M4-L2 Problem 2

Here we will revisit the phase diagram problem from the logistic regression module. Your task will be to code a one-vs-rest support vector classifier.

Work through this notebook, filling in code as requested, to implement the OvR classifier.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.svm import SVC

x1 = np.array([7.4881350392732475,16.351893663724194,22.427633760716436,29.04883182
x2 = np.array([0.11120957227224215,0.1116933996874757,0.14437480785146242,0.1181820
X = np.vstack([x1,x2]).T
y = np.array([0,2,2,2,2,2,0,2,2,2,2,2,0,0,2,0,1,2,0,0,1,1,1,2,0,1,0,1,1,1,0,0,1,1,1


def plot_data(X, y, title="Phase of simulated material", newfig=True):
    xlim = [0,52.5]
    ylim = [0,1.05]
    markers = [dict(marker="o", color="royalblue"), dict(marker="s", color="crimson
    labels = ["Solid", "Liquid", "Vapor"]

    if newfig:
        plt.figure(dpi=150)

    for i in range(1+max(y)):
        plt.scatter(X[y==i,0], X[y==i,1], s=60, **(markers[i]), edgecolor="black",

    plt.title(title)
    plt.legend(loc="upper right")
    plt.xlim(xlim)
    plt.ylim(ylim)
    plt.xlabel("Temperature, K")
    plt.ylabel("Pressure, atm")
    plt.box(True)

def plot_ovr_colors(classifiers, res=40):
    xlim = [0,52.5]
    ylim = [0,1.05]
    xvals = np.linspace(*xlim,res)
    yvals = np.linspace(*ylim,res)
    x,y = np.meshgrid(xvals,yvals)
    XY = np.concatenate((x.reshape(-1,1),y.reshape(-1,1)),axis=1)
    if type(classifiers) == list:
        color = classify_ovr(classifiers,XY).reshape(res,res)
    else:
        color = classifiers(XY).reshape(res,res)
    cmap = ListedColormap(["lightblue","lightcoral","palegreen"])
    plt.pcolor(x, y, color, shading="nearest", zorder=-1, cmap=cmap,vmin=0,vmax=2)
    return
```

Binomial classification function

You are given a function that performs binomial classification by using sklearn's `SVC` tool: `prob = get_ovr_decision_function(X, y, A, kernel, C)`

To use it, input:

- `X`, an array in which each row contains (x,y) coordinates of data points
- `y`, an array that specifies the class each point in `X` belongs to
- `A`, the class of the group (0, 1, or 2 in this problem) -- classifies into A or "rest"
- `kernel`, the kernel to use for the SVM
- `C`, the inverse regularization strength to use for the SVM

The function outputs a decision function (`decision()` in this case), which can be used to evaluate each `X`, giving positive values for class A, and negative values for [not A].

In [2]:

```
def get_ovr_decision_function(X, y, A, kernel="linear",C=1000):
    y_new = -1 + 2*(y == A).astype(int)

    model = SVC(kernel=kernel,C=C)
    model.fit(X, y_new)

    def decision(X):
        pred = model.decision_function(X)
        return pred.flatten()

    return decision
```

Coding an OvR classifier

Now you will create a one-vs-rest classifier to do multinomial classification. This will generate a binomial classifier for each class in the dataset, when compared against the rest of the classes. Then to predict the class of a new point, classify it using each of the binomial classifiers, and select the class whose binomial classifier decision function returns the highest value.

Complete the two functions we have started:

- `generate_ovr_decision_functions(X, y)` which returns a list of binary classifier probability functions for all possible classes (0, 1, and 2 in this problem)
- `classify_ovr(decisions, X)` which loops through a list of ovr classifiers and gets the decision function evaluation for each point in `X`. Then taking the

highest decision function value for each, return the overall class predictions for each point.

In [3]:

```
def generate_ovr_decision_functions(X, y, kernel="linear", C=1000):
    # YOUR CODE GOES HERE
    print(kernel,C)
    decision0 = get_ovr_decision_function(X, y, 0,kernel=kernel,C=C)
    decision1 = get_ovr_decision_function(X, y, 1,kernel=kernel,C=C)
    decision2 = get_ovr_decision_function(X, y, 2,kernel=kernel,C=C)
    decisions = [decision0,decision1,decision2]
    return decisions


def classify_ovr(decisions, X):
    # YOUR CODE GOES HERE
    n = X.shape[0]
    # for i in probs:
    #     preds = np.append(preds,np.array([i(xy)]))
    # preds = preds.reshape(3,n)

    final_preds = np.zeros(n)

    # for i in range(preds.shape[1]):
    #     max_index = np.where(preds[:,i] == np.max(preds[:,i]))
    #     if max_index[0] == 0:
    #             final_preds[i] = 0
    #     elif max_index[0] ==  1:
    #             final_preds[i] = 1
    #     else:
    #         final_preds[i] = 2
    decision0 = decisions[0](X)
    decision1 = decisions[1](X)
    decision2 = decisions[2](X)
    # print(decision0)
    # print(decision1)
    # print(decision2)
    for i in range(n):
        if decision0[i] > decision1[i] and decision0[i] > decision2[i]:
            final_preds[i] = 0
        elif decision1[i] > decision2[i] and decision1[i] > decision0[i]:
            final_preds[i] = 1
        else:
            final_preds[i] = 2
    print(final_preds)
    return final_preds
```

Testing the classifier

In [4]:

```
kernel = "linear"
C = 1000

decisions = generate_ovr_decision_functions(X, y, kernel, C)
preds = classify_ovr(decisions, X)
```

```
accuracy = np.sum(preds == y) / len(y) * 100
print("True Classes:", y)
print(" Predictions:", preds)
print("    Accuracy:", accuracy, r"%")
```

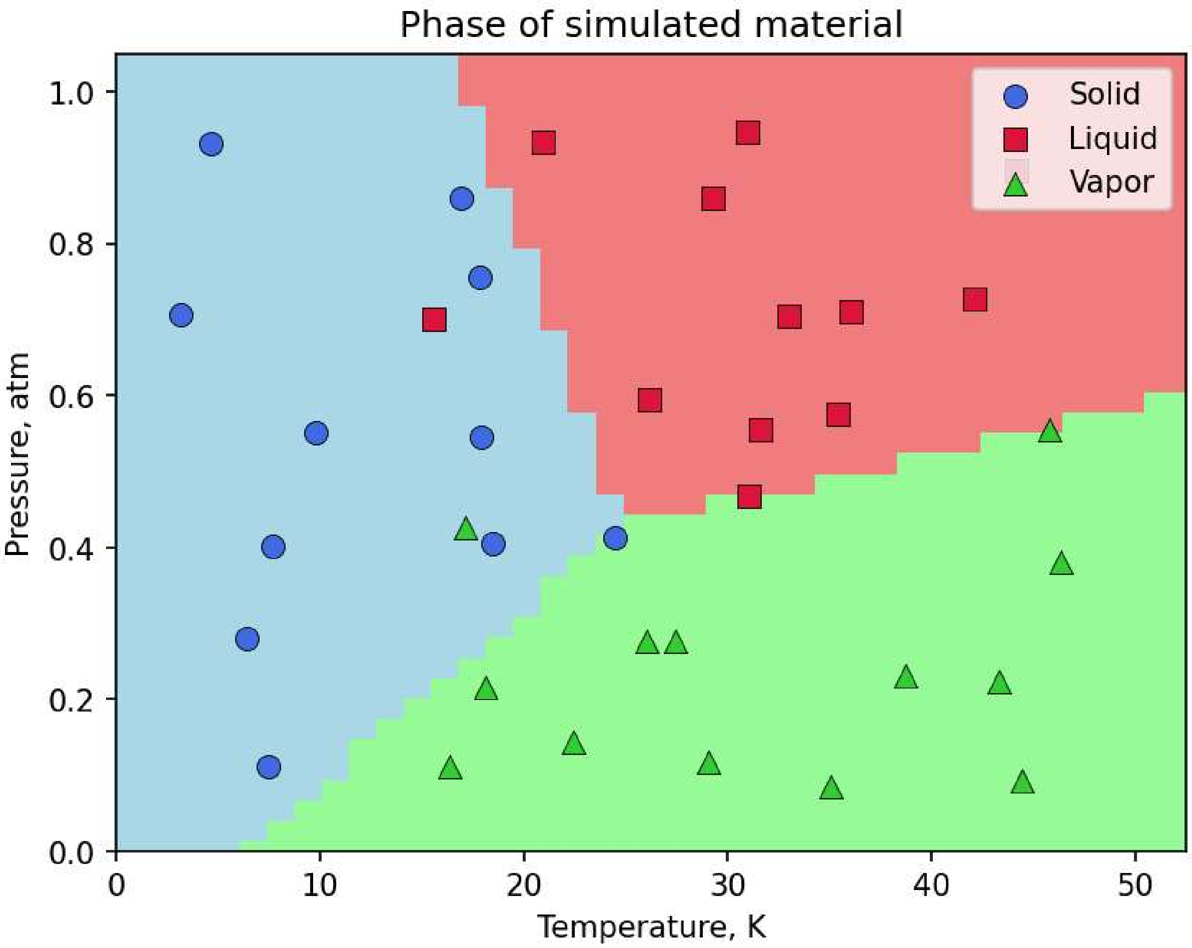
```
linear 1000
[0. 2. 2. 2. 2. 2. 0. 2. 2. 2. 2. 2. 0. 0. 0. 2. 1. 2. 0. 0. 1. 1. 1. 2.
 0. 0. 0. 1. 1. 1. 0. 0. 1. 1. 1. 1.]
True Classes: [0 2 2 2 2 2 0 2 2 2 2 2 0 0 2 0 1 2 0 0 1 1 1 2 0 1 0 1 1 1 0 0 1 1 1
1]
 Predictions: [0. 2. 2. 2. 2. 2. 0. 2. 2. 2. 2. 2. 0. 0. 0. 2. 1. 2. 0. 0. 1. 1. 1.
2.
 0. 0. 0. 1. 1. 1. 0. 0. 1. 1. 1. 1.]
    Accuracy: 91.66666666666666 %
```

Plotting results

In [5]:

```
plot_data(X,y)
plot_ovr_colors(decisions)
plt.show()
```

```
[0. 0. 0. ... 1. 1. 1.]
```



Modifying the SVC

Now go back and change the kernel and C value; observe how the results change.

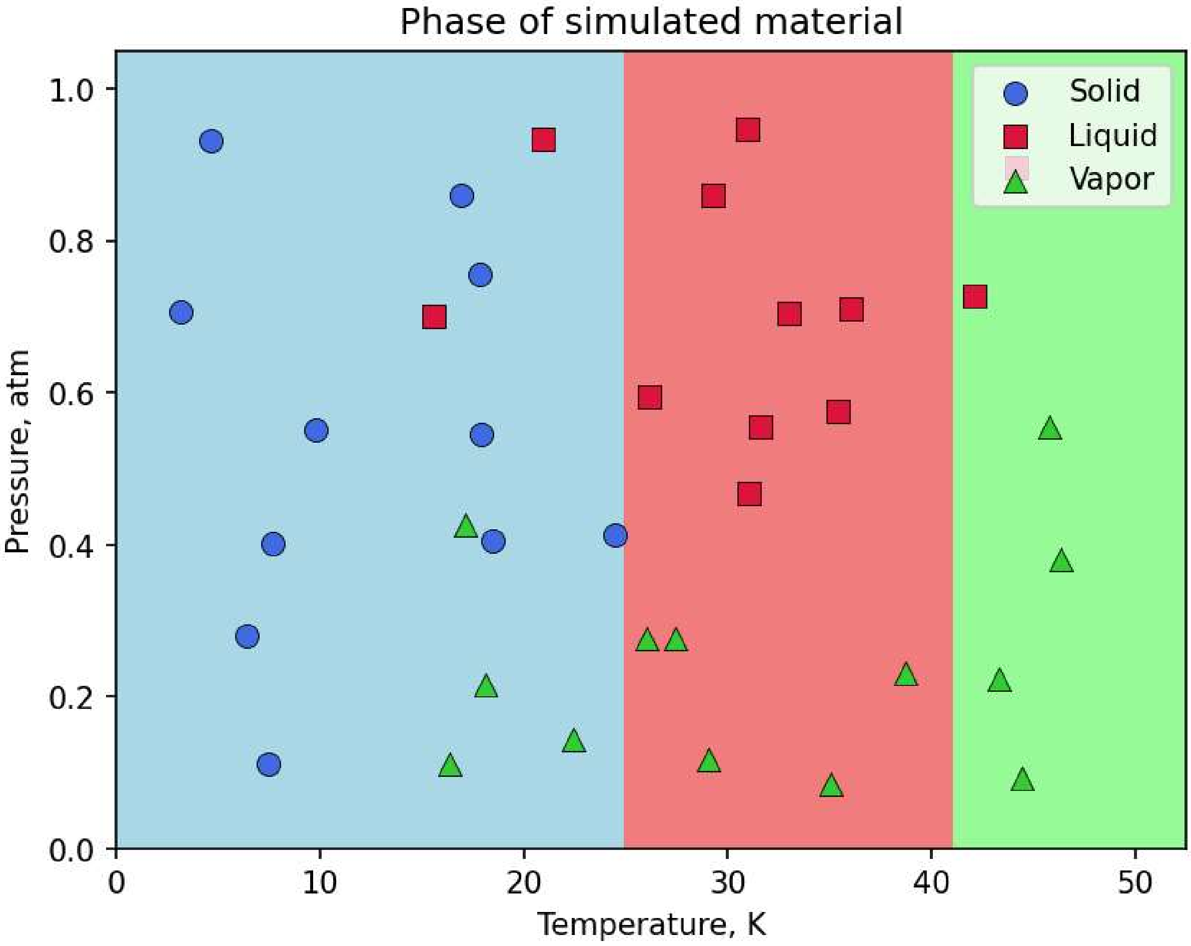
In [6]:

```
kernel = "rbf"  # CHANGE THIS
C = 0.00001             # CHANGE THIS

decisions = generate_ovr_decision_functions(X, y, kernel, C)
preds = classify_ovr(decisions, X)
accuracy = np.sum(preds == y) / len(y) * 100
print("True Classes:", y)
print(" Predictions:", preds)
print("    Accuracy:", accuracy, r"%")

plot_data(X,y)
plot_ovr_colors(decisions)
plt.show()
```

```
rbf 1e-05
[0. 0. 0. 1. 1. 2. 0. 0. 1. 1. 1. 2. 0. 0. 0. 0. 1. 2. 0. 0. 1. 1. 1. 2.
 0. 0. 0. 1. 1. 2. 0. 0. 0. 1. 1. 2.]
True Classes: [0 2 2 2 2 2 0 2 2 2 2 2 0 0 2 0 1 2 0 0 1 1 1 2 0 1 0 1 1 1 0 0 1 1 1
1]
 Predictions: [0. 0. 0. 1. 1. 2. 0. 0. 1. 1. 1. 2. 0. 0. 0. 0. 1. 2. 0. 0. 1. 1. 1.
2.
 0. 0. 0. 1. 1. 2. 0. 0. 0. 1. 1. 2.]
    Accuracy: 63.888888888888886 %
[0. 0. 0. ... 2. 2. 2.]
```



In [ ]: