Rutuja Dukhande
1001730748
CSE 6363: Machine Learning

Assignment 3

Support Vector Machine

1. The Negative η Case

```
b = self._b
L1 = alpha1 + s*(alpha2 - L)
H1 = alpha1 + s*(alpha2 - H)

f1 = y1 * (e1 + b) - alpha1 * self._k(i1, i1) - s * alpha2 * self._k(i1, i2)
f2 = y2 * (e2 + b) - alpha2 * self._k(i2, i2) - s * alpha1 * self._k(i1, i2)
ol = (
    L1 * f1
    + L * f2
    + 1 / 2 * L1**2 * self._k(i1, i1)
    + 1 / 2 * L**2 * self._k(i2, i2)
    + s * L * L1 * self._k(i1, i2)
)
oh = (
    H1 * f1
    + H * f2
    + 1 / 2 * H1**2 * self._k(i1, i1)
    + 1 / 2 * H**2 * self._k(i2, i2)
    + s * H * H1 * self._k(i1, i2)
)
if ol < (oh - self._eps):
  a2 = L
elif ol > oh + self._eps:
  a2 = H
else:
  a2 = alpha2
```

vector x. In any event, SMO will work even when η is not positive, in which case the objective function Ψ should be evaluated at each end of the line segment:

$$
\begin{aligned}
f_1 &= y_1(E_1 + b) - \alpha_1 K(\vec{x}_1, \vec{x}_1) - s\alpha_2 K(\vec{x}_1, \vec{x}_2),\\
f_2 &= y_2(E_2 + b) - s\alpha_1 K(\vec{x}_1, \vec{x}_2) - \alpha_2 K(\vec{x}_2, \vec{x}_2),\\
L_1 &= \alpha_1 + s(\alpha_2 - L),\\
H_1 &= \alpha_1 + s(\alpha_2 - H),\\
\Psi_L &= L_1 f_1 + L f_2 + \tfrac{1}{2} L_1^2 K(\vec{x}_1, \vec{x}_1) + \tfrac{1}{2} L^2 K(\vec{x}_2, \vec{x}_2) + sLL_1 K(\vec{x}_1, \vec{x}_2),\\
\Psi_H &= H_1 f_1 + H f_2 + \tfrac{1}{2} H_1^2 K(\vec{x}_1, \vec{x}_1) + \tfrac{1}{2} H^2 K(\vec{x}_2, \vec{x}_2) + sHH_1 K(\vec{x}_1, \vec{x}_2).
\end{aligned}
\tag{19}
$$

SMO will move the Lagrange multipliers to the end point that has the lowest value of the objective function. If the objective function is the same at both ends (within a small ε for round-off error) and the kernel obeys Mercer's conditions, then the joint minimization cannot make progress. That scenario is described below.

***Reference**: Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines John C. Plat*

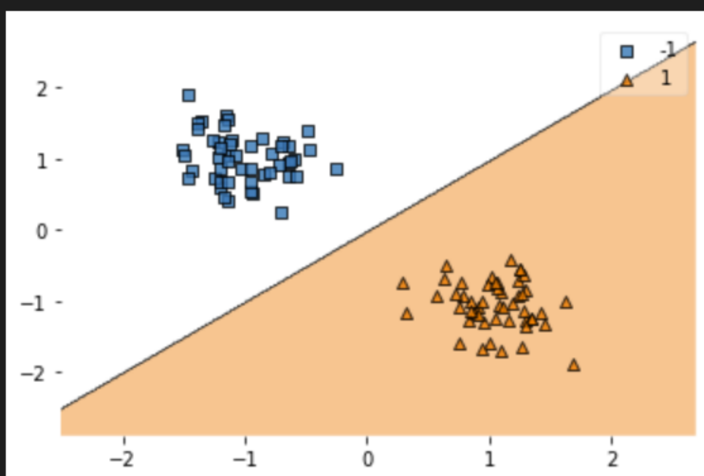
2. **Non-linear SVM**

1. **Linear kernel**

```
print(f"weights={model1._weights}")
print(f"b={model1._b}")

fig = plt.figure()
ax = plot_decision_regions(samples, targets.astype(np.int_), model1)
fig.add_subplot(ax)
plt.show()
```

```
weights=[ 0.99626423 -1.00255756]
b=0.03149400131475239
```

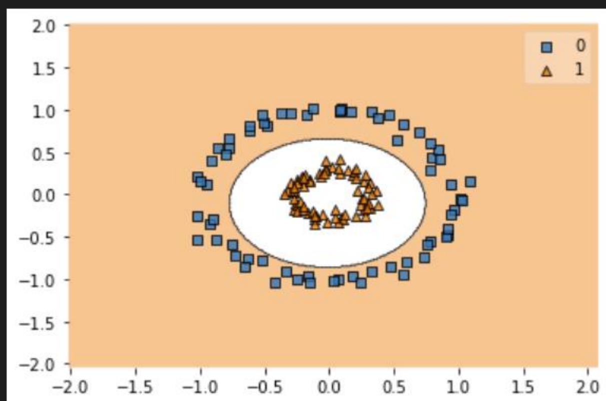


2. Poly Kernel

```
print(f"weights={model._weights}")
print(f"b={model._b}")

fig = plt.figure()
ax = plot_decision_regions(X_train, y_train, model)
fig.add_subplot(ax)
plt.show()
```

```
weights=[0.03647515 0.33212398]
b=69.76106057415655
```



3. Multi-class SVM

Using sklearn with Poly kernel

```python
# Fitting the model with training data

poly = OneVsRestClassifier(svm.SVC(kernel='poly', degree=3, C=1))

poly.fit(X_train, y_train)


# Making a prediction on the test set
prediction = poly.predict(X_test)

# Evaluating the model
print("Accuracy:",metrics.accuracy_score(y_test, prediction))
```

```
Accuracy: 0.7
```

Using sklearn with linear kernel

```python
# Fitting the model with training data

poly1 = OneVsRestClassifier(svm.SVC(kernel='linear', degree=3, C=1))

poly1.fit(X_train, y_train)


# Making a prediction on the test set
prediction1 = poly1.predict(X_test)

# Evaluating the model
print("Accuracy:",metrics.accuracy_score(y_test, prediction1))
```

```
Accuracy: 0.6
```