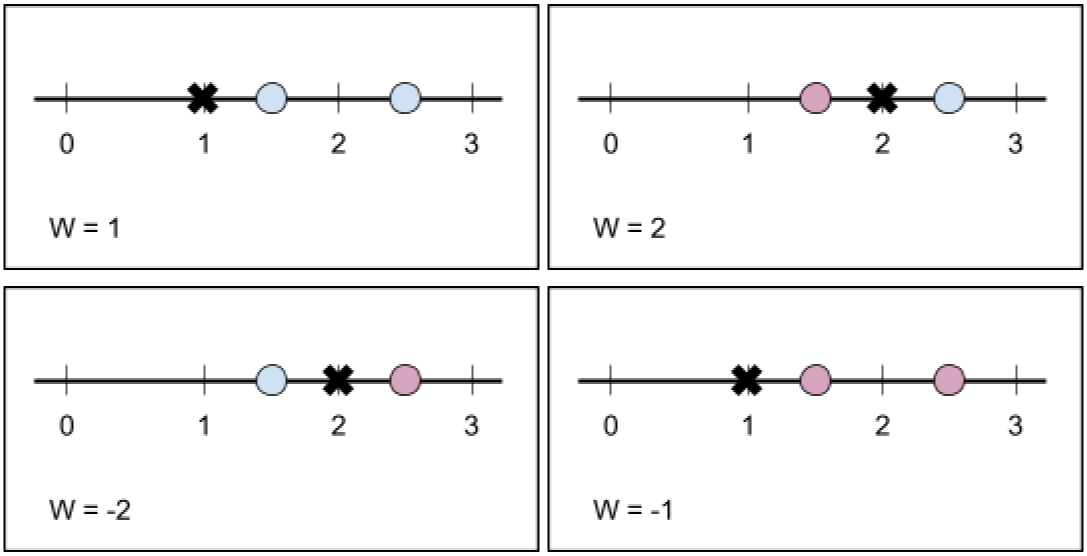
Taylor Tanita
A11216947

COGS 118A: Assignment 5

1 Shattering

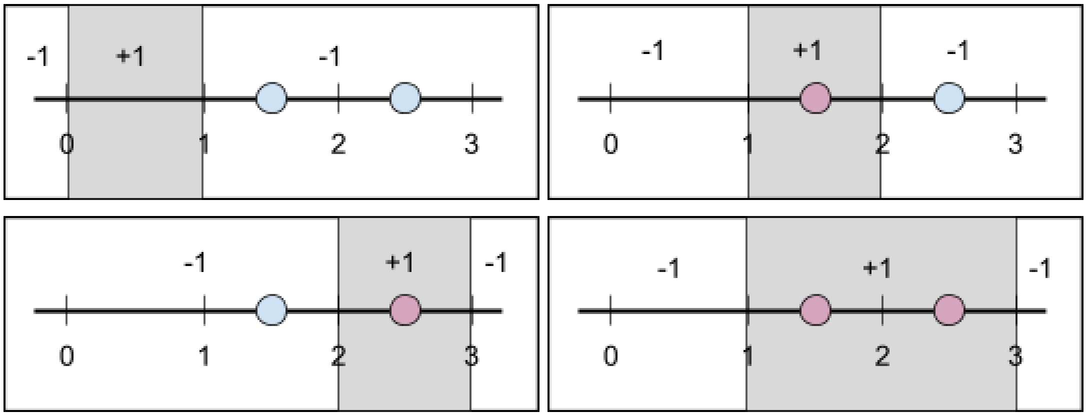
1. $f(x, w, b) = sign(x \cdot w + b)$

   VC-dimension = 2



2. $f(x, q, b) = sign(q \cdot x \cdot x + b)$

   VC-dimension = 2



$qx^2 + b = 0$

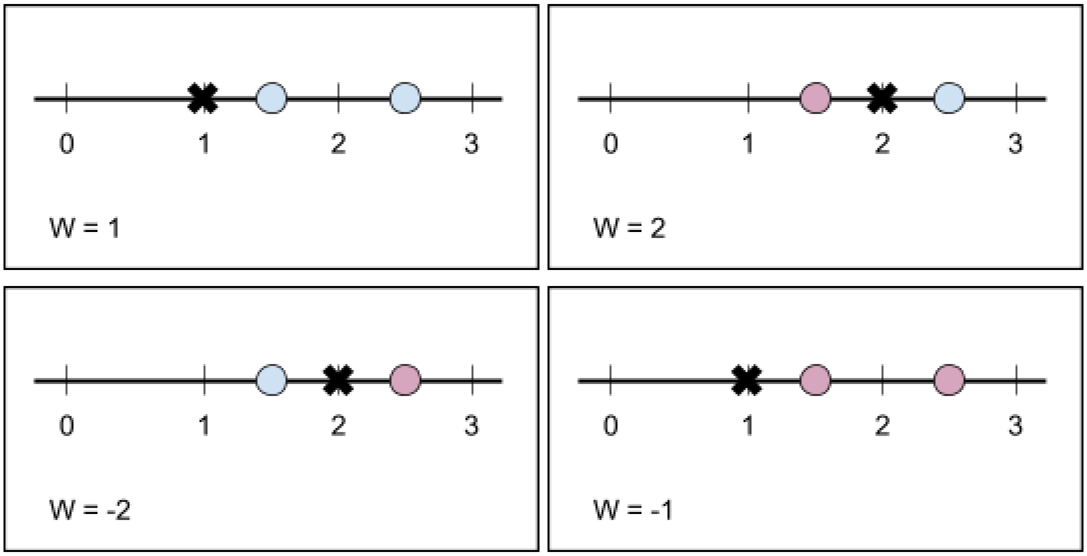
$qx^2 = b$ (because $b \in R,$ sign does not matter because you can just flip it)

$x^2 = b/q$

$x = \pm\sqrt{\frac{b}{q}}$

Can manipulate b and q to fit any interval you need

3. $f(x, w, b) \ = \ sign((x \cdot w + b)^2)$
VC-dimension = 2



$(x \cdot w + b)^2 \ = w^2x^2 + 2wbx + b^2$

Using quadratic formula, $x \ = \ \frac{-2wb \pm \sqrt{4w^2b^2 - 4w^2b^2}}{4w^2}$

$x \ = \ \frac{-2wb}{4w^2} \ = \ \frac{-b}{2w} \ = \ \frac{b}{2w}$ (because sign is irrelevant)

Can manipulate b and w to be any point

Taylor Tanita
A11216947

2 Ridge Regression

$$w* \ = \ arg\ min_w\ b\ \times \|w\|^2 \ + \ \sum_{i=1}^{n} (y_i \ - \ w \cdot x_i)^2$$

$$w* \ = \ arg\ min_w\ b\ \|w\|^2 \ + \ \|Y - X^T W\|^2$$

Let $f \ = \ b\ \|w\|^2 \ + \ \|Y - X^T W\|^2$

- $f' \ = \ 2bw - 2X(Y - X^T w)$
- Set $f' \ = \ 0$

  $2bw - 2XY + 2XX^T w \ = \ 0$

  $bw - XY + XX^T w \ = \ 0$
- Solve for w

  $(b + XX^T)w - XY = 0$

  $w \ = \ (bI + XX^T)^{-1} XY$

  $w \ = \ X(bI + X^T X)^{-1} Y$

$$w* \ = \ \sum_{i=1}^{n} \alpha_i\, x_i$$

$$w* \ = \ X\alpha$$

Set w’s to equal each other

$X\alpha \ = X(bI + X^T X)^{-1} Y$

$\alpha \ = (bI + X^T X)^{-1} Y$

3 Decision Tree

Output:

```
Q3: DECISION TREE
For 80/20 split
Optimal D: 2
Validation Error: 0.106704260652
Training Error: 0.0889679715302
Testing Error: 0.1

For 60/40 split
Optimal D: 8
Validation Error: 0.120933923957
Training Error: 0.00947867298578
Testing Error: 0.107142857143
```

80/20
Optimal D: 2
Training error: 8.90%
Validation error: 10.67%
Testing error: 10%

60/40
Optimal D: 8
Training error: .95%
Validation error: 12.09%
Testing error: 10.71%

4 k-Nearest Neighbors

Output:

```
Q4: k-NEAREST NEIGHBORS
For 80/20 split
occurrences where k[1,3,5,7] are best: [9, 0, 1, 0]
Optimal k: 1
Validation Error: 0.149135802469
Training Error: 0.145614035088
Testing Error: 0.171428571429

For 60/40 split
occurrences where k[1,3,5,7] are best: [5, 3, 2, 0]
Optimal k: 1
Validation Error: 0.158518518519
Training Error: 0.161129568106
Testing Error: 0.171428571429
```

80/20
Optimal k: 1
Training error: 14.56%
Validation error: 14.91%
Testing error: 17.14%

60/40
Optimal k: 1
Training error: 16.11%
Validation error: 15.85%
Testing error: 17.14%

Taylor Tanita
A11216947

a6.py

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

#method to convert data into 0,1's
def convert(elmt):
  if (elmt[0][0] == u'b'):
    return 1
  else:
    return 0

def convert_array(y):
  yy = np.array([[-1]])
  (length,_) = y.shape
  for i in range(length):
    yy = np.append(yy,[[convert(y[i])]],axis=0)
  return yy[1:]

# 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)

# helper method
# 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


###### METHODS FOR Q3 ######

# method to compute error using specific D
# reports the error associated with that D value
def compute_error(x,y,D):
  (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
  clf = tree.DecisionTreeClassifier(max_depth=D)
  # if 400 is indicated, it means the default D was used
  if (D == 400):
    clf = tree.DecisionTreeClassifier()
  for i in range(5):
    # removing 1/5 of the data to test on
    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

  error = error/5
#  print('error = '+str(error))
```

```
  return error

# runs cross validation multiple times
def run(x_train,y_train,D):
  error = 0.0
  for i in range(30):
    error = error + compute_error(x_train,y_train,D)
  error = error/30
  return error

# method to pick which D is best
# takes in training data and list of D values to test out
# reports the D value
def pick_D(x_train,y_train,D_choices):
  error = 1
  D = D_choices[0]
  iterate = len(D_choices)
  for i in range(iterate):
    e = run(x_train,y_train,D_choices[i])
#    print('for D = '+str(D_choices[i])+', e = '+str(e))
    if (e < error):
      error = e
      D = D_choices[i]
  return (D,error)

###### METHODS FOR Q4 ######

# computes euclidean distance
def distance(a,b):
  d = numpy.linalg.norm(a-b)
  return d

# returns k nearest neighbors of target
def get_neighbors(x_data,y_data,target,k):
  (length,_) = x_data.shape
  # distances contains tuples of (euclid norm, class)
  distances = []
  for i in range(length):
    d = np.linalg.norm(target-x_data[i])
    distances = distances + [(d,y_data[i])]
  # sorts distances from smallest to largest
  distances.sort(key=lambda x:x[0])
  # number of occurances of neighbors of 0 or 1 class
  num_0 = 0
  num_1 = 0
  # finds k nearest neighbors
  for i in range(k):
    (dis,clas) = distances[i]
    if (clas == 0):
      num_0 = num_0 + 1
    else:
      num_1 = num_1 + 1

  # return whichever neighbors were more present
  if (num_0 > num_1):
    return 0
  else:
    return 1

# takes in test data and x training data and returns a list of the predicted
# values for y
# returns list of predicted classes
def predict_class(x_data,y_data,x_target,k):
  (length,_) = x_target.shape
  classes = []
  for i in range(length):
    c = get_neighbors(x_data,y_data,x_target[i],k)
    classes = classes + [c]
  return classes


# cross validation
# returns error with associated k
def run_k(x,y,k):
  (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 = 0.0
  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)

    y_predict = predict_class(xx,yy,to_testx,k)
    e = test(y_predict,to_testy)
#    print('  error: ' + str(e))
    error = error + e

  error = error/5
  return error

# method that see's which k produces least error
def pick_k(x_train,y_train,k_choices):
  error = 1
  k = k_choices[0]
  iterate = len(k_choices)
  for i in range(iterate):
    e = run_k(x_train,y_train,k_choices[i])
#    print('k = '+str(k_choices[i])+', error: '+str(e))
    if (e < error):
      error = e
      k = k_choices[i]
  return (k,error)


# shuffling data and running many times
def multi_k(x,y,percent,k_choices):
  occ = [0,0,0,0]
  err = [0.0,0.0,0.0,0.0]

  for i in range(10):
    (x_train,y_train,x_test,y_test) = select_data(x,y,percent)
    (k,error) = pick_k(x_train,y_train,k_choices)

    if(k==1):
      occ[0] = occ[0]+1
      err[0] = err[0]+error
    if(k==3):
      occ[1] = occ[1]+1
      err[1] = err[1]+error
    if(k==5):
      occ[2] = occ[2]+1
      err[2] = err[2]+error
    if(k==7):
      occ[3] = occ[3]+1
      err[3] = err[3]+error
```

```
print('occurrences where k[1,3,5,7] are best: '+str(occ))
i = occ.index(max(occ))
final_error = err[i]/occ[i]
k = 1
if(i==1):
  k = 3
if(i==2):
  k = 5
if(i==3):
  k = 7
return(k,final_error)
```

Taylor Tanita
A11216947

a6_ex.py

```
from a6 import *

data = sio.loadmat('ionosphere.mat')
x = data['X']
y = data['Y']
y = convert_array(y)

##### Q3 #####
print('Q3: DECISION TREE')
### 80/20 split
percent = 0.8
print('For 80/20 split')
(x_train,y_train,x_test,y_test) = select_data(x,y,percent)
D_choices = [1,2,4,8,16,32,64,128,256,400]

# getting appropriate D value to use
(D,training_error) = pick_D(x_train,y_train,D_choices)
if (D==400):
  print('Optimal D: default')
else:
  print('Optimal D: '+str(D))
print('Validation Error: '+str(training_error))

clf = tree.DecisionTreeClassifier(max_depth=D)
if (D == 400):
  clf = tree.DecisionTreeClassifier()
clf.fit(x_train,y_train)
predict = clf.predict(x_train)
training_error = test(predict,y_train)
print('Training Error: '+str(training_error))

# Testing
clf = tree.DecisionTreeClassifier(max_depth=D)
if (D == 400):
  clf = tree.DecisionTreeClassifier()
clf.fit(x_train,y_train)
predict = clf.predict(x_test)
testing_error = test(predict,y_test)
print('Testing Error: '+str(testing_error)+'\n')


### 60/40 split
percent = 0.6
print('For 60/40 split')
(x_train,y_train,x_test,y_test) = select_data(x,y,percent)
D_choices = [1,2,4,8,16,32,64,128,256,400]
```

```
# getting appropriate D value to use
(D,training_error) = pick_D(x_train,y_train,D_choices)
if (D==400):
  print('Optimal D: default')
else:
  print('Optimal D: '+str(D))
print('Validation Error: '+str(training_error))

clf = tree.DecisionTreeClassifier(max_depth=D)
if (D == 400):
  clf = tree.DecisionTreeClassifier()
clf.fit(x_train,y_train)
predict = clf.predict(x_train)
training_error = test(predict,y_train)
print('Training Error: '+str(training_error))

# Testing
clf = tree.DecisionTreeClassifier(max_depth=D)
if (D == 400):
  clf = tree.DecisionTreeClassifier()
clf.fit(x_train,y_train)
predict = clf.predict(x_test)
testing_error = test(predict,y_test)
print('Testing Error: '+str(testing_error)+'\n')


###### Q4 #####
print('Q4: k-NEAREST NEIGHBORS')

percent = 0.8
(x_train,y_train,x_test,y_test) = select_data(x,y,percent)
print('For 80/20 split')
k_choices = [1,3,5,7]
(k,training_error) = multi_k(x_train,y_train,percent,k_choices)
print('Optimal k: '+str(k))
print('Validation Error: '+str(training_error))
e_train = run_k(x_train,y_train,k)
print('Training Error: '+str(e_train))
e = run_k(x_test,y_test,k)
print('Testing Error: '+str(e)+'\n')


percent = 0.6
(x_train,y_train,x_test,y_test) = select_data(x,y,percent)
print('For 60/40 split')
k_choices = [1,3,5,7]
(k,training_error) = multi_k(x_train,y_train,percent,k_choices)
print('Optimal k: '+str(k))
print('Validation Error: '+str(training_error))
e_train = run_k(x_train,y_train,k)
print('Training Error: '+str(e_train))
e = run_k(x_test,y_test,k)
print('Testing Error: '+str(e)+'\n')
```