Homework Assignment 6

1 (8 points) Conceptual Questions

1. Which one below best describes what support vectors are:

A. The decision boundary.
B. The positive and the negative planes.
[✓ C. The training samples that determine the positive and the negative planes.]
D. The test samples that determine the positive and the negative planes.

2. When tuning the hyper-parameters of a model, we find the hyper-parameters that

A. minimize error on the test set
B. minimize error on the test set
C. maximize the margin of the classifier
[✓ D. minimize error on the validation set]

3. Select all that apply. k-fold cross validation is

[✓ A. Not necessary when you have huge amounts of labelled data]
[✓ B. A way to do model selection while minimizing over-fitting]
C. The only way to do hyper-parameter tuning
[✓ D. Subject to the bias-variance trade-off]

4. When you increase the k in cross-validation while keeping the dataset the same, you

A. Get a decrease in both the generalization error and the validation error
B. Get an increase both the generalization error and the validation error
C. Get a decrease in the variability of the validation error across folds
[✓ D. Get an increase in the variability of the validation error across folds]

2 (10 points) Cross-Validation

Given a training dataset $S_{\text{training}} = \{(x_i, y_i)\}, i = 1, \ldots, 6\}$ where $x_i \in R$ is the feature scalar and $y_i \in \{-1, +1\}$ is the corresponding label. The data points in the dataset S_{training} are given below:

$$(x_1, y_1) = (2, -1), (x_2, y_2) = (7, -1), (x_3, y_3) = (4, +1),$$
$$(x_4, y_4) = (1, -1), (x_5, y_5) = (3, +1), (x_6, y_6) = (6, +1).$$

Suppose you are training a linear classifier $f(x; a, b)$ = sign$(ax + b)$ with 2-fold cross-validation where sign(z) is defined as:

$$\text{sign}(z) = \begin{cases} 1, \ z \geq 0 \\ -1, \ z < 0 \end{cases}$$

- You have split the dataset S_{training} into:

$$S_1 = \{(x_1, y_1), (x_2, y_2), (x_3, y_3)\}$$
$$S_2 = \{(x_4, y_4), (x_5, y_5), (x_6, y_6)\}$$

- After training the classifier $f(x; a, b)$ on S_1, you have obtained the parameters $a_1 = -1, b_1 = 5$ and then try to validate the classifier on S_2.

- After training the classifier $f(x; a, b)$ on S_2, you have obtained the parameters $a_2 = 2, b_2 = -3$ and then try to validate the classifier on S_1.

Please finish the tasks below:

1. Calculate the **average training error** in the 2-fold cross-validation.

Note: The definition of average training error is the mean of the error of classifier $f(x; a_1, b_1)$ on S_1 and the error of classifier $f(x; a_2, b_2)$ on S_2.

$$S_1 = \frac{1}{3} \text{ and } S_2 = \frac{0}{3} = 0$$

$$\boxed{\text{Average Training Error } \Rightarrow \frac{1}{6}}$$

2. Calculate the **average validation error** (i.e. the cross-validation error) in the 2-fold cross-validation.

$$\boxed{\text{Avearge validation error } \Rightarrow \frac{2}{3}}$$

3 (12 points) Shattering

In this problem, consider a classifier $f(x; a, b)$ = sign$(ax^T x - b)$ where the feature vector $x = [x_1, x_2] >$ ∈ R^2 and its prediction $f(x; a, b)$ ∈ $\{-1, +1\}$. Besides, a, b ∈ R are the model parameters and sign(z) is defined as:

$$\text{sign}(z) = \begin{cases} 1, \ z \geq 0 \\ -1, \ z < 0 \end{cases}$$

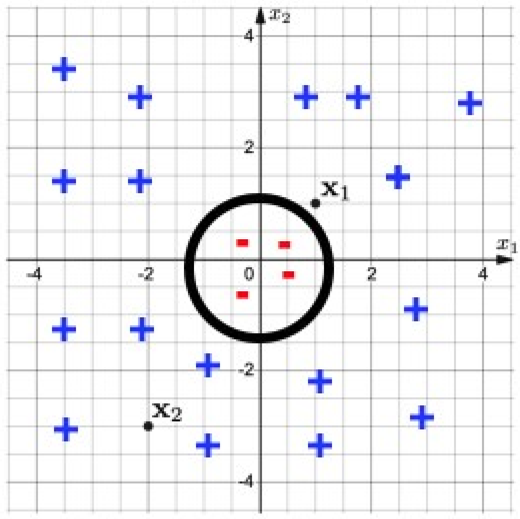
The classifier $f(x; a, b)$ performs a binary classification on an input feature vector x under model parameters a, b ∈ R that can be learned.

In this question, please attempt to show that if the classifier $f(x; a, b)$ can shatter two points, $x_1 = [1, 1]^T$ and $x_2 = [-2, -3]^T$.

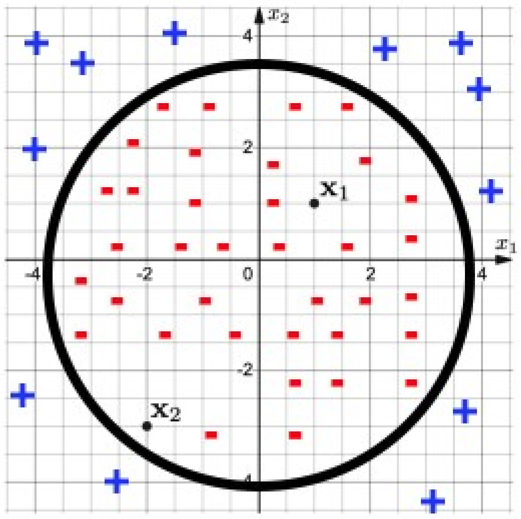
- If you think $f(x; a, b)$ can shatter x_1 and x_2, you need to show that for each possible label configuration (y_1, y_2) ∈ $\{(+1, +1), (-1, -1), (+1, -1), (-1, +1)\}$ of x_1 and x_2, there exists a classifier $f(x; a, b)$ that classifies the x_1 and x_2 correctly, and you should illustrate each classifier by the following steps:

```
- Draw the decision boundary of the classifier.
- Shade the area where the classifier makes the positive prediction
```

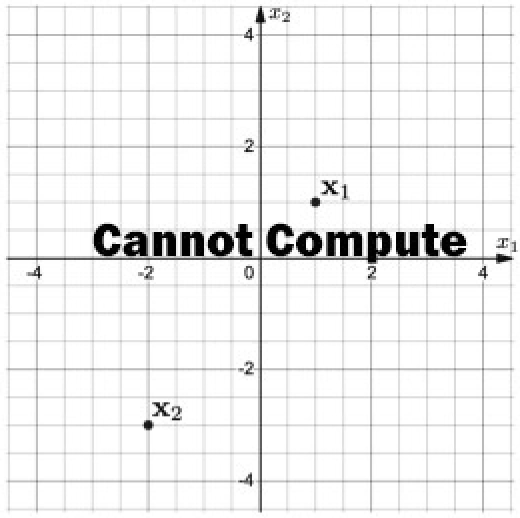
- If you think classifier $f(x; a, b)$ cannot shatter x_1 and x_2, please explain the reason.



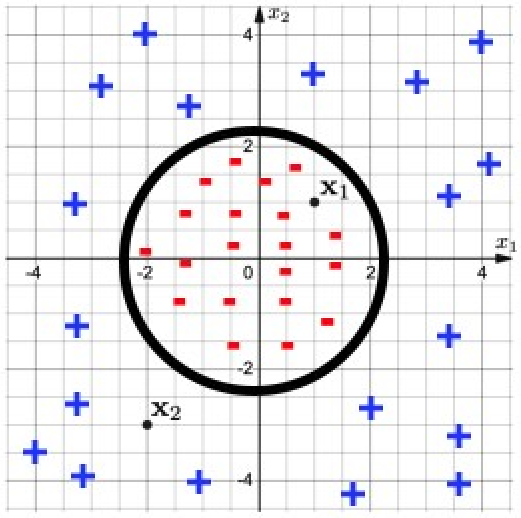
$(y_1, y_2) = (+1, +1)$



$(y_1, y_2) = (-1, -1)$



$(y_1, y_2) = (+1, -1)$



$(y_1, y_2) = (-1, +1)$

Answer: $f(x; a, b)$ cannot be shattered since there is no way to classify $(y_1, y_2) = (+1, -1)$. That is, we are unable to draw a circle that allows x_1 to have a +1 classificaiton and x_2 to have -1 classification

4 (10 points) SVM: Gradient

Given a training dataset $S_{\text{training}} = \{(x_i, y_i)\}, i = 1, \dots, n\}$, we wish to optimize the loss $\mathcal{L}(w, b)$ of a linear SVM classifier:

$$\mathcal{L}(w,b) = \frac{1}{2}||w||_2^2 + C\sum_{i=1}^{n}(1 - y_i(w^T x_i + b))_+ \ \ (1)$$

where $(z)_+ = \max(0, z)$ is called the rectifier function and C is a scalar constant

The optimal weight vector w^* and the bias b^* used to build the SVM classifier are defined as follows:

$$w^*, b^* = \text{argmin}_{w,b}\mathcal{L}(w,b) \ \ (2)$$

In this problem, we attempt to obtain the optimal parameters w^* and b^* by using a standard gradient descent algorithm.

Hint: To derive the derivative of $\mathcal{L}(w, b)$, please consider two cases:

(a) $1 - y_i(w^T x_i + b) \geq 0$,
(b) $1 - y_i(w^T x_i + b) < 0$

1. Derive the derivative: $\frac{\partial \mathcal{L}(w,b)}{\partial w}$

$$\frac{\mathcal{L}(w,b)}{\partial w} = w + C\sum_{i=1}^{n}\begin{cases}0, \ \text{if}\, y_i(w^T x_i + b) \geq 1\\ -y_i x_i, \text{otherwise}\end{cases}$$

2. Derive the derivative: $\frac{\partial \mathcal{L}(w,b)}{\partial b}$

$$\frac{\mathcal{L}(w,b)}{\partial b} = C\sum_{i=1}^{n}\begin{cases}0, \ \text{if}\, y_i(w^T x_i + b) \geq 1\\ -y_i, \text{otherwise}\end{cases}$$

5 (10 points) SVM: Margin

As shown in the Figure 3, we have the decision boundary (marked as a black line) defined as Eq. 3 given w,b:

$$w^T x + b = 0 \ \ (3)$$

In parallel to the decision boundary, we have the positive plane (marked as a red line)defined as Eq. 4 and the negative plane (marked as a blue line) defined as Eq. 5:

$$w^T x + b = +1 \ \ (4)$$
$$w^T x + b = -1 \ \ (5)$$

We pick an arbitrary point x^- on the negative plane, and draw a purple line thatpasses x^- and is perpendicular to the negative plane. The intersection between this purple line and the positive plane can be denoted as x^+. Thus, we have the following Eq.6 that indicates the relation between x^+ and x^-:

$$x^+ = x^- + \lambda w \ \ (6)$$

where λ ∈ R is an undetermined scalar. The margin M is defined as the distancebetween the positive and the negative planes, which can be calculated from Eq.7:

$$M = ||x^+ - x^-||_2 = \sqrt{< \lambda w, \lambda w >} \ \ (7)$$

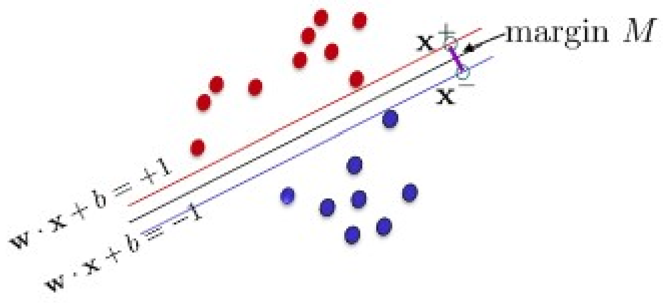

$\mathbf{x}^+$
margin M
$\mathbf{x}^-$
$\mathbf{w} \cdot \mathbf{x} + b = +1$
$\mathbf{w} \cdot \mathbf{x} + b = -1$

Figure 3: The decision boundary, the positive plane and the negative plane.

Please derive the following according to Eq. 7:

$$M = \frac{2}{\sqrt{< w, w >}}$$

Hint: You can firstly represent λ in the form of w by using Eq. 4, 5, 6

$$(4) \Rightarrow \ x^+ = \frac{1-b}{w}, \ (5) \Rightarrow \ x^- = \frac{-b-1}{w}$$
$$(6) \Rightarrow \ \lambda = \frac{x^+ - x^-}{w}$$

$$\text{plugging } x^+ \text{ and } x^- \text{ into } \lambda$$

$$\Rightarrow \lambda = \frac{\frac{1-b}{w} - \frac{-b-1}{w}}{w}$$
$$\Rightarrow \lambda = \frac{2}{ww}$$

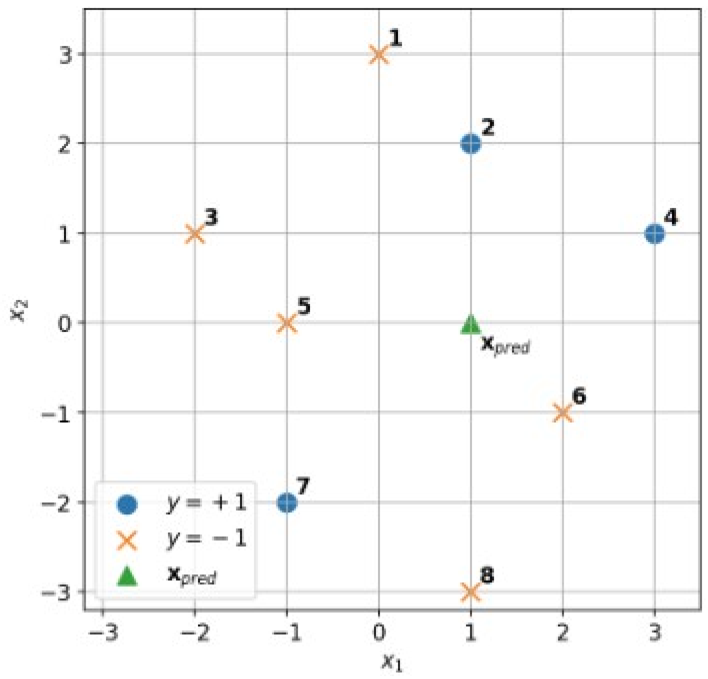
$$M = \sqrt{< \lambda w, \lambda w >} \ldots (7)$$

$$\text{plugging } \lambda \text{ in } (7),$$

$$\Rightarrow M = \sqrt{(\frac{2w}{ww})(\frac{2w}{ww})^T}$$
$$\Rightarrow M = \sqrt{\frac{4}{w^t w}}$$
$$\Rightarrow M = \frac{2}{\sqrt{w^T w}}$$
$$\boxed{\Rightarrow M = \frac{2}{\sqrt{< w, w >}}}$$

6 (10 points) K Nearest Neighbors

Consider a training dataset $S_{\text{training}} = \{(x_i, y_i), i = 1, 2, \ldots, 9\}$ where each data point (x, y) has a feature vector $x = [x_1, x_2]^T$ and the corresponding label $y \in \{-1, +1\}$. The points with the corresponding labels in the dataset are shown in the figure below.



In the figure above, the index i for each training example x_i is given in **bold** near the point. You are asked to predict the label of a point $x_{\text{pred}} = [1, 0]^T$ shown in the figure as a triangle using k nearest neighbors (k-NN) method under **Euclidean distance**.

1. List the indices of all the data points in S_{training} and their corresponding labels.

$$
\begin{gathered}
(x_1, y_1) \Rightarrow x_1 = [0, 3], y_1 = -1 \\
(x_2, y_2) \Rightarrow x_2 = [1, 2], y_2 = +1 \\
(x_3, y_3) \Rightarrow x_3 = [-2, 1], y_3 = -1 \\
(x_4, y_4) \Rightarrow x_4 = [3, 1], y_4 = +1 \\
(x_5, y_5) \Rightarrow x_5 = [-1, 0], y_5 = -1 \\
(x_6, y_6) \Rightarrow x_6 = [2, -1], y_6 = -1 \\
(x_7, y_7) \Rightarrow x_7 = [-1, -2], y_7 = +1 \\
(x_8, y_8) \Rightarrow x_8 = [1, -3], y_8 = -1
\end{gathered}
$$

2. Determine the predicted label for x_{pred} using the k-NN with different k.

(a) k=1 $\Rightarrow \boxed{\text{label is} - 1}$

(b) k=3 $\Rightarrow \boxed{\text{label is} - 1}$

(c) k=5 $\Rightarrow \boxed{\text{label is} + 1}$

7 (20 points) Coding: K Nearest Neighbors

In this problem, you need to implement the k nearest neighbors (k-NN) algorithmand apply it to the binary classification. Here we use the modified Iris dataset $S = \{(x_i, y_i)\}$ where each feature vector $x \in R^2$ and label $y \in \{-1, +1\}$. You are not allowed to use **sklearn.neighbors.KNeighborsClassifier()** in your code, but you can use it to validate your implementation.

- Load the modified Iris dataset. The dataset S is split to three subsets: The training set S_{training}, the validation set $S_{\text{validation}}$ and the test set S_{test}. In the code, we use X_train, Y_train for the feature vectors and labels of the training set respectively. Similar notations are also used for the validation and the test sets.

- Implement k-NN algorithm in 3 steps.

```
1. For each feature vector x you are predicting a label, you need to calcu
latethe distances between this
   feature vector x and all the feature vectorsin the training set Straini
ng.

2. Then  sort  all  distances  in  ascending  order  and  pick  the  label
s  for  the k minimum distances.

3. Count the number of negative labels Ny=-1, and the number of the positi
velabels Ny=+1 from k labels picked
   in step 2. Use the following decision ruleto predict label ˆy for each
feature vector x:
```

$$\hat{y} = \begin{cases} +1, \ N_{y=-1} < N_{y=+1}, \\ -1, \ N_{y=-1} \geq N_{y=+1} \end{cases}$$

Here we assume **Euclidean distance** as the distance metric. For more details, please refer to the code and the corresponding part in the slides.

- Use the validation set to obtain optimal k^*. In k-NN, there is a hyper-parameter k which adjusts the number of nearest neighbors. You would need to perform a grid search on the following list of k:

$$k \in \{1, 2, 3\}$$

For each k, you need to form a k-NN classifier with the training set S_{training}. Then, use the classifier to make predictions on the validation set $S_{\text{validation}}$ and calculate the error $e_{\text{validation}}$. We aim to obtain the best hyper-parameter k^* corresponding to the minimum validation error $e^*_{\text{validation}}$ among all ks.

- Use the obtained classifier corresponding to the best hyper-parameter k^* to calculate the test error e_{test} on test set S_{test}.

In [1]:
```
%config InlineBackend.figure_format = 'retina'
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
import scipy
from matplotlib.colors import ListedColormap
from functools import partial
```

Load the modified Iris dataset

In [2]:
```
# Iris dataset.
iris = datasets.load_iris()      # Load Iris dataset.

X = iris.data                    # The shape of X is (150, 4), which mea
ns
                                 # there are 150 data points, each data
point
                                 # has 4 features.

Y = (iris.target > 1.5).reshape(-1,1).astype(np.float) # The shape of Y
is (150, 1), which means
                                 # there are 150 data points, each data
point
                                 # has 1 target value.
Y[Y==0] = -1                     # Convert labels from {0, 1} to {-1, 1}

X_and_Y = np.hstack((X, Y))      # Stack them together for shuffling.
np.random.seed(1)                # Set the random seed.
np.random.shuffle(X_and_Y)       # Shuffle the data points in X_and_Y ar
ray

print(X.shape)
print(Y.shape)
print(X_and_Y[0])                # Should be: [5.8 4.  1.2 0.2 -1. ].
```

```
(150, 4)
(150, 1)
[ 5.8  4.   1.2  0.2 -1. ]
```

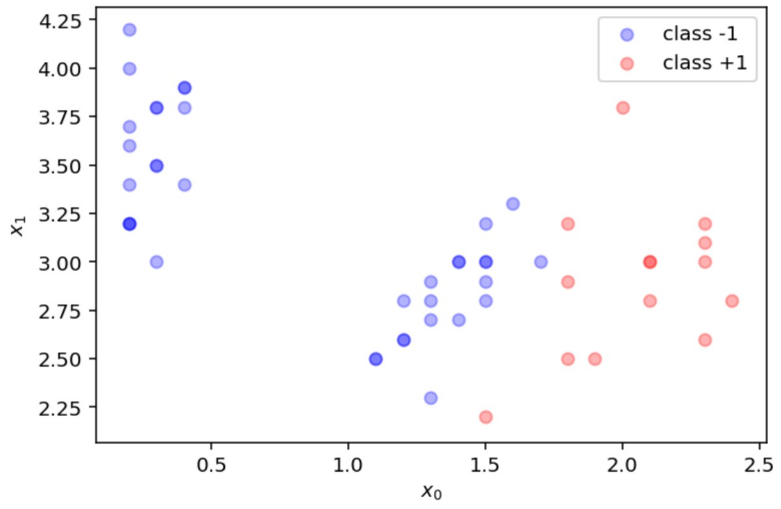
In [3]:
```
# Divide the data points into training set and test set.
X_shuffled = X_and_Y[:,:4]
Y_shuffled = X_and_Y[:,4]

X_train = X_shuffled[:50][:, [3,1]]     # Shape: (50,2)
Y_train = Y_shuffled[:50]               # Shape: (50,)
X_val   = X_shuffled[50:100][:, [3,1]]  # Shape: (50,2)
Y_val   = Y_shuffled[50:100]            # Shape: (50,)
X_test  = X_shuffled[100:][:, [3,1]]    # Shape: (50,2)
Y_test  = Y_shuffled[100:]              # Shape: (50,)
print(X_train.shape)
print(Y_train.shape)
print(X_val.shape)
print(Y_val.shape)
print(X_test.shape)
print(Y_test.shape)
```

```
(50, 2)
(50,)
(50, 2)
(50,)
(50, 2)
(50,)
```

Visualization

In [4]:
```
def vis(X, Y, knn_classifier=None):
    # Visualize k-NN.
    if knn_classifier is not None:
        # Calculate min, max and create grids.
        h = .02
        x0_min, x0_max = X[:, 0].min() - 0.1, X[:, 0].max() + 0.1
        x1_min, x1_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1
        x0s, x1s = np.meshgrid(np.arange(x0_min, x0_max, h),
                               np.arange(x1_min, x1_max, h))
        xs = np.stack([x0s, x1s], axis=-1).reshape(-1, 2)

        # Predict class using kNN classifier and data.
        ys_pred = np.array([knn_classifier(x) for x in xs])
        ys_pred = ys_pred.reshape(x0s.shape)

        # Put the result into a color plot.
        # Color map: #00AAFF - blue, #FFAAAA - red, #AAFFAA - green

        cmap_light = ListedColormap(['#00AAFF', '#FFAAAA'])
        plt.pcolormesh(x0s, x1s, ys_pred, cmap=cmap_light, alpha=0.3)

    indices_neg1 = (Y == -1).nonzero()[0]
    indices_pos1 = (Y == 1).nonzero()[0]
    plt.scatter(X[:,0][indices_neg1], X[:,1][indices_neg1],
                c='blue', label='class -1', alpha=0.3)
    plt.scatter(X[:,0][indices_pos1], X[:,1][indices_pos1],
                c='red', label='class +1', alpha=0.3)
    plt.legend()
    plt.xlabel('$x_0$')
    plt.ylabel('$x_1$')

    plt.show()
```

In [5]:
```
# Visualize training set.
vis(X_train, Y_train)
# Note that some points have darker color since there can be
# multiple points at the same location.
```



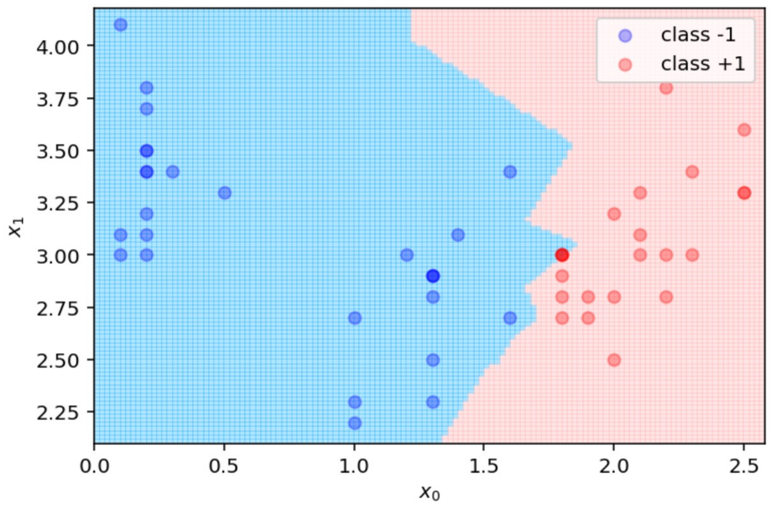
k Nearest Neighbors

In [6]:
```
# Euclidean distance.
def calc_distance(x1, x2):
    dist = np.square(x1-x2).sum()
    return np.sqrt(dist)

# k nearest neighbor predictor.
def f_knn(x, X_train, Y_train, k):
    # Create the list of (distance, label) pairs.
    dist_label_pairs = []
    for xi, yi in zip(X_train, Y_train):
        # Calculate the distance.
        dist = calc_distance(xi, x)
        # Add a (distance, label) pair to the list.
        dist_label_pairs.append((dist, yi))

    # Sort the pairs by distance (ascending).
    sorted_dist_label_pairs = sorted(dist_label_pairs, key=lambda x:x
[0])

    # Obtain the first k pairs (corresponding to k smallest distances).
    k_dist_label_pairs = sorted_dist_label_pairs[:k]       ######## To
be filled. ########

    # Extract the labels of the k pairs.
    k_labels            = np.array(k_dist_label_pairs)[:,-1] ######## To
be filled. ########

    # Count the number of +1 predictions and -1 predictions.
    pos1_in_k_labels = 0
    neg1_in_k_labels = 0
    for label in k_labels:
        if label == +1:
            pos1_in_k_labels += 1
        elif label == -1:
            neg1_in_k_labels += 1

    # Make the prediction based on counts.
    if pos1_in_k_labels > neg1_in_k_labels:
        y_pred = +1
    else:
        y_pred = -1

    return y_pred

# Judge function: 1(a != b). It supports scalar, vector and matrix.
def judge(a, b):
    return np.array(a != b).astype(np.float32)

# Calculate error given feature vectors X and labels Y.
def calc_error(X, Y, knn_classifier):
    e = 0
    n = len(X)
    for (xi, yi) in zip(X, Y):
        e += judge(yi, knn_classifier(xi))
    e = 1.0 * e / n
    return e
```

Visualize the results

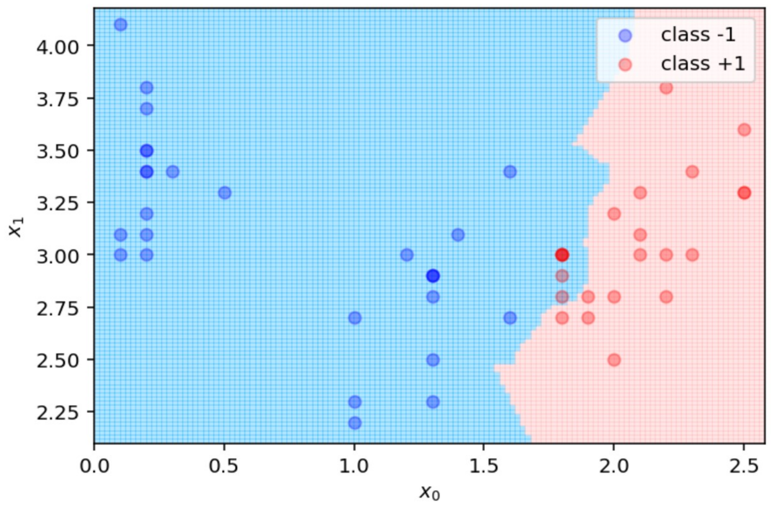
In [7]:
```
opt_val_error = 1.0
opt_k = None

# Try different k(s).
for k in [1,2,3]:
    # Visualize
    #   1. Validation set (as points).
    #   2. Decision boundary from training set (as background).
    print("k={}".format(k))
    # Create a k-NN classifier with training set.
    knn_classifier = partial(f_knn, X_train=X_train, Y_train=Y_train, k
=k)
    # Visualization.
    vis(X_val, Y_val, knn_classifier)
    # Calculate validation error.
    val_error = calc_error(X_val, Y_val, knn_classifier)
    print("Validation error: {}\n".format(val_error))
    if val_error < opt_val_error:
        opt_val_error = val_error
        opt_k = k
        opt_knn_classifier = knn_classifier
```

```
k=1
```



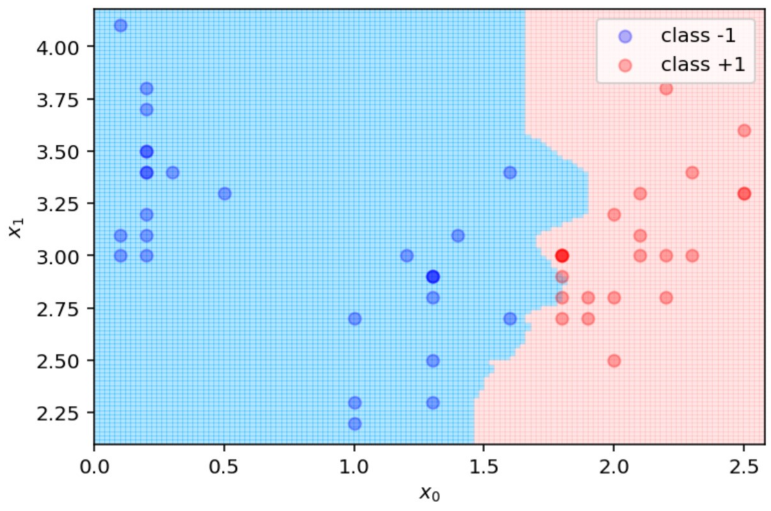
```
Validation error: 0.08
```

```
k=2
```



```
Validation error: 0.12
```

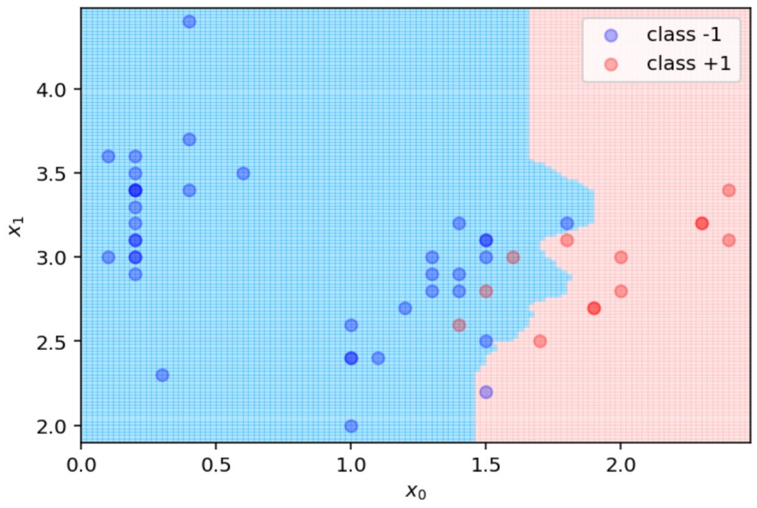
```
k=3
```



```
Validation error: 0.02
```

In [8]:
```
print("Best k={}".format(opt_k))
test_error = calc_error(X_test, Y_test, opt_knn_classifier)
vis(X_test, Y_test, opt_knn_classifier)
print("Test error: {}".format(test_error))
```

```
Best k=3
```



```
Test error: 0.08
```

8 (20 points) Coding: Decision Tree

In this problem, you need to implement the decision tree algorithm and apply it to the binary classification. Here we use the Ionosphere dataset $S = \{(x_i, y_i)\}$ where each feature vector $x \in R^{3,4}$ and label $y \in \{-1, +1\}$. You are allowed to use the functions from scikit-learn in this question.

- Load the Ionosphere dataset. The dataset S is split to two subsets: The training set S_{training} and the test set S_{test}. In the code, we use X train, Y train for the feature vectors and labels of the training set respectively. Similar notations are also used for the test set.

- Train the decision tree classifier with the entropy criterion. In the decision ree, there is a hyper-parameter D which controls the maximum depth. Youwould need to perform a grid search on the following list of D:

$$D \in \{1, 2, 3, 4, 5\}$$

For each D, you need to form a decision tree classifier with the training set S_{training}. Specifically, you need to conduct a 10-fold cross-validation on S_{training} and calculate the cross-validation error $\bar{e}$ (i.e. average validation error over the splits in cross-validation). We aim to obtain the best hyper-parameter D^* corresponding to the minimum cross-validation error $\bar{e}^*$ among all Ds.

- Use the obtained classifier corresponding to the best hyper-parameter D$*$ to calculate the test error etest on test set S_{test}.

Decision Tree with Scikit-Learn

In [9]:
```
%config InlineBackend.figure_format = 'retina'
import scipy.io as sio
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn import tree
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
```

Load the Ionosphere dataset

In [10]:
```
# Ionosphere dataset.
X_and_Y = np.load('ionosphere.npy').astype(np.float32) # Load data from
file.

np.random.seed(1)                # Set the random seed.
np.random.shuffle(X_and_Y)       # Shuffle the data.
X = X_and_Y[:, 0:-1]             # First column to second last column: F
eatures.
Y = X_and_Y[:, -1]               # Last column: Labels.
Y[Y==0] = -1                     # Convert labels from {0, 1} to {-1,
1}.

print(X.shape)       # (351, 34)
print(Y.shape)       # (351,)
print(X_and_Y[0])
# The result should be:
# [ 1.        0.       -0.205     0.2875    0.23      0.1       0.2825    0.31
75
#   0.3225    0.35      0.36285 -0.34617  0.0925    0.275    -0.095     0.21
#  -0.0875    0.235    -0.34187  0.31408 -0.48     -0.08      0.29908   0.33
176
#  -0.58     -0.24      0.3219   -0.28475 -0.47      0.185    -0.27104 -0.31
228
#   0.40445   0.0305    1.      ]
```

```
(351, 34)
(351,)
[ 1.        0.       -0.205     0.2875    0.23      0.1       0.2825    0.31
75
  0.3225    0.35      0.36285 -0.34617  0.0925    0.275    -0.095     0.21
 -0.0875    0.235    -0.34187  0.31408 -0.48     -0.08      0.29908   0.33
176
 -0.58     -0.24      0.3219   -0.28475 -0.47      0.185    -0.27104 -0.31
228
  0.40445   0.0305    1.      ]
```

In [11]:
```
# Divide the data points into training set and test set.
X_shuffled = X
Y_shuffled = Y
X_train = X_shuffled[:200]          # Shape: (200, 34)
Y_train = Y_shuffled[:200]          # Shape: (200,)
X_test = X_shuffled[200:]           # Shape: (151,4)
Y_test = Y_shuffled[200:]           # Shape: (151,)
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
```

```
(200, 34)
(200,)
(151, 34)
(151,)
```

Decision Tree Using Scikit-Learn

In [12]:
```
# Perform grid search for best max depth.

# 1. Create a decision tree classifier.
#    Hint: You can use tree.DecisionTreeClassifier()
#          We use "entropy" as the criterion. The random state should b
e
#          set to 1 for consistent results. Other options are left at d
efault.
estimator = tree.DecisionTreeClassifier(criterion='entropy', random_sta
te=1)
        ######## To be filled. ########
# 2. Create a grid searcher with cross-validation.
D_list = [1, 2, 3, 4, 5]
param_grid = {'max_depth': D_list}
#    Hint: You can use GridSearchCV()
#          Please set a 10-fold cross-validation.
grid_search = GridSearchCV(estimator,param_grid, cv=10) ######## To be
filled. ########
# 3. Use the grid searcher to fit the training set.
#    - This grid searcher will try every max depth in the list.
#    - For each max depth, a cross-validation is applied to the trainin
g set,
#      that is, it creates several (training subset, validation subset)
pairs.
#      Note: Sometimes the validation subset is called as "test" subse
t, but it
#            is not the subset of real test set.
#        - For each pair, a decision tree classifier will be trained on
the
#          training subset and evaluated on validation subset.
#        - The average validation scores will be kept.
#
#    Hint: You can simply use .fit() function of the grid searcher.
grid_search.fit(X_train,Y_train) ######## To be filled. ########
```

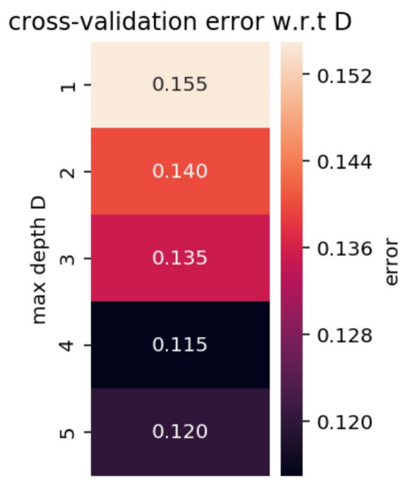
Out[12]:
```
GridSearchCV(cv=10,
             estimator=DecisionTreeClassifier(criterion='entropy',
                                              random_state=1),
             param_grid={'max_depth': [1, 2, 3, 4, 5]})
```

Visualize the Results

In [13]:
```
print(grid_search.cv_results_['mean_test_score'])
```

```
[0.845 0.86  0.865 0.885 0.88 ]
```

In [14]:
```
# Draw heatmaps for result of grid search.
def draw_heatmap(errors, D_list, title):
    plt.figure(figsize = (2,4))
    ax = sns.heatmap(errors, annot=True, fmt='.3f', yticklabels=D_list,
xticklabels=[])
    ax.collections[0].colorbar.set_label('error')
    ax.set(ylabel='max depth D')
    bottom, top = ax.get_ylim()
    ax.set_ylim(bottom + 0.5, top - 0.5)
    plt.title(title)
    plt.show()

# Draw heatmaps of cross-validation errors (in cross-validation).
# Hint: You can use .cv_results_['mean_test_score'] to obtain
#       cross-validation accuracy (that is, average validation accuracy
over
#       different splits in the cross-validation). You need to convert
it
#       to the error.
#       Note that you need to reshape the results to shape (?, 1), whic
h is
#       needed by draw_heatmap().
cross_val_errors = (1 - grid_search.cv_results_['mean_test_score']).res
hape(5,1)
draw_heatmap(cross_val_errors, D_list, title='cross-validation error w.
r.t D')
```



In [15]:
```
# Show the best max depth.
# Hint: You can use the .best_params_ of the grid searcher
#       to obtain the best parameter(s).
best_max_depth = grid_search.best_params_######## To be filled. #######
#
print("Best max depth D: {}".format(best_max_depth))

# Calculate the test error.
# Hint: You can use .best_estimator_.predict() to make predictions.

#test_error = (((grid_search.best_estimator_.predict(X_test) -Y_test)/
2)**2).sum()/len(Y_test)######## To be filled. ########
test_error = 1 - accuracy_score(y_true=Y_test, y_pred=grid_search.predi
ct(X_test))
print("Test error: {}".format(test_error))
```

```
Best max depth D: {'max_depth': 4}
Test error: 0.1258278145695364
```