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 corresponding 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()

In [1]:
```python
from google.colab import drive
drive.mount('content')
```

Mounted at content

In [11]:
```python
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
import numpy as np
from sklearn.svm import SVC
```

In [12]:
```python
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)
```

In [13]:
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=0)
X_train, X_cv, y_train, y_cv = train_test_split(X_train,y_train, test_size=0.4, random_state=0)
```

In [14]:
```python
clf = SVC(C=100,gamma=0.001)
clf.fit(X_train, y_train)
```

Out[14]:
```
SVC(C=100, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

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)

In [15]:
```python
import math
alphas = [i for i in list(clf.dual_coef_)]
def decision_function(X_cv):
  classes = []
  for xq in X_cv:
    sum = 0
    for i , j in enumerate(clf.support_vectors_):
      dist = np.linalg.norm(j - xq)
      sum = sum + (alphas[0][i]*math.exp(-0.001*np.square(dist)))
    classes.extend(sum+clf.intercept_)
  return classes
```

In [16]:
```python
d_fun = decision_function(X_cv)
print(d_fun[0:5])
```

```
[-4.059409504510256, -2.323440307294163, -0.7630431048146846, 1.8841593841742252, 1.0122689248
729415]
```

In [17]:
```python
f_cv = [i for i in clf.decision_function(X_cv)]
print(f_cv[0:5])
```

```
[-4.059409504510256, -2.323440307294163, -0.7630431048146846, 1.8841593841742252, 1.0122689248
729415]
```

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

Let the output of a learning method be $f(x)$. To get cali-
· brated 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 ex-
ample, if the model learns to discriminate the train set per-
fectly and orders all the negative examples before the posi-
tive examples, then the sigmoid transformation will output
just a 0,1 function. So we need to use an independent cali-
bration 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 train-
ing 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

1. 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

1. 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

TASK F

In [18]:
```python
def initialize_weights():
    w=np.zeros(1)
    b=0
    return w,b

import math
def sigmoid(z):
    sig = 1/(1 + math.exp(-z))

    return sig

def logloss(y_true,y_pred):
    n = len(y_true)
    loss = 0
    for i in range(n):
        loss +=  (y_true[i] *math.log(y_pred[i],10)) + (1-y_true[i])*math.log((1-y_pred[i]),10)
    log_loss = -loss/n
        #'''In this function, we will compute log loss '''

    return log_loss

def gradient_dw(x,y,w,b,alpha,N):
    y1 = np.dot(w.T,x) + b
    sig = sigmoid(y1)
    dw = x*(y - sig) -(alpha/N)*w.T

    return dw

def gradient_db(x,y,w,b):
    y1 = np.dot(w.T,x) + b
    sig = sigmoid(y1)

    db = y - sig
        #'''In this function, we will compute gradient w.r.to b '''
    return db
```

In [42]:
```python
def train(X_train,y_train,epochs,alpha,eta0):
    train_loss  = list()
    w,b = initialize_weights()
    N = len(X_train)
    for i in range(epochs):
        for i in range(len(X_train)):
            dw = gradient_dw(X_train[i],y_train[i],w,b,alpha,N)
            db = gradient_db(X_train[i],y_train[i],w,b)
            w = w + eta0*dw
            b = b + eta0*db

        y_train_pred = []
```

```
        for i in range(len(X_train)):
          z = np.dot(w.T,X_train[i]) + b
          sig = sigmoid(z)
          y_train_pred.append(sig)
        train_loss.append(logloss(y_train,y_train_pred))

      return w,b,train_loss
```

In [43]:
```
f_train = [i for i in clf.decision_function(X_train)]


N_plus = 0
N_minus = 0
for i in y_train:
  if i == 1:
    N_plus +=1
  else:
    N_minus +=1

y_train = [(N_plus+1)/(N_plus+2) if i == 1 else 1/(N_minus+2) for i in y_train]

import pandas as pd
data = {'f_train':f_train, 'y_train':y_train}
df = pd.DataFrame(data)
x_train = df['f_train']
y_train = df['y_train']


alpha=0.0001
eta0=0.0001
N=len(x_train)
epochs=100

w,b,train_loss=train(x_train,y_train,epochs,alpha,eta0)
```
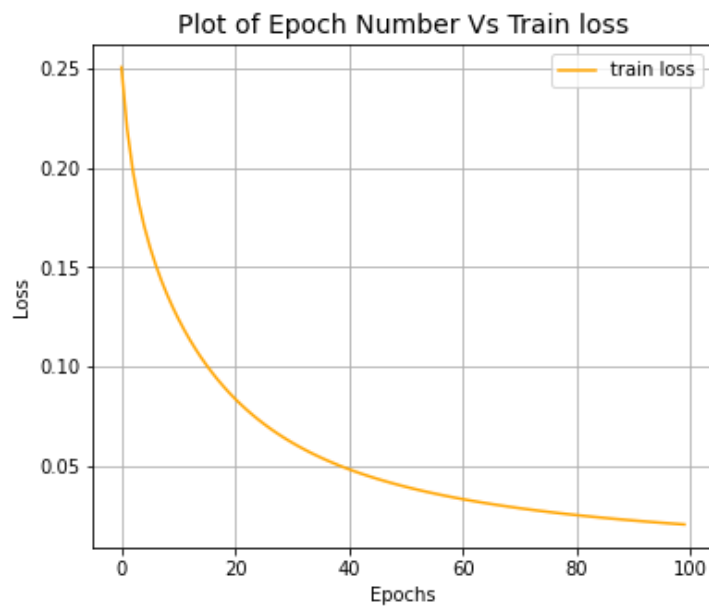
In [44]:
```
epoch = np.arange(100)
import matplotlib.pyplot as plt
plt.figure( figsize=(6,5))
plt.grid()
plt.plot(epoch,train_loss,color='orange')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title('Plot of Epoch Number Vs Train loss',fontsize = 14)
plt.legend(['train loss'])
plt.show()
```

Plot of Epoch Number Vs Train loss

In [40]: `print(w,b)`

[0.34062765] -2.851242297812991

In [41]:
```python
f_test = [i for i in decision_function(X_test)]

for i in range(10):
    print(f'P(Y=1|X_test{i}) = {round((1/(1 + math.exp(-(w*f_test[i]+b)))),3)}')
```

P(Y=1|X_test0) = 0.017
P(Y=1|X_test1) = 0.055
P(Y=1|X_test2) = 0.031
P(Y=1|X_test3) = 0.052
P(Y=1|X_test4) = 0.021
P(Y=1|X_test5) = 0.096
P(Y=1|X_test6) = 0.037
P(Y=1|X_test7) = 0.029
P(Y=1|X_test8) = 0.02
P(Y=1|X_test9) = 0.061