3 (10 points) Shattering

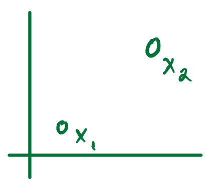
Use shattering to derive the VC-dimension for classifiers below. Show your work.

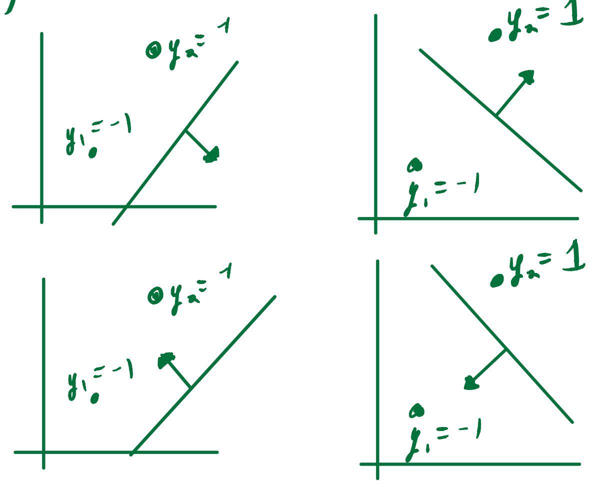
1) $f(x; w, b) = \mathrm{sign}(x \times w + b)$

2) $f(x; w, b) = \mathrm{sign}((x \times w + b)^2)$

where $x, w, q, b \in \mathbb{R}$, and w, q and b are free parameters.

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

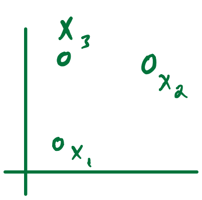


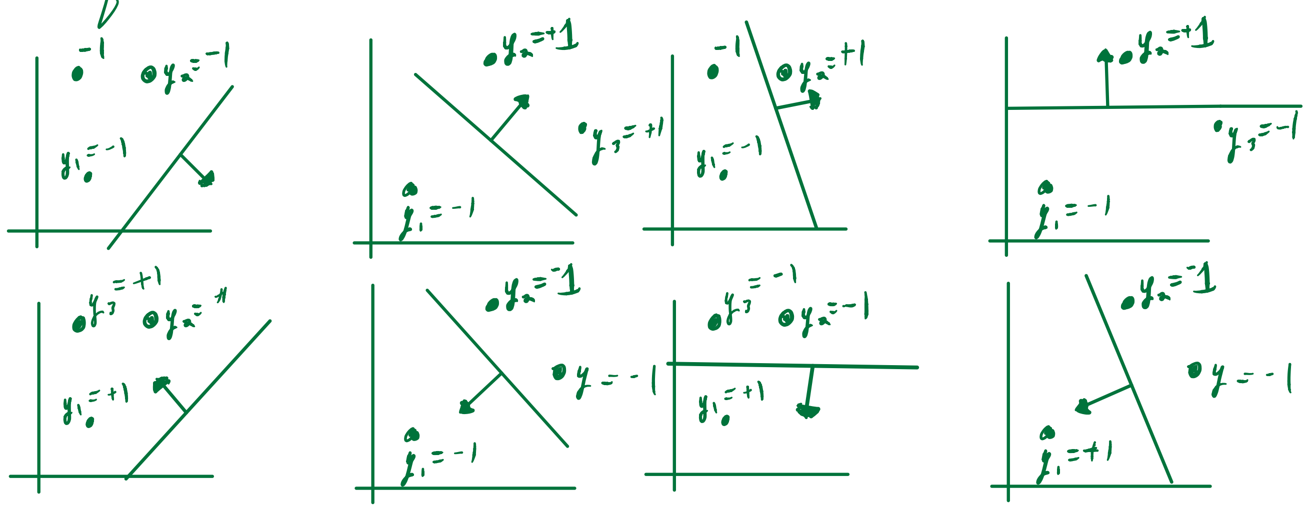


✓

2. $f(x; w, b) = \mathrm{sign}((x \cdot w + b)^2)$

for 3 points



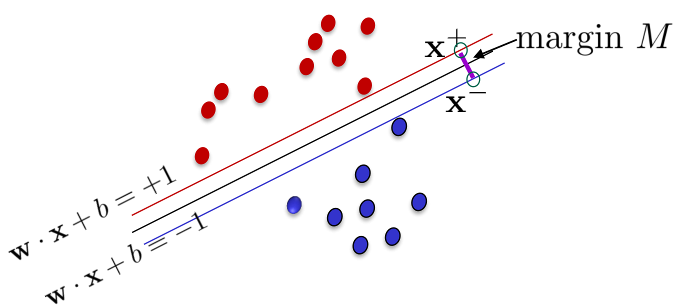


4 (10 points) Support Vector Machine 1

As shown in the figure, two boundaries are shifted to be parallel to the decision boundary in black, which is $\mathbf{w} \cdot \mathbf{x} + b = 0$. The equations of the boundaries are given in the figure. We first pick an arbitrary point $\mathbf{x}^-$ on the negative plane such that $\mathbf{w} \cdot \mathbf{x}^- + b = -1$; we then draw a line that passes $\mathbf{x}^-$ and is perpendicular to the negative plane; the intersection between this line and the positive plane can be denoted as $\mathbf{x}^+$ with $\mathbf{w} \cdot \mathbf{x}^+ + b = 1$. We thus have the following equations:

$$\mathbf{w} \cdot \mathbf{x}^- + b = -1,$$
$$\mathbf{w} \cdot \mathbf{x}^+ + b = +1,$$
$$\mathbf{x}^+ = \mathbf{x}^- + \lambda \mathbf{w},$$

where $\mathbf{x}^-$ is any point that lies on the blue boundary and $\mathbf{x}^+$ is any point that lies on the red boundary, $\mathbf{w}, b$ are given, and λ is an unknown parameter. Margin, M, is the distance between the two boundaries, which can be calculated as $M = ||\mathbf{x}^+ - \mathbf{x}^-||_2 = \sqrt{< \lambda\mathbf{w}, \lambda\mathbf{w} >}$. Please derive M to be parameterized by known parameters only (not containing λ).



Hint: (1) Derive λ based on the three equations given above (2) Plug in the value of λ you derive in (1) to $M = ||\mathbf{x}^+ - \mathbf{x}^-||_2 = \sqrt{< \lambda\mathbf{w}, \lambda\mathbf{w} >}$.

$x^+ = x^- + \lambda w$

$M = |\lambda w|$

$\lambda = \frac{2}{\langle w, w \rangle}$

$||w||_2 = \sqrt{\langle w, w \rangle}$

$M = ||x^+ - x^-||_2$

$= ||\lambda w||_2 \in \mathbb{R}$

$M = ||\lambda w||_2 = \frac{2\sqrt{\langle w, w \rangle}}{\langle w, w \rangle}$

$= \frac{2}{\sqrt{\langle w, w \rangle}}$

$x^- = \frac{-1+b}{w}$ $\quad x^+ = \frac{1-b}{w}$

$\lambda w = x^+ + x^-$

$\lambda w = \frac{1-b-(-1-b)}{w} = \frac{2}{w}$

$\lambda = \frac{2}{w_2}$

$M = \sqrt{\frac{2w}{w} - \frac{2w}{w^2}} = \sqrt{\frac{2}{w} - \frac{2}{w}}$

$= \boxed{\frac{2}{w}}$

HW 5 - LDA, VC-Dimensions, Shattering, SVM, Cross-Validation, and Grid Search

1.1 - Structural Risk Minimization:

D - *We do not need to compute the testing error for each model to perform SRM*

1.2 - Cross-Validation:

D - *a higher k, does not neccessarily lead to a more optimal result*

Q2 LDA

In [1]:

```
import numpy as np
from numpy.linalg import inv, norm
import matplotlib.pyplot as plt
%config InlineBackend.figure_format = 'retina'
```

In [2]:

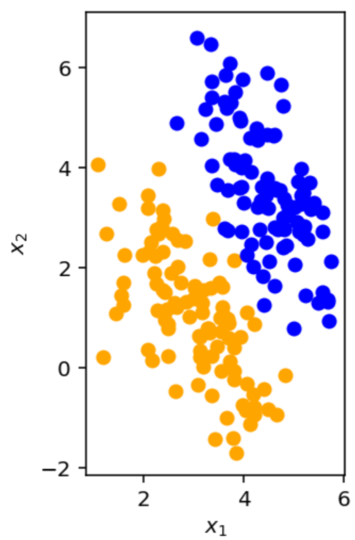
```
def print_latex(mat):
    s = r'\begin{bmatrix}' + '\n'
    for i in range(mat.shape[0]):
        for j in range(mat.shape[1]):
            if j != mat.shape[1] - 1:
                s += '{: .4f} & '.format(mat[i,j])
            else:
                s += '{: .4f} \\\\\n'.format(mat[i,j])
    s += r'\end{bmatrix}'
    print(s)
```

In [3]:
```
# Load the data and visualize.
Xs = np.load('lda.npy')

X_0 = np.matrix(Xs[:, 0:2]).T  # Shape: (2, 100).
X_1 = np.matrix(Xs[:, 2:4]).T  # Shape: (2, 100).

print(X_0.shape, X_1.shape)
plt.scatter(X_0[0].tolist(), X_0[1].tolist(), color='orange')
plt.scatter(X_1[0].tolist(), X_1[1].tolist(), color='blue')
plt.axis('scaled')
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.show()
```

```
(2, 100) (2, 100)
```



In [16]:
```
# (a) Compute mean of each class.
mu_0_0 = X_0[0].mean()
mu_0_1 = X_0[1].mean()
mu_0 = np.array([mu_0_0, mu_0_1])
mu_0 = mu_0.reshape(2,1)
mu_1_0 = X_1[0].mean()
mu_1_1 = X_1[1].mean()
mu_1 = np.array([mu_0_0, mu_0_1])
mu_1 = mu_1.reshape(2,1)

print(mu_0.shape, mu_1.shape)
print('mu_0=\n{},\nmu_1=\n{}'.format(mu_0, mu_1))
```

```
(2, 1) (2, 1)
mu_0=
[[2.98351552]
 [1.06453902]],
mu_1=
[[2.98351552]
 [1.06453902]]
```

In [5]:
```
# (b) Compute the covariance matrix for each class, Sigma_0 and Sigma_
1.
# Sigma_0, Sigma_1 = np.cov(X_0, rowvar = True), np.cov(X_1, rowvar =
True)
Sigma_0, Sigma_1 = np.cov(X_0), np.cov(X_1)
print_latex(Sigma_0)
print_latex(Sigma_1)
```

```
\begin{bmatrix}
 0.7063 & -0.6905 \\
-0.6905 &  1.6147 \\
\end{bmatrix}
\begin{bmatrix}
 0.4898 & -0.5748 \\
-0.5748 &  1.6767 \\
\end{bmatrix}
```

$$\begin{bmatrix} 0.7063 & -0.6905 \\ -0.6905 & 1.6147 \end{bmatrix}$$
$$\begin{bmatrix} 0.4898 & -0.5748 \\ -0.5748 & 1.6767 \end{bmatrix}$$

In [17]:
```
# (c) Find the optimal w_star and w_tilde_star with unit length.
# numerator = (mu_0 - mu_1)**2 # TODO: Multiply in Weights
# inverse_sigma = np.linalg.inv((Sigma_0 + Sigma_1)) # TODO: Mult in W
# denominator = np.linalg.norm(np.multiply(inverse_sigma,(mu_0 - mu_1)
))
# print(denominator)
# # w_star = np.dot(numerator,denominator)
w_star = np.dot(inv(Sigma_0 + Sigma_1),(mu_0 - mu_1))
# w_star = np.dot((Sigma_0 + Sigma_1)**-1,((mu_0 - mu_1)))
w_tilde_star = (w_star/np.sqrt(sum(w_star**2)))
print(w_star.shape, w_tilde_star.shape)
print('w_star=\n{},\nw_tilde_star=\n{}'.format(w_star, w_tilde_star))
```

```
(2, 1) (2, 1)
w_star=
[[0.]
 [0.]],
w_tilde_star=
[[nan]
 [nan]]
```

```
/Users/brody/anaconda2/envs/env36/lib/python3.5/site-packages/ipyker
nel_launcher.py:9: RuntimeWarning: invalid value encountered in true
_divide
  if __name__ == '__main__':
```

$$\begin{bmatrix} -2.78310591 - 1.06989259] \\ -1.06989259 - 1.01138952 \end{bmatrix}$$

$$\begin{bmatrix} -0.83693718 - 0.32173871 \\ -0.32173871 - 0.30414563 \end{bmatrix}$$

In [20]:

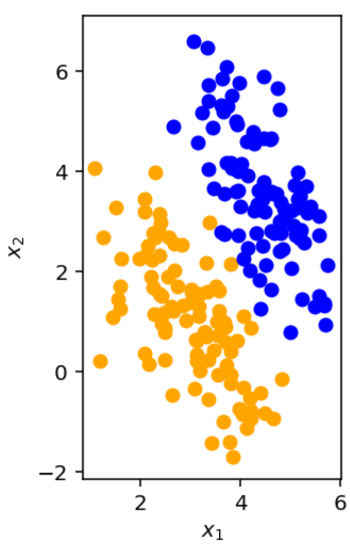
```
# (d) Compute the projection and plot the figure.
Xproj_0 = np.zeros((2,100))
Xproj_1 = np.zeros((2,100))

for i in range(len(X_0.T)):
    Xproj_0[0][i] = (np.dot(np.dot(w_tilde_star,X_0.T[i]),w_tilde_star
))[0]
    Xproj_0[1][i] = (np.dot(np.dot(w_tilde_star,X_0.T[i]),w_tilde_star
))[1]

    Xproj_1[0][i] = (np.dot(np.dot(w_tilde_star,X_0.T[i]),w_tilde_star
))[0]
    Xproj_1[1][i] = (np.dot(np.dot(w_tilde_star,X_0.T[i]),w_tilde_star
))[1]

print(XProj_0.shape, Xproj_1.shape)
plt.scatter(X_0[0].tolist(), X_0[1].tolist(), color='orange')
plt.scatter(X_1[0].tolist(), X_1[1].tolist(), color='blue')
plt.scatter(Xproj_0[0].tolist(), Xproj_0[1].tolist(), color='yellow')
plt.scatter(Xproj_1[0].tolist(), Xproj_1[1].tolist(), color='green')
plt.axis('scaled')
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.show()
```

```
(2, 100) (2, 100)
```



In [21]:
```
import scipy.io as sio
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn import svm
from sklearn.model_selection import GridSearchCV
%config InlineBackend.figure_format = 'retina'
```

Q5 Linear SVM

In [22]:
```
from sklearn.utils import shuffle

# 1) Load data.
X_and_Y = np.load('arrhythmia.npy')    # Load data from file
X_and_Y = np.matrix(shuffle(X_and_Y))   # Shuffle the data.
X = np.matrix(X_and_Y[:,0:278])   # First column to second to last col
umn: Features (numerical values)
Y = np.matrix(X_and_Y[:,279])    # Last column: Labels (0 or 1)
print(X.shape, Y.shape)       # Check the shapes.
```

```
(452, 278) (452, 1)
```

In [25]:
```
from sklearn.model_selection import train_test_split
# 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)

# Get features from test set.
X_train_val, X_test, Y_train_val, Y_test = train_test_split(X, Y, test
_size=0.2, random_state=42) # Get features from train + val set.
print(X_train_val.shape, X_test.shape, Y_train_val.shape, Y_test.shape
)
```

```
(361, 278) (91, 278) (361, 1) (91, 1)
```

In [30]:

```
# 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.mod
el_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

def svc_linear_classifier(X,y,cross_val):
    C_list = [0.00001,0.0001,0.001,0.01,0.1]
    y_flat = np.ravel(Y_train_val)
    parameters = {'kernel':['linear'], 'C':C_list}
    svc = SVC(gamma='auto')
    classifier = GridSearchCV(SVC(),
                              parameters,
                              cv=cross_val)
    classifier.fit(X,y_flat)
    means = classifier.cv_results_['mean_test_score']
    stds = classifier.cv_results_['std_test_score']
    SVC_arr = []
    print("Best parameters set found on test set:")
    print(classifier.best_params_)
    for mean, std, params in zip(means, stds, classifier.cv_results_['
params']):
        print("%0.3f (+/-%0.03f) for %r"
              % (mean, std * 2, params))
        SVC_arr.append(mean)

    return np.mean(X), np.mean(y)
```

In [31]:

```
# 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, x
ticklabels=[])
    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 hi
nt:
#
# demo_acc        = np.array([[0.8],
#                             [0.7]])
# demo_C_list     = [0.1, 1]
# draw_heatmap_linear(demo_acc, 'demo accuracy', demo_C_list)
#

# accuracy = [x[1] for x in classifier.grid_scores_]
# train_acc = np.array(accuracy).reshape(len(C_list))

# Gamma Values
# accuracy_array = np.array(SVC_arr)
# train_acc = np.array([[0.8],
#                       [0.7]])
# draw_heatmap_linear(train_acc, 'train accuracy', C_list)

# val_acc = np.array([[0.001],
#                     [0.01],
#                     [0.1],
#                     [1.0]])
# draw_heatmap_linear(val_acc, 'val accuracy', C_list)
C_list = [0.00001,0.0001,0.001,0.01,0.1]
train_acc, test_acc = svc_linear_classifier(X_train_val, Y_train_val,
3)
draw_heatmap_linear(train_acc.T, 'training accuracy', C_list)
draw_heatmap_linear(test_acc.T, 'validatoin accuracy', C_list)
```

```
Best parameters set found on test set:
{'kernel': 'linear', 'C': 0.0001}
0.720 (+/-0.043) for {'kernel': 'linear', 'C': 1e-05}
0.745 (+/-0.043) for {'kernel': 'linear', 'C': 0.0001}
0.734 (+/-0.070) for {'kernel': 'linear', 'C': 0.001}
0.698 (+/-0.075) for {'kernel': 'linear', 'C': 0.01}
0.643 (+/-0.096) for {'kernel': 'linear', 'C': 0.1}

-----------------------------------------------------------------
```

```
-------
ValueError                                Traceback (most recent cal
l last)
<ipython-input-31-beb5ff264713> in <module>()
     37 C_list = [0.00001,0.0001,0.001,0.01,0.1]
     38 train_acc, test_acc = svc_linear_classifier(X_train_val,
Y_train_val, 3)
---> 39 draw_heatmap_linear(train_acc.T, 'training accuracy', C_list
)
     40 draw_heatmap_linear(test_acc.T, 'validatoin accuracy',
C_list)

<ipython-input-31-beb5ff264713> in draw_heatmap_linear(acc, acc_desc
, C_list)
      4 def draw_heatmap_linear(acc, acc_desc, C_list):
      5     plt.figure(figsize = (2,4))
----> 6     ax = sns.heatmap(acc, annot=True, fmt='.3f', yticklabels
=C_list, xticklabels=[])
      7     ax.collections[0].colorbar.set_label("accuracy")
      8     ax.set(ylabel='$C$')

~/anaconda2/envs/env36/lib/python3.5/site-packages/seaborn/matrix.py
in heatmap(data, vmin, vmax, cmap, center, robust, annot, fmt, annot
_kws, linewidths, linecolor, cbar, cbar_kws, cbar_ax, square, xtickl
abels, yticklabels, mask, ax, **kwargs)
    515     plotter = _HeatMapper(data, vmin, vmax, cmap, center, ro
bust, annot, fmt,
    516                           annot_kws, cbar, cbar_kws,
xticklabels,
--> 517                           yticklabels, mask)
    518
    519     # Add the pcolormesh kwargs here

~/anaconda2/envs/env36/lib/python3.5/site-packages/seaborn/matrix.py
in __init__(self, data, vmin, vmax, cmap, center, robust, annot, fmt
, annot_kws, cbar, cbar_kws, xticklabels, yticklabels, mask)
    109         else:
    110             plot_data = np.asarray(data)
--> 111             data = pd.DataFrame(plot_data)
    112
    113         # Validate the mask and convet to DataFrame

~/anaconda2/envs/env36/lib/python3.5/site-packages/pandas/core/frame
.py in __init__(self, data, index, columns, dtype, copy)
    359             else:
    360                 mgr = self._init_ndarray(data, index, column
s, dtype=dtype,
--> 361                                          copy=copy)
    362         elif isinstance(data, (list, types.GeneratorType)):
    363             if isinstance(data, types.GeneratorType):

~/anaconda2/envs/env36/lib/python3.5/site-packages/pandas/core/frame
.py in _init_ndarray(self, values, index, columns, dtype, copy)
```

```
    511         # by definition an array here
    512         # the dtypes will be coerced to a single dtype
--> 513         values = _prep_ndarray(values, copy=copy)
    514
    515         if dtype is not None:

~/anaconda2/envs/env36/lib/python3.5/site-packages/pandas/core/frame
.py in _prep_ndarray(values, copy)
   6255             values = values.reshape((values.shape[0], 1))
   6256         elif values.ndim != 2:
-> 6257             raise ValueError('Must pass 2-d input')
   6258
   6259         return values

ValueError: Must pass 2-d input

<matplotlib.figure.Figure at 0x1a12bb3f98>
```

In [ ]:

```
# 5) Use the best C to calculate the test accuracy.

test_acc = # See above 'Best parameters found on test set'
print(test_acc)
```

Question 6: Implement Grid Search and Cross-Validation

In [33]:

```
import scipy.io as sio
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.svm import SVC
from sklearn.model_selection import KFold
from sklearn.utils import shuffle

%config InlineBackend.figure_format = 'retina'
```

Part 1

```
1: Divide training and validation into 4 folds
2: Train the SVM w/ RBF kernel for fold round.
      Each round choose a subset as validation set & fold - 1 as the train
ing set
3: Calc Trng acc and Valid_acc every round
4: Finally, return the avg training acc and avg valid acc over all rounds
```

In [37]:

```
def svc_linear_classifier(X,y,n_folds):
    C_list = [0.00001,0.0001,0.001,0.01,0.1]
    X_train_folds = []
    X_valid_folds = []
    Y_train_folds = []
    Y_valid_folds = []
    train_score = []
    valid_score = []
    y_flat = np.ravel(y)
    kf = KFold(n_splits=n_folds)     #Split train_val set
    kf.get_n_splits(X)
    # Train the SVM with RBF kernel for fold rounds
    classiifer = SVC(C=C_list,kernel='linear',degree=n_folds)
    for train_index, valid_index in kf.split(X):
        X_train_folds.append(X[train_index])
        X_valid_folds.append(X[valid_index])
        Y_train_folds.append(y[train_index])
        Y_valid_folds.append(y[valid_index])
        # Each round choose one different subset as validation set
        # All of the other data points (all the other fold - 1 subsets
) are the training set.
        for i in range(n_folds):
            tmp_res = classiifer.fit(X_train_folds[i], X_valid_folds[i
])
            valid_pred = tmp_res.predict((X_valid_folds[i]))
            valid_acc = sum(valid_pred == Y_valid_folds[i])/len(Y_vali
d_folds[i])
            valid_score.append(valid_acc)
        # Calculate the training accuracy and validation accuracy ever
y round.
        means = classifier.cv_results_['mean_test_score']
        stds = classifier.cv_results_['std_test_score']



    # Return the avg training accuracy & average validation accuracy o
ver all rounds.
    return np.mean(train_score), np.mean(valid_score)
```

Part 2: Grid Search

In [38]:
```python
# GridSearch implements both fit and transform
def grid_search(X, y, fold, C_list):
    train_acc = []
    valid_acc = []
    for c in C_list:
        train, valid = svc_linear_classifier(X, y, fold)
        train_acc.append(train)
        valid_acc.append(valid)
    train_acc = np.atleast_2d(train_acc)
    valid_acc = np.atleast_2d(valid_acc)
    return(train_acc, valid_acc)
```

Part 3: Heatmap

In [39]:
```python
C_list = [0.00001,0.0001,0.001,0.01,0.1]
training_acc, validation_acc = grid_search(X_train_val, Y_train_val, 3
, C_list)
draw_heatmap_linear(training_acc.T, 'training accuracy', C_list)
draw_heatmap_linear(validation_acc.T, 'validatoin accuracy', C_list)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent cal
l last)
<ipython-input-39-51633fb2604c> in <module>()
      1 C_list = [0.00001,0.0001,0.001,0.01,0.1]
----> 2 training_acc, validation_acc = grid_search(X_train_val,
Y_train_val, 3, C_list)
      3 draw_heatmap_linear(training_acc.T, 'training accuracy',
C_list)
      4 draw_heatmap_linear(validation_acc.T, 'validatoin accuracy',
C_list)

<ipython-input-38-42ef680517a5> in grid_search(X, y, fold, C_list)
      4     valid_acc = []
      5     for c in C_list:
----> 6         train, valid = svc_linear_classifier(X, y, fold)
      7         train_acc.append(train)
      8         valid_acc.append(valid)

<ipython-input-37-88debdd8fd84> in svc_linear_classifier(X, y, n_fol
ds)
     20         # All of the other data points (all the other fold -
1 subsets) are the training set.
     21         for i in range(n_folds):
---> 22             tmp_res = classiifer.fit(X_train_folds[i],
X_valid_folds[i])
     23             valid_pred = tmp_res.predict((X_valid_folds[i]))
     24             valid_acc = sum(valid_pred == Y_valid_folds[i])/
len(Y_valid_folds[i])
```

```
~/anaconda2/envs/env36/lib/python3.5/site-packages/sklearn/svm/base.
py in fit(self, X, y, sample_weight)
    147         self._sparse = sparse and not callable(self.kernel)
    148 
--> 149         X, y = check_X_y(X, y, dtype=np.float64, order='C',
accept_sparse='csr')
    150         y = self._validate_targets(y)
    151 

~/anaconda2/envs/env36/lib/python3.5/site-packages/sklearn/utils/val
idation.py in check_X_y(X, y, accept_sparse, dtype, order, copy, for
ce_all_finite, ensure_2d, allow_nd, multi_output, ensure_min_samples
, ensure_min_features, y_numeric, warn_on_dtype, estimator)
    576                             dtype=None)
    577     else:
--> 578         y = column_or_1d(y, warn=True)
    579         _assert_all_finite(y)
    580     if y_numeric and y.dtype.kind == 'O':

~/anaconda2/envs/env36/lib/python3.5/site-packages/sklearn/utils/val
idation.py in column_or_1d(y, warn)
    612         return np.ravel(y)
    613 
--> 614     raise ValueError("bad input shape {0}".format(shape))
    615 
    616 

ValueError: bad input shape (121, 278)
```