9E and 9F: Finding the Probability P(Y==1|X)

9E: 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 $sign(\sum_{i=1}^{n}(y_i\alpha_iK(x_i,x_q)) + intercept)$, 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()

To implement this decision_function()

In [1]:

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

In [2]:

```
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 [3]:

```
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=42
)
X_train,X_val,y_train,y_val = train_test_split(X_train,y_train,test_size=0.2,rando
m_state=24)
print(X_train.shape, y_train.shape)
print(X_val.shape, y_val.shape)
print(X_test.shape, y_test.shape)
```

```
(3200, 5) (3200,)
(800, 5) (800,)
(1000, 5) (1000,)
```

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 [4]:

```
# you can write code here
gamma = 0.001
clf = SVC(gamma=gamma, C=100)
clf.fit(X_train,y_train)
```

Out[4]:

```
SVC(C=100, 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,
```

```
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [5]:

```
def K(xq):
    val = 0
    for alpha,xi in zip(clf.dual_coef_[0],clf.support_vectors_): #the dual_coef_[i
] contains label[i]*alpha[i]
        val += alpha*np.exp(-gamma*np.linalg.norm(xi-xq)**2)
    return val+clf.intercept_.item()
```

In [6]:

```
def dec_fun(X_val):
    fcv = []
    for xq in X_val:
        fcv.append(K(xq))
    return(np.array(fcv))
```

In [7]:

```
dec_fun(X_val)[:5]
```

Out[7]:

```
array([-2.69065026, -4.01123357, -2.48966713,  1.35046624, -2.59528514])
```

In [8]:

```
clf.decision_function(X_val)[:5]
```

Out[8]:

```
array([-2.69065026, -4.01123357, -2.48966713,  1.35046624, -2.59528514])
```

We observe that the custom function gives same values as the inbuilt decision function.

9F: Implementing Platt Scaling to find P(Y==1|X)

Check this [PDF]

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

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

$$y_+ = \frac{N_+ + 1}{N_+ + 2};\ y_- = \frac{1}{N_- + 2} \tag{4}$$

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

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.

```
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

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

In [9]:

```
fcv = dec_fun(X_val)
y_val_cpy = y_val.astype('float')
unique,counts = np.unique(y_val_cpy,return_counts=True)
print(unique,counts)
y_val_cpy[y_val_cpy==unique[0]]=1.0/(counts[0]+2)
y_val_cpy[y_val_cpy==unique[1]]=(counts[1]+1.0)/(counts[1]+2)
```

```
[0. 1.] [550 250]
```

In [10]:

```python
def sigmoid(w,x,b):
    return 1/(1+np.exp(-(w@x+b)))
```

In [11]:

```python
def update_weights(X,y,w,b,lamda,alpha,N):
    w_new = (1-alpha*lamda/N)*w + alpha*X*(y-sigmoid(w,X.T,b))
    b_new = b + alpha*(y-sigmoid(w,X.T,b))
    return w_new,b_new
```

In [12]:

```python
def next_batch(X, y, batchSize):
    # loop over our dataset `X` in mini-batches of size `batchSize`
    for i in np.arange(0, X.shape[0], batchSize):
        # yield a tuple of the current batched data and labels
        yield (X[i:i + batchSize], y[i:i + batchSize])
```

In [13]:

```python
def compute_log_loss(A,n):# your code
    loss=0
    for Y in A:
        loss += Y[0]*math.log(Y[1])+(1-Y[0])*math.log(1-Y[1])
    loss = -loss/n
    return loss
```

In [14]:

```python
w = np.zeros((1,))
b = 0
lamda  = 0.0001
alpha = 0.0001
N = len(fcv)
w.shape
```

Out[14]:

```
(1,)
```

In [15]:

```python
import math
lossHistoryTrain = []
lossHistoryTest = []
epochs = range(1,30)
for epoch in epochs:
    # initialize the total loss for the epoch
    epochLossTrain = []
    epochLossTest = []
    # loop over our data in batches
    for (batchX, batchY) in next_batch(fcv, y_val_cpy, 1):
        preds = sigmoid(w,batchX,b)
        loss = -(batchY*math.log(preds)+(1-batchY)*math.log(1-preds))
        epochLossTrain.append(loss)
        w, b = update_weights(batchX,batchY,w,b,lamda,alpha,N)
```

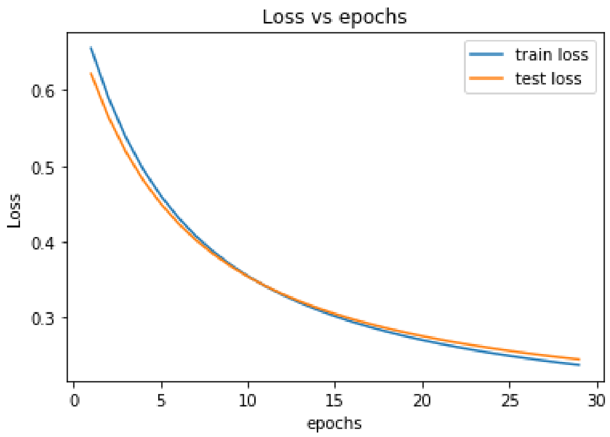
```
    avgLossTrain = np.average(epochLossTrain)
    lossHistoryTrain.append(avgLossTrain)
    print("iteration:{}".format(epoch))
    print("Training Loss:{}".format(avgLossTrain))
    y_pred = [sigmoid(w,x.reshape(-1,1),b) for x in dec_fun(X_test)]
    avgLossTest = compute_log_loss(zip(y_test,y_pred),len(y_test))
    lossHistoryTest.append(avgLossTest)
    print("Test Loss:{}".format(avgLossTest))
    print('='*75)
print('Final Weights:')
print(w)
print('Final Intercept:',b)
```

```
iteration:1
Training Loss:0.6552991440625178
Test Loss:0.6216799191875771
===========================================================================
iteration:2
Training Loss:0.5899504093091643
Test Loss:0.5645912433935767
===========================================================================
iteration:3
Training Loss:0.5376450468017597
Test Loss:0.5185955466686065
===========================================================================
iteration:4
Training Loss:0.4953452949234486
Test Loss:0.4811179312138139
===========================================================================
iteration:5
Training Loss:0.4607177842269182
Test Loss:0.4502053772357796
===========================================================================
iteration:6
Training Loss:0.43201116620223556
Test Loss:0.4243957071932323
===========================================================================
iteration:7
Training Loss:0.40792005329857134
Test Loss:0.40259569163253994
===========================================================================
iteration:8
Training Loss:0.38746914857339987
Test Loss:0.3839836238254411
===========================================================================
iteration:9
Training Loss:0.3699243012691913
Test Loss:0.3679364008743245
===========================================================================
iteration:10
Training Loss:0.35472727885274635
Test Loss:0.3539765039079248
===========================================================================
iteration:11
Training Loss:0.34144882027064943
Test Loss:0.3417338531083492
===========================================================================
iteration:12
Training Loss:0.32975501183164174
Test Loss:0.3309183847754427
===========================================================================
iteration:13
Training Loss:0.31938314266484286
Test Loss:0.3213002439334413
===========================================================================
iteration:14
Training Loss:0.31012425112115755
```

```
Test Loss:0.3126953673758365З
================================================================================
iteration:15
Training Loss:0.30181039569982493
Test Loss:0.3049548927351467
================================================================================
iteration:16
Training Loss:0.29430527895800895
Test Loss:0.2979573002257887
================================================================================
iteration:17
Training Loss:0.2874972693182899
Test Loss:0.29160252219300625
================================================================================
iteration:18
Training Loss:0.2812941533109972
Test Loss:0.2858074827676511
================================================================================
iteration:19
Training Loss:0.2756191487461773
Test Loss:0.28050268691278957
================================================================================
iteration:20
Training Loss:0.27040784585353755
Test Loss:0.27562958702863327
================================================================================
iteration:21
Training Loss:0.26560583812342137
Test Loss:0.27113853126412746
================================================================================
iteration:22
Training Loss:0.2611668707347844
Test Loss:0.26698715111215393
================================================================================
iteration:23
Training Loss:0.25705138105688347
Test Loss:0.26313908375226086
================================================================================
iteration:24
Training Loss:0.2532253388312813
Test Loss:0.25956295170834337
================================================================================
iteration:25
Training Loss:0.24965931739417382
Test Loss:0.2562315419516265
================================================================================
iteration:26
Training Loss:0.24632774449024541
Test Loss:0.25312114082534004
================================================================================
iteration:27
Training Loss:0.2432082937828996
Test Loss:0.2502109916324353
================================================================================
iteration:28
Training Loss:0.24028138741329827
Test Loss:0.24748284948063623
================================================================================
iteration:29
Training Loss:0.23752978683073997
Test Loss:0.24492061377030055
================================================================================
Final Weights:
[0.90106319]
Final Intercept: [-0.1014738]
```

In [17]:

```python
import matplotlib.pyplot as plt
plt.plot(epochs,lossHistoryTrain)
plt.plot(epochs,lossHistoryTest)
plt.legend(['train loss','test loss'])
plt.title('Loss vs epochs')
plt.xlabel('epochs')
plt.ylabel('Loss')
plt.show()
```



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