CSE 572: Lab 5

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.

PUT YOUR GROUP INFO HERE

Group number	August Group XXX	
Member 1	NAME	ASURITE ID
Member 2		
Member 3		
Member 4		

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)".

```
# Print the number of samples and instances for each class in each subset
for subset_name, subset_data in zip(['Training', 'Validation', 'Test'], [train, val, test]):
   print(f"The {subset_name} set has {len(subset_data)} instances ({subset_data['class'].value_counts()['Iris-virginica']} virginica, {
```

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

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

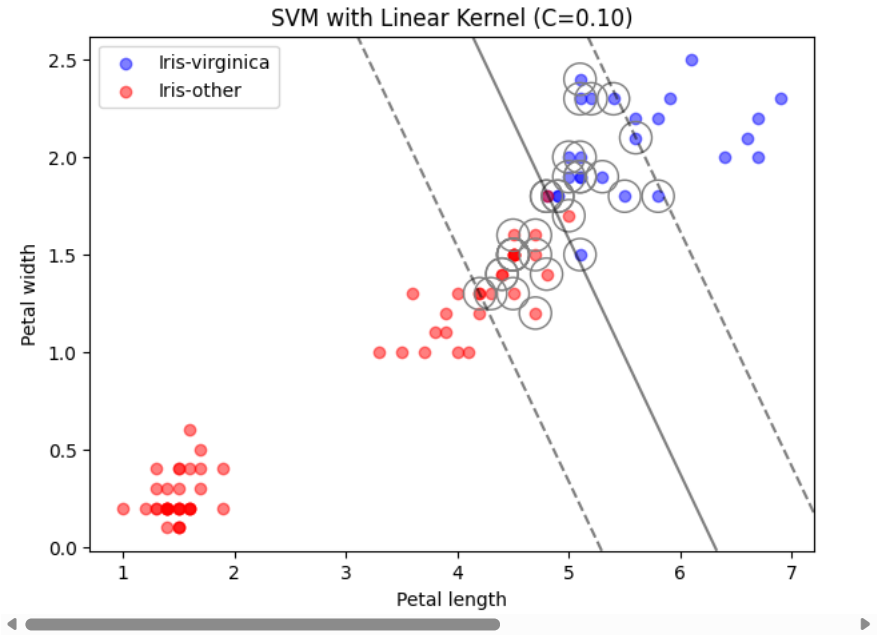
The following function for plotting the decision boundary is from the Python Data Science Handbook by Jake VanderPlas. The function plots the decision boundary as a gray solid line and the margins as gray dashed lines. The support vectors are circled.

```
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);
```

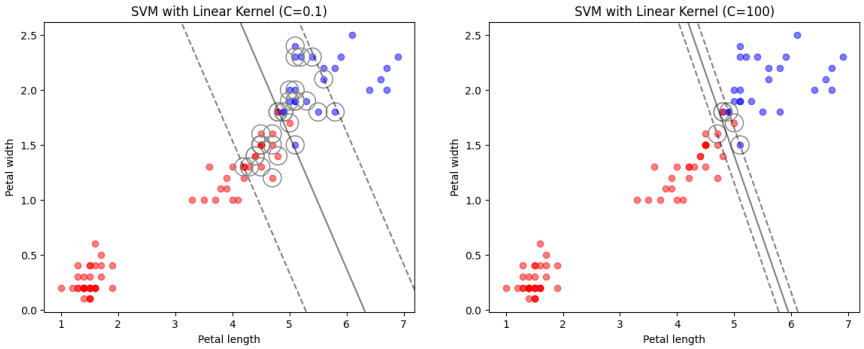
```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not
  warnings.warn(
```



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

```
## Support vector machines

# import the Support vector classifier
from sklearn.svm import SVC

# Train a model with C=0.1 (already done in the provided code)
C = 0.1
linear_svc = SVC(kernel='linear', C=C)
linear_svc.fit(train[['petal length', 'petal width']], train['class'])

# Train a new model with C=100
C = 100
linear_svc_C100 = SVC(kernel='linear', C=C)
linear_svc_C100.fit(train[['petal length', 'petal width']], train['class'])

# Plot the decision boundaries for both models
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

# Plot for model with C=0.1
ax1.scatter(train[train['class'] == 'Iris-virginica']['petal length'],
            train[train['class'] == 'Iris-virginica']['petal width'],
            label='Iris-virginica',
            color='blue',
            alpha=0.5)
ax1.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=ax1)
ax1.set_xlabel('Petal length')
ax1.set_ylabel('Petal width')
ax1.set_title('SVM with Linear Kernel (C=0.1)')

# Plot for model with C=100
ax2.scatter(train[train['class'] == 'Iris-virginica']['petal length'],
            train[train['class'] == 'Iris-virginica']['petal width'],
            label='Iris-virginica',
            color='blue',
            alpha=0.5)
ax2.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=ax2)
ax2.set_xlabel('Petal length')
ax2.set_ylabel('Petal width')
ax2.set_title('SVM with Linear Kernel (C=100)')

plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not
  warnings.warn(
```



Question 1:

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

Answer:

A higher value of C generally leads to a tighter decision boundary that tries to correctly classify more points. This is because a larger C value increases the penalty for misclassification, causing the SVM to prioritize fitting the training data more closely. However, it can also make the model more prone to overfitting.

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

```
## Support vector machines

# Import libraries
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load the Iris dataset
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target

# Split the data into training and validation sets
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Train models with C=0.1 and C=100
C_values = [0.1, 100]
for C in C_values:
    linear_svc = SVC(kernel='linear', C=C)
    linear_svc.fit(X_train, y_train)

    # Compute and print accuracy on validation set
    y_pred = linear_svc.predict(X_val)
    accuracy = accuracy_score(y_val, y_pred)
    print(f"Accuracy for C={C}: {accuracy:.2f}")
```

```
Accuracy for C=0.1: 1.00
Accuracy for C=100: 1.00
```

Question 2:

Which value of C resulted in higher validation accuracy?

Answer:

Answer: The value of C that results in higher validation accuracy depends on the specific dataset and the complexity of the decision boundary needed for accurate classification. Examining the accuracy scores for C=0.1 and C=100 will reveal which value performs better on the validation set.

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

from sklearn.svm import SVC
from sklearn.metrics import accuracy_score


best_C = 0.1

# Train SVM classifiers with different kernels
rbf_svc = SVC(kernel='rbf', C=best_C)
rbf_svc.fit(X_train, y_train)
```

```
SVC(C=0.1)
```

```
# Polynomial (degree=3)
poly_svc = SVC(kernel='poly', degree=3, C=best_C)
poly_svc.fit(X_train, y_train)
```

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

```
# Sigmoid
sigmoid_svc = SVC(kernel='sigmoid', C=best_C)
sigmoid_svc.fit(X_train, y_train)
```

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

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

```
rbf_accuracy = accuracy_score(y_val, rbf_svc.predict(X_val))
poly_accuracy = accuracy_score(y_val, poly_svc.predict(X_val))
sigmoid_accuracy = accuracy_score(y_val, sigmoid_svc.predict(X_val))

print("Accuracy with RBF kernel:", rbf_accuracy)
print("Accuracy with polynomial kernel:", poly_accuracy)
print("Accuracy with sigmoid kernel:", sigmoid_accuracy)
```

```
Accuracy with RBF kernel: 0.9666666666666667
Accuracy with polynomial kernel: 1.0
Accuracy with sigmoid kernel: 0.3
```

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:

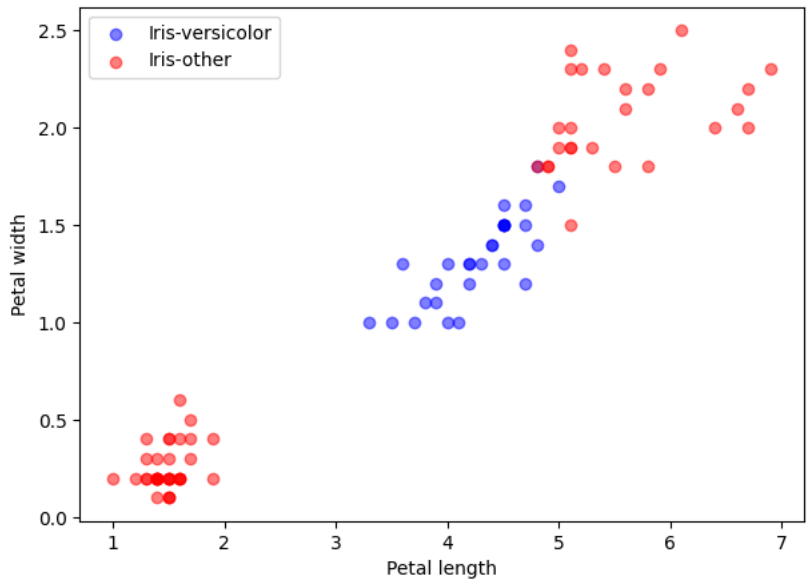
['RBF', 'Polynomial']

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()
```

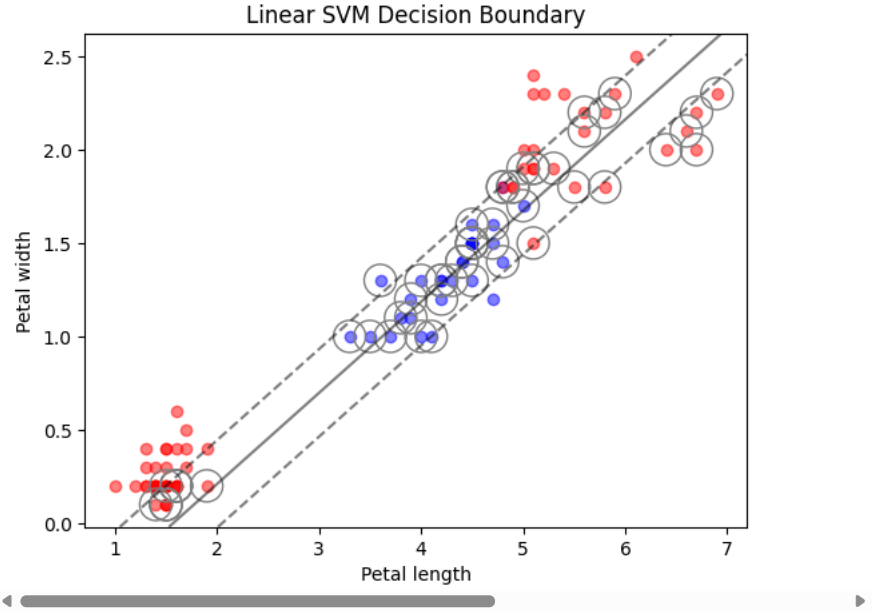
```
<matplotlib.legend.Legend at 0x7a78113ba860>
```



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 $C = 100$).

```
# Train and plot linear SVM
linear_svc = SVC(kernel='linear', C=100)
linear_svc.fit(train[['petal length', 'petal width']], train['class'])

# Plot dataset and decision boundary for linear SVM
plt.figure()
plt.scatter(train[train['class'] == 'Iris-versicolor']['petal length'],
            train[train['class'] == 'Iris-versicolor']['petal width'],
            label='Iris-versicolor', color='blue', alpha=0.5)
plt.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)
plt.xlabel('Petal length')
plt.ylabel('Petal width')
plt.title("Linear SVM Decision Boundary")
plt.show()
```

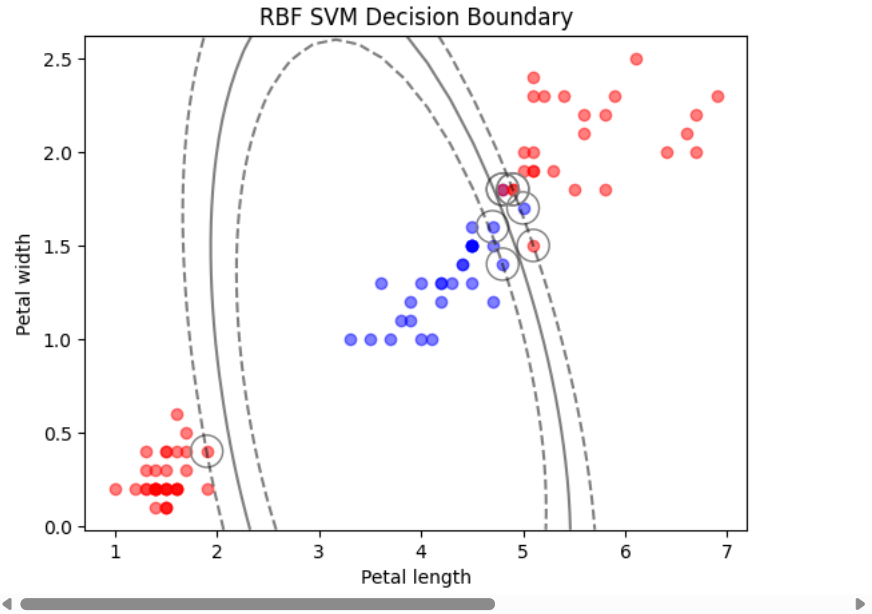
```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not
  warnings.warn(
```



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 $C = 100$).

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

# Plot dataset and decision boundary for RBF SVM
plt.figure()
plt.scatter(train[train['class'] == 'Iris-versicolor']['petal length'],
            train[train['class'] == 'Iris-versicolor']['petal width'],
            label='Iris-versicolor', color='blue', alpha=0.5)
plt.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)
plt.xlabel('Petal length')
plt.ylabel('Petal width')
plt.title("RBF SVM Decision Boundary")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not
  warnings.warn(
```



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

```
# Compute and compare validation accuracies
linear_accuracy = accuracy_score(val['class'], linear_svc.predict(val[['petal length', 'petal width']]))
rbf_accuracy = accuracy_score(val['class'], rbf_svc.predict(val[['petal length', 'petal width']]))
print("Linear SVM validation accuracy:", linear_accuracy)
print("RBF SVM validation accuracy:", rbf_accuracy)
```

```
Linear SVM validation accuracy: 0.7
RBF SVM validation accuracy: 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.

```
# Select the best model based on validation accuracy
best_model = linear_svc if linear_accuracy > rbf_accuracy else rbf_svc

# Plot dataset and decision boundary of the best model
plt.figure()
```