8E and 8F: Finding the Probability P(Y==1|X)

8E: Implementing Decision Function of SVM RBF Kernel

After we train a kernel SVM model, we will be getting support vectors and their corresponsing coefficients α_i

Check the documentation for better understanding of these attributes:

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Attributes:

support_ : *array-like, shape = [n_SV]*
Indices of support vectors.

support_vectors_ : *array-like, shape = [n_SV, n_features]*
Support vectors.

n_support_ : *array-like, dtype=int32, shape = [n_class]*
Number of support vectors for each class.

dual_coef_ : *array, shape = [n_class-1, n_SV]*
Coefficients of the support vector in the decision function. For multiclass, coefficient for all 1-vs-1 classifiers. The layout of the coefficients in the multiclass case is somewhat non-trivial. See the section about multi-class classification in the SVM section of the User Guide for details.

coef_ : *array, shape = [n_class * (n_class-1) / 2, n_features]*
Weights assigned to the features (coefficients in the primal problem). This is only available in the case of a linear kernel.

coef_ is a readonly property derived from `dual_coef_` and `support_vectors_`.

intercept_ : *array, shape = [n_class * (n_class-1) / 2]*
Constants in decision function.

fit_status_ : *int*
0 if correctly fitted, 1 otherwise (will raise warning)

probA_ : *array, shape = [n_class * (n_class-1) / 2]*

probB_ : *array, shape = [n_class * (n_class-1) / 2]*
If probability=True, the parameters learned in Platt scaling to produce probability estimates from decision values. If probability=False, an empty array. Platt scaling uses the logistic function `1 / (1 + exp(decision_value * probA_ + probB_))` where `probA_` and `probB_` are learned from the dataset [R20c70293ef72-2]. For more information on the multiclass case and training procedure see section 8 of [R20c70293ef72-1].

As a part of this assignment you will be implementing the `decision_function()` of kernel SVM, here decision_function() means based on the value return by `decision_function()` model will classify the data point either as positive or negative

Ex 1: In logistic regression After traning the models with the optimal weights w we get, we will find the value $\frac{1}{1+\exp(-(wx+b))}$, if this value comes out to be < 0.5 we will mark it as negative class, else its positive class

Ex 2: In Linear SVM After traning the models with the optimal weights w we get, we will find the value of $sign(wx + b)$, if this value comes out to be -ve we will mark it as negative class, else its positive class.

Similarly in Kernel SVM After traning the models with the coefficients α_i we get, we will find the value of $sign(\sum_{i=1}^{n}(y_i \alpha_i K(x_i, x_q)) + intercept)$, here $K(x_i, x_q)$ is the RBF kernel. If this value comes out to be -ve we will mark x_q as negative class, else its positive class.

RBF kernel is defined as: $K(x_i, x_q)$ = $exp(-\gamma ||x_i - x_q||^2)$

For better understanding check this link: https://scikit-learn.org/stable/modules/svm.html#svm-mathematical-formulation

Task E

> 1. Split the data into X_{train}(60), X_{cv}(20), X_{test}(20)
>
> 2. Train $SVC(gamma = 0.001, C = 100.)$ on the (X_{train}, y_{train})
>
> 3. Get the decision boundry values f_{cv} on the X_{cv} data i.e. f_{cv} `= decision_function(` X_{cv} `)` you need to implement this decision_function()

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
import numpy as np
```

```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
```

```
X, y  = make_classification(n_samples=5000, n_features=5, n_redundant=2,
                           n_classes=2, weights=[0.7], class_sep=0.7, random_state=15)
```

Pseudo code

clf = SVC(gamma=0.001, C=100.)

clf.fit(Xtrain, ytrain)

def decision_function(Xcv, ...): #use appropriate parameters

for a data point x_q in Xcv:

#write code to implement $(\sum_{i=1}^{\text{all the support vectors}}(y_i \alpha_i K(x_i, x_q)) + intercept)$, here the values y_i, α_i, and $intercept$ can be obtained from the trained model

return *# the decision_function output for all the data points in the Xcv*

fcv = decision_function(Xcv, ...) *# based on your requirement you can pass any other parameters*

Note: Make sure the values you get as fcv, should be equal to outputs of clf.decision_function(Xcv)

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.20,random_state=15)
```

```
x_train, x_cv, y_train, y_cv = train_test_split(x_train,y_train, test_size=0.25,random_state=15)
```

```
# svc classifier and we are using rfb kernal in that

rbf=SVC(kernel="rbf",C=100,gamma=0.001)
```

```
#fitting the data

rbf.fit(x_train,y_train)
```

```
SVC(C=100, gamma=0.001)
```

```
sup_vecs=rbf.support_vectors_
dual_coefs=rbf.dual_coef_
intercept=rbf.intercept_
```

```
# refer : ##https://stackoverflow.com/questions/28503932/calculating-decision-function-of-svm-manually
# refer : https://stackoverflow.com/questions/28503932/calculating-decision-function-of-svm-manually

def decision_function(x_cv,gamma):

    predict=[]
    decision=[]

    # for each datapoint in x_cv dataset
    for x_query_point in x_cv:
      # intiating with 0
        decision__function = 0

        # for each point j in supp_vecs
        for K in range(len(sup_vecs)):
             norm2 = np.linalg.norm(sup_vecs[K, :] -x_query_point)**2                   # calculating the kernel(K(xi,xq)
             decision__function = decision__function + dual_coefs[0, K] * np.exp(-gamma*norm2)

              # MOST IMPORTANTLY HERE WE ARE INCREMENTING THE INTERCEPT TERM                                 # calculating the sign
        decision__function += intercept

        # defining the decison function limit if its less then 0 else 1
        decision.append(decision__function)
        if (decision__function)<0:
```

```
            predict.append(0)
        else:
         # else 1
            predict.append(1)

            # predicting the decision using numpy array
    return np.array(predict),decision
```

```
gamma=0.001
f_cv,decision=decision_function(x_cv,gamma)
print(f_cv)

# IN ABOVE CODE SNIPPET IMPLEMENTING THE DECISION FUNCTION OF CV DATA WITH GAMMA VALUE 0.001
```

```
[1 0 0 0 0 0 0 0 1 0 1 0 1 1 0 1 1 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1
 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 0 0 1
 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 1 0
 0 0 0 0 0 0 1 0 0 1 0 1 1 0 0 0 1 1 0 1 0 0 0 1 0 0 0 0 1 1 0 0 1 1 0 0 0
 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0
 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1 0 0 1
 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1
 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0 1 0 0 0 0 1
 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 0 1 0 0 1 1 0 0 1 1 0 1 0 0 1 1 1 0 0 0 0 0
 1 1 0 0 0 0 0 0 0 1 0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0
 0 0 1 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 1 1 0
 1 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0
 1 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 1 1 1 0 1 0 1 0 1 1 1 1 0 1 1 1
 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 1 1 1 0 1 0 1 0 0 0 1 1 1 1 1 0 1 1 0 0 1 1 1 0 0 0 0 0 1 1 0 0 1 0
 0 0 1 0 1 1 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 0 0 0 1
 0 0 1 1 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 0 1 0 1 1 0 1
 0 1 0 0 0 1 0 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 0 0 0 0 1
 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 1 0 0 0 1
 0 0 1 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 1 1 0 1 0
 0 0 0 1 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0 1
 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 0 1 1 0 0 0 0
 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0
 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 0 0 0 0 0
```

```
 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 1 0 0 1 0
 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1 1 0 1 0 1 0 0 1 1 0 0 0 1 0 0 0 1 1
 0]
```

```
print(np.array(decision).T)
```

```
[[ 9.78553590e-01 -1.11760405e+00 -2.04720532e+00 -3.09565428e+00
  -2.79889732e+00 -3.19223178e+00 -3.06506013e+00 -2.65326082e+00
   3.33031294e+00 -1.27804286e+00  1.23713395e+00 -2.26620102e+00
   1.15545642e+00  1.55336237e+00 -3.18756150e-01  1.20293177e+00
   1.88290414e+00  1.19637781e+00 -2.78451489e-01 -2.12325762e+00
   1.96221487e+00 -4.75759780e-01  1.08589552e+00 -2.55752818e+00
  -2.89278575e+00 -2.43600679e+00 -3.82456176e+00 -2.45783486e+00
  -2.27106674e+00 -1.23683699e+00 -1.16338685e+00 -2.10126553e+00
  -1.80723683e+00 -2.97522570e+00 -1.78272899e+00 -2.57605838e+00
   1.08577450e+00 -1.38072948e-01 -2.30545739e+00 -2.29909830e+00
   7.42068597e-01 -2.97019249e+00 -2.90071729e+00  1.41309686e+00
   2.78861618e+00 -1.89767766e+00 -1.98425643e+00 -1.45561992e+00
  -2.89596097e+00 -2.67027549e+00 -2.39828370e+00 -3.04643574e+00
  -1.56860504e+00  3.60540603e-01 -1.74975096e+00 -1.77366133e+00
  -2.78686913e+00 -2.13619233e+00 -1.90398564e+00  1.33960705e+00
  -3.24020418e+00  8.97308121e-01 -2.21557816e+00 -2.99426446e+00
  -4.94047716e-02  1.59483473e+00 -1.83496676e+00  7.95082106e-02
  -2.79323852e+00 -3.98279565e+00 -1.79837344e+00 -2.00613380e+00
  -1.97130396e+00  2.07687621e+00 -2.41070545e+00 -1.65161037e+00
  -3.07902438e+00 -3.17027645e-01 -2.85843081e+00 -4.65434071e+00
   5.51521648e-01 -4.49305200e+00 -3.76755323e+00 -8.63617712e-01
  -1.60777008e+00 -3.16398958e+00 -3.78415750e+00 -1.08713016e+00
  -2.21569698e+00 -2.69281638e+00 -4.39129734e+00 -2.58718677e+00
  -1.94996291e+00  9.55121457e-01 -2.25689728e+00 -1.79146566e+00
   1.46475046e+00 -1.61124209e+00  1.09078368e-01 -2.00541103e+00
  -9.96893289e-01 -2.70779082e+00 -1.76887567e+00 -3.78126963e+00
  -2.82995150e+00 -1.38797363e+00 -1.85961356e+00 -3.64882982e+00
   2.54385614e+00  1.68983589e+00 -2.43212936e+00 -1.40083552e+00
  -7.43796022e-01 -4.27483365e+00 -2.51198025e+00 -3.07676558e-01
  -2.55917854e+00  1.74299350e+00 -2.88922226e+00 -3.41397052e+00
   5.16065245e-01 -1.00630919e+00  1.54105178e+00  1.93021728e+00
  -2.75067087e+00 -1.71170224e+00 -3.86017913e-01  3.28147869e-01
   1.55476940e+00 -2.41901755e+00  2.43495429e+00 -6.92820420e-01
  -1.86825933e+00 -1.29327697e+00  1.58473334e+00 -3.64073602e+00
```

```
 -3.83519908e-01 -1.17292273e+00 -2.58480286e+00  1.72842330e+00
  1.25061440e+00 -2.85785061e+00 -2.38553121e+00  2.32235330e+00
  2.10194986e+00 -3.44510618e+00 -2.92868489e-01 -3.23908167e+00
  1.55751513e+00 -2.47843205e+00  2.18001557e+00 -1.20165501e+00
 -4.01948985e-01  1.78028779e+00 -3.27550034e+00 -3.06912256e+00
 -3.52004623e+00 -2.48539060e+00 -2.93959754e+00 -4.77021920e-01
 -2.59316966e+00 -2.90351656e+00 -1.46597057e+00  8.56265650e-01
 -2.36234254e+00 -3.73547703e+00  4.95282596e-02 -2.10923052e-01
 -2.90912800e+00 -2.33290215e+00  1.52248752e+00  1.48791501e+00
 -4.13498090e+00 -2.27268616e+00 -2.97313927e+00  3.47689519e-01
 -4.08266391e+00 -1.93805452e+00 -2.47266391e+00 -1.79711210e+00
 -2.84698945e+00 -2.24385224e+00 -2.93091108e+00 -2.01071939e+00
 -3.19778593e+00 -1.59651801e+00 -1.67436943e+00 -2.84951635e+00
 -1.66361709e+00 -2.39813527e+00 -4.34716445e-01  9.25867786e-01
 -1.54384069e-01  8.76104455e-01 -1.86226192e+00 -3.24401380e+00
 -4.76941729e-01 -2.11483641e+00 -2.43535141e+00 -2.44623939e+00
 -2.78785281e+00  1.49657837e+00 -3.10132048e+00 -1.58929029e+00
 -3.61986801e+00 -1.49480139e+00 -2.79801098e+00 -2.29219451e+00
 -3.52218636e+00 -2.81049414e+00 -2.29185795e+00  1.06526260e+00
 -3.33931008e+00 -2.78758109e+00  1.84947921e+00 -2.88574205e+00
 -2.43024877e+00 -2.41347029e+00 -2.68611335e+00 -2.98952530e+00
  2.24683602e+00 -2.95715269e+00 -3.04103655e-02 -2.49347304e+00
 -2.00957021e+00 -3.08899226e+00  2.03283356e+00 -2.04916266e+00
  1.65369691e+00 -4.00913706e+00 -2.05403809e+00 -9.46102721e-01
```

```
rbf.decision_function(x_cv)
```

```
array([ 9.78553590e-01, -1.11760405e+00, -2.04720532e+00, -3.09565428e+00,
       -2.79889732e+00, -3.19223178e+00, -3.06506013e+00, -2.65326082e+00,
        3.33031294e+00, -1.27804286e+00,  1.23713395e+00, -2.26620102e+00,
        1.15545642e+00,  1.55336237e+00, -3.18756150e-01,  1.20293177e+00,
        1.88290414e+00,  1.19637781e+00, -2.78451489e-01, -2.12325762e+00,
        1.96221487e+00, -4.75759780e-01,  1.08589552e+00, -2.55752818e+00,
       -2.89278575e+00, -2.43600679e+00, -3.82456176e+00, -2.45783486e+00,
       -2.27106674e+00, -1.23683699e+00, -1.16338685e+00, -2.10126553e+00,
       -1.80723683e+00, -2.97522570e+00, -1.78272899e+00, -2.57605838e+00,
        1.08577450e+00, -1.38072948e-01, -2.30545739e+00, -2.29909830e+00,
        7.42068597e-01, -2.97019249e+00, -2.90071729e+00,  1.41309686e+00,
        2.78861618e+00, -1.89767766e+00, -1.98425643e+00, -1.45561992e+00,
       -2.89596097e+00, -2.67027549e+00, -2.39828370e+00, -3.04643574e+00,
```

```
-1.56860504e+00,  3.60540603e-01, -1.74975096e+00, -1.77366133e+00,
-2.78686913e+00, -2.13619233e+00, -1.90398564e+00,  1.33960705e+00,
-3.24020418e+00,  8.97308121e-01, -2.21557816e+00, -2.99426446e+00,
-4.94047716e-02,  1.59483473e+00, -1.83496676e+00,  7.95082106e-02,
-2.79323852e+00, -3.98279565e+00, -1.79837344e+00, -2.00613380e+00,
-1.97130396e+00,  2.07687621e+00, -2.41070545e+00, -1.65161037e+00,
-3.07902438e+00, -3.17027645e-01, -2.85843081e+00, -4.65434071e+00,
 5.51521648e-01, -4.49305200e+00, -3.76755323e+00, -8.63617712e-01,
-1.60777008e+00, -3.16398958e+00, -3.78415750e+00, -1.08713016e+00,
-2.21569698e+00, -2.69281638e+00, -4.39129734e+00, -2.58718677e+00,
-1.94996291e+00,  9.55121457e-01, -2.25689728e+00, -1.79146566e+00,
 1.46475046e+00, -1.61124209e+00,  1.09078368e-01, -2.00541103e+00,
-9.96893289e-01, -2.70779082e+00, -1.76887567e+00, -3.78126963e+00,
-2.82995150e+00, -1.38797363e+00, -1.85961356e+00, -3.64882982e+00,
 2.54385614e+00,  1.68983589e+00, -2.43212936e+00, -1.40083552e+00,
-7.43796022e-01, -4.27483365e+00, -2.51198025e+00, -3.07676558e-01,
-2.55917854e+00,  1.74299350e+00, -2.88922226e+00, -3.41397052e+00,
 5.16065245e-01, -1.00630919e+00,  1.54105178e+00,  1.93021728e+00,
-2.75067087e+00, -1.71170224e+00, -3.86017913e-01,  3.28147869e-01,
 1.55476940e+00, -2.41901755e+00,  2.43495429e+00, -6.92820420e-01,
-1.86825933e+00, -1.29327697e+00,  1.58473334e+00, -3.64073602e+00,
-3.83519908e-01, -1.17292273e+00, -2.58480286e+00,  1.72842330e+00,
 1.25061440e+00, -2.85785061e+00, -2.38553121e+00,  2.32235330e+00,
 2.10194986e+00, -3.44510618e+00, -2.92868489e-01, -3.23908167e+00,
 1.55751513e+00, -2.47843205e+00,  2.18001557e+00, -1.20165501e+00,
-4.01948985e-01,  1.78028779e+00, -3.27550034e+00, -3.06912256e+00,
-3.52004623e+00, -2.48539060e+00, -2.93959754e+00, -4.77021920e-01,
-2.59316966e+00, -2.90351656e+00, -1.46597057e+00,  8.56265650e-01,
-2.36234254e+00, -3.73547703e+00,  4.95282596e-02, -2.10923052e-01,
-2.90912800e+00, -2.33290215e+00,  1.52248752e+00,  1.48791501e+00,
-4.13498090e+00, -2.27268616e+00, -2.97313927e+00,  3.47689519e-01,
-4.08266391e+00, -1.93805452e+00, -2.47266391e+00, -1.79711210e+00,
-2.84698945e+00, -2.24385224e+00, -2.93091108e+00, -2.01071939e+00,
-3.19778593e+00, -1.59651801e+00, -1.67436943e+00, -2.84951635e+00,
-1.66361709e+00, -2.39813527e+00, -4.34716445e-01,  9.25867786e-01,
-1.54384069e-01,  8.76104455e-01, -1.86226192e+00, -3.24401380e+00,
-4.76941729e-01, -2.11483641e+00, -2.43535141e+00, -2.44623939e+00,
-2.78785281e+00,  1.49657837e+00, -3.10132048e+00, -1.58929029e+00,
-3.61986801e+00, -1.49480139e+00, -2.79801098e+00, -2.29219451e+00,
-3.52218636e+00, -2.81049414e+00, -2.29185795e+00,  1.06526260e+00,
-3.33931008e+00, -2.78758109e+00,  1.84947921e+00, -2.88574205e+00,
```

```
-2.43024877e+00, -2.41347029e+00, -2.68611335e+00, -2.98952530e+00,
 2.24683602e+00, -2.95715269e+00, -3.04103655e-02, -2.49347304e+00,
-2.00957021e+00, -3.08899226e+00,  2.03283356e+00, -2.04916266e+00,
```

both the output are same

8F: Implementing Platt Scaling to find P(Y==1|X)

Let the output of a learning method be $f(x)$. To get calibrated probabilities, pass the output through a sigmoid:

$$P(y = 1|f) = \frac{1}{1 + exp(Af + B)} \qquad (1)$$

where the parameters A and B are fitted using maximum likelihood estimation from a fitting training set (f_i, y_i). Gradient descent is used to find A and B such that they are the solution to:

$$\underset{A,B}{argmin}\{-\sum_i y_i log(p_i) + (1 - y_i) log(1 - p_i)\}, \qquad (2)$$

where

$$p_i = \frac{1}{1 + exp(Af_i + B)} \qquad (3)$$

Two questions arise: where does the sigmoid train set come from? and how to avoid overfitting to this training set?

If we use the same data set that was used to train the model we want to calibrate, we introduce unwanted bias. For example, if the model learns to discriminate the train set perfectly and orders all the negative examples before the positive examples, then the sigmoid transformation will output just a 0,1 function. So we need to use an independent calibration set in order to get good posterior probabilities. This, however, is not a draw back, since the same set can be used for model and parameter selection.

To avoid overfitting to the sigmoid train set, an out-of-sample model is used. If there are N_+ positive examples and N_- negative examples in the train set, for each training example Platt Calibration uses target values y_+ and y_- (instead of 1 and 0, respectively), where

$$y_+ = \frac{N_+ + 1}{N_+ + 2};\ y_- = \frac{1}{N_- + 2} \qquad (4)$$

For a more detailed treatment, and a justification of these particular target values see (Platt, 1999).

Check this PDF

TASK F

> 4. Apply SGD algorithm with (f_{cv}, y_{cv}) and find the weight W intercept b `Note: here our data is of one dimensional so we will have a one dimensional weight vector i.e W.shape (1,)`

> Note1: Don't forget to change the values of y_{cv} as mentioned in the above image. you will calculate y+, y- based on data points in train data

> Note2: the Sklearn's SGD algorithm doesn't support the real valued outputs, you need to use the code that was done in the `'Logistic Regression with SGD and L2'` Assignment after modifying loss function, and use same parameters that used in that assignment.

```
def log_loss(w, b, X, Y):
    N = len(X)
    sum_log = 0
    for i in range(N):
        sum_log += Y[i]*np.log10(sig(w, X[i], b)) + (1-Y[i])*np.log10(1-sig(w, X[i], b))
    return -1*sum_log/N
```

if Y[i] is 1, it will be replaced with y+ value else it will replaced with y- value

5. For a given data point from X_{test}, $P(Y=1|X) = \frac{1}{1+exp(-(W*f_{test}+b))}$ where f_{test} `= decision_function(` X_{test} `)`, W and b will be learned as metioned in the above step

Note: in the above algorithm, the steps 2, 4 might need hyper parameter tuning, To reduce the complexity of the assignment we are excluding the hyerparameter tuning part, but intrested students can try that

If any one wants to try other calibration algorithm istonic regression also please check these tutorials

1. http://fa.bianp.net/blog/tag/scikit-learn.html#fn:1
2. https://drive.google.com/open?id=1MzmA7QaP58RDzocB0RBmRiWfl7Co_VJ7
3. https://drive.google.com/open?id=133odBinMOIVb_rh_GQxxsyMRyW-Zts7a
4. https://stat.fandom.com/wiki/Isotonic_regression#Pool_Adjacent_Violators_Algorithm

creating dataset

```
X, y = make_classification (n_samples=5000, n_features=5, n_redundant=2,
                           n_classes=2, weights=[0.7], class_sep=0.7, random_state=15)          # HERE CREATING THE RANDOM SAMPL
```

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.20,random_state=15)          # SPLITTING THE DATA
```

```
x_train, x_cv, y_train, y_cv = train_test_split(x_train,y_train, test_size=0.25,random_state=15)      # SPLITING THE DATA INTO 2 PART
```

```
rbf=SVC(kernel="rbf",C=100,gamma=0.001)                     # HERE WE ARE USING RBF KERNAL AND ITS ALSO A DEFAULT KERNAL IN SVM
```

```
rbf.fit(x_train,y_train)                               # FIT THE DATA
```

```
SVC(C=100, gamma=0.001)
```

```
gamma=0.001                         # STARTING THE GAMMA VALUE FROM 0.001
```

```
sup_vecs=rbf.support_vectors_
dual_coefs=rbf.dual_coef_                # refer :  https://stackoverflow.com/questions/22816646/the-dimension-of-dual-coef-in-sklear
intercept=rbf.intercept_
```

```
def decision_function(x_cv,gamma):

    predict=[]
    decision=[]

    # for each query point in x_cv dataset
    for x_query_point in x_cv:

      # initiating decision function with 0
        decision__function = 0
        for K in range(len(sup_vecs)):
```

```
            norm2 = np.linalg.norm(sup_vecs[K, :] -x_query_point)**2  # calculating the kernel(K(xi,xq)
            decision__function = decision__function + dual_coefs[0, K] * np.exp(-gamma*norm2) # calculating the sign

 # now will increment the intercept term
       decision__function += intercept
       # now will append the decision function
       decision.append(decision__function)
       # defining the limit for decison function  if its less than 0 then 0 else 1
       if (decision__function)<0:                ##https://stackoverflow.com/questions/28503932/calculating-decision-function-of-svm
           predict.append(0)
       else:
         # else 1
           predict.append(1)
           # returning the using numpy array
    return np.array(decision)
```

```
f_cv=decision_function(x_cv,gamma)                            # calculating f_cv based on decision function
```

DEFINING THE SIGMOID FUNCTION

```
def sigmoid(w,x,b):
  # defining the sigmoid function
    return 1/(1+np.exp(-(np.dot(x,w.T)+b)))       #return 1/1+e(-x)
```

THIS IS COST FUNCTION FOR LOGESTIC REGRESSION

```
def logloss(w,x,y,b,reg=0):                                  #HERE WE ARE GIVING THE PARAMETER TO LOGLOSS FUNCTION
```

```
    val=sigmoid(w,x,b)
    # defining the loss function
    return -np.mean(y*np.log10(val)+(1-y)*np.log10(1-val))+reg                      # cost function of logistic regressioN
    # HERE WE ARE SIMPLY EXECUTING THE FORMULA OF COST FUNCTION
```

```
count_one=list(y_cv).count(1)          # HERE WE ARE COMPUTING THE Y+ AS 1    AND Y- AS 0
count_zero=list(y_cv).count(0)

    # calculating y+ and y_
y_positive=(count_one+1)/(count_one+2)

y_negative=1/(count_zero+2)
```

```
def update(y_cv,y_plus,y_minize):
    u_cv=[]
    for point in y_cv:
      # if value 1 then it will be positive                          # update function convert y_cv into y+,y_
        if point==1:
                u_cv.append(y_positive)                   # APPENDING THE Y+ TO VALUE 1 IN CV DATA
        else:
  # else that will be negative
              u_cv.append(y_negative)                 #      # APPENDING THE Y- TO VALUE 1 IN CV DATA
    return(np.array(u_cv))
```

```
y_cv=update(y_cv,y_positive,y_negative)         # HERE COMPUTING THE BOTH Y+ AND Y- IN CV DATA
```

```
w = np.zeros_like(f_cv[0])# initial weight vector
b = 0      # initial intercept value
eta0  = 0.0001     # learning rate
alpha = 0.0001       # lambda value

print(len(y_cv))
print(len(f_cv))
```

```
1000
1000
```

```
initial=logloss(w,f_cv,y_cv,b)                     # SO AS PER INSTRUCTION PRINTING THE INTIAL LOG HERE
print('this is log loss we got as starting')
print("Initial log loss ==>",initial)
```

```
this is log loss we got as starting
Initial log loss ==> 0.3010299956639812
```

SGD alorithm for calculating optimal w and b

```
# refer : https://stackoverflow.com/questions/64739896/implementing-stochastic-gradient-descent
# refer : https://datascience.stackexchange.com/questions/30786/implementation-of-stochastic-gradient-descent-in-python

def sgd_algo(f_cv,y_cv,eta0,alpha,w,b,epoch):
    t=0.001 # tolerence
    test_loss=[]
    epoc=[]   # empty variable created for epochs value in future
    for i in  range(0,epoch):
        epoc.append(i)  # now will append the epoch value
        for K in range(0,N):


          # refer : sgd assginment code snippet taken
            reg=alpha/2*np.dot(w.T,w)
                                                                  #regulrization term
            # writing the equation to update the weight
            w = ((1-eta0*(alpha/N))*w)+((eta0*f_cv[K])*(y_cv[K]-sigmoid(w,f_cv[K],b)))                       # updating weight vector

            # defining the intercept term
            b = b+(eta0*(y_cv[K]-sigmoid(w,f_cv[K],b)))

                       # updatind intercept
        test=logloss(w,f_cv,y_cv,b,reg)
```

```
        test_loss.append(test)

        if i<=t :
            continue
            # here we are a condition if the difference between test last previus value is greatar than t then continue else break
            if abs(test_loss[i]-test_loss[i-1])>t:           # here we are checking  convergence
                continue
            else:
                break
    return w,b,epoc,test_loss
```

```
epoch=50                            # HERE WE HAVE GIVEN THE EPOCH VALUE 40
we,be,epo,loss=sgd_algo(f_cv,y_cv,eta0, alpha,w,b,epoch)
print('this is the optimal weight for model  ')
print("optimal weight = ",we)
print('==>'*25)
print('this is optimial intercept value of model')
print("optimal intercept = ",be)
```

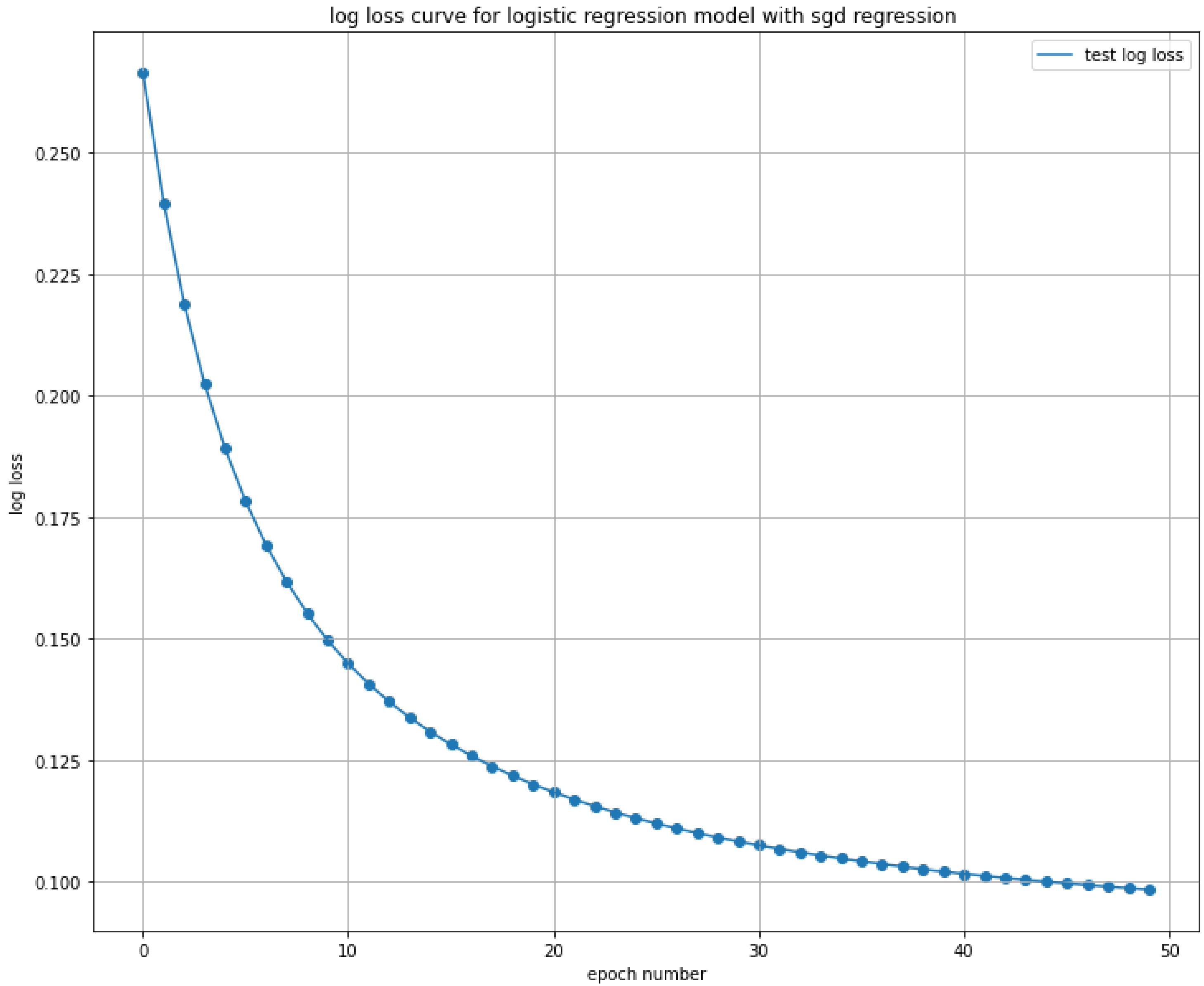
```
this is the optimal weight for model
optimal weight =  [1.18379931]
==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>
this is optimial intercept value of model
optimal intercept =  -0.16239206514508156
```

```
# refer : https://www.geeksforgeeks.org/pyplot-in-matplotlib/

%matplotlib inline
import matplotlib.pyplot as plt
plt.figure(figsize=(12,10))
plt.grid()
plt.plot(epo,loss, label='test log loss')
plt.scatter(epo,loss)
plt.title('log loss curve for logistic regression model with sgd regression ')
plt.xlabel('epoch number')
plt.ylabel("log loss")
```

```
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f0780c391d0>
```



OBSERVATION: LOG-LOSS CURVE ON LOGISTIC REGRESSION

1. AS WE CAN SEE THAT LOSS IS GETTING DECRESING AS NUMBER OF EPOCH INCREASING

2. AS WE HAVE RUN TILL 50 EPOCHS LOSS DECREASING EXPONETIOALLY

```
test_data=decision_function(x_test,gamma)                           # printing the decision fucntion
```

```
def probability(test_data,w,b):

# compute the probability
    probability_P =1/(1+np.exp(-w*test_data+b))
    return probability_P
```

```
prob=probability(test_data,we,be)
print("THESE ARE THE PROBABILITY OF FIRST 20 VALUE ")
print('==>'*25)
print(prob[:20])                                    # finally lets print the  20 probability value
```

```
THESE ARE THE PROBABILITY OF FIRST 20 VALUE
==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>
[[0.22872309]
 [0.9230178 ]
 [0.26044606]
 [0.68005024]
 [0.00180578]
 [0.93105676]
 [0.17322549]
 [0.91092284]
```

```
 [0.17943987]
 [0.88754708]
 [0.10588928]
 [0.89353501]
 [0.06657624]
 [0.02170994]
 [0.0223607 ]
 [0.07905585]
 [0.05595814]
 [0.81707152]
 [0.01397373]
 [0.00603034]]
```

THIS IS 20 PREDICTED PROBABILITY VALUE

✓ 0s completed at 1:18 PM