takes in two lists and returns the percent error
# i.e. the percentage of time where the two lists dffer
def test(predict,y_test):
  y = y_test.ravel()
  total = len(predict)
  errors = 0.0
  for i in range(total):
    if (predict[i] != y[i]):
      errors = errors + 1.0
  return errors/total

# function to settle ties in errors to ensure lowest C is used
def min_func((_,error,c)):
# weigh the error significantly more so that the only difference will occur in a
# tie
  score = error * 1000
  if (c == 100):
    score = score - .000001
  if (c == 10):
    score = score - .000002
  if (c == 1):
    score = score - .000003
```

```
  if (c == .1):
    score = score - .000004
  if (c == .01):
    score = score - .000005
  if (c == .001):
    score = score - .000006
  return score

# method to pick the best C
# tries 4 different C values
# returns the best C value and the error associated with it
def pick_C(x,y,x_test,y_test,svm_type):
# test the different C values
  (val,error) = test_c(x,y,x_test,y_test,10,svm_type)
  (val_100,err_100) = test_c(x,y,x_test,y_test,100,svm_type)
  (val_1,err_1) = test_c(x,y,x_test,y_test,1,svm_type)
  (val_p1,err_p1) = test_c(x,y,x_test,y_test,.1,svm_type)
  (val_01,err_01) = test_c(x,y,x_test,y_test,.01,svm_type)
  (val_001,err_001) = test_c(x,y,x_test,y_test,.001,svm_type)

# put results in a list to extract the minimum
  values=[(val_100,err_100,100),(val,error,10),(val_1,err_1,1),(val_p1,err_p1,.1)]
  values = values + [(val_01,err_01,.01),(val_001,err_001,.001)]

  (_,error,c) = min(values,key=min_func)

# getting correct interval of c's
  c1 = 0
  c2 = 0

  if (c == 100):
    c1 = 10
    c2 = 100
  elif (c == .001):
    c1 = .001
    c2 = .01

  for i in range(1,5):
    (vv,_,cc) = values[i]
    if (c == cc):
      (v1,_,cc1) = values[i-1]
      (v2,_,cc2) = values[i+1]
      if (v1 < v2):
        c1 = cc
        c2 = cc1
      else:
        c1 = cc2
        c2 = cc
```

```
# once interval obtained, do binary search on that interval to find C that
# generated minimum error
  (c,error) = binary(c1,c2,x,y,x_test,y_test,svm_type)

  return (c,error)


# binary search method
def binary(c1,c2,x,y,x_test,y_test,svm_type):
# calculate errors for endpoints and mid value
  mid = (c1 + c2)/2.0
  (val_1,err_1) = test_c(x,y,x_test,y_test,c1,svm_type)
  (val_mid,err_mid) = test_c(x,y,x_test,y_test,mid,svm_type)
  (val_2,err_2) = test_c(x,y,x_test,y_test,c2,svm_type)

# threshold
  if (abs(c2 - c1) < .0001):
    return (c1,err_1)

# picking which endpoint to compare middle value to
  c_val = c1
  error = 0
  if (val_1 < val_2):
    c_val = c1
    error = err_1
  else:
    c_val = c2
    error = err_2

# if the middle value has a lower error, continue searching, if not, return the
# current C value and error
  if ((val_mid < val_1) | (val_mid < val_2)):
    if (c_val < val_mid):
      return binary(c_val,mid,x,y,x_test,y_test,svm_type)
    else:
      return binary(mid,c_val,x,y,x_test,y_test,svm_type)
  else:
    return (c_val,error)

# method for finding gamma.
# because gamma's default is usually sufficient, just checked against two other
# options and reported the gamma val and error associated with it
def test_gamma(x,y,x_test,y_test,c):
  (size,width) = x.shape
  mod = size % 5
  xx = x[mod:]
  yy = y[mod:]
```

```
  x_part = np.split(xx,5)
  y_part = np.split(yy,5)

  x_part[0] = np.concatenate((x_part[0],x[:mod]),axis=0)
  y_part[0] = np.concatenate((y_part[0],y[:mod]),axis=0)

# 5-fold cross validation
  error = 0
  error1 = 0
  error2 = 0
  clf = svm.SVC(C=c)
  clf1 = svm.SVC(gamma=.1,C=c)
  clf2 = svm.SVC(gamma=.01,C=c)
  for i in range(5):
    X = copy.deepcopy(x_part)
    to_testx = X.pop(i)
    xx = np.concatenate((X[0],X[1]),axis=0)
    xx = np.concatenate((xx,X[2]),axis=0)
    xx = np.concatenate((xx,X[3]),axis=0)

    Y = copy.deepcopy(y_part)
    to_testy = Y.pop(i)
    yy = np.concatenate((Y[0],Y[1]),axis=0)
    yy = np.concatenate((yy,Y[2]),axis=0)
    yy = np.concatenate((yy,Y[3]),axis=0)

    clf.fit(xx,yy)
    predict = clf.predict(to_testx)
    e = test(predict,to_testy)
    error = error + e

    clf1.fit(xx,yy)
    predict1 = clf1.predict(to_testx)
    e1 = test(predict1,to_testy)

    clf2.fit(xx,yy)
    predict2 = clf2.predict(to_testx)
    e2 = test(predict2,to_testy)
    error2 = error2 + e2

  error = error/5
  error1 = error1/5
  error2 = error2/5

# determining which gamma value was best
  g = 0.0
  if (error1 < error):
```

```
  predict_test = clf1.predict(x_test)
  e = test(predict_test,y_test)
  return ('.1', e)
if (error2 < error):
  predict_test = clf2.predict(x_test)
  e = test(predict_test,y_test)
  return ('.01', e)
else:
  predict_test = clf.predict(x_test)
  e = test(predict_test,y_test)
  return('default', e)
```

Taylor Tanita
A11216947

a_ex.py (execution)

```
from a5 import *

# setting up the data for the problems
# PROBLEM 1
problem = 1
data = sio.loadmat('ionosphere.mat')
x = data['X']
y = data['Y']
y = convert_array(y,problem)

# PROBLEM 2
problem = 2
data2 = sio.loadmat('fisheriris.mat')
meas = data2['meas']
species = data2['species']
species = convert_array(species,problem)


# PROBLEM 3
problem = 3
data3 = sio.loadmat('arrhythmia.mat')
X = data3['X']
X = convert_x(X)
Y = data3['Y']
Y = convert_array(Y,problem)

# method to perform svm on data that has been altered to correct form
def output(x,y):
# using different percentages, error for linear svm and rbf kernal are reported
  percent = 0.8

# need to run experiment multiple times- I ran it 10 times each and averaged the
# data.
# for gamma, I took the most frequently reported gamma opposed tot he average
  (x_train,y_train,x_test,y_test) = select_data(x,y,percent)
  C8 = 0.0; error8 = 0.0; C8_rbf = 0.0; error8_rbf = 0.0; g8 = ''

# to figure out most frequently reported data, list set up
  gam = [('default',0),('.1',0),('.01',0)]

# perform experiment 10 times
  for i in range(10):
#   pick C for Linear and RBF kernal
    (iC8,ierror8) = pick_C(x_train,y_train,x_test,y_test,1)
    (iC8_rbf,ierror8_rbf) = pick_C(x_train,y_train,x_test,y_test,0)
    (ig8, ierror8_rbf) = test_gamma(x_train,y_train,x_test,y_test,iC8_rbf)
```

```
#   updating gamma list
    C8 = C8 + iC8
    error8 = error8 + ierror8
    C8_rbf = C8_rbf + iC8_rbf
    error8_rbf = error8_rbf + ierror8_rbf
    if (ig8 == 'default'):
      (_,v) = gam[0]
      gam[0] = ('default',(v+1))
    if (ig8 == '.1'):
      (_,v) = gam[1]
      gam[1] = ('.1',(v+1))
    if (ig8 == '.01'):
      (_,v) = gam[2]
      gam[2] = ('.01',(v+1))

# averaging data out
  C8 = C8/10.0
  error8 = error8/10.0
  C8_rbf = C8_rbf/10.0
  error8_rbf = error8_rbf/10.0
  (g8,_) = max(gam, key=lambda x:x[1])

  print('\n')
  print('Linear SVM at 80/20: C = ' + str(C8) + ', error = ' + str(error8))
  print('RBF SVM at 80/20: C = '+str(C8_rbf)+', gamma = '+g8+', error = '+str(error8_rbf))
  print('\n')

###
# same exact code used for .6 an .4 as .8
###
  percent = 0.6
  (x_train,y_train,x_test,y_test) = select_data(x,y,percent)
  C6 = 0.0; error6 = 0.0; C6_rbf = 0.0; error6_rbf = 0.0; g6 = ''
  gam = [('default',0),('.1',0),('.01',0)]
  for i in range(10):
    (iC6,ierror6) = pick_C(x_train,y_train,x_test,y_test,1)
    (iC6_rbf,ierror6_rbf) = pick_C(x_train,y_train,x_test,y_test,0)
    (ig6, ierror6_rbf) = test_gamma(x_train,y_train,x_test,y_test,iC6_rbf)

    C6 = C6 + iC6
    error6 = error6 + ierror6
    C6_rbf = C6_rbf + iC6_rbf
    error6_rbf = error6_rbf + ierror6_rbf
    if (ig6 == 'default'):
      (_,v) = gam[0]
      gam[0] = ('default',(v+1))
    if (ig6 == '.1'):
      (_,v) = gam[1]
```

```
    gam[1] = ('.1',(v+1))
  if (ig6 == '.01'):
    (_,v) = gam[2]
    gam[2] = ('.01',(v+1))
C6 = C6/10.0
error6 = error6/10.0
C6_rbf = C6_rbf/10.0
error6_rbf = error6_rbf/10.0
(g6,_) = max(gam, key=lambda x:x[1])
(C6,error6) = pick_C(x_train,y_train,x_test,y_test,1)
(C6_rbf,error6_rbf) = pick_C(x_train,y_train,x_test,y_test,0)
(g6, error6_rbf) = test_gamma(x_train,y_train,x_test,y_test,C6_rbf)
print('Linear SVM at 60/40: C = ' + str(C6) + ', error = ' + str(error6))
print('RBF SVM at 60/40: C = '+str(C6_rbf)+', gamma = '+g6+', error = '+str(error6_rbf))
print('\n')

percent = 0.4
(x_train,y_train,x_test,y_test) = select_data(x,y,percent)
C4 = 0.0; error4 = 0.0; C4_rbf = 0.0; error4_rbf = 0.0; g4 = ''
gam = [('default',0),('.1',0),('.01',0)]
for i in range(10):
  (iC4,ierror4) = pick_C(x_train,y_train,x_test,y_test,1)
  (iC4_rbf,ierror4_rbf) = pick_C(x_train,y_train,x_test,y_test,0)
  (ig4, ierror4_rbf) = test_gamma(x_train,y_train,x_test,y_test,iC4_rbf)

  C4 = C4 + iC4
  error4 = error4 + ierror4
  C4_rbf = C4_rbf + iC4_rbf
  error4_rbf = error4_rbf + ierror4_rbf
  if (ig4 == 'default'):
    (_,v) = gam[0]
    gam[0] = ('default',(v+1))
  if (ig4 == '.1'):
    (_,v) = gam[1]
    gam[1] = ('.1',(v+1))
  if (ig4 == '.01'):
    (_,v) = gam[2]
    gam[2] = ('.01',(v+1))
C4 = C4/10.0
error4 = error4/10.0
C4_rbf = C4_rbf/10.0
error4_rbf = error4_rbf/10.0
(g4,_) = max(gam, key=lambda x:x[1])
(C4,error4) = pick_C(x_train,y_train,x_test,y_test,1)
(C4_rbf,error4_rbf) = pick_C(x_train,y_train,x_test,y_test,0)
(g4, error4_rbf) = test_gamma(x_train,y_train,x_test,y_test,C4_rbf)
print('Linear SVM at 40/60: C = ' + str(C4) + ', error = ' + str(error4))
print('RBF SVM at 40/60: C = '+str(C4_rbf)+', gamma = '+g4+', error = '+str(error4_rbf))
```

```
  print('\n')


# calling output on the data
print('Problem 1\n')
output(x,y)
print('Problem 2\n')
output(meas,species)
print('Problem 3\n')
output(X,Y)
```

Taylor Tanita
A11216947

Output (reported in table):

```
Taylors-MacBook-Pro-8:assignment5 taylortanita$ python
Python 2.7.10 (default, Oct 23 2015, 18:05:06)
[GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from a5_ex import *
Problem 1

Linear SVM at 80/20: C = 0.775, error = 0.114285714286
RBF SVM at 80/20: C = 10.0, gamma = .1, error = 0.0285714285714

Linear SVM at 60/40: C = 0.775, error = 0.15
RBF SVM at 60/40: C = 100, gamma = default, error = 0.05

Linear SVM at 40/60: C = 0.1, error = 0.142857142857
RBF SVM at 40/60: C = 10, gamma = default, error = 0.0761904761905


Problem 2

Linear SVM at 80/20: C = 0.1, error = 0.0
RBF SVM at 80/20: C = 1.0, gamma = default, error = 0.0

Linear SVM at 60/40: C = 0.1, error = 0.0
RBF SVM at 60/40: C = 1, gamma = default, error = 0.0

Linear SVM at 40/60: C = 0.1, error = 0.0
RBF SVM at 40/60: C = 1, gamma = default, error = 0.0


Problem 3

Linear SVM at 80/20: C = 10.1008, error = 0.355555555556
RBF SVM at 80/20: C = 0.01, gamma = default, error = 0.488888888889

Linear SVM at 60/40: C = 0.001, error = 0.35
RBF SVM at 60/40: C = 0.01, gamma = default, error = 0.461111111111

Linear SVM at 40/60: C = 0.83125, error = 0.376383763838
RBF SVM at 40/60: C = 0.01, gamma = default, error = 0.568265682657
```

Taylor Tanita
A11216947

Output (when ran one more time):

```
Taylors-MacBook-Pro-8:assignment5 taylortanita$ python
Python 2.7.10 (default, Oct 23 2015, 18:05:06)
[GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from a5_ex import *
Problem 1

Linear SVM at 80/20: C = 1.0, error = 0.142857142857
RBF SVM at 80/20: C = 10.0, gamma = .1, error = 0.1

Linear SVM at 60/40: C = 10, error = 0.15
RBF SVM at 60/40: C = 10, gamma = .1, error = 0.0428571428571

Linear SVM at 40/60: C = 0.055, error = 0.166666666667
RBF SVM at 40/60: C = 10, gamma = .1, error = 0.0619047619048


Problem 2

Linear SVM at 80/20: C = 0.1, error = 0.0
RBF SVM at 80/20: C = 1.0, gamma = default, error = 0.0

Linear SVM at 60/40: C = 0.1, error = 0.0
RBF SVM at 60/40: C = 1, gamma = default, error = 0.0

Linear SVM at 40/60: C = 0.1, error = 0.0
RBF SVM at 40/60: C = 1, gamma = default, error = 0.0


Problem 3

Linear SVM at 80/20: C = 19.75166875, error = 0.362222222222
RBF SVM at 80/20: C = 0.01, gamma = default, error = 0.5

Linear SVM at 60/40: C = 0.001, error = 0.283333333333
RBF SVM at 60/40: C = 0.01, gamma = default, error = 0.472222222222

Linear SVM at 40/60: C = 0.001, error = 0.343173431734
RBF SVM at 40/60: C = 0.01, gamma = default, error = 0.479704797048
```