CS 405/605 Data Science

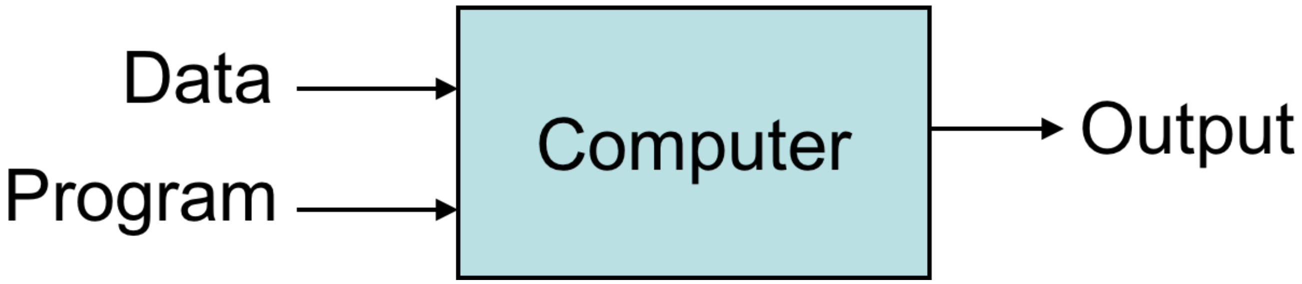
Dr. Qianqian Tong

Introduction to ML: agenda

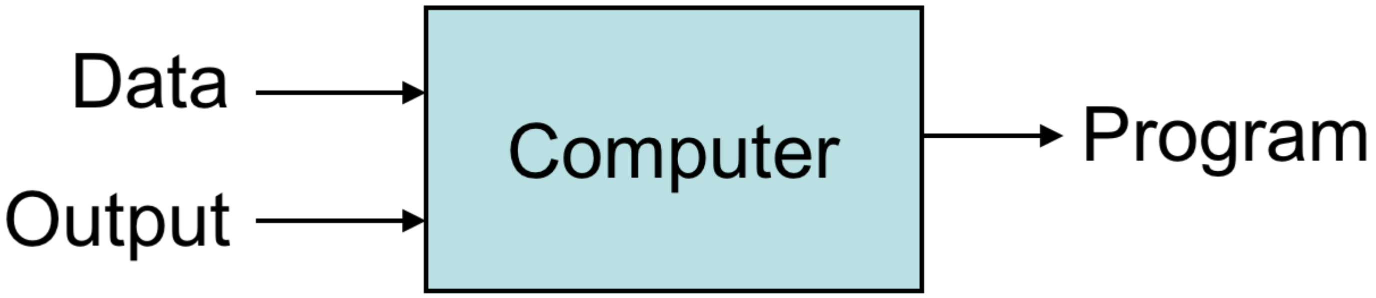
- ML basis
- Linear Regression
- Classification: SVM (kernel)
- Decision Tree & random forest
- Validation
- Dimensionality - PCA
- Clustering: Kmeans
- Visualization

Introduction to ML

- Traditional Programming

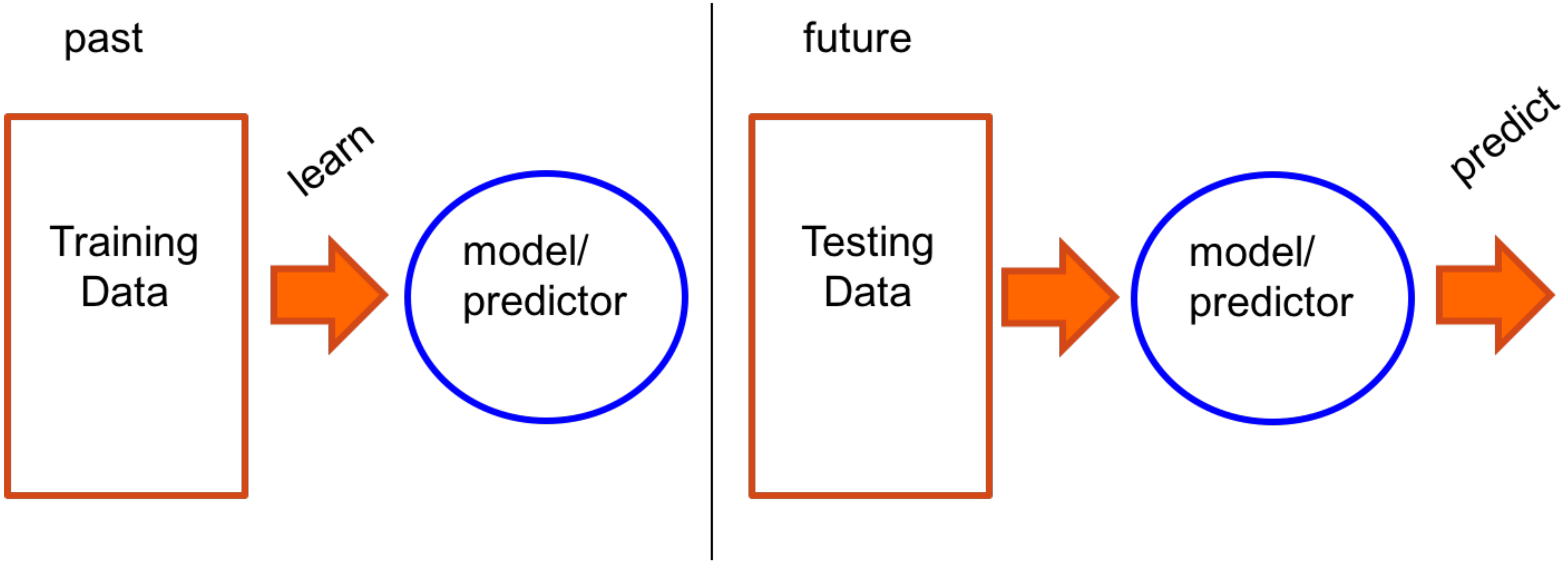


- Machine Learning



Introduction to ML

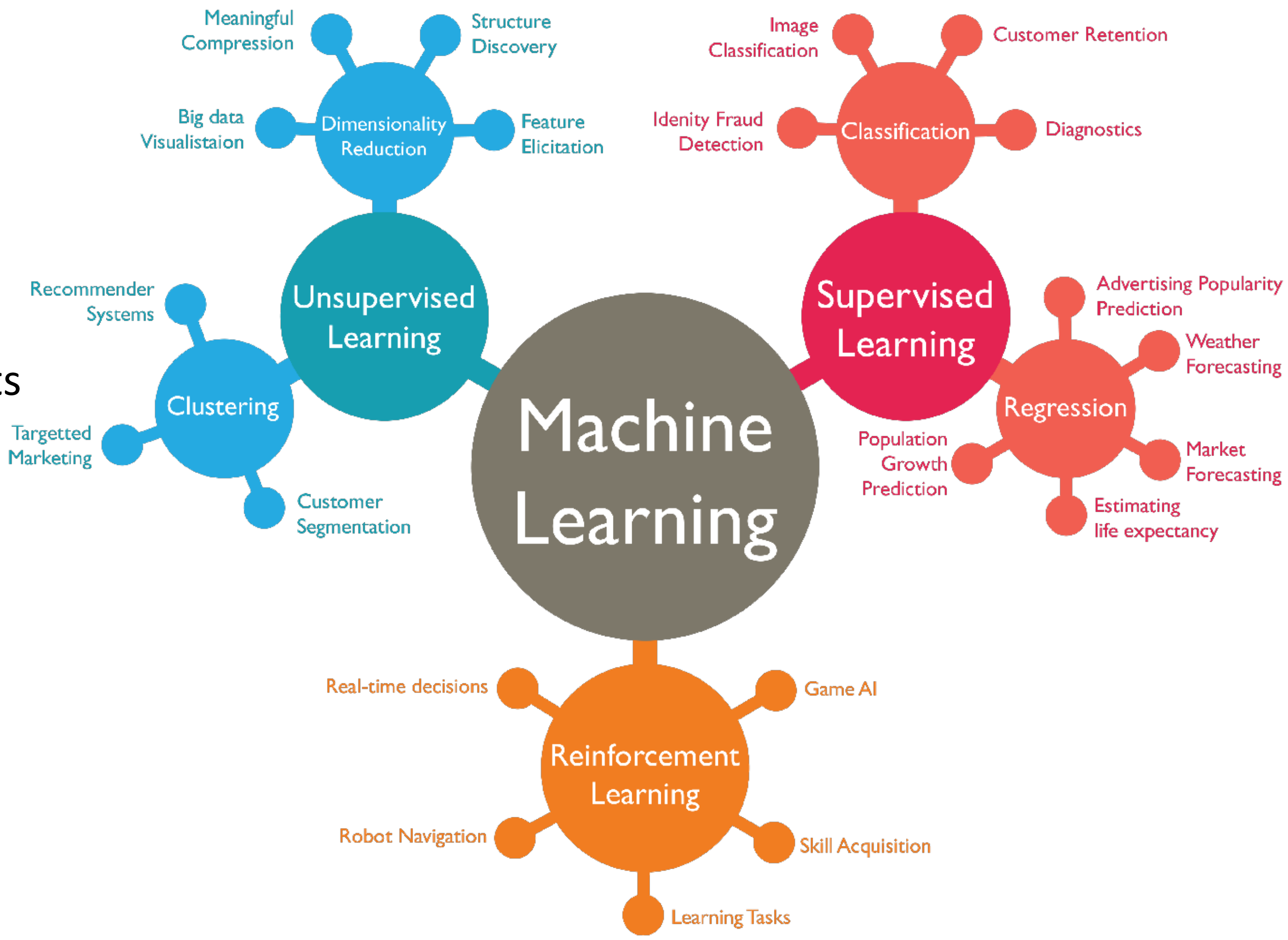
Machine learning is about predicting the future based on the past.



Introduction to ML

Types of Machine Learning Algorithms

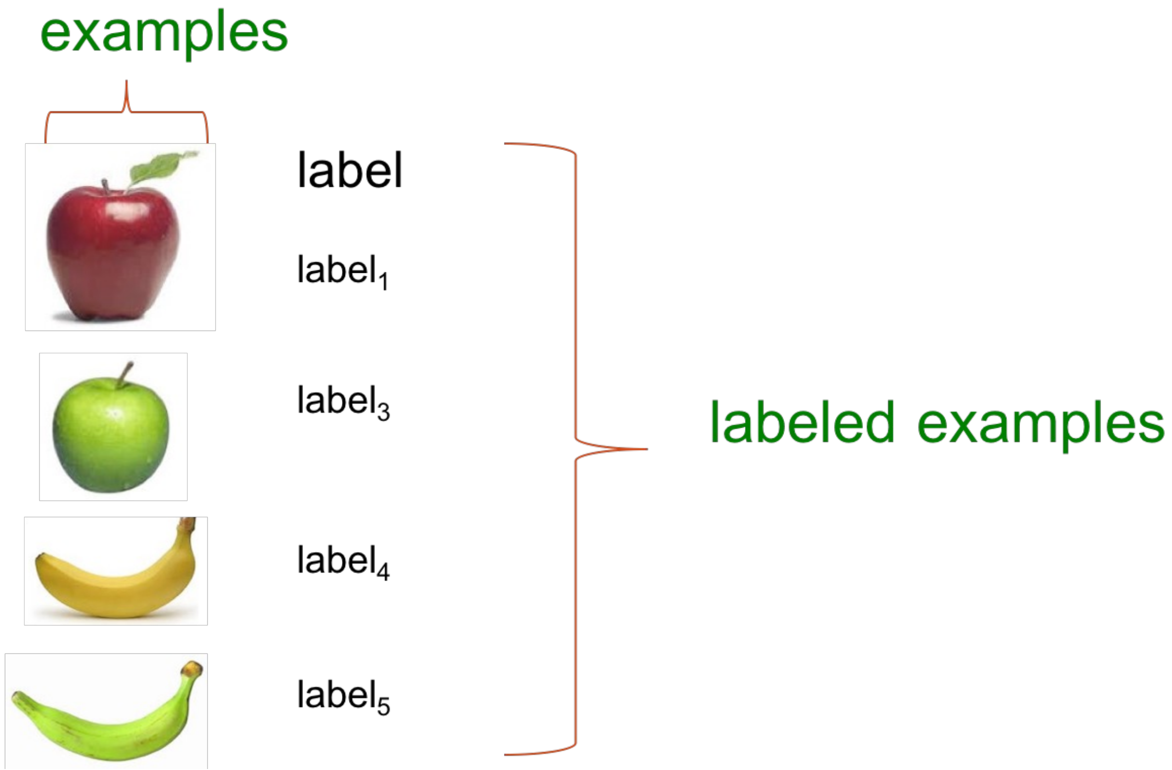
- **Supervised (inductive) learning**
    - Training data includes desired outputs
    - Classification
    - Regression/Prediction
- **Unsupervised learning**
    - Training data does not include desired outputs
- **Semi-supervised learning**
    - Training data includes a few desired outputs
- **Reinforcement learning**
    - Rewards from sequence of actions



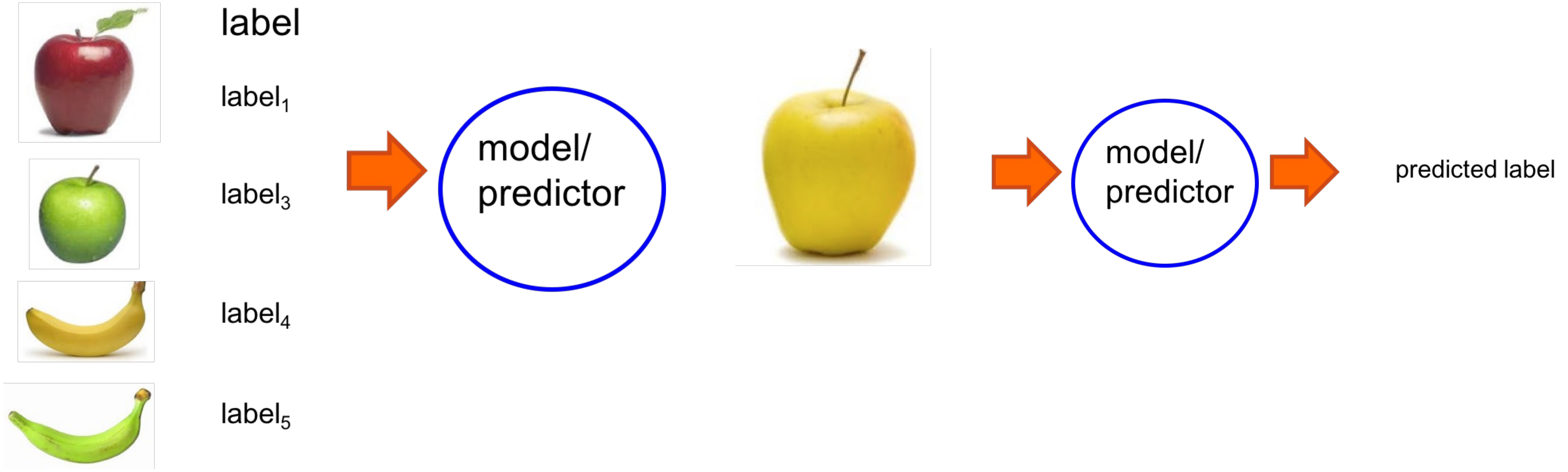
Introduction to ML

Types of Machine Learning Algorithms

- **Supervised (inductive) learning**
  - Training data includes desired outputs
  - Classification
  - Regression/Prediction
- **Unsupervised learning**
  - Training data does not include desired outputs
- **Semi-supervised learning**
  - Training data includes a few desired outputs
- **Reinforcement learning**
  - Rewards from sequence of actions



Introduction to ML



Introduction to ML

Unsupervised learning: given data, i.e. examples, but no labels

- Clustering: Grouping similar instances

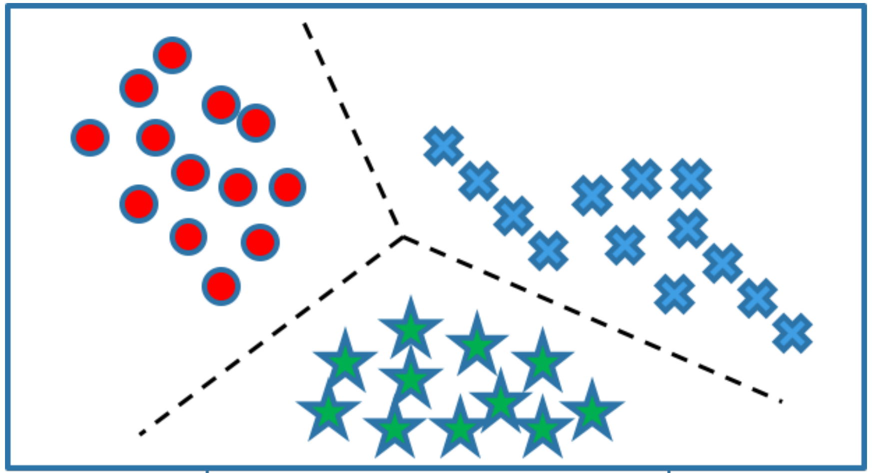




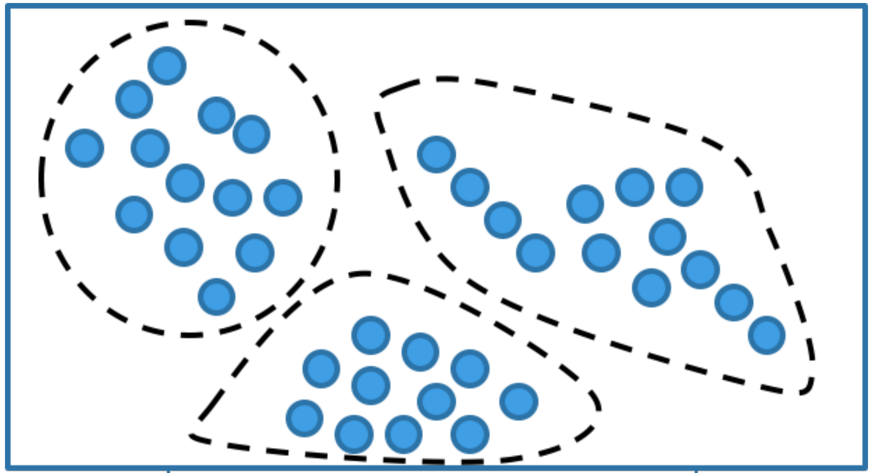
UNC GREENSBORO

Introduction to ML

Supervised Versus Un-Supervised learning



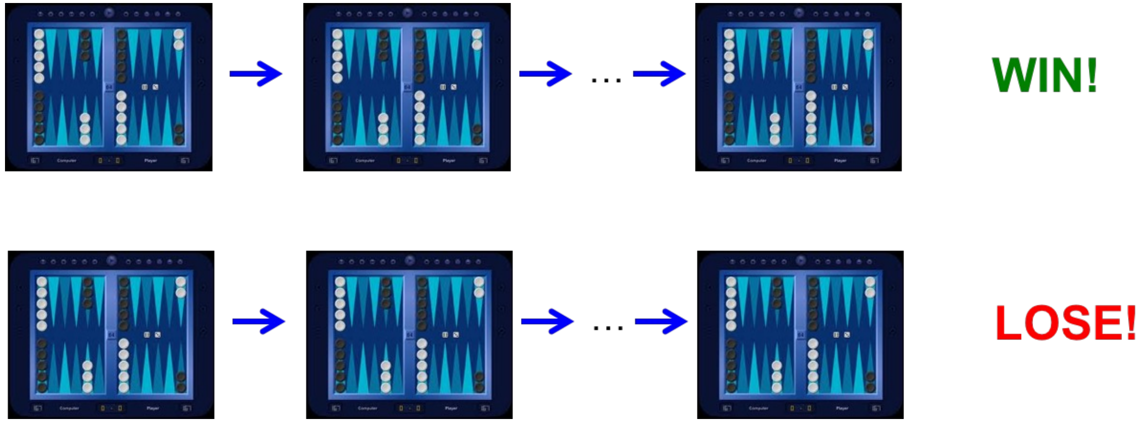
Supervised learning



Unsupervised learning

Introduction to ML

Reinforcement Learning



Given a *****sequence****** of examples/states and a *****reward****** after completing that sequence, learn to predict the action to take in for an individual example/state

Introduction to ML

- Tens of thousands of machine learning algorithms
  - Hundreds new every year
- Every machine learning algorithm has three components:

  Representation ----- Evaluation ----- Optimization

Representation

- What is the model design landscape?

- Decision trees
- Sets of rules / Logic programs
- Instances
- Graphical models (Bayes/Markov nets)
- Neural networks
- Support vector machines
- Model ensembles
- ...

Evaluation

- How is the model doing?

- Accuracy
- Precision and recall
- Squared error
- Likelihood
- Posterior probability
- Cost / Utility
- Margin
- Entropy
- K-L divergence
- ...

Optimization

- How can we get better models?

- Combinatorial optimization
  - Greedy search
- Convex optimization
  - Gradient descent
- Nonconvex optimization
  - Stochastic gradient methods
- Constrained optimization
  - Linear programming

UNC GREENSBORO

Introduction to ML

Growth of Machine Learning

- Machine learning is preferred approach to
  - Speech recognition, Natural language processing
  - Computer vision
  - Medical outcomes analysis
  - Robot control
  - Web search
  - Finance
  - Social Networks

This trend is accelerating

- Improved machine learning algorithms
- Improved data capture, networking, faster computers
- Software too complex to write by hand
- New sensors / IO devices
- Big Data

Scikit-learn

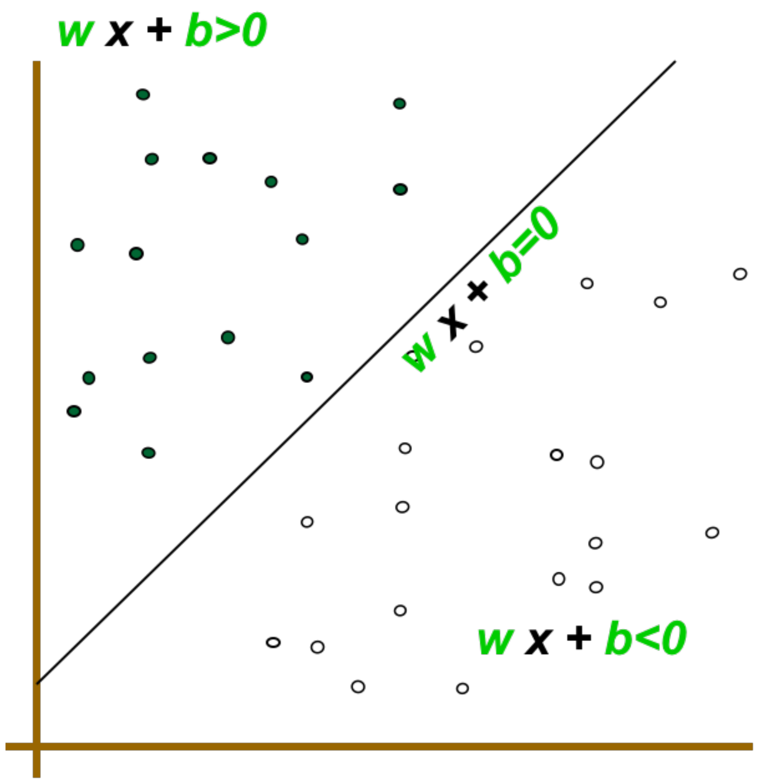
- **Supervised Learning**
  - https://scikit-learn.org/stable/supervised_learning.html
- **Un-Supervised Learning**
  - https://scikit-learn.org/stable/unsupervised_learning.html

- Implement a model: from sklearn.linear_model import LinearRegression
- Train the model: model.fit( )
- Predict: model.predict( )
- Visualize the outcome: plt.plot( )

- **Introduction to Machine Learning**
  - https://github.com/q-tong/CS405-605-Data-Science/blob/main/Fall2023/Lecture/4.Machine%20Learning/Machine_learning/0-Machine_Learning_Overview.ipynb
- **Introduction to Scikit-Learn: Machine Learning with Python**
  - https://github.com/q-tong/CS405-605-Data-Science/blob/main/Fall2023/Lecture/4.Machine%20Learning/Machine_learning/1-Machine-Learning-Intro.ipynb
- **Basic Principles of Machine Learning**
  - https://github.com/q-tong/...... /2-Basic-Principles.ipynb

SVM: Support Vector Machines

- Supervised learning
- **Classification (categorical)**
- Regression (continuous)



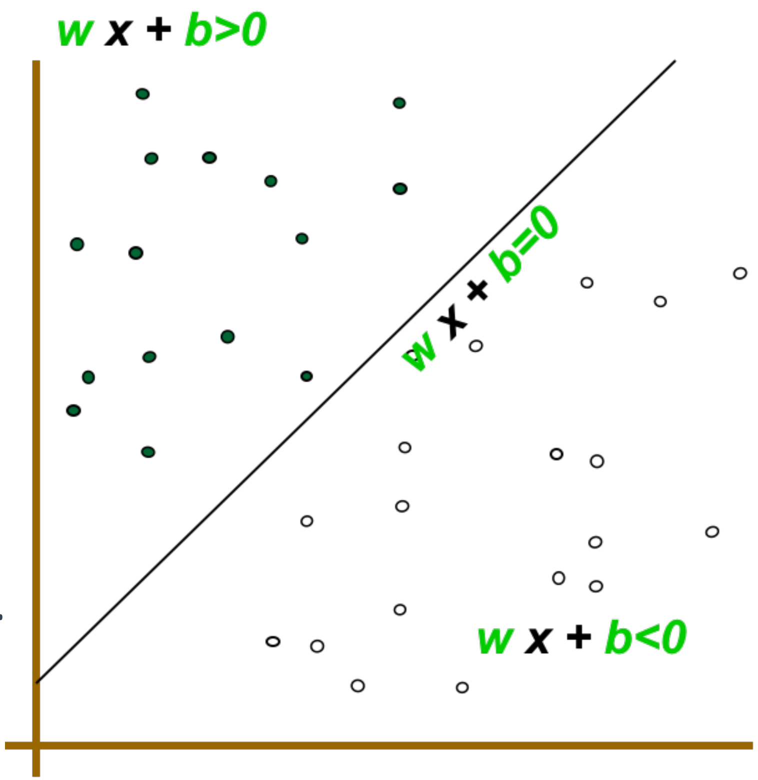
SVM

- **Key Concepts**
- Hyperplane:

  A decision boundary that separates different classes.
- Support Vectors:

  Data points closest to the hyperplane.
- Margin:

  Distance between the hyperplane and the support vectors.



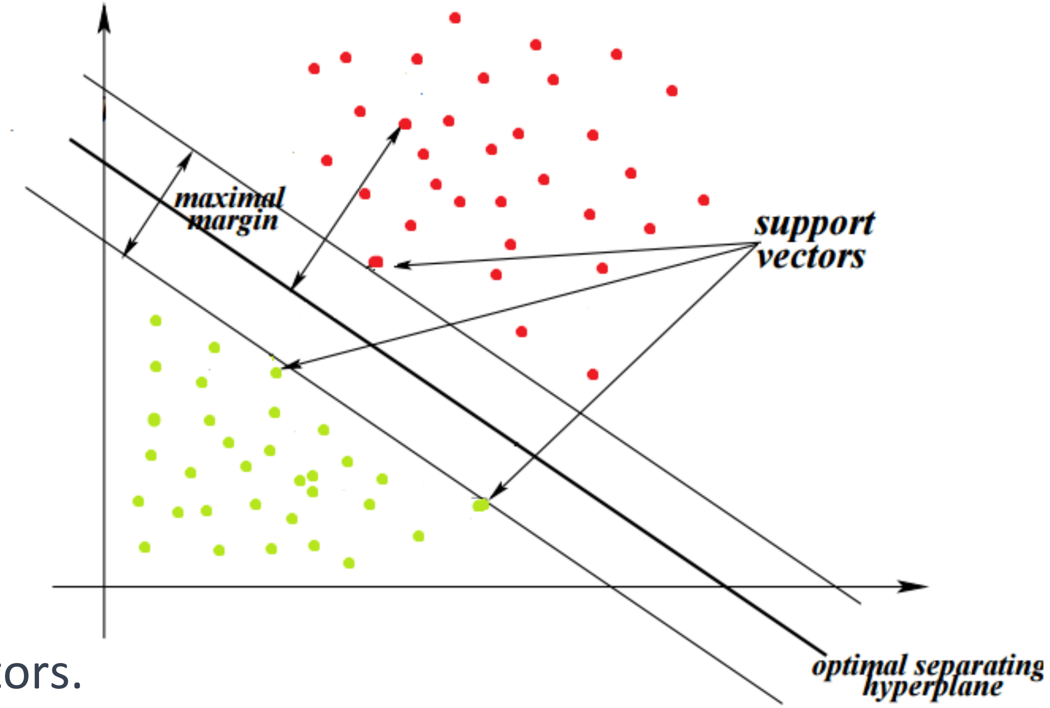
SVM

- **Key Concepts**
- Hyperplane:

  A decision boundary that separates different classes.
- Support Vectors:

  Data points closest to the hyperplane.
- Margin:

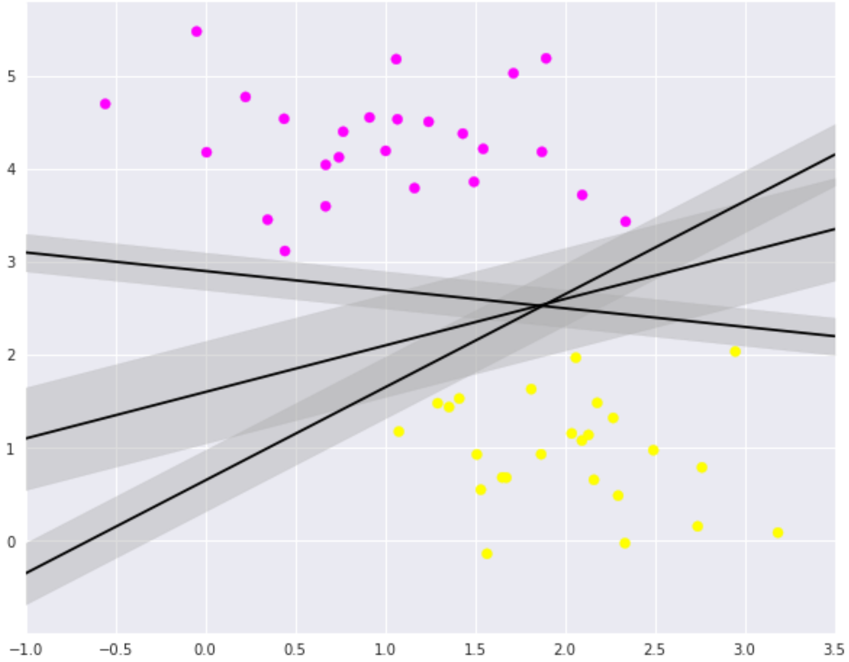
  Distance between the hyperplane and the support vectors.



Maximizing the *Margin*

SVM - Maximizing the Margin

- What support vector machine does is to not only draw a line, but consider a ***region*** about the line of some given width.



UNC GREENSBORO

SVM – Scikit-learn

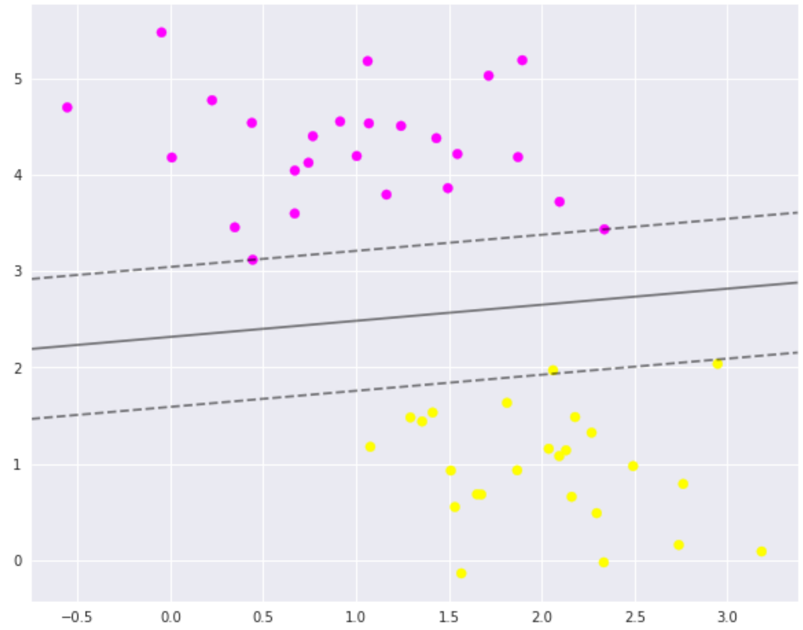
```
from sklearn.svm import SVC # "Support Vector Classifier"
clf = SVC(kernel='linear')
clf.fit(X, y)
```

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

- To better visualize what's happening here, let's create a quick convenience function that will plot SVM decision boundaries for us:

```
def plot_svc_decision_function(clf, ax=None):
    """Plot the decision function for a 2D SVC"""
    if ax is None:
        ax = plt.gca()
    x = np.linspace(plt.xlim()[0], plt.xlim()[1], 30)
    y = np.linspace(plt.ylim()[0], plt.ylim()[1], 30)
    Y, X = np.meshgrid(y, x)
    P = np.zeros_like(X)
    for i, xi in enumerate(x):
        for j, yj in enumerate(y):
            P[i, j] = clf.decision_function([xi, yj])
    # plot the margins
    ax.contour(X, Y, P, colors='k',
               levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])
```

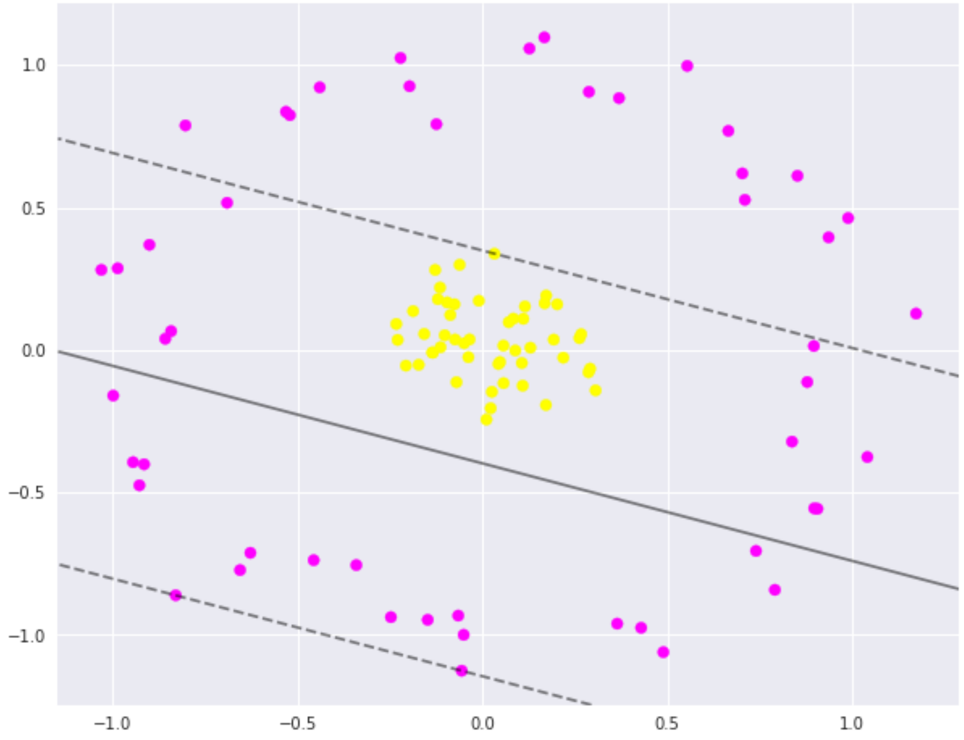
```
plt.figure(figsize=(10,8))
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='spring')
plot_svc_decision_function(clf);
```



Going further: Kernel Methods

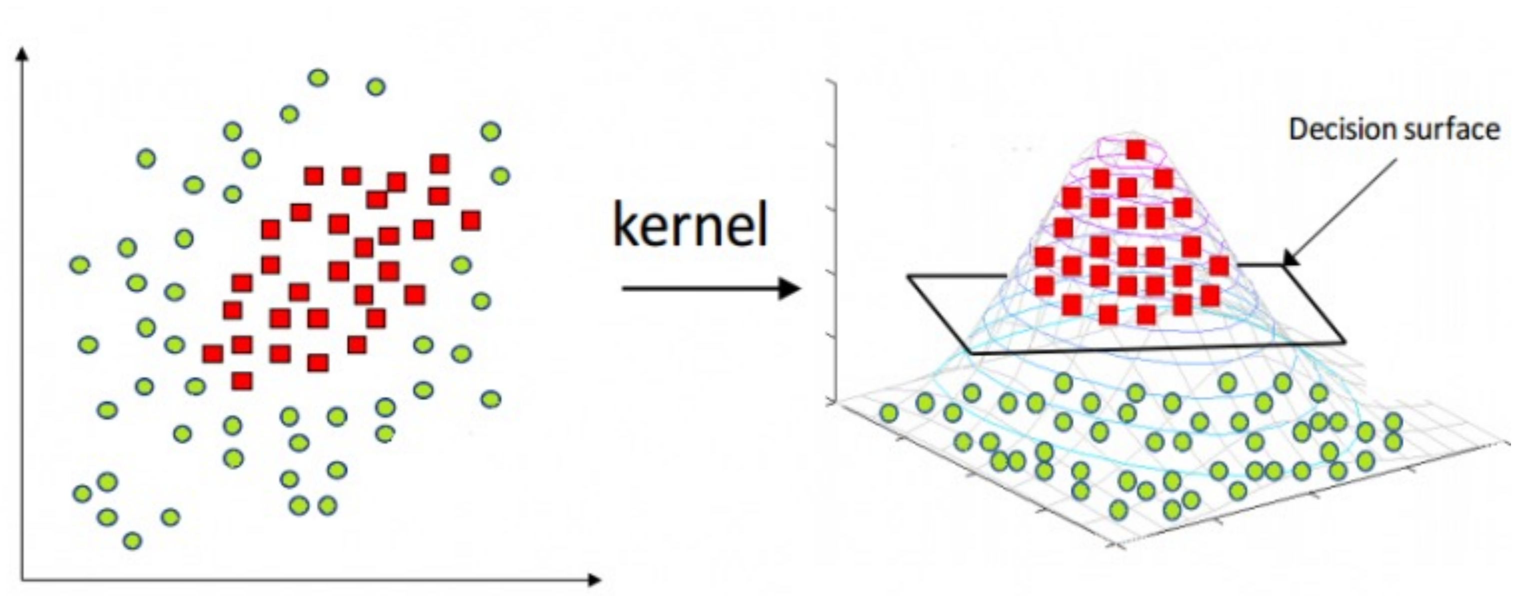
- Where SVM gets incredibly exciting is when it is used in conjunction with **kernels**.

To motivate the need for kernels, let's look at some data which is not linearly separable:



- The ***kernel trick*** in SVM
  - transforms 2-d data into a 3-d space using a ***kernel***
  - in such a 3-d space a linear hyperplane can be used to separate classes
  - Radial basis function (rbf)

$$K(X_i, X_j) = exp(-(X_i^2 + X_j^2))$$

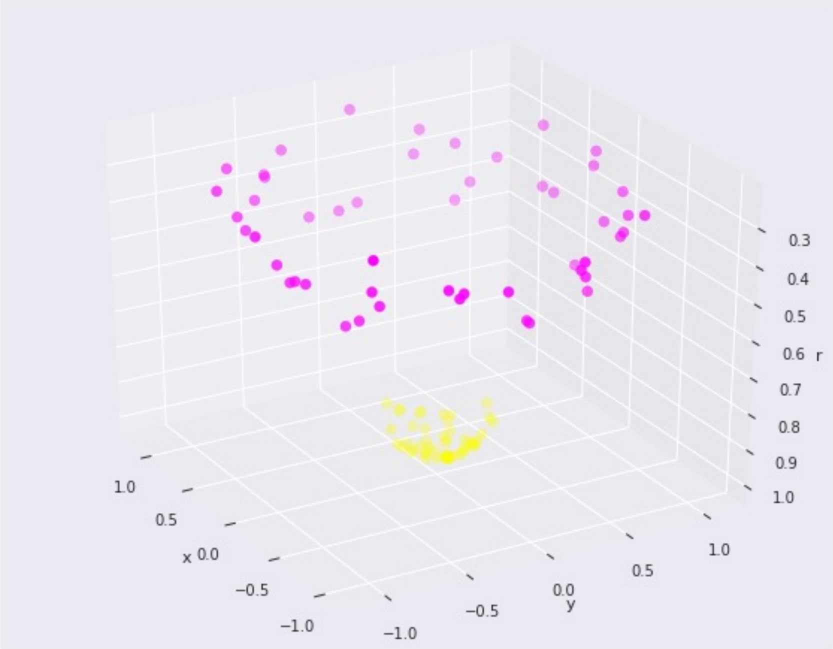


- Use rbf:

```
r = np.exp(-(X[:, 0] ** 2 + X[:, 1] ** 2))
```

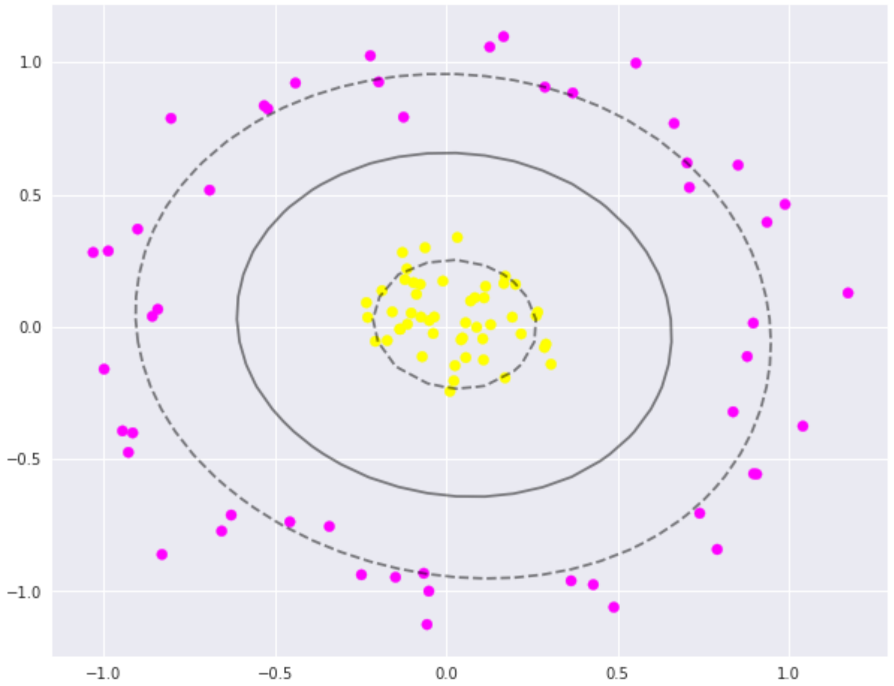
- If we plot this along with our data,

```
from mpl_toolkits import mplot3d

def plot_3D(elev=30, azim=30):
    plt.figure(figsize=(10, 8))
    ax = plt.subplot(projection='3d')
    ax.scatter3D(X[:, 0], X[:, 1], r, c=y, s=50, cmap='spring')
    ax.view_init(elev=elev, azim=azim)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('r')

interact(plot_3D, elev=[-150, 150], azip=(-180, 180));
```



- We can see that with this additional dimension, the data becomes trivially linearly separable! This is a relatively simple kernel; SVM has a more sophisticated version of this kernel built-in to the process. This is accomplished by using kernel='rbf', short for radial basis function:

```
clf = SVC(kernel='rbf')
clf.fit(X, y)

plt.figure(figsize=(10, 8))
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='spring')
plot_svc_decision_function(clf)
plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1],
            s=200, facecolors='none');
```



In-Class Exercise:
SVM Implementation with the Iris Dataset

Objective: Implement an SVM classifier using the Iris dataset and visualize the decision boundaries.

Steps:

- Import the necessary libraries.
- Load the Iris dataset.
- Split the dataset into training and testing subsets.
- Implement an SVM classifier and train it on the training subset.
- Evaluate the classifier on the testing subset and print the accuracy.
- Visualize the decision boundaries using appropriate plotting tools.

Example code::

```
# Step 1: Import the necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
# Step 2: Load the Iris dataset
iris = datasets.load_iris()
X = iris.data[:, :2]  # taking only the first two features for easy visualization
y = iris.target
```

```
# Step 3: Split the dataset into training and testing subsets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

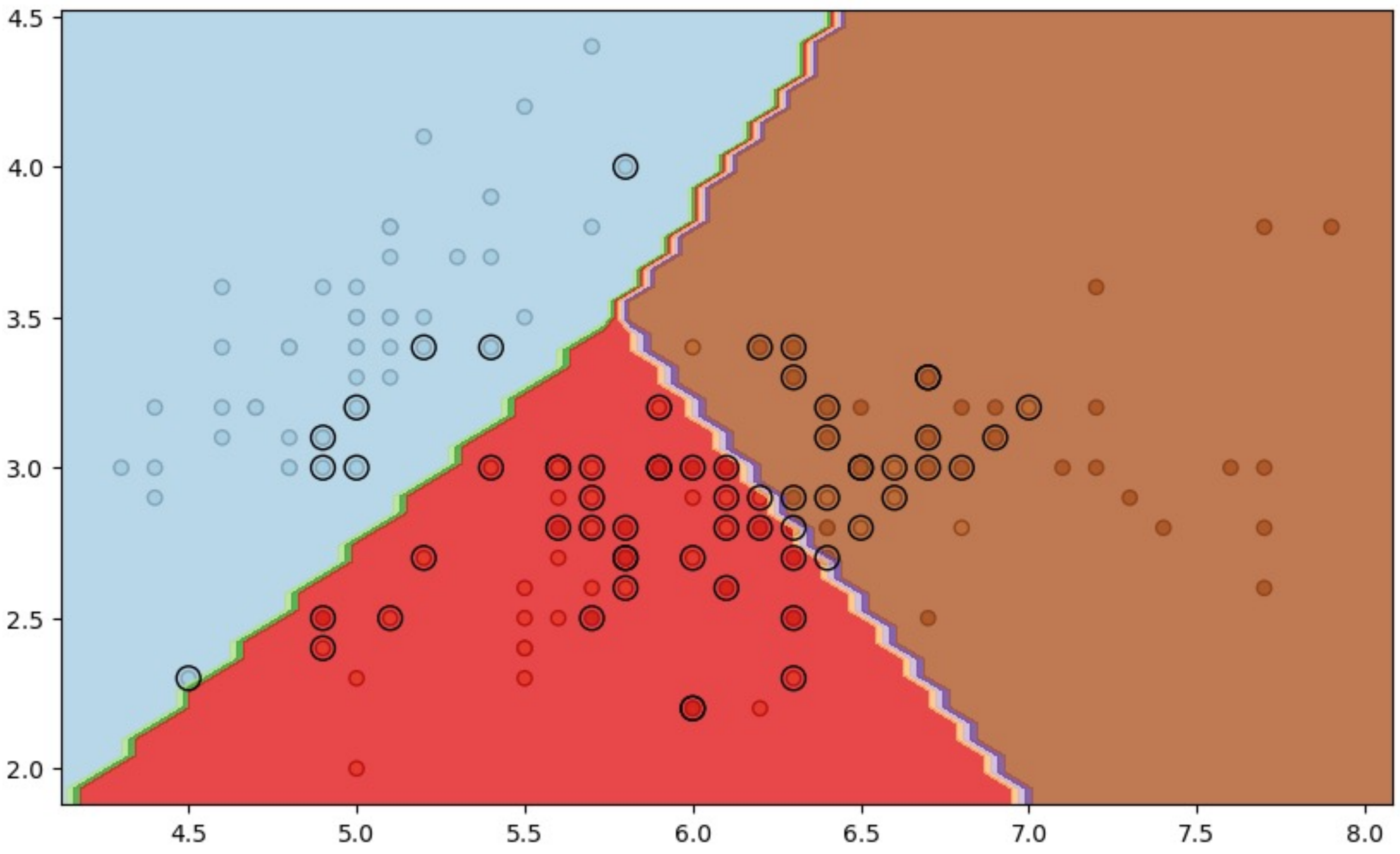
```
# Step 4: Implement an SVM classifier and train it
clf = SVC(kernel='linear')
clf.fit(X_train, y_train)
```

```
SVC
SVC(kernel='linear')
```

```
# Step 5: Evaluate the classifier
y_pred = clf.predict(X_test)
print(f"Accuracy: {accuracy_score(y_test, y_pred) * 100:.2f}%")
```

```
Accuracy: 80.00%
```

```
# Step 6: Visualize the decision boundaries
plt.figure(figsize=(10, 6))
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', cmap=plt.cm.Paired)
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

# create a grid to evaluate model
xx = np.linspace(xlim[0], xlim[1], 50)
yy = np.linspace(ylim[0], ylim[1], 50)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = clf.decision_function(xy)

# For multiclass SVM, get the class with the maximum decision function value for each point
Z = np.argmax(Z, axis=1).reshape(XX.shape)

# plot decision boundary
contour = ax.contourf(XX, YY, Z, alpha=0.8, cmap=plt.cm.Paired)

# plot support vectors
ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=100, facecolors='none', edgecolors='k')
plt.show()
```


4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0