CSE572-Lab6-key

October 17, 2022

1 CSE 572: Lab 6

In this lab, you will practice implementing different variations of the Support Vector Machine (SVM) classifier.

To execute and make changes to this notebook, click File > Save a copy to save your own version in your Google Drive or Github. Read the step-by-step instructions below carefully. To execute the code, click on each cell below and press the SHIFT-ENTER keys simultaneously or by clicking the Play button.

When you finish executing all code/exercises, save your notebook then download a copy (.ipynb file). Submit the following **three** things: 1. a link to your Colab notebook, 2. the .ipynb file, and 3. a pdf of the executed notebook on Canvas.

To generate a pdf of the notebook, click File > Print > Save as PDF.

1.0.1 Load the iris dataset

Load the dataset. For visualization purposes for the first exercise, we will convert the dataframe to have only two features (petal length and petal width) and two classes (Iris-virginica and Iris-other).

[1]:
```
import pandas as pd

data = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/
 ↪iris/iris.data',header=None)
data.columns = ['sepal length', 'sepal width', 'petal length', 'petal width',␣
 ↪'class']

data.head()
```

[1]:
```
   sepal length  sepal width  petal length  petal width        class
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa
```

[2]:
```
# Drop the sepal features
data = data.drop(['sepal length', 'sepal width'], axis=1)
```

```
data.head()
```

```
[2]:    petal length  petal width        class
     0           1.4          0.2  Iris-setosa
     1           1.4          0.2  Iris-setosa
     2           1.3          0.2  Iris-setosa
     3           1.5          0.2  Iris-setosa
     4           1.4          0.2  Iris-setosa
```

```
[3]: # Replace the Iris-setosa and Iris-versicolor classes with Iris-other
     data['class'] = data['class'].replace('Iris-versicolor', 'Iris-other')
     data['class'] = data['class'].replace('Iris-setosa', 'Iris-other')
```

Next, we will split our dataset into three subsets: training (60%), validation (20%), and test (20%).

```
[4]: import numpy as np

     # The first parameter is the shuffled data frame
     # The second parameter is the split indices which are at 60% of the data and␣
       ↪80% of the data
     train, val, test = np.split(data.sample(frac=1, random_state=42), [int(.
       ↪6*len(data)), int(.8*len(data))])
```

Print the number of samples in each of the three subsets and the number of instances from each class. For example, for the training set you might print "The training set has ___ instances (___ virginica, ___ other)".

```
[5]: # YOUR CODE HERE

     print('Train set has {} instances ({} virginica, {} other)'.format(train.
       ↪shape[0],
                                                                    ␣
       ↪train[train['class'] == 'Iris-virginica'].shape[0],
                                                                    ␣
       ↪train[train['class'] == 'Iris-other'].shape[0]))

     print('Validation set has {} instances ({} virginica, {} other)'.format(val.
       ↪shape[0],
                                                                         ␣
       ↪val[val['class'] == 'Iris-virginica'].shape[0],
                                                                         ␣
       ↪val[val['class'] == 'Iris-other'].shape[0]))

     print('Test set has {} instances ({} virginica, {} other)'.format(test.shape[0],
                                                                   ␣
       ↪test[test['class'] == 'Iris-virginica'].shape[0],
↪test[test['class'] == 'Iris-other'].shape[0]))
```

```
Train set has 90 instances (26 virginica, 64 other)
Validation set has 30 instances (12 virginica, 18 other)
Test set has 30 instances (12 virginica, 18 other)
```

1.1 Support vector machines

Support vector machines (SVMs) are a supervised learning method that finds the hyperplane (or set of hyperplanes) in the n-dimensional feature space (where n is the number of input features) which maximizes the distance to the nearest training samples from each class. Maximizing this margin ensures that the decision boundary will be as generalizable as possible to new, unseen data points.

[6]:
```
# import the Support vector classifier
from sklearn.svm import SVC
```

The main hyperparameter to choose is the regularization parameter C, which represents the strength of the penalty incurred during training for allowing samples to be closer to the margin boundary (since a perfect decision boundary is not attainable for most problems).

SVM also uses a kernel function K to map samples to a higher dimensional space (this is referred to as the "kernel trick"). The SVM implementation in scikit-learn gives four options for the kernel function: linear (this is the standard SVM without non-linear kernel), polynomial, radial basis function (RBF), and sigmoid.

The below example uses a linear kernel with $C = 0.1$.

[7]:
```
C = 0.1
linear_svc = SVC(kernel='linear', C=C)
```

[8]:
```
# train the SVM classifier
linear_svc.fit(train[['petal length','petal width']], train['class'])
```

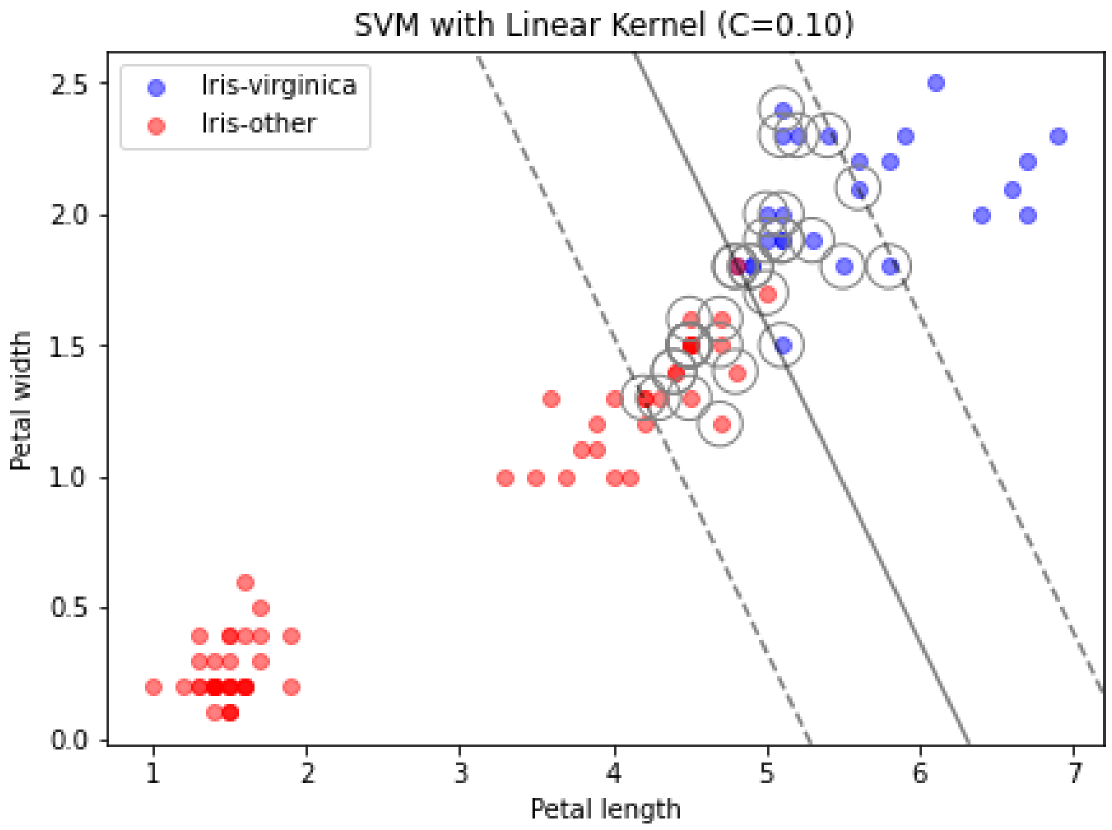
[8]:
```
SVC(C=0.1, kernel='linear')
```

The following function for plotting the decision boundary is from the Python Data Science Handbook by Jake VanderPlas. The function plots the decision boundary as a gray solid line and the margins as gray dashed lines. The support vectors are circled.

[9]:
```
import matplotlib.pyplot as plt

def plot_svc_decision_function(model, ax=None, plot_support=True):
    """Plot the decision function for a 2D SVC"""
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()
    # create grid to evaluate model
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    # plot decision boundary and margins
    ax.contour(X, Y, P, colors='k',
               levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])

    # plot support vectors
    if plot_support:
        ax.scatter(model.support_vectors_[:, 0],
                   model.support_vectors_[:, 1],
                   s=300, linewidth=1, facecolors='none', edgecolors='gray');
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
```

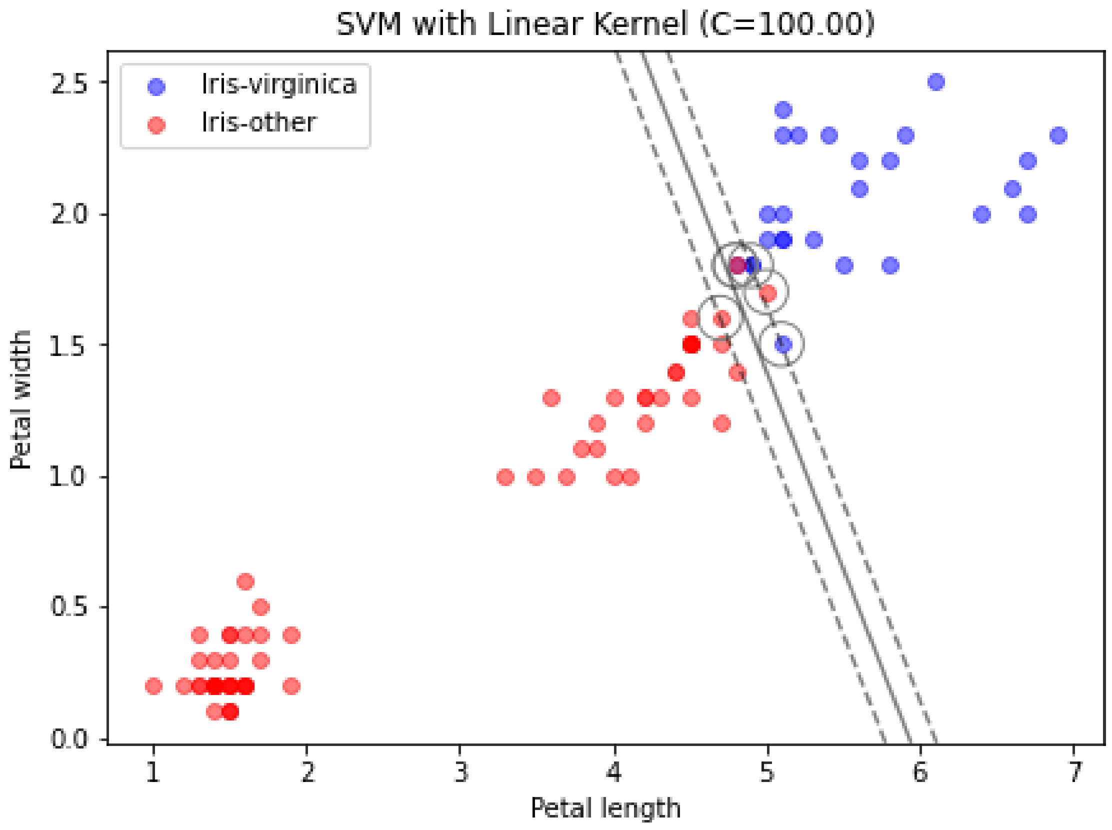
Plot the dataset and the learned decision boundary.

[10]:
```
fig, ax = plt.subplots(1, figsize=(7,5))
# Plot the setosa instances
ax.scatter(train[train['class'] == 'Iris-virginica']['petal length'],
               train[train['class'] == 'Iris-virginica']['petal width'],
               label='Iris-virginica',
               color='blue',
               alpha=0.5)
# Plot the other instances
ax.scatter(train[train['class'] == 'Iris-other']['petal length'],
               train[train['class'] == 'Iris-other']['petal width'],
               label='Iris-other',
               color='red',
               alpha=0.5)

plot_svc_decision_function(linear_svc, ax=ax)
ax.set_xlabel('Petal length')
ax.set_ylabel('Petal width')
ax.legend()
ax.set_title('SVM with Linear Kernel (C=%0.2f)' % C);
```



Train a new model with $C = 100$ and plot the resulting decision boundary.

[11]:
```
# YOUR CODE HERE

C = 100
linear_svc_c100 = SVC(kernel='linear', C=C)

# train the SVM classifier
linear_svc_c100.fit(train[['petal length','petal width']], train['class'])

fig, ax = plt.subplots(1, figsize=(7,5))
# Plot the setosa instances
ax.scatter(train[train['class'] == 'Iris-virginica']['petal length'],
               train[train['class'] == 'Iris-virginica']['petal width'],
               label='Iris-virginica',
               color='blue',
               alpha=0.5)
# Plot the other instances
ax.scatter(train[train['class'] == 'Iris-other']['petal length'],
               train[train['class'] == 'Iris-other']['petal width'],
               label='Iris-other',
                color='red',
                alpha=0.5)

plot_svc_decision_function(linear_svc_c100, ax=ax)
ax.set_xlabel('Petal length')
ax.set_ylabel('Petal width')
ax.legend()
ax.set_title('SVM with Linear Kernel (C=%0.2f)' % C);
```



Question 1:

How did a higher value of C affect the decision boundary? Why?

Answer:

YOUR ANSWER HERE

Higher values of C place more emphasis on minimizing the training error, thus there are fewer points considered as support vectors and the margin is tighter.

Compute and print the accuracy of each classifier (with $C = 0.1$ and $C = 100$) on the validation set.

[12]:
```
from sklearn.metrics import accuracy_score
# YOUR CODE HERE

val_pred = linear_svc.predict(val[['petal length','petal width']])
val_pred_100 = linear_svc_c100.predict(val[['petal length','petal width']])

print('Accuracy on val data for C=0.1: %.2f' % (accuracy_score(val['class'],␣
 ↪val_pred)))
print('Accuracy on val data for C=100: %.2f' % (accuracy_score(val['class'],␣
 ↪val_pred_100)))
```

```
Accuracy on val data for C=0.1: 0.90
Accuracy on val data for C=100: 0.97
```

Question 2:

Which value of C resulted in higher validation accuracy?

Answer:

YOUR ANSWER HERE

C=100

In addition to linear SVM, we can also implement SVM with polynomial, radial basis function (RBF), and sigmoid kernels. Train an SVM classifier with each kernel separately. Set C to be the value of C with highest validation accuracy from Question 2.

You may need to consult the sklearn documentation. The polynomial kernel requires the `degree` parameter to be passed; use `degree=3`.

[13]:
```
# RBF
# YOUR CODE HERE

rbf_svc = SVC(kernel='rbf', C=100).fit(train[['petal length','petal width']],␣
 ↪train['class'])
```

[14]:
```
# Polynomial (order=3)
# YOUR CODE HERE

poly_svc = SVC(kernel='poly', degree=3, C=100).fit(train[['petal length','petal␣
 ↪width']], train['class'])
```

[15]:
```
# Sigmoid
# YOUR CODE HERE

sigm_svc = SVC(kernel='sigmoid', C=100).fit(train[['petal length','petal␣
 ↪width']], train['class'])
```

Compute and print the validation accuracy for each of the 3 classifiers.

[16]:
```
# YOUR CODE HERE

val_pred_rbf = rbf_svc.predict(val[['petal length','petal width']])
val_pred_poly = poly_svc.predict(val[['petal length','petal width']])
val_pred_sigm = sigm_svc.predict(val[['petal length','petal width']])

print('Accuracy on val data for RBF kernel: %.2f' %␣
 ↪(accuracy_score(val['class'], val_pred_rbf)))
print('Accuracy on val data for polynomial kernel: %.2f' %␣
 ↪(accuracy_score(val['class'], val_pred_poly)))
print('Accuracy on val data for sigmoid kernel: %.2f' %␣
 ↪(accuracy_score(val['class'], val_pred_sigm)))
```

```
Accuracy on val data for RBF kernel: 0.97
Accuracy on val data for polynomial kernel: 0.97
Accuracy on val data for sigmoid kernel: 0.43
```

Question 3:

Which of the four kernels (the three above + linear) gave the highest validation accuracy? (If multiple tied for the highest accuracy, list all of them.)

Answer:

YOUR ANSWER HERE

linear, RBF, and polynomial kernel all had 97% validation accuracy

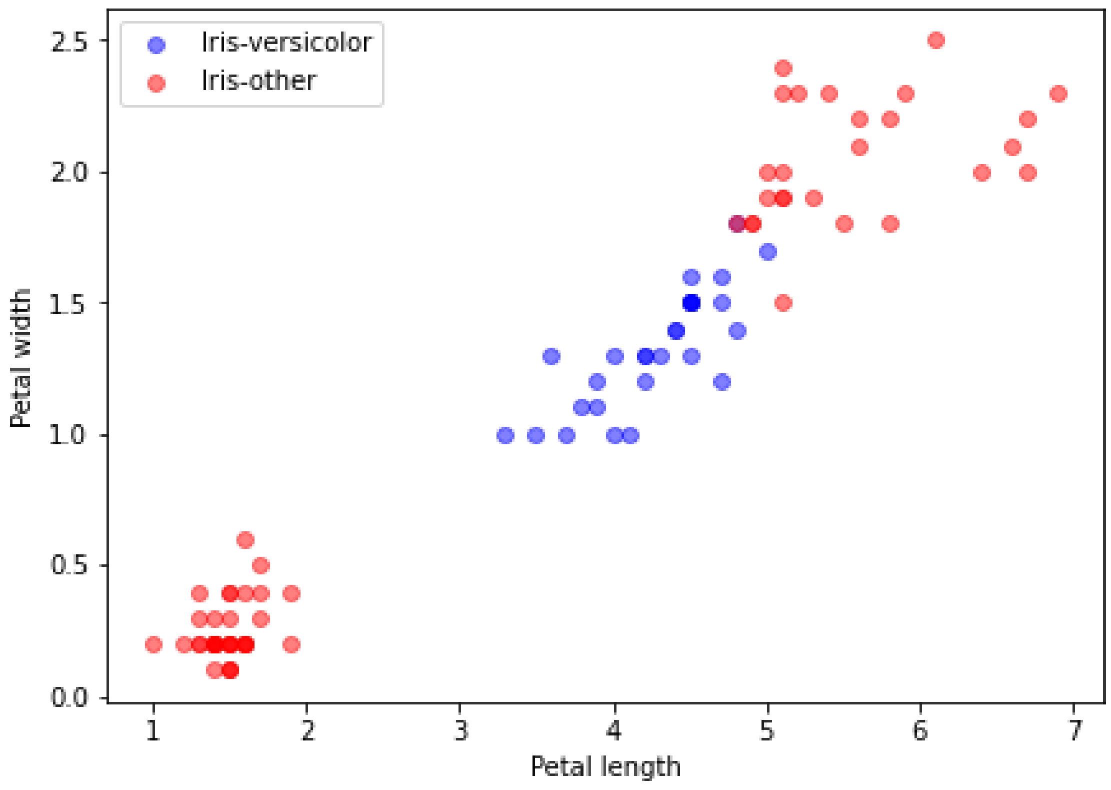
1.1.1 SVM with non-linear decision boundary

For the Iris-virginica vs. Iris-other version of the Iris dataset, the two classes were mostly linearly separable with a small number of classification errors. However, if we instead wanted to classify Iris-versicolor vs. Iris-other, the two classes would not be linearly separable. Below, we load the dataset again and convert it to Iris-versicolor vs. Iris-other and plot the dataset as a scatter plot.

[17]:
```
# Load the dataset
data = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/
 ↪iris/iris.data',header=None)
data.columns = ['sepal length', 'sepal width', 'petal length', 'petal width',␣
 ↪'class']

# Drop the sepal features
data = data.drop(['sepal length', 'sepal width'], axis=1)

# Replace the Iris-virginica and Iris-setosa classes with Iris-other
data['class'] = data['class'].replace('Iris-virginica', 'Iris-other')
data['class'] = data['class'].replace('Iris-setosa', 'Iris-other')

# Split the data into train/val/test
```

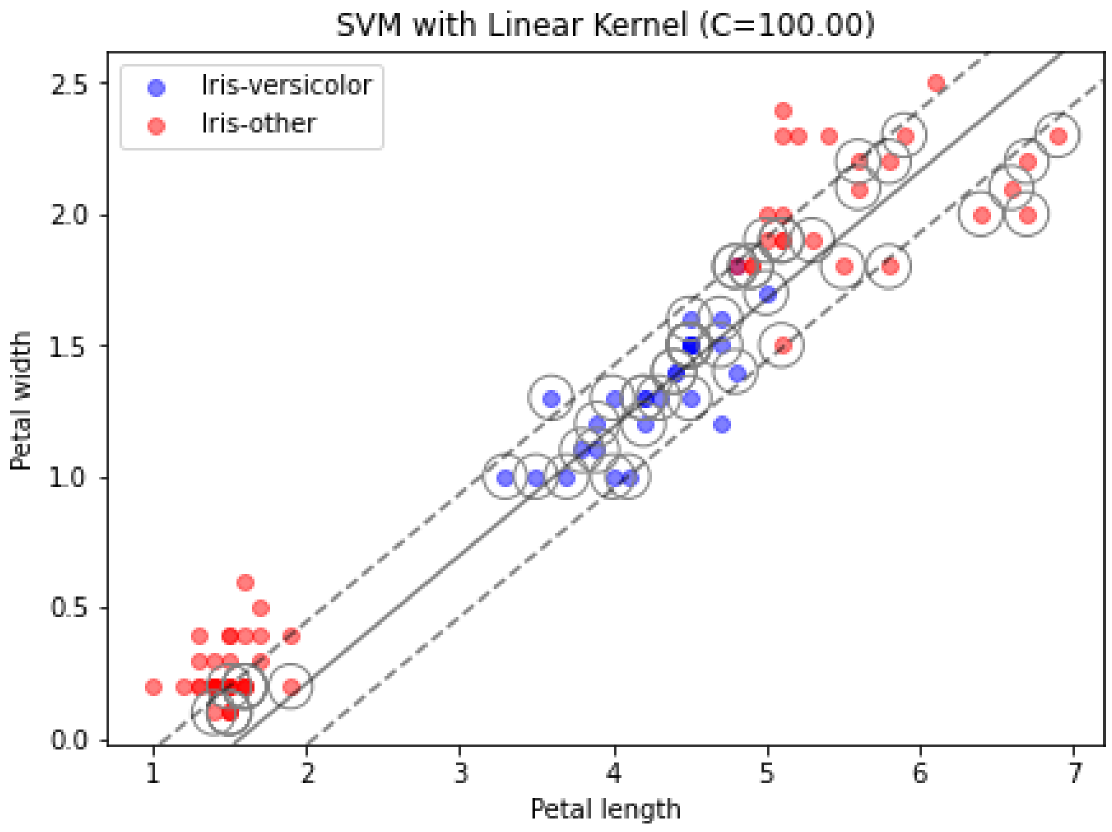
```
train, val, test = np.split(data.sample(frac=1, random_state=42), [int(.
 ↪6*len(data)), int(.8*len(data))])

# Plot the dataset
fig, ax = plt.subplots(1, figsize=(7,5))
# Plot the versicolor instances
ax.scatter(train[train['class'] == 'Iris-versicolor']['petal length'],
               train[train['class'] == 'Iris-versicolor']['petal width'],
               label='Iris-versicolor',
               color='blue',
               alpha=0.5)

# Plot the other instances
ax.scatter(train[train['class'] == 'Iris-other']['petal length'],
               train[train['class'] == 'Iris-other']['petal width'],
               label='Iris-other',
               color='red',
               alpha=0.5)

ax.set_xlabel('Petal length')
ax.set_ylabel('Petal width')
ax.legend()
```

```
[17]: <matplotlib.legend.Legend at 0x7fb140df9b70>
```



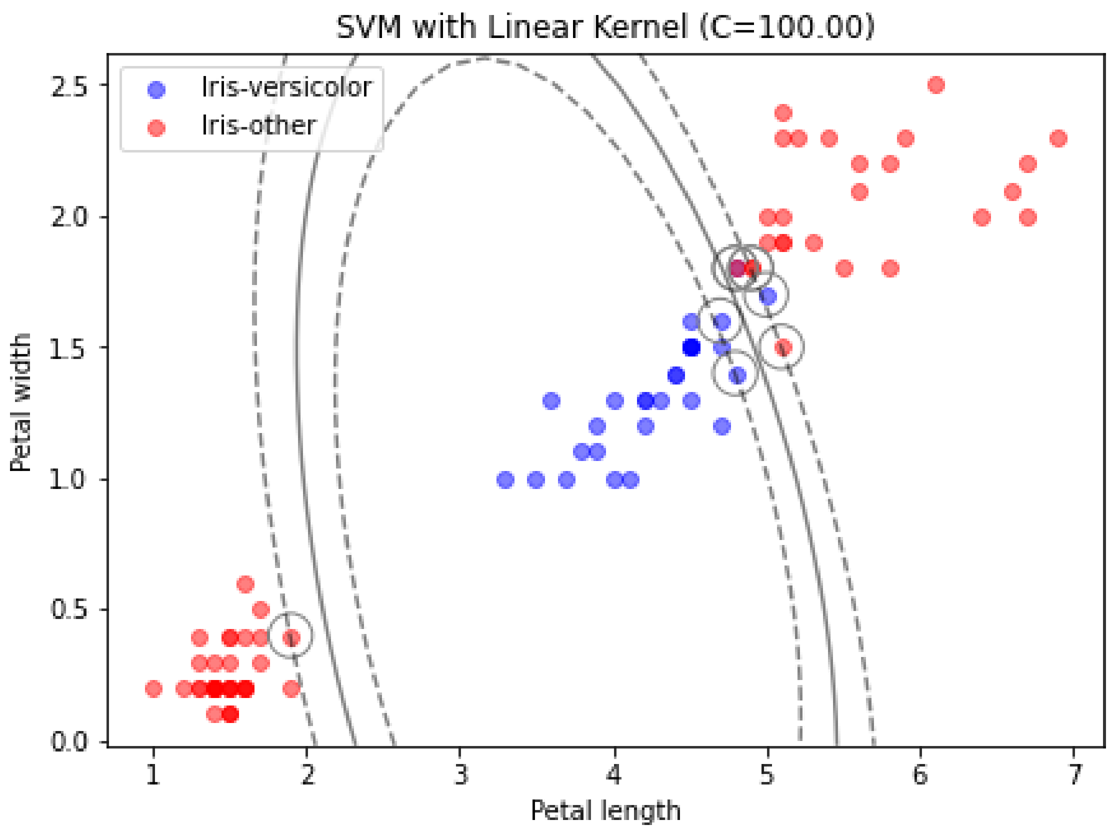
If we train a linear SVM to classify these instances, the accuracy will be low. Train a linear SVM and plot the decision boundary to show this (use $C = 100$).

[18]:
```
# YOUR CODE HERE

C = 100
linear_svc_c100 = SVC(kernel='linear', C=C)

# train the SVM classifier
linear_svc_c100.fit(train[['petal length','petal width']], train['class'])

fig, ax = plt.subplots(1, figsize=(7,5))
# Plot the versicolor instances
ax.scatter(train[train['class'] == 'Iris-versicolor']['petal length'],
               train[train['class'] == 'Iris-versicolor']['petal width'],
               label='Iris-versicolor',
               color='blue',
               alpha=0.5)
# Plot the other instances
ax.scatter(train[train['class'] == 'Iris-other']['petal length'],
               train[train['class'] == 'Iris-other']['petal width'],
               label='Iris-other',
               color='red',
               alpha=0.5)

plot_svc_decision_function(linear_svc_c100, ax=ax)
ax.set_xlabel('Petal length')
ax.set_ylabel('Petal width')
ax.legend()
ax.set_title('SVM with Linear Kernel (C=%0.2f)' % C);
```



The radial basis function kernel will allow us to learn an elliptical decision boundary that will better fit the data. Train an SVM with RBF kernel and plot the decision boundary (use $C = 100$).

[19]:
```
# YOUR CODE HERE

C = 100
rbf_svc_c100 = SVC(kernel='rbf', C=C)

# train the SVM classifier
rbf_svc_c100.fit(train[['petal length','petal width']], train['class'])

fig, ax = plt.subplots(1, figsize=(7,5))
# Plot the versicolor instances
ax.scatter(train[train['class'] == 'Iris-versicolor']['petal length'],
               train[train['class'] == 'Iris-versicolor']['petal width'],
               label='Iris-versicolor',
               color='blue',
               alpha=0.5)
# Plot the other instances
ax.scatter(train[train['class'] == 'Iris-other']['petal length'],
               train[train['class'] == 'Iris-other']['petal width'],
                label='Iris-other',
                color='red',
                alpha=0.5)

plot_svc_decision_function(rbf_svc_c100, ax=ax)
ax.set_xlabel('Petal length')
ax.set_ylabel('Petal width')
ax.legend()
ax.set_title('SVM with Linear Kernel (C=%0.2f)' % C);
```



Compute and print the validation accuracy of your linear and RBF SVM classifiers.

[20]:
```
# YOUR CODE HERE

val_pred_rbf = rbf_svc_c100.predict(val[['petal length','petal width']])
val_pred_linear = linear_svc_c100.predict(val[['petal length','petal width']])

print('Accuracy on val data for RBF kernel: %.2f' %␣
 ↪(accuracy_score(val['class'], val_pred_rbf)))
print('Accuracy on val data for linear SVM: %.2f' %␣
 ↪(accuracy_score(val['class'], val_pred_linear)))
```

```
Accuracy on val data for RBF kernel: 0.97
Accuracy on val data for linear SVM: 0.70
```

Finally, use the best model (the one with the highest validation accuracy in the last cell) to compute the final accuracy on our test set.

[21]:
```
# YOUR CODE HERE

test_pred_rbf = rbf_svc_c100.predict(test[['petal length','petal width']])

print('Accuracy on test data for RBF kernel: %.2f' %␣
↪(accuracy_score(test['class'], test_pred_rbf)))
```

```
Accuracy on test data for RBF kernel: 0.97
```

1.1.2 Put everything together

In this last section, we load the full iris dataset including all features and the three classes and split it into train/val/test subsets. Then you will implement the following steps: 1. Train an SVM with RBF kernel using all four input features and all 3 classes. 2. Evaluate whether $C = 0.1$, $C = 1$, $C = 10$, or $C = 100$ gives better validation accuracy. Print the validation accuracy for both. 3. Use the model with the setting of C that gave the highest validation accuracy to make predictions for the *test* set and compute/print the final accuracy on the test set.

[22]:
```
# Load the dataset
data = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/
↪iris/iris.data',header=None)
data.columns = ['sepal length', 'sepal width', 'petal length', 'petal width',␣
↪'class']

# Split the data into train/val/test
train, val, test = np.split(data.sample(frac=1, random_state=42), [int(.
↪6*len(data)), int(.8*len(data))])
```

[23]:
```
# Train SVMs and evaluate validation accuracy
# YOUR CODE HERE

# Train SVMs
rbf_svc_c01 = SVC(kernel='rbf', C=0.1).fit(train[train.columns[:-1]],␣
↪train['class'])
rbf_svc_c1 = SVC(kernel='rbf', C=1).fit(train[train.columns[:-1]],␣
↪train['class'])
rbf_svc_c10 = SVC(kernel='rbf', C=10).fit(train[train.columns[:-1]],␣
↪train['class'])
rbf_svc_c100 = SVC(kernel='rbf', C=100).fit(train[train.columns[:-1]],␣
↪train['class'])

# Predict on validation set and compute accuracy
```

```
val_pred_c01 = rbf_svc_c01.predict(val[val.columns[:-1]])
val_pred_c1 = rbf_svc_c1.predict(val[val.columns[:-1]])
val_pred_c10 = rbf_svc_c10.predict(val[val.columns[:-1]])
val_pred_c100 = rbf_svc_c100.predict(val[val.columns[:-1]])

print('Accuracy on val data for RBF kernel with C=0.1: %.2f' %␣
 ↪(accuracy_score(val['class'], val_pred_c01)))
print('Accuracy on val data for RBF kernel with C=1: %.2f' %␣
 ↪(accuracy_score(val['class'], val_pred_c1)))
print('Accuracy on val data for RBF kernel with C=10: %.2f' %␣
 ↪(accuracy_score(val['class'], val_pred_c10)))
print('Accuracy on val data for RBF kernel with C=100: %.2f' %␣
 ↪(accuracy_score(val['class'], val_pred_c100)))
```

```
Accuracy on val data for RBF kernel with C=0.1: 0.73
Accuracy on val data for RBF kernel with C=1: 0.93
Accuracy on val data for RBF kernel with C=10: 0.97
Accuracy on val data for RBF kernel with C=100: 0.93
```

[24]:
```
# Use best model to predict test set and compute test accuracy
# YOUR CODE HERE

# Predict on test set and compute accuracy
test_pred = rbf_svc_c10.predict(test[test.columns[:-1]])

print('Accuracy on test data for RBF kernel with C=10: %.2f' %␣
 ↪(accuracy_score(test['class'], test_pred)))
```

```
Accuracy on test data for RBF kernel with C=10: 0.97
```