20180221_COGS118a_Hw5

February 22, 2018

1 Shattering

1.1

See fig 1.

1.2

See fig 2

1.3

VC-Dimension = 0

Because the function is squared and there is no hyper parameter on the outside of the square term, all outputs from the function are positive. Hence, the classifier will never be able to classify a negative label correctly, thus there the VC dimesion is 0.

2 Support Vector Machine

```
In [5]: import scipy.io as sio
        import matplotlib.pyplot as plt
        import numpy as np
        import seaborn as sns
        import sys
        from sklearn import svm
        from sklearn.model_selection import GridSearchCV
        %config InlineBackend.figure_format = 'retina'
```

2.1 Q1 Support Vector Machine

2.1.1 Linear SVM

```
In [2]: # 1) Load data.

        X_and_Y = np.load('./arrhythmia.npy')     # Load data from file.
        print(f'X_and_Y.shape: {X_and_Y.shape}')
        np.random.shuffle(X_and_Y)     # Shuffle the data.
        X = X_and_Y[:, 0:X_and_Y.shape[1] - 1]     # First column to second last column: Features
```

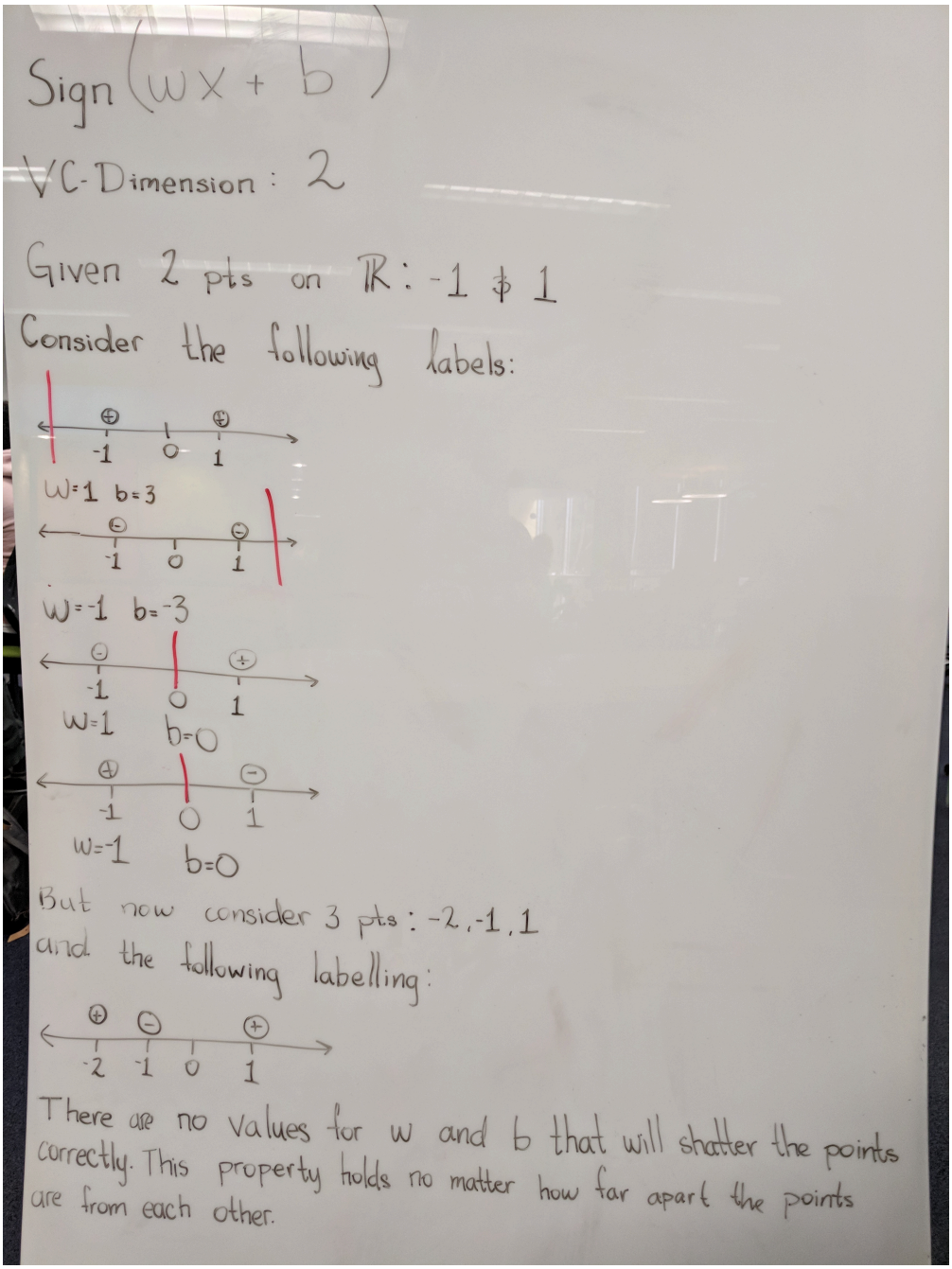

Sign (wx + b)
VC-Dimension: 2
Given 2 pts on ℝ: -1 & 1
Consider the following labels:
-1 0 1
w=1 b=3
-1 0 1
w=-1 b=-3
-1 0 1
w=1 b=0
-1 0 1
w=-1 b=0
But now consider 3 pts: -2, -1, 1
and the following labelling:
-2 -1 0 1
There are no values for w and b that will shatter the points correctly. This property holds no matter how far apart the points are from each other.

fig 1

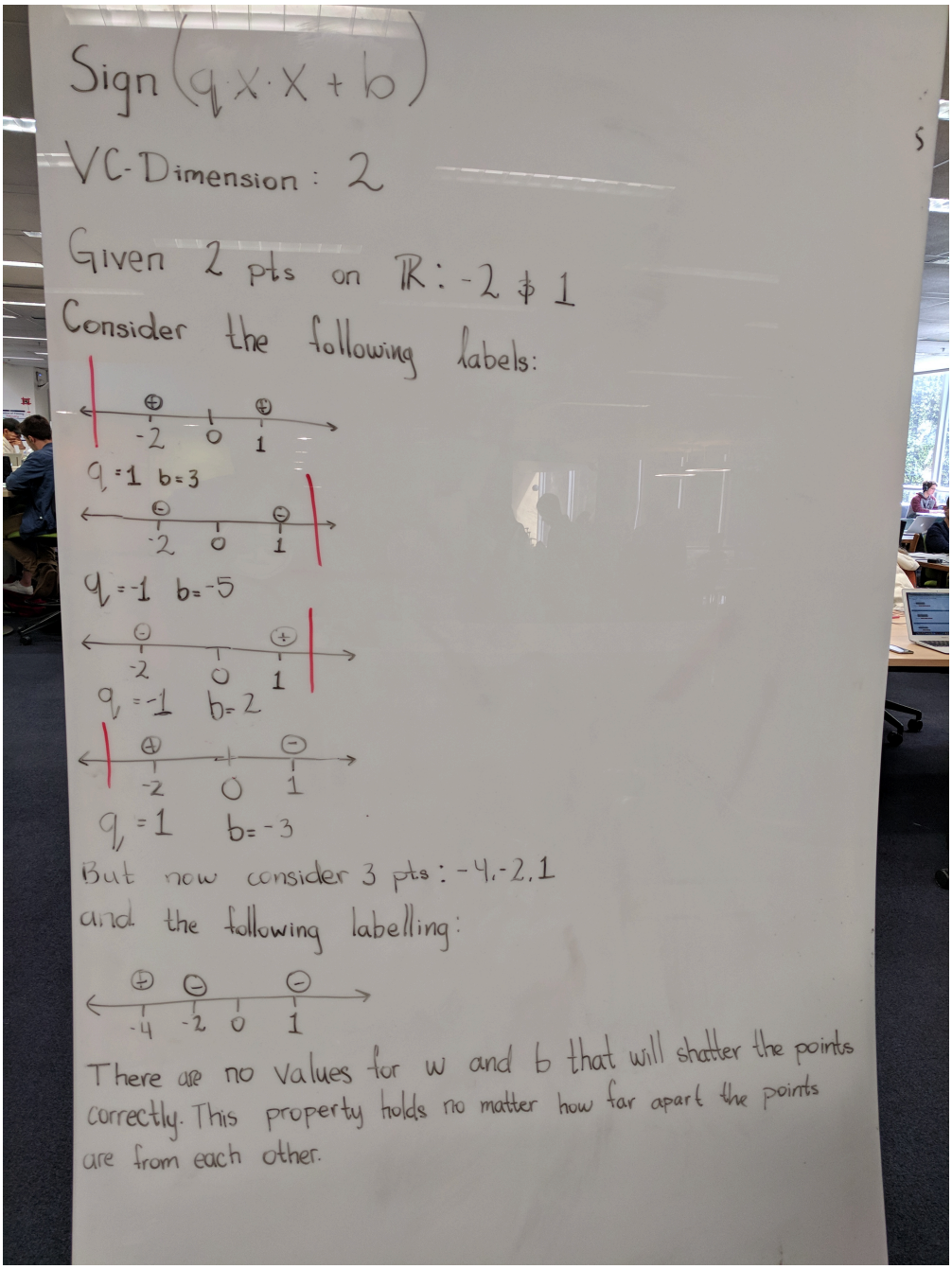

Sign $(q_1 x \cdot X + b)$
VC-Dimension : 2
Given 2 pts on $\mathbb{R}$: -2 & 1
Consider the following labels:
$q_1 = 1$ $b = 3$
$q_1 = -1$ $b = -5$
$q_1 = -1$ $b = 2$
$q_1 = 1$ $b = -3$
But now consider 3 pts : -4, -2, 1
and the following labelling:
There are no values for w and b that will shatter the points correctly. This property holds no matter how far apart the points are from each other.

fig 2

```
        Y = X_and_Y[:, X_and_Y.shape[1] - 1]    # Last column: Labels (0 or 1)
        print(X.shape, Y.shape)       # Check the shapes.
```

```
X_and_Y.shape: (452, 280)
(452, 279) (452,)
```

In [3]:
```
        # 2) Split the dataset into 2 parts:
        #    (a) Training set + Validation set  (80% of all data points)
        #    (b) Test set                       (20% of all data points)
        eighty_percent = round(len(X) * .8)
        X_train_val = X[:eighty_percent] # Get features from train + val set.
        X_test      = X[eighty_percent:] # Get features from test set.
        Y_train_val = Y[:eighty_percent] # Get labels from train + val set.
        Y_test      = Y[eighty_percent:] # Get labels from test set.
        print(X_train_val.shape, X_test.shape, Y_train_val.shape, Y_test.shape)
```

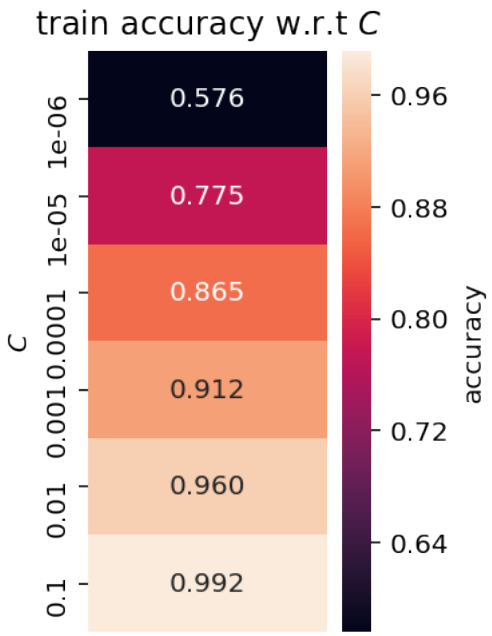
```
(362, 279) (90, 279) (362,) (90,)
```

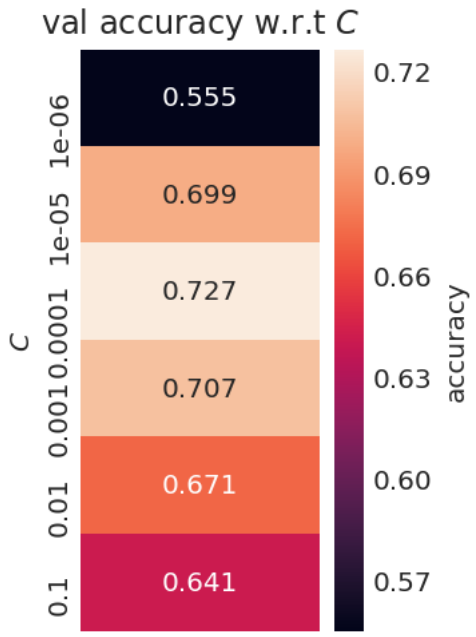
In [4]:
```
        # 3) Consider linear kernel. Perform grid search for best C
        #    with 3-fold cross-validation. You can use svm.SVC() for SVM
        #    classifier and use GridSearchCV() to perform such grid search.
        #    For more details, please refer to the sklearn documents:
        #         http://scikit-learn.org/stable/modules/svm.html
        #         http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridS

        svc = svm.SVC()
        C_list     = [10 ** -6, 10 ** -5, 10 ** -4, 10 ** -3, 10 ** -2, 10 ** -1] # Different C
        parameters = {'kernel':['linear'], 'C': C_list}
        classifier = GridSearchCV(svc, parameters, scoring='accuracy', return_train_score=True)
        classifier.fit(X_train_val, Y_train_val)
```

Out[4]:
```
GridSearchCV(cv=None, error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'kernel': ['linear'], 'C': [1e-06, 1e-05, 0.0001, 0.001, 0.01, 0.1]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring='accuracy', verbose=0)
```

In [5]:
```
        # 4) Draw heatmaps for result of grid search and find
        #    best C for validation set.

        def draw_heatmap_linear(acc, acc_desc, C_list):
            plt.figure(figsize = (2,4))
            ax = sns.heatmap(acc, annot=True, fmt='.3f', yticklabels=C_list, xticklabels=[])
```

```
    ax.collections[0].colorbar.set_label("accuracy")
    ax.set(ylabel='$C$')
    plt.title(acc_desc + ' w.r.t $C$')
    sns.set_style("whitegrid", {'axes.grid' : False})
    plt.show()

#
# You can use the draw_heatmap_linear() to draw a heatmap to visualize
# the accuracy w.r.t. C and gamma. Some demo code is given below as hint:
#
# demo_acc        = np.array([[0.8],
#                             [0.7]])
# demo_C_list     = [0.1, 1]
# draw_heatmap_linear(demo_acc, 'demo accuracy', demo_C_list)
#

train_acc = classifier.cv_results_['mean_train_score'].reshape(-1, 1)
draw_heatmap_linear(train_acc, 'train accuracy', C_list)

val_acc = classifier.cv_results_['mean_test_score'].reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', C_list)
```






```
In [6]: C_star = .0001 # based on 'val accuracy' heatmap above.
```

```
In [7]: # 5) Use the best C to calculate the test accuracy.
        from sklearn.metrics import accuracy_score
        clf = svm.SVC(C = C_star)
        clf.fit(X_train_val, Y_train_val)
        y_pred = clf.predict(X_test)

        test_acc = accuracy_score(Y_test, y_pred)
        print(test_acc)
```

```
0.555555555556
```

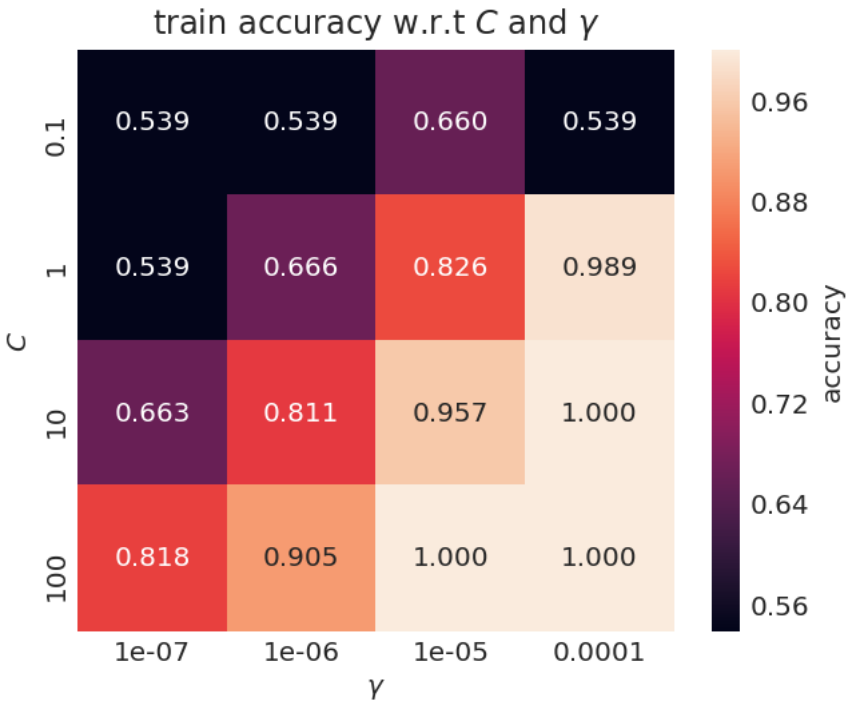
2.1.2 SVM with RBF Kernel

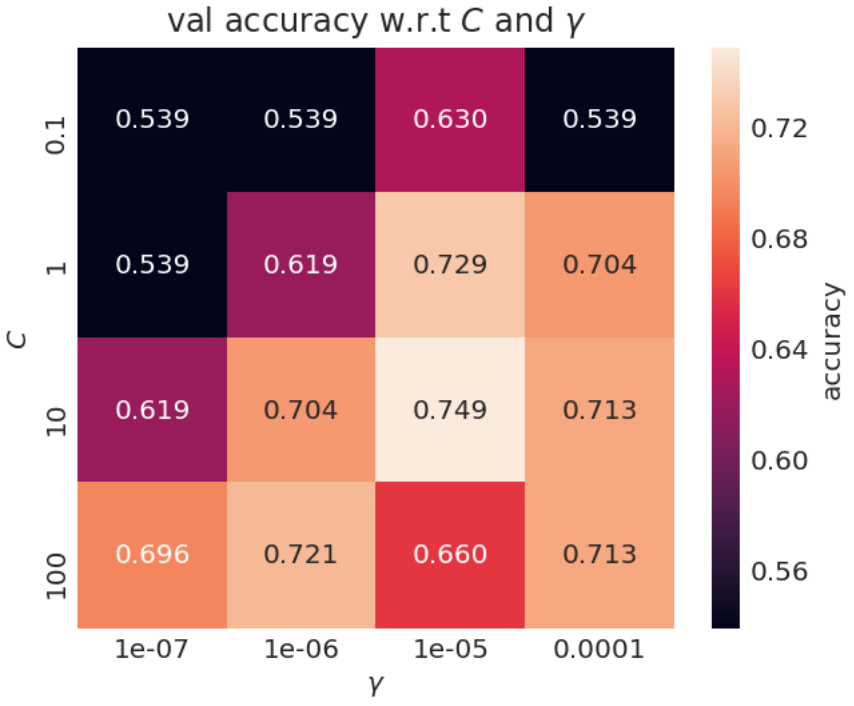
```
In [8]: # 1) Consider RBF kernel. Perform grid search for best C and gamma
        #    with 3-fold cross-validation. You can use sum.SVC() for SVM
        #    classifier and use GridSearchCV() to perform such grid search.
        #    For more details, please refer to the sklearn documents:
        #         http://scikit-learn.org/stable/modules/svm.html
        #         http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridS

        svc = svm.SVC()
        C_list     = [0.1, 1, 10, 100] # Different C to try.
        gamma_list = [10 ** -7 , 10 ** -6 , 10 ** -5 , 10 ** -4] # Different gamma to try.
```

```
        parameters = {'kernel':['rbf'], 'C': C_list, 'gamma': gamma_list}
        classifier = GridSearchCV(svc, parameters, scoring='accuracy', return_train_score=True)
        classifier.fit(X_train_val, Y_train_val)
```

```
Out[8]: GridSearchCV(cv=None, error_score='raise',
               estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
          decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
          max_iter=-1, probability=False, random_state=None, shrinking=True,
          tol=0.001, verbose=False),
               fit_params=None, iid=True, n_jobs=1,
               param_grid={'kernel': ['rbf'], 'C': [0.1, 1, 10, 100], 'gamma': [1e-07, 1e-06, 1e
               pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
               scoring='accuracy', verbose=0)
```

```
In [10]: # 2) Draw heatmaps for result of grid search and find
         #    best C and gamma for validation set.

         def draw_heatmap_RBF(acc, acc_desc, gamma_list, C_list):
             plt.figure(figsize = (5,4))
             ax = sns.heatmap(acc, annot=True, fmt='.3f',
                              xticklabels=gamma_list, yticklabels=C_list)
             ax.collections[0].colorbar.set_label("accuracy")
             ax.set(xlabel = '$\gamma$', ylabel='$C$')
             plt.title(acc_desc + ' w.r.t $C$ and $\gamma$')
             sns.set_style("whitegrid", {'axes.grid' : False})
             plt.show()

         #
         # You can use the draw_heatmap_RBF() to draw a heatmap to visualize
         # the accuracy w.r.t. C and gamma. Some demo code is given below as hint:
         #
         # demo_acc        = np.array([[0.8, 0.7],
         #                             [0.7, 0.9]])
         # demo_C_list     = [0.1, 1]
         # demo_gamma_list = [0.01, 0.1]
         # draw_heatmap_RBF(demo_acc, 'demo accuracy', demo_gamma_list, demo_C_list)
         #

         train_acc = classifier.cv_results_['mean_train_score'].reshape(4,4)
         draw_heatmap_RBF(train_acc, 'train accuracy', gamma_list, C_list)

         val_acc   = classifier.cv_results_['mean_test_score'].reshape(4,4)
         draw_heatmap_RBF(val_acc, 'val accuracy',gamma_list, C_list)
```


train accuracy w.r.t C and γ
0.539
0.539
0.660
0.539
0.539
0.666
0.826
0.989
0.663
0.811
0.957
1.000
0.818
0.905
1.000
1.000
0.1
1
10
100
C
1e-07
1e-06
1e-05
0.0001
γ
0.96
0.88
0.80
0.72
0.64
0.56
accuracy


val accuracy w.r.t C and γ
0.539
0.539
0.630
0.539
0.539
0.619
0.729
0.704
0.619
0.704
0.749
0.713
0.696
0.721
0.660
0.713
0.1
1
10
100
C
1e-07
1e-06
1e-05
0.0001
γ
0.72
0.68
0.64
0.60
0.56
accuracy

In [11]:
```
C_star = 10
gamma_star = 10 ** -5
```

In [12]:
```
# 3) Use the best C and gamma to calculate the test accuracy.

clf = svm.SVC(C = C_star, gamma = gamma_star)
clf.fit(X_train_val, Y_train_val)
y_pred = clf.predict(X_test)

test_acc = accuracy_score(Y_test, y_pred)
print(test_acc)
```

```
0.811111111111
```

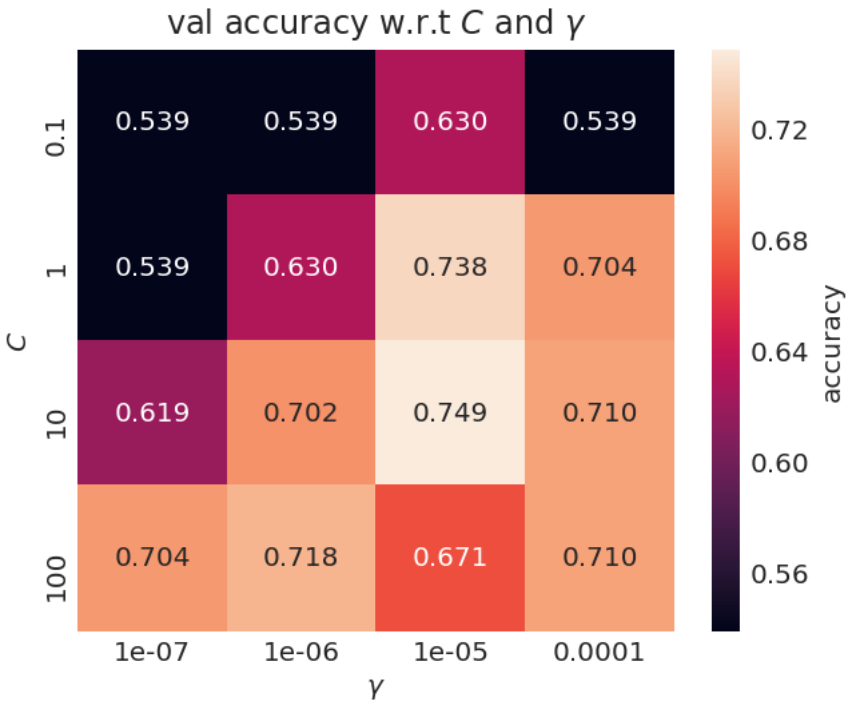
2.1.3 Re-implementation of Cross-validation and Grid Search

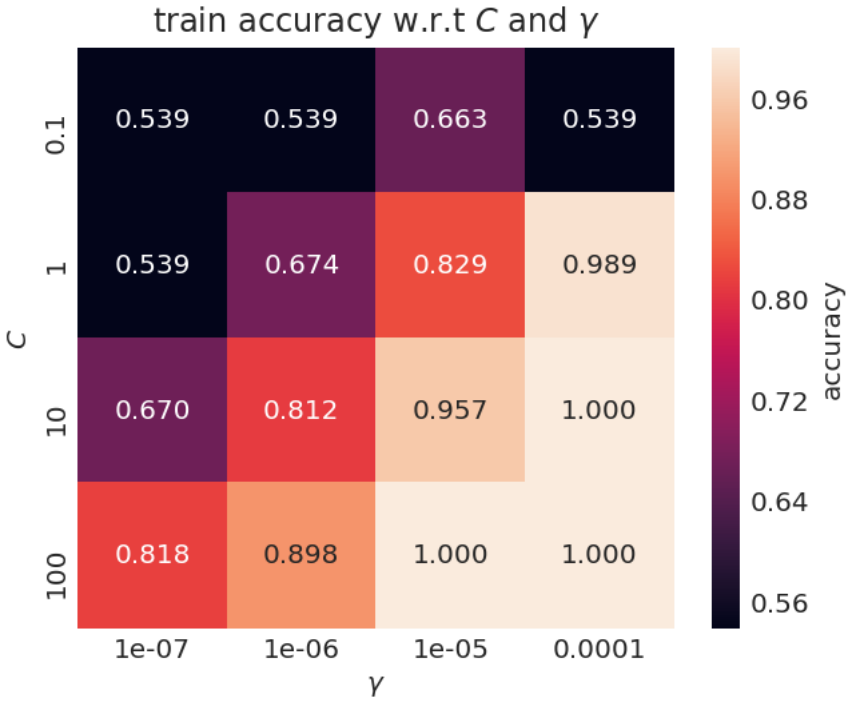
In [13]:
```
# 1) Implement a simple cross-validation.

def simple_cross_validation(X_train_val, Y_train_val, C, gamma, fold):
    """
    A simple cross-validation function.
    We assume the SVM with the RBF kernel.

    X_train_val: Features for train and val set.
                 Shape: (num of data points, num of features)
    Y_train_val: Labels for train and val set.
                 Shape: (num of data points,)
    C:           Parameter C for SVM.
    gamma:       Parameter gamma for SVM.
    fold:        The number of folds to do the cross-validation.

    Return the average accuracy on validation set.
    """
    val_acc_list = []
    train_acc_list = []
    start = 0
    for i in range(fold):
        end = int(min(round(len(X_train_val) / fold, 0) * (i + 1), len(X_train_val)))
        X_val = X_train_val[start: end]
        Y_val = Y_train_val[start: end]
        X_train = X_train_val[0: start]
        X_train = np.concatenate((X_train, X_train_val[end:]))
        Y_train = Y_train_val[0: start]
        Y_train = np.concatenate((Y_train, Y_train_val[end:]))

        clf = svm.SVC(C = C, gamma = gamma)
        clf.fit(X_train, Y_train)
```

```
            val_acc_list.append(clf.score(X_val, Y_val))
            train_acc_list.append(clf.score(X_train, Y_train))

            start = end

        return sum(val_acc_list) / len(val_acc_list), \
               sum(train_acc_list) / len(train_acc_list)
```

In [14]:
```
# 2) Implement the grid search function.
def simple_GridSearchCV_fit(X_train_val, Y_train_val, C_list, gamma_list, fold):
    """
    A simple grid search function for C and gamma with cross-validation.
    We assume the SVM with the RBF kernel.

    X_train_val: Features for train and val set.
                 Shape: (num of data points, num of features)
    Y_train_val: Labels for train and val set.
                 Shape: (num of data points,)
    C_list:      The list of C values to try.
    gamma_list:  The list of gamma values to try.
    fold:        The number of folds to do the cross-validation.

    Return the val and train accuracy matrix of cross-validation.
    All combinations of C and gamma are
    included in the matrix. Shape: (len(C_list), len(gamma_list))
    """
    val_acc_matrix   = np.empty([len(C_list), len(gamma_list)])
    train_acc_matrix = np.empty([len(C_list), len(gamma_list)])

    for i in range(len(C_list)):
        for j in range(len(gamma_list)):
            accs = simple_cross_validation(X_train_val, Y_train_val, C_list[i], gamma_l
            val_acc_matrix[i][j] = accs[0]
            train_acc_matrix[i][j] = accs[1]

    return val_acc_matrix, train_acc_matrix
```

In [15]:
```
# 3) Perform grid search with 3-fold cross-validation.
#    Draw heatmaps for result of grid search and find
#    best C and gamma for validation set.
val_acc_matrix, train_acc_matrix = \
    simple_GridSearchCV_fit(X_train_val, Y_train_val, C_list, gamma_list, 3)

draw_heatmap_RBF(val_acc_matrix, 'val accuracy', gamma_list, C_list)
draw_heatmap_RBF(train_acc_matrix, 'train accuracy', gamma_list, C_list)
```


val accuracy w.r.t C and γ
0.539 0.539 0.630 0.539
0.539 0.630 0.738 0.704
0.619 0.702 0.749 0.710
0.704 0.718 0.671 0.710
0.1
1
10
100
C
1e-07
1e-06
1e-05
0.0001
γ
0.72
0.68
0.64
0.60
0.56
accuracy


train accuracy w.r.t C and γ
0.539 0.539 0.663 0.539
0.539 0.674 0.829 0.989
0.670 0.812 0.957 1.000
0.818 0.898 1.000 1.000
0.1
1
10
100
C
1e-07
1e-06
1e-05
0.0001
γ
0.96
0.88
0.80
0.72
0.64
0.56
accuracy

2.1.4 Implement Linear SVM

In [50]:
```
from numpy import linalg as LA
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import math
%config InlineBackend.figure_format = 'retina'
%matplotlib inline
from sklearn.utils import shuffle
import scipy.io as sio
plt.rcParams['figure.figsize'] = 8,8

X_and_Y_train = np.load('./hw4-q2-logistic-train.npy')
X_train = X_and_Y_train[:, :2]    # Shape: (70,2)
X_train = np.matrix(np.hstack((np.ones((len(X_train), 1)), X_train))) # Shape: (70, 3)
Y_train = X_and_Y_train[:, 2]    # Shape: (70,)
```

In [51]:
```
X_and_Y_test = np.load('./hw4-q2-logistic-test.npy')
X_test = X_and_Y_test[:, :2]   # Shape: (30,2)
X_test = np.matrix(np.hstack((np.ones((len(X_test), 1)), X_test))) # Shape: (30, 3)
Y_test = X_and_Y_test[:, 2]    # Shape: (70,)
```

In [52]:
```
mpl.style.use('seaborn')

fig = plt.figure()
plt.scatter([X_train[Y_train==0, 1]], [X_train[Y_train==0, 2]], marker='x', color='b',
plt.scatter([X_train[Y_train==1, 1]], [X_train[Y_train==1, 2]], marker='o', color='r',
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend(loc='upper right', fontsize=10)
plt.title('Training data')
plt.show()
```


```
In [198]: def L_prime_theta(X, Y, theta):
              v = np.zeros((X_train.shape[1], 1)).T
              for i, x in enumerate(X):
                  if (Y[i] * X[i].dot(theta) < 1):
                      v += (2*theta).T - (X[i] * Y[i])
                  else:
                      v += (2*theta).T
              return  v.T
```

```
In [177]: def L_theta(X, Y, theta):
              return (LA.norm(theta, 2) ** 2) +  np.sum(np.maximum(np.zeros((X.shape[0], 1)), 1
```

```
In [199]: learning_rate = 0.00001
          n_iter = 10000
          theta = np.zeros((X_train.shape[1], 1))
          # We will keep track of training loss over iterations
          iterations = [0]
          L_theta_list = [L_theta(X_train, Y_train, theta)]
          for i in range(n_iter):
              gradient = L_prime_theta(X_train, Y_train, theta)
              theta_new = theta - learning_rate * gradient
              iterations.append(i+1)
```

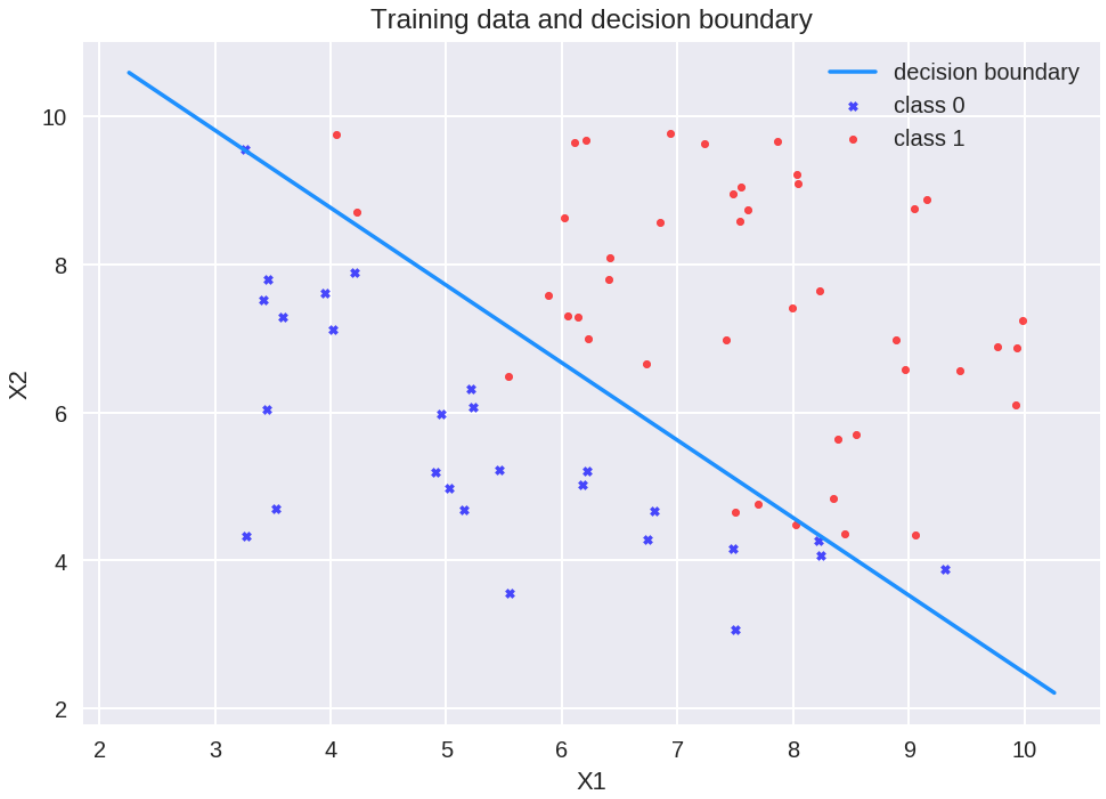
```
            L_theta_list.append(L_theta(X_train, Y_train, theta_new))

            if np.linalg.norm(theta_new - theta, ord = 1) < 0.0001:
                print("gradient descent has converged after " + str(i) + " iterations")
                break
            theta = theta_new

        print ("theta vector: \n" + str(theta))
```

```
gradient descent has converged after 57 iterations
theta vector:
[[ 0.01094796]
 [ 0.08087694]
 [ 0.0772241 ]]
```

Results on Training Data

```
In [208]: prediction = X_train.dot(theta) > 1
          training_accuracy = np.sum(prediction == Y_train.reshape(-1, 1))*1.0/X_train.shape[0]
          print(prediction.shape, Y_train.shape)
          print ("training accuracy: " + str(training_accuracy))

          x = np.arange(np.min(X_train[:,1])-1,np.max(X_train[:,1])+1,1.0)
          y = (-theta[1][0]-theta[1][0]*x)/theta[2][0] + 14
          plt.scatter([X_train[Y_train==0, 1]], [X_train[Y_train==0, 2]], marker='x', color='b',
          plt.scatter([X_train[Y_train==1, 1]], [X_train[Y_train==1, 2]], marker='o', color='r',


          plt.xlabel('X1')
          plt.ylabel('X2')
          plt.plot(x,y.T, 'dodgerblue', label='decision boundary')
          plt.title('Training data and decision boundary')

          plt.legend(loc='upper right', fontsize=10)
          plt.show()
```

```
(70, 1) (70,)
training accuracy: 0.928571428571
```



Results on Test Data

```
In [213]: prediction = X_test.dot(theta) > 1
          testing_accuracy = np.sum(prediction == Y_test.reshape(-1, 1))*1.0/X_test.shape[0]
          print ("testing accuracy: " + str(testing_accuracy))

          x = np.arange(np.min(X_train[:,1])-1,np.max(X_train[:,1])+1,1.0)
          y = (-theta[0][0]-theta[1][0]*x)/theta[2][0] + 12
          plt.scatter([X_test[Y_test==0, 1]], [X_test[Y_test==0, 2]], marker='x', color='b', alp
          plt.scatter([X_test[Y_test==1, 1]], [X_test[Y_test==1, 2]], marker='o', color='r', alp


          plt.xlabel('X1')
          plt.ylabel('X2')
          plt.plot(x,y.T, 'dodgerblue', label='decision boundary')
          plt.title('Testing data and decision boundary')

          plt.legend(loc='upper right', fontsize=10)

testing accuracy: 0.833333333333
```

Out[213]: <matplotlib.legend.Legend at 0x7fb1e64908d0>