# CSE 572: Lab 7

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.

## 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).

```
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()
```

| | 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 |

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

data.head()
```

| | 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 |

```
# 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%).

```
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)".

```
# YOUR CODE HERE
print("\n ==============Train============ \n")
print(train['class'].value_counts())
print("\n =============Test============== \n")
print(test['class'].value_counts())
print("\n ============Val================ \n")
print(val['class'].value_counts())

print("\n ================================ \n")
print('Train set: {} Number of Instances: (virginica: {}, other: {})'.format(train.shape[0], train[train['class'] == 'Iris-virginica'].shape[
print('Validation set: {} Number of Instances: (virginica: {}, other: {})'.format(val.shape[0],val[val['class'] == 'Iris-virginica'].shape[0]
print('Test set: {} Number of Instances: (virginica: {}, other: {})'.format(test.shape[0],test[test['class'] == 'Iris-virginica'].shape[0],te
```

```
 ==============Train============

Iris-other        64
Iris-virginica    26
Name: class, dtype: int64

 =============Test==============

Iris-other        18
Iris-virginica    12
Name: class, dtype: int64

 ============Val================

Iris-other        18
Iris-virginica    12
Name: class, dtype: int64

 ================================

Train set: 90 Number of Instances: (virginica: 26, other: 64)
Validation set: 30 Number of Instances: (virginica: 12, other: 18)
Test set: 30 Number of Instances: (virginica: 12, other: 18)
```

## 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.

```
# 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 $C = 0.1$.

```
C = 0.1
linear_svc = SVC(kernel='linear', C=C)
```

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

```
SVC(C=0.1, kernel='linear')
```

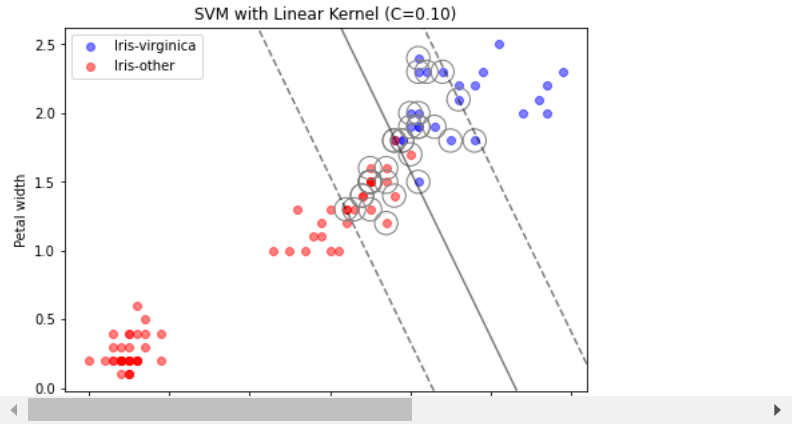
The following function for plotting the decision boundary is from the Python Data Science Handbook by Jake VanderPlas. The function plots the decision boundary as a gray solid line and the margins as gray dashed lines. The support vectors are circled.

```
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.

```
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);
```

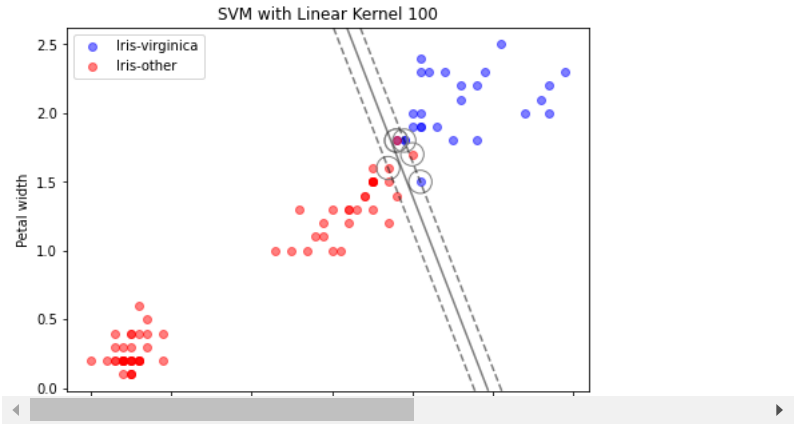
```
/usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X do
  warnings.warn(
```




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

```
# YOUR CODE HERE

C = 100
linear_svc_100 = SVC(kernel='linear', C=C)

# train the SVM classifier
linear_svc_100.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_100, ax=ax)
ax.set_xlabel('Petal length')
ax.set_ylabel('Petal width')
ax.legend()
ax.set_title('SVM with Linear Kernel {}'.format(C));
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X dc
  warnings.warn(
```




**Question 1:**

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

**Answer:**

YOUR ANSWER HERE

When C is higher, the model strives to fit the training data as closely as possible because it is penalized more severely for incorrectly classifying points. Due to overfitting, where the model memorizes the training data and may not generalize well to new data, the decision boundary becomes more complex.

The decision boundary is made simpler and the generalization to new data is better with a smaller value of C, but the training error is larger.

In other words, C controls how much margin slack there is; a lower value of C results in less slack and a more complicated boundary i.e. focus is 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.

```
from sklearn.metrics import accuracy_score
# YOUR CODE HERE

val_Acc = linear_svc.predict(val[['petal length','petal width']])
val_Acc_100 = linear_svc_100.predict(val[['petal length','petal width']])
print('Val Data Accuracy with params: C=0.1: {}'.format((accuracy_score(val['class'],val_Acc))))
```

```
print('Val Data Accuracy with params: C=100: {}'.format((accuracy_score(val['class'],val_Acc_100))))
```

```
Val Data Accuracy with params: C=0.1: 0.9
Val Data Accuracy with params: C=100: 0.9666666666666667
```

**Question 2:**

Which value of $C$ resulted in higher validation accuracy?

**Answer:**

YOUR ANSWER HERE C = 100 resulted in high accuracy.

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`.

```
# RBF
# YOUR CODE HERE
rbf = SVC(kernel='rbf', C=100).fit(train[['petal length','petal width']],train['class'])
```

```
# Polynomial (degree=3)
# YOUR CODE HERE
poly = SVC(kernel='poly', degree=3, C=100).fit(train[['petal length','petal width']], train['class'])
```

```
# Sigmoid
# YOUR CODE HERE

sigm = 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.

```
# YOUR CODE HERE

val_Acc_rbf = rbf.predict(val[['petal length','petal width']])
val_Acc_poly = poly.predict(val[['petal length','petal width']])
val_Acc_sigm = sigm.predict(val[['petal length','petal width']])
print('Val Accuracy for RBF kernel: {}'.format((accuracy_score(val['class'], val_Acc_rbf))))
print('Val Accuracy for polynomial kernel:  {}'.format((accuracy_score(val['class'], val_Acc_poly))))
print('Val Accuracy for sigmoid kernel:  {}'.format((accuracy_score(val['class'], val_Acc_sigm))))
```

```
Val Accuracy for RBF kernel: 0.9666666666666667
Val Accuracy for polynomial kernel:  0.9666666666666667
Val Accuracy for sigmoid kernel:  0.43333333333333335
```

**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 with C=100, RBF, and Polynomial kernel all had 97% validation accuracy.

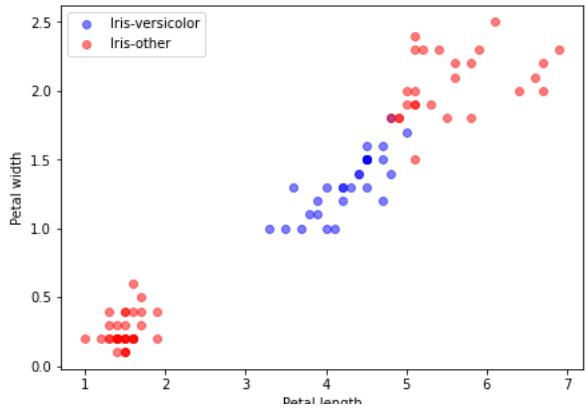
## SVM with non-linear decision boundary

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.

```
# 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()
```

```
<matplotlib.legend.Legend at 0x7ff24b4a3d30>
```

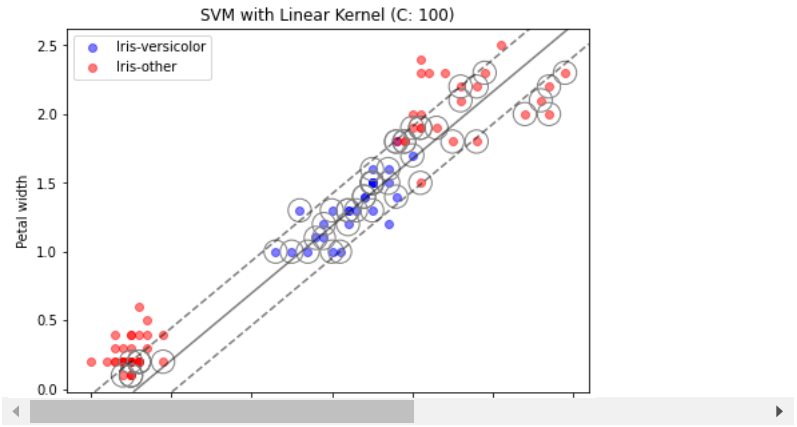



If we train a linear SVM to classify these instances, the accuracy will be low. Train a linear SVM and plot the decision boundary to show this (use $C = 100$).

```
# YOUR CODE HERE

C = 100
linear_svc_100 = SVC(kernel='linear', C=C)


linear_svc_100.fit(train[['petal length','petal width']], train['class'])
fig, ax = plt.subplots(1, figsize=(7,5))


ax.scatter(train[train['class'] == 'Iris-versicolor']['petal length'],
train[train['class'] == 'Iris-versicolor']['petal width'],
label='Iris-versicolor',
color='blue',
alpha=0.5)

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_100, ax=ax)
ax.set_xlabel('Petal length')
ax.set_ylabel('Petal width')
```

```
ax.legend()
ax.set_title('SVM with Linear Kernel (C: {})'.format(C));
```

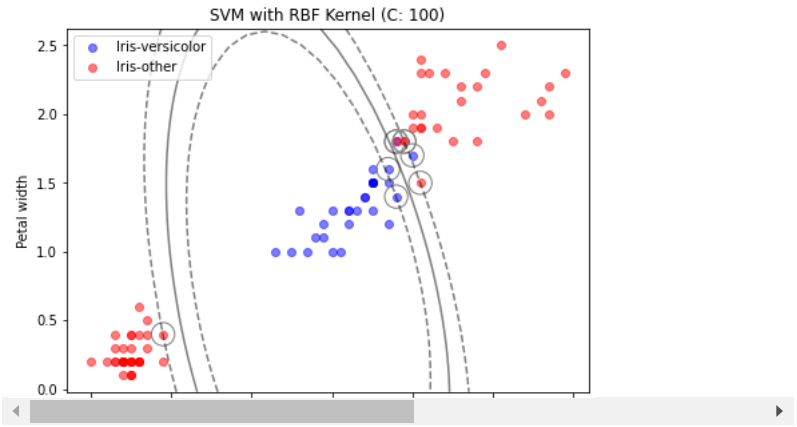
```
/usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X do
  warnings.warn(
```




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 $C = 100$).

```
# YOUR CODE HERE
C = 100
rbf_100 = rbf = SVC(kernel='rbf', C=C)


rbf_100.fit(train[['petal length','petal width']],train['class'])
fig, ax = plt.subplots(1, figsize=(7,5))


ax.scatter(train[train['class'] == 'Iris-versicolor']['petal length'],
train[train['class'] == 'Iris-versicolor']['petal width'],
label='Iris-versicolor',
color='blue',
alpha=0.5)

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_100, ax=ax)
ax.set_xlabel('Petal length')
ax.set_ylabel('Petal width')
ax.legend()
ax.set_title('SVM with RBF Kernel (C: {})'.format(C));
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X do
  warnings.warn(
```




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

```
# YOUR CODE HERE
val_pred_rbf = rbf_100.predict(val[['petal length','petal width']])
val_pred_linear = linear_svc_100.predict(val[['petal length','petal width']])

print('Val Accuracy for linear SVM: {}'.format(accuracy_score(val['class'], val_pred_linear)))
print('Val Accuracy for RBF kernel: {}'.format(accuracy_score(val['class'], val_pred_rbf)))
```

```
Val Accuracy for linear SVM: 0.7
Val Accuracy for RBF kernel: 0.9666666666666667
```

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.

```
# YOUR CODE HERE
test_Y = test[['petal length','petal width']]
test_Acc = rbf_100.predict(test_Y)
print('Accuracy on test data for RBF kernel with C=100: {}'.format((accuracy_score(test['class'], test_Acc))))
```

```
Accuracy on test data for RBF kernel with C=100: 0.9666666666666667
```

✓ 0s completed at 6:49 AM