Assignment 3

Support Vector Machines:

Negative ETA Case:

The following code was added:

```
        if eta > 0:
            a2 = alpha2 + y2 * (e1 - e2) / eta
            if a2 <= L:
                a2 = L
            elif a2 >= H:
                a2 = H
        # TODO: the negative case
        else:
            print(f"[DEBUG] smo_step: eta = {eta}")

            f1 = (y1*(e1 + self._b)) - (alpha1 * k11) - (s*alpha2*k12)
            f2 = (y2*(e2 + self._b)) - (s*alpha1*k12) - (alpha2*k22)
            L1 = alpha1 + s*(alpha2 - L)
            H1 = alpha1 + s*(alpha2 - H)

            Lobj = (L1*f1) + (L*f2) + (0.5*(L1**2)*k11) + (0.5*(L**2)*k22) + (s*L*L1*k12)
            Hobj = (H1*f1) + (H*f2) + (0.5*(H1**2)*k11) + (0.5*(H**2)*k22) + (s*H*H1*k12)

            if (Lobj < Hobj- self.eps):
                a2 = L
            elif (Lobj > (Hobj + self.eps)):
                  a2 = H
            else:
                  a2 = alpha2


        if np.abs(a2 - alpha2) < 1e-3 * (a2 + alpha2 + 1e-3):
            return 0

        a1 = alpha1 + s * (alpha2 - a2)
```

Non-Linear SVM:

The following code was added in poly kernel function:

Non-linear SVM

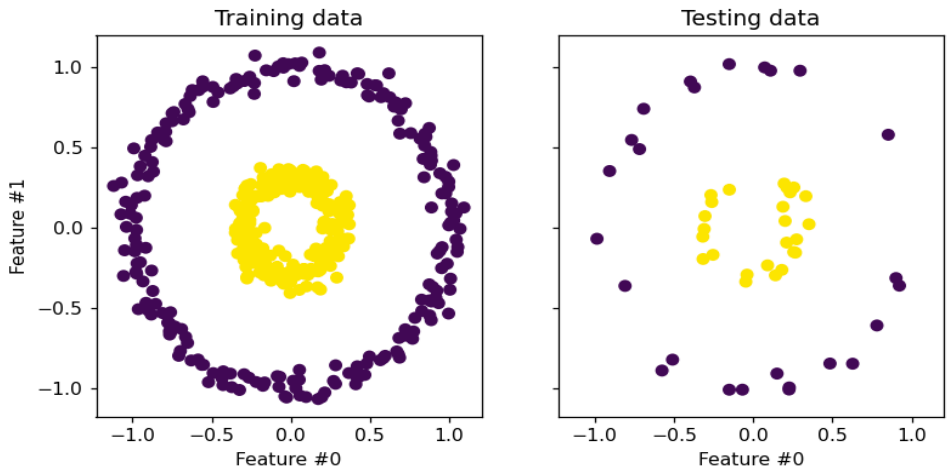
In [7]:

```
def linear_kernel(x1, x2):
    return x1.T @ x2

def poly_kernel(x1, x2, d = 2, c = 0):
    return (x1 @ x2.T + c)**d


class p_svm():
    def __init__(self, kernel='linear', c=1.0, tol=1e-3, maxiter=1000):
        self._kernel = kernel
        self._tol = tol
        self._maxiter = maxiter
        self.eps = 0.001

        if self._kernel == 'linear':
            self._k = linear_kernel
        elif self._kernel == 'poly':
            self._k = poly_kernel

        self._c = c
```

Make Circles Dataset plot:

Only 500 samples are considered from 1000 samples to avoid long training durations.

Figure 3

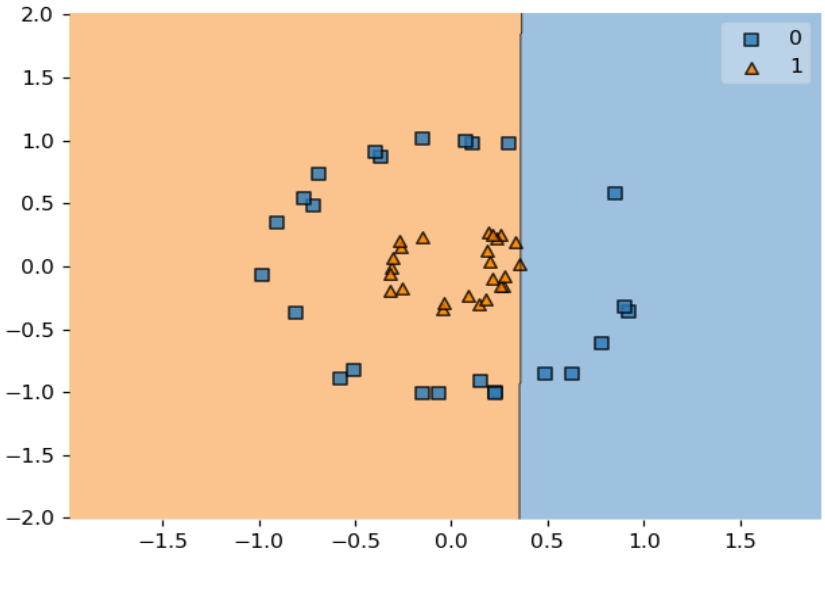


Classification Results for the dataset:

Using sklearn.SVC with Linear Kernel: (acc = 62 %)

```
coef_=[[-1.41066491  0.00310126]]
intercept=[0.50648098]
Accuracy of linear svc =  0.62
```

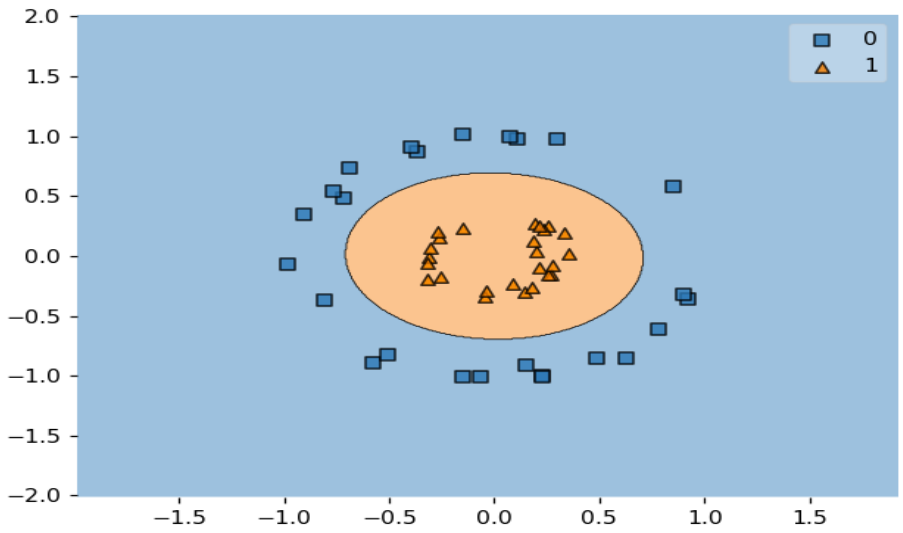
Figure 4



Using sklearn.SVC with Poly Kernel: (acc = 100%)

```
intercept=[1.56765424]
Accuracy of poly svc =  1.0
```

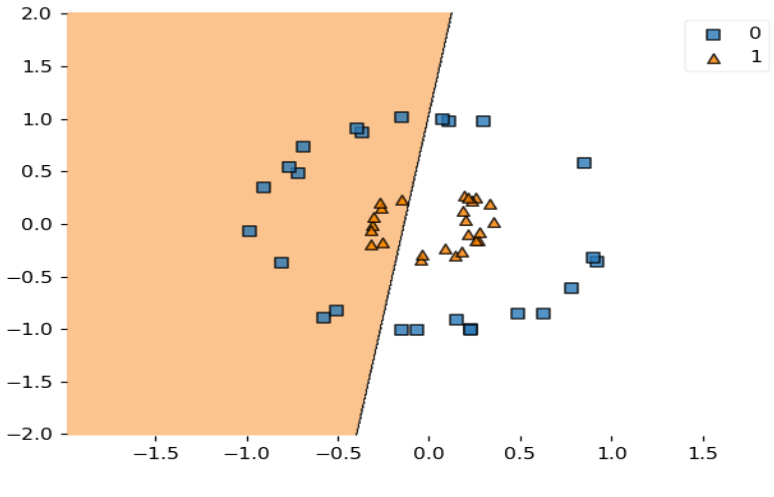
Figure 5



Using my defined SVM with Linear Kernel: (acc = 56%)

```
[-8.08094331  1.05091353]
1.1008373584163835
Accuracy of poly svc =  0.56
```

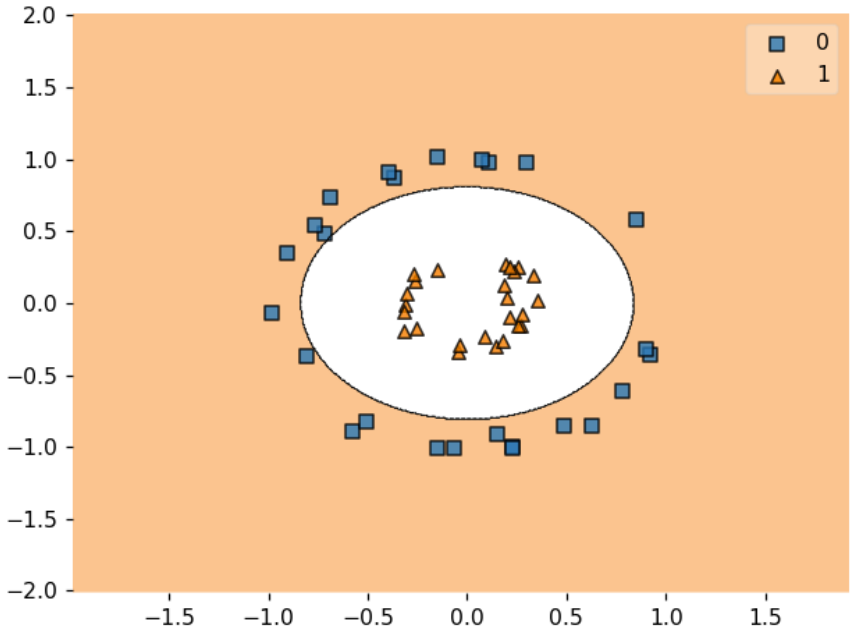
Figure 10



Using my defined SVM with Poly Kernel: (acc = 100%)

```
7.406700995421142
Accuracy of poly svc =  100.0 %
b=7.406700995421142
```

Figure 11



According to the testing results for Non-Linear SVM, for linear kernel, the accuracy of sklearn's SVC (62%) is better my defined model (56%). On the other hand for poly kernel, both sklearn's and my defined model perform good with 100% accuracy.

Multi – class SVM:

Model 1: Type 0 vs (Type 1 and Type 2)

Accuracy Summary

My defined SVM with Linear Kernel	sklearn SVC with Linear Kernel	My defined SVM with Poly Kernel	sklearn SVC with Poly Kernel
13.3334%	**100%**	**0.0%**	**100%**

Model 2: Type 1 vs (Type 0 and Type 2)

Accuracy Summary

My defined SVM with Linear Kernel	sklearn SVC with Linear Kernel	My defined SVM with Poly Kernel	sklearn SVC with Poly Kernel
46.66664%	**73.3333%**	**46.66664%**	**93.33333%**

Model 3: Type 2 vs (Type 0 and Type 1)

Accuracy Summary

My defined SVM with Linear Kernel	sklearn SVC with Linear Kernel	My defined SVM with Poly Kernel	sklearn SVC with Poly Kernel
86.66667%	**100%**	**86.666667%**	**100%**

According to the testing results for Multi-class SVM, the defined model classifies the Type 2 (Model 3) better than the other two types with accuracy of 86.6667%. The accuracy of sklearn classifiers is better in all the cases.