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

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 [27]:
```python
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
import numpy as np
from sklearn.svm import SVC
from matplotlib import pyplot as plt
```

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

X.shape, y.shape
```

Out[2]:
```
((5000, 5), (5000,))
```

In [3]:
```python
X[0], y[0]
```

Out[3]:
```
(array([ 0.35375589,  0.28929301,  0.48523022,  0.62203148, -1.24936153]), 0)
```

In [4]:
```python
y
```

Out[4]:
```
array([0, 1, 0, ..., 1, 0, 1])
```

In [5]:
```python
from collections import Counter

Counter(y)
```

Out[5]:
```
Counter({0: 3486, 1: 1514})
```

In [6]:
```python
np.count_nonzero(y)
```

Out[6]:
```
1514
```

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 [7]:
```python
from sklearn.metrics.pairwise import euclidean_distances

GAMMA = 0.001

# you can write your code here
def rbfKernel(xi, xq, gamma=GAMMA):
    ''' Manual Impl RBF Kernel'''
    l2_norm = np.linalg.norm(xi-xq)  # Euc-Distance
    squared_norm = l2_norm**2
    # sq_l2_norm = euclidean_distances(xi, xq, squared=True)  // Alternative to linalg.norm()
    return np.exp(-gamma * squared_norm)
```

In [8]:
```python
# Validating the RBF Kernel Func
from sklearn.metrics.pairwise import rbf_kernel

x1 = np.array([1, 2])
x2 = np.array([2, 1])

a1 = rbf_kernel(x1.reshape(1,-1), x2.reshape(1,-1), gamma=0.001)
#print(np.asscalar(a1))
print(a1.item())

a2 = rbfKernel(x1, x2, 0.001)
print(a2)
```

```
0.9980019986673331
0.9980019986673331
```

In [9]:
```python
def decision_fun(x, alpha_coef, sup_vecs, intercept):
    '''alpha_coef = alpha_i * y_i for corresponding support vector'''
    n = len(x)
    res = np.zeros(n)
    for i in range(n):
        res[i] = sum([(cls_alp * rbfKernel(sv, x[i])) for sv, cls_alp in zip(sup_vecs, alpha_coef)]) + intercept

    return res
```

In [10]:
```python
# Train Test SPlit
# Ref 3 Split : https://stackoverflow.com/questions/38250710/how-to-split-data-into-3-sets-train-validation-and-test
size = len(y)
o1, o2 = int(.6*size), int(.8*size)
x_train, x_cv, x_test = np.split(X, [o1, o2])
print('X -< ', x_train.shape, x_cv.shape, x_test.shape)

y_train, y_cv, y_test = np.split(y, [o1, o2])
print('Y -< ',y_train.shape, y_cv.shape, y_test.shape)
```

```
X -<  (3000, 5) (1000, 5) (1000, 5)
Y -<  (3000,) (1000,) (1000,)
```

In [11]:
```python
# Classifier (Score Verification Sample Check)

clf = SVC(gamma=GAMMA, C=100)
clf.fit(x_train, y_train)
print(clf.score(x_train, y_train))

from sklearn.metrics import accuracy_score

print(accuracy_score(clf.predict(x_train), y_train))
```

```
0.9286666666666666
0.9286666666666666
```

NOTE: Looking At Score It seems that there are some outliers in Training Dataset

In [12]:
```python
# IGNORE (Check)
clf.predict(x_train).shape
```

Out[12]:
```
(3000,)
```

In [13]:
```python
#Params
print('Support Vectors Indexes : ', len(clf.support_))
print('# Support Vectors Per class : ', clf.n_support_)
print('Alphas (ie Langrangians): ', len(clf.dual_coef_[0]))
print('Intercept : ', clf.intercept_[0])

# https://stackoverflow.com/questions/33860938/how-to-get-all-alpha-values-of-scikit-learn-svm-classifier
alphas = clf.dual_coef_[0]
sv_idxs = clf.support_
intercept = clf.intercept_[0]

#print('Alpha : ', alphas[-30:])


supp_vecs = x_train[sv_idxs]
supp_vecs_yi = y_train[sv_idxs]

# Sample Check index & corresp support vector
supp_vecs[4] == x_train[sv_idxs[4]]
```

```
Support Vectors Indexes :  538
# Support Vectors Per class :  [269 269]
Alphas (ie Langrangians):  538
Intercept :  2.8407827279266704
```

Out[13]:
```
array([ True,  True,  True,  True,  True])
```

In [14]:
```python
# alphas := alphai * coefi
# REF : https://stackoverflow.com/questions/33860938/how-to-get-all-alpha-values-of-scikit-learn-svm-classifier

# Exact Prediction
pred_d_cv = decision_fun(x_cv, alphas, supp_vecs, intercept)

# Signed (decisive prediction)
pred_s_cv = np.where(pred_d_cv < 0, 0, np.ones(len(pred_d_cv))).astype(int)

# Original Prediction by trained sklearn classifer
orig_pred_cv = clf.predict(x_cv)

is_same = (pred_s_cv == orig_pred_cv).all()
print('Same : ', is_same)
```

```
Same :  True
```

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

Check this PDF

TASK F

> 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. 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

4) Apply Custom SGD (Logistic Reg) Algo

In [15]:
```python
cnt_plus = np.count_nonzero(y_train)
cnt_minus = len(y_train) - cnt_plus

print(f'+ve :- {cnt_plus}, -ve :- {cnt_minus}')

y_plus = (cnt_plus + 1) / (cnt_plus + 2)
y_minus = 1 / (cnt_minus + 2)

print(y_plus, y_minus)
print(type(y_plus))
print(round(y_plus, 4), round(y_minus, 4))

y_plus, y_minus = round(y_plus, 4), round(y_minus, 4)
```

```
+ve :- 894, -ve :- 2106
0.9988839285714286 0.0004743833017077799
<class 'float'>
0.9989 0.0005
```

In [16]:
```python
f_cv = pred_d_cv  # predicted y_cv (via manual func impl)

# Modify the label (ie )
new_y_cv = np.where(y_train == 0, y_minus, y_plus)

cnt_plus == np.count_nonzero(new_y_cv == y_plus)
```

Out[16]:
```
True
```

In [17]:
```python
Counter(new_y_cv)
```

Out[17]:
```
Counter({0.0005: 2106, 0.9989: 894})
```

In [30]:
```python
#from custom_sgd_lr import train
import custom_sgd_lr as logreg
import importlib, sys

#print(sys.modules['custom_sgd_lr'] is not None)
# NOTE : If the import module is altered whilst trial & error then
#        Either you need to restart the kernel
#        or reload the module again
#        because once the module is imported its cached till kernel is restarted (ie all modules info are flushed out)
if sys.modules['custom_sgd_lr'] is not None:
    logreg = importlib.reload(logreg)

alpha=0.0001
eta0=0.0001
epochs=20

w_cv, b_cv, tr_loss, nEpochs = logreg.train(f_cv, new_y_cv, y_plus, y_minus, epochs, alpha, eta0)

print(f'W : {w_cv}, B : {b_cv}')
```

```
Epoch0 :- Loss : 2.315357193724607
Epoch1 :- Loss : 2.315357193724607
W : 0.04049328210637219, B : -0.03811484564365198
```

5) Predict the Test Data

In [31]:
```python
# Exact Prediction

# Predicted Test Point (SVM)
f_test = decision_fun(x_test, alphas, supp_vecs, intercept)

# Prediction (Calibration | Logistic Reg)
z = np.dot(w_cv, f_test) + b_cv
pred_calib_test = 1.0 / (1 + np.exp(-z))

# Signed (decisive prediction)
pred_label_test = np.where(pred_calib_test < 0.5, 0, np.ones(len(pred_d_cv))).astype(int)

#np.count_nonzero(y_test == pred_label_test)
```

In [32]:
```python
np.count_nonzero(y_test == pred_label_test) # Correctly Predicted after Calibration test
```

Out[32]:
```
892
```

In [33]:
```python
len(y_test) - 892  # Falsely Predicted after Calibration test
```

Out[33]:
```
108
```

In [36]:
```python
epochs, nEpochs
```

Out[36]:
```
(20, 2)
```

In [37]:
```python
tr_loss
```

Out[37]:
```
[2.315357193724607, 2.315357193724607]
```

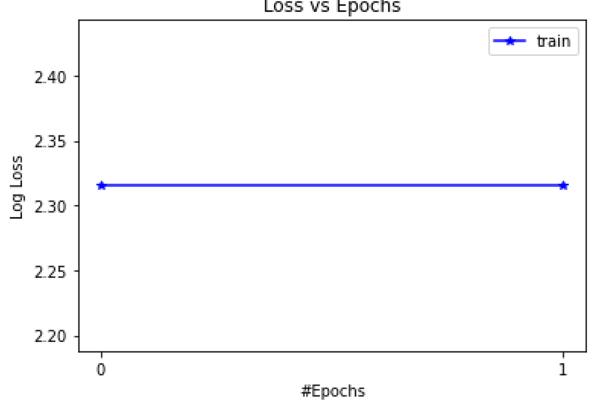
In [40]:
```python
# Plot
xlbls = range(nEpochs)

plt.plot(xlbls, tr_loss, 'b*-', label='train', markersize=6)
plt.title('Loss vs Epochs')
plt.xlabel('#Epochs')
plt.ylabel('Log Loss')

plt.xticks(xlbls)

plt.legend()
```

Out[40]:
```
<matplotlib.legend.Legend at 0x1aa0c866cd0>
```



Observations

After Plat Scaling (ie Calibration), it took algo to train in fewer epoch comparatively

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