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 [59]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
import numpy as np
from sklearn.svm import SVC
import math
from sklearn.model_selection import train_test_split
```

In [60]:
```
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)

In [61]:
```
#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_final,X_cv,y_train_final,y_cv = train_test_split(X_train,y_train,test_size = 0.25,random_state = 0)
#train the algorithm
print("Shape of the X_train:{0} and Y_train:{1}".format(X_train_final.shape,y_train_final.shape))
print("Shape of the X_test : {0} and y_test : {1}".format(X_test.shape,y_test.shape))
print("Shape of the Validation :{0} and Y_cv : {1}".format(X_cv.shape,y_cv.shape))
support_vector_classifier = SVC(gamma = 0.001,C = 100)
support_vector_classifier.fit(X_train_final,y_train_final)
```

```
Shape of the X_train:(3000, 5) and Y_train:(3000,)
Shape of the X_test : (1000, 5) and y_test : (1000,)
Shape of the Validation :(1000, 5) and Y_cv : (1000,)
```

Out[61]: `SVC(C=100, gamma=0.001)`

In [62]:
```
class SupportVectorClassifier:

    def __init__(self,gamma = 0.001,C = 100):

        self.gamma = gamma
        self.C = C

    def fit(self,X_train,y_train):

        model = SVC(gamma = self.gamma,C = self.C).fit(X_train,y_train)
        return model

    def compute_similarity_matrix(self,xi,xq,gamma):

        norm = np.sum(( xi - xq) ** 2 ,axis = -1)

        similarity_function  = np.exp(- (gamma) * norm)

        return similarity_function

    def kernal_gram_matrix(self,Xi,Xj,gamma):
        kernal_matrix_similarity = np.zeros((Xi.shape[0],Xj.shape[0]))
        for idx,row in enumerate(Xi):
            for idx_col,column in enumerate(Xj):
                rbf_kernal = self.compute_similarity_matrix(row,column,self.gamma)
                kernal_matrix_similarity[idx][idx_col] = rbf_kernal
        return kernal_matrix_similarity

    def decision_function(self,XCV,support_vectors,dual_coeff,intercept):

        #compute the kernal similarity Matrix
        kernal_gram_matixes= self.kernal_gram_matrix(XCV, support_vectors,self.gamma)

        #yi* alpha i * kernal(xi,xq)
        decision_fun = dual_coeff * kernal_gram_matixes
        decision_custom = np.sum(decision_fun,axis = -1) + intercept

        return decision_custom.reshape(-1,1)
```

In [63]:
```
svc = SupportVectorClassifier(gamma = 0.001,C = 100)
support_vector_classifier = svc.fit(X_train_final,y_train_final)
#intercept b
intercept = support_vector_classifier.intercept_

#dual_coefficient yi*alphai

svm_dual_coeff_ = support_vector_classifier.dual_coef_

#get the support_vector
support_vector = support_vector_classifier.support_vectors_

f_cv = svc.decision_function(X_cv,support_vector,svm_dual_coeff_,intercept)
```

In [64]:
```
sk_df = support_vector_classifier.decision_function(X_cv)
sk_df = sk_df.reshape(-1,1)
```

In [65]:
```
#check the difference between the custom predction and actual predictiom
difference = f_cv - sk_df
differece_error = [i  for i in range(len(difference)) if difference[i] > 1e-10]
print(differece_error)
```

```
[]
```

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

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 [66]:
```
#implemented Calibrated CV

#input actual_y -> function(x)-> ycv
#construct a data set

def label_count(Y):
    positive_label,negative_label = 0,0
    for label in range(len(Y)):
        if Y[label] == 1:
            positive_label += 1
        else:
            negative_label +=1
    assert((positive_label +negative_label) == len(Y) )
    return positive_label,negative_label
no_of_positive_point,no_of_negative_point = label_count(y_train_final)
print("Positive Label : {0} and Negative Label : {1}".format(no_of_positive_point,no_of_negative_point))
y_positive,y_negative = (no_of_positive_point + 1) / (no_of_positive_point + 2), 1 / (no_of_negative_point + 2)
print("Calibrated positive point:{0} and Calibrated Negative Point : {1}".format(y_positive,y_negative))
```

```
Positive Label : 906 and Negative Label : 2094
Calibrated positive point:0.998898678414097 and Calibrated Negative Point : 0.00047709923664122136
```

In [67]:
```
#In platt Scaling We build a new validation dataset- create a bucket and computing the averagefrom our case have no of points in low we directly apply y_avg
def average_y(Y,y_positive,y_negative):

    y_bucket = []
    for label in range(len(Y)):
        if Y[label] == 1:
            y_bucket.append(y_positive)
        else:
            y_bucket.append(y_negative)
    return y_bucket
y_cv_modified = average_y(y_cv,y_positive,y_negative)
```

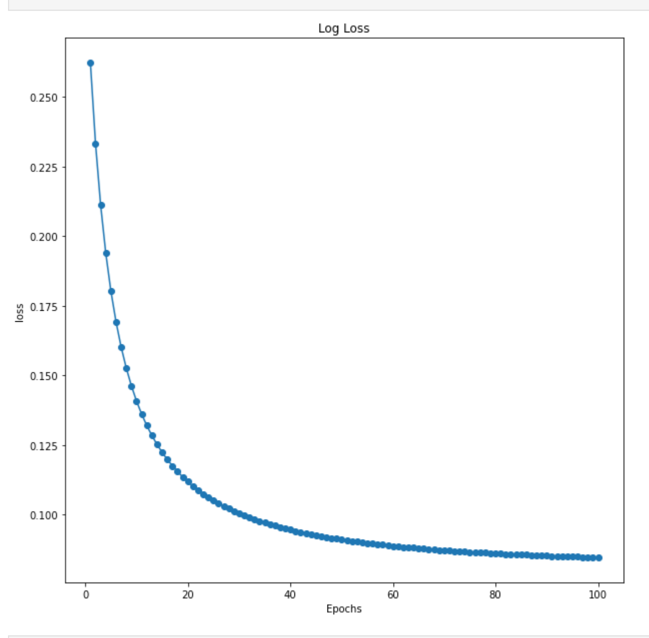
In [68]:
```
class SGD:
    def __init__(self,learning_rate,epochs):

        self.learning_rate = learning_rate
        self.epochs = epochs

    def sigmoid(self,z):
        sigmoid_of_Z= 1 / (1 + np.exp(-z))
        return sigmoid_of_Z

    def gradient(self,x,y,w,b,alpha,N):
        dw = ((x * ((y - self.sigmoid(np.dot(w,x) + b))) - ((alpha / N) * w)))
        db = y - self.sigmoid(np.dot(w,x) + b)
        return dw,db
    def logloss(self,y_true,y_pred):

        loss = 0
        n = len(y_true)
        if(len(y_true) == len(y_pred)):
            for l in range(len(y_true)):

                loss += (y_true[l] * math.log10(y_pred[l])) + ((1-y_true[l]) * math.log10(1-y_pred[l]))

            loss = (-1 / n) * loss

        return loss

    def fit(self,X,Y,alpha):

        loss_list = []
        wait = 0
        #initialize weight
        w = 0
        b = 0
        print(self.learning_rate)
        for epoch in range(1,epochs+1):

            for data in range(len(X)):
                #compute gradient
                dw,db= self.gradient(X[data],Y[data],w,b,alpha,len(X))
                #update weight
                w = w +  self.learning_rate * dw
                #update bias
                b = b +  self.learning_rate * db
            y_pred  = [self.sigmoid(np.dot(w,X[data]) + b) for data in range(len(X))]
            loss = self.logloss(Y,np.array(y_pred))

            loss_list.append(loss)
            if epoch >= 2:
                if loss_list[epoch-2]  == loss_list[epoch-1]:
                    print("No improvement in training")
                    break

            print('Epoch--{0}'.format(epoch) )

            print("Training Loss  : {0} ".format(loss[0]))

        return loss_list,w,b
    def predict(self,x,w,b):
        z = np.dot(w,x) + b
        return self.sigmoid(z)
```

In [69]:
```
learning_rate = 1e-4
epochs = 100
alpha=0.0001
y_cv_modified = np.array(y_cv_modified).reshape(-1,1)
sgd = SGD(learning_rate,epochs)
```

In [70]:
```
loss_list,w,b= sgd.fit(f_cv,y_cv_modified,alpha)
print("Optimized W:{0} and b:{1}".format(w,b))
```

```
0.0001
Epoch--1
Training Loss  : 0.26234390399314991
Epoch--2
Training Loss  : 0.23328949504531726
Epoch--3
Training Loss  : 0.21112989001987448
Epoch--4
Training Loss  : 0.19390095342750988
Epoch--5
Training Loss  : 0.1802395179581706
Epoch--6
Training Loss  : 0.16920443188047013
Epoch--7
Training Loss  : 0.16014004963550305
Epoch--8
Training Loss  : 0.15258251814728516
Epoch--9
Training Loss  : 0.14619776722588454
Epoch--10
Training Loss  : 0.14074077868139112
Epoch--11
Training Loss  : 0.13602866186090992
Epoch--12
Training Loss  : 0.1319226272846691
Epoch--13
Training Loss  : 0.12831572881244926
Epoch--14
Training Loss  : 0.12512439016558863
Epoch--15
Training Loss  : 0.12228245064105836
Epoch--16
Training Loss  : 0.11973691421734095
Epoch--17
Training Loss  : 0.11744486869036001
Epoch--18
Training Loss  : 0.11537122090892515
Epoch--19
Training Loss  : 0.1134870096789125
Epoch--20
Training Loss  : 0.11176813331551005
Epoch--21
Training Loss  : 0.11019437877586269
Epoch--22
Training Loss  : 0.10874867287090426
Epoch--23
Training Loss  : 0.10741649892603101
Epoch--24
Training Loss  : 0.1061854380530169
Epoch--25
Training Loss  : 0.10504480524062751
Epoch--26
Training Loss  : 0.10398535829020995
Epoch--27
Training Loss  : 0.10299906322149323
Epoch--28
Training Loss  : 0.10207890382703495
Epoch--29
Training Loss  : 0.10121872601838272
Epoch--30
Training Loss  : 0.10041310979663653
Epoch--31
Training Loss  : 0.09965726331235482
Epoch--32
Training Loss  : 0.09894693470715851
Epoch--33
Training Loss  : 0.09827833836006895
Epoch--34
Training Loss  : 0.0976480928728578
Epoch--35
Training Loss  : 0.09705316867628976
Epoch--36
Training Loss  : 0.09649084356375484
Epoch--37
Training Loss  : 0.09595866479027237
Epoch--38
Training Loss  : 0.09545441663524254
Epoch--39
Training Loss  : 0.09497609253320682
Epoch--40
Training Loss  : 0.09452187104051173
Epoch--41
Training Loss  : 0.09409009503663503
Epoch--42
Training Loss  : 0.0936792536640946
Epoch--43
Training Loss  : 0.09328796659580965
Epoch--44
Training Loss  : 0.09291497028774075
Epoch--45
Training Loss  : 0.09255910593087173
Epoch--46
Training Loss  : 0.09221930886265292
Epoch--47
Training Loss  : 0.091894599235923
Epoch--48
Training Loss  : 0.09158407377461211
Epoch--49
Training Loss  : 0.09128689847149252
Epoch--50
Training Loss  : 0.09100230210482986
Epoch--51
Training Loss  : 0.09072957046884425
Epoch--52
Training Loss  : 0.09046804122800774
Epoch--53
Training Loss  : 0.09021709931792915
Epoch--54
Training Loss  : 0.08997617282631465
Epoch--55
Training Loss  : 0.08974472929657472
Epoch--56
Training Loss  : 0.08952227240436805
Epoch--57
Training Loss  : 0.08930833896393585
Epoch--58
Training Loss  : 0.08910249622669368
Epoch--59
Training Loss  : 0.08890433943934649
Epoch--60
Training Loss  : 0.08871348963291924
Epoch--61
Training Loss  : 0.08852959161763156
Epoch--62
Training Loss  : 0.08835231216162523
Epoch--63
Training Loss  : 0.08818133833417724
Epoch--64
Training Loss  : 0.08801637599634181
Epoch--65
Training Loss  : 0.0878571484239491
Epoch--66
Training Loss  : 0.08770339504962435
Epoch--67
Training Loss  : 0.0875548703120069
Epoch--68
Training Loss  : 0.08741134260166905
Epoch--69
Training Loss  : 0.08727259329439284
Epoch--70
Training Loss  : 0.0871384158634822
Epoch--71
Training Loss  : 0.0870086150636789
Epoch--72
Training Loss  : 0.08688300618004377
Epoch--73
Training Loss  : 0.08676141433584818
Epoch--74
Training Loss  : 0.08664367385414927
Epoch--75
Training Loss  : 0.08652962766825541
Epoch--76
Training Loss  : 0.08641912677677396
Epoch--77
Training Loss  : 0.086312029739364
Epoch--78
Training Loss  : 0.08620820220969132
Epoch--79
Training Loss  : 0.08610751650243058
Epoch--80
Training Loss  : 0.08600985119145745
Epoch--81
Training Loss  : 0.0859150907366405
Epoch--82
Training Loss  : 0.08582312513689555
Epoch--83
Training Loss  : 0.08573384960736886
Epoch--84
Training Loss  : 0.08564716427882373
Epoch--85
Training Loss  : 0.08556297391746323
Epoch--86
Training Loss  : 0.08548118766360062
Epoch--87
Training Loss  : 0.08540171878770683
Epoch--88
Training Loss  : 0.08532448446251112
Epoch--89
Training Loss  : 0.08524940554993635
Epoch--90
Training Loss  : 0.08517640640175544
Epoch--91
Training Loss  : 0.08510541467295472
Epoch--92
Training Loss  : 0.08503636114686645
Epoch--93
Training Loss  : 0.08496917957122135
Epoch--94
Training Loss  : 0.0849038065043314
Epoch--95
Training Loss  : 0.08484018117068305
Epoch--96
Training Loss  : 0.08477824532527921
Epoch--97
Training Loss  : 0.08471794312611652
Epoch--98
Training Loss  : 0.0846592210142411
Epoch--99
Training Loss  : 0.0846020276008571
Epoch--100
Training Loss  : 0.08454631356101536
Optimized W:[1.44961872] and b:[-0.18626539]
```

In [71]:
```
import matplotlib.pyplot as plt
plt.figure(figsize = (10,10))
plt.plot(range(1,epochs + 1),loss_list)
plt.scatter(range(1,epochs+1),loss_list)
plt.title("Log Loss")
plt.xlabel("Epochs")
plt.ylabel("loss")
plt.show()
```



In [72]:
```
#test Data
ftest = svc.decision_function(X_test,support_vector,svm_dual_coeff_,intercept)
test_prediction  = [sgd.predict(ftest[data],w[0],b[0])[0] for data in range(len(ftest))]
```

In [73]:
```
print("test Prediction : {0}".format(test_prediction))
```

```
test Prediction : [0.004969085753400267, 0.47172988567687335, 0.06292162797181014, 0.4095663384280869, 0.01140033712095906, 0.9259763715110881, 0.123656385165
41687, 0.046788111997427245, 0.0091663688641821, 0.587843417715812, 0.015006597986503008, 0.00396070783736178, 0.029818039457043463, 0.14609579168440315, 0.01
4945209957721714, 0.03131214377426784, 0.9345603976529385, 0.0052418777140559245, 0.4110100860659789, 0.8895417931530457, 0.014402329226791405, 0.002199656329
178635, 0.009941930326802193, 0.13125844227504047, 0.9466558956954717, 0.028231504596045367, 0.965887174631899, 0.7253491644589101, 0.948790049442062, 0.00193
81617031063703, 0.01344490819041161, 0.08800046071647795, 0.02966517906702783, 0.9688470781895671, 0.038380802429151935, 0.018434573396832455, 0.0227480869792
84786, 0.806287544725349, 0.892218807517814, 0.020030795953398742, 0.8998408320921583, 0.03615966297659763, 0.17937112970496885, 0.45612959165443817, 0.025736
08492544919, 0.008026124639616194, 0.958247539026555, 0.8539575223687195, 0.024180882862586836, 0.8968458788928236, 0.030012366497741493, 0.7835538366509989,
0.9553303684625589, 0.014202752891805257, 0.0539202064150608, 0.6597687572978111, 0.0054050745074294745, 0.8608393647651055, 0.0022143795629986553, 0.02579408
8814771014, 0.8902287186289125, 0.9504415022696011, 0.2488237642383243, 0.390105482328496966, 0.8639629160260216, 0.7302045241577895, 0.2544252845808563, 0.047
14246543115774, 0.8142335830158376, 0.15777663425233646, 0.07132266205980414, 0.005880248077556567, 0.007625974173235556, 0.8452587836266152, 0.84567691089522
07, 0.9296716072186454, 0.3499841377882236, 0.8072556703568875, 0.9498612381451831, 0.2658891507486397, 0.08364924976124825, 0.0033073348738980463, 0.01050982
3039617118, 0.8584156005944402, 0.011041964521947594, 0.41141101960332166, 0.3150029323290646, 0.9166495157867433, 0.030519894810188714, 0.004782004997925106
6, 0.057633478052011876, 0.18174992034862006, 0.8855282672161411, 0.8808485636288477, 0.9960134812790495, 0.006370944945224001, 0.010043439410350064, 0.032557
951593329276, 0.5331007155772204, 0.019868402083566796, 0.9290117624814149, 0.03429012594586714, 0.04055165531874831, 0.0979613270570139, 0.3708494376533416
6, 0.005263772996877466, 0.06538537805672935, 0.006246145878278894, 0.00023554922034736125, 0.5171147407787401, 0.8324093313802556, 0.018927157105622403, 0.059
20888642780235, 0.0015135669898467736, 0.3178972546911113, 0.9401033885550046, 0.9181950637936352, 0.24099072727277512, 0.012588693851300084, 0.80083982231082
14, 0.7155478403936302, 0.035677973439124724, 0.04166456408577889, 0.04850990530708536, 0.005925588712795146, 0.067546593696552, 0.004377918092143205, 0.0066
96283822948996, 0.8823764432122138, 0.21573323705842623, 0.04121885356200099, 0.8373895031423978, 0.0034564170779443913, 0.8641679588178953, 0.00383157961166613
79, 0.0065301085397466985, 0.009467324607675537, 0.041402210539976586, 0.022411622351850853, 0.5366319044042956, 0.008691779496129145, 0.6287954774357731, 0.09
373895900730056, 0.03809431105766375, 0.9801852226553932, 0.9375013013367913, 0.08458413183104609, 0.7823134835907389, 0.019399350267034928, 0.033977929836693
66, 0.01307187064626153, 0.004163884438875327, 0.012139062968719519, 0.7373334282401842, 0.008535996242115715, 0.3665590787115734, 0.014096605578407587, 0.025
381306064780847, 0.6439357444238846, 0.03693760816724338, 0.09617083556940423, 0.006823839075358012, 0.2082559925455328, 0.025277238901981115, 0.5653988779966
552, 0.9205868984762204, 0.02094339129613084, 0.05413481494309434, 0.9152077235984286, 0.8806850094678386, 0.5109881603239929, 0.023159201711443447, 0.6979043
938723505, 0.0016307473632282226, 0.017154186811187282, 0.03557663026602945, 0.023946902596287057, 0.0022722657915608766, 0.8963698100590378, 0.0284674933904
7677, 0.7800706031231566, 0.8497327839350607, 0.021116382453205487, 0.16995354366558602, 0.02333454474884013, 0.0152684230521062, 0.016915846615419766, 0.7554
714178279204, 0.13330549339244072, 0.0036350637598909384, 0.038272692301725024, 0.8278841303166204, 0.8211667217211371, 0.0060130160655642185, 0.1502621762993
3938, 0.013572066201862285, 0.01487990862468774, 0.16500089015362723, 0.009673834515673479, 0.0011592600233922337, 0.0038807482310747547, 0.000965074838795534,
0.5990877444101601, 0.9932564676312478, 0.013985845812317958, 0.9127276885519151, 0.1319287181529915, 0.14971036293571285, 0.5180186777706162, 0.9066598188025
005, 0.06969656732210415, 0.6513611359419914, 0.03388148790564795, 0.001825190302476183, 0.8891102151104239, 0.9153939760166888, 0.012506345492306748, 0.56371
71178935163, 0.04073224380053033, 0.78382795079373469, 0.9639621915152675, 0.9106857354874239, 0.10460068723975229, 0.04219295267773605, 0.912246240603363, 0.
8869816725114472, 0.10038113558353977, 0.8810818133621212, 0.00415860873388059, 0.3054677544133251, 0.19122719111715467, 0.038136983749512174, 0.0146741540726
89887, 0.3134846837836528, 0.05893551644832439, 0.8460992065482258, 0.021746654392935082, 0.2767222846511429, 0.9152773859375932, 0.002416601755752803, 0.0027
542798351156837, 0.04643254623173652, 0.006366716891215406, 0.9590998127786221, 0.021245478414767528, 0.20583503368657008, 0.7319861560756777, 0.0318260381993
9563, 0.035826047144150004, 0.026886755585613074, 0.05019931915975108, 0.011401659878814543, 0.1600804112938701, 0.0019660202022518014, 0.09153841670702809,
0.8989851192847265, 0.7513455254732313, 0.07907632373097792, 0.981095277496359, 0.7916562878750087, 0.9251754259419104, 0.20597658250289974, 0.967693431673784
2, 0.0037546283328799084, 0.5298728968763671, 0.036746773759317744, 0.01534778136489834, 0.8871424032083762, 0.019360749810780224, 0.010804444555959558, 0.0107
78960275007314, 0.9386057324603535, 0.06003416752769395, 0.13272415024276268, 0.022078768216568824, 0.9442177047396915, 0.21227192535551162, 0.009826128477741
667, 0.0132330514499037, 0.8876952448731813, 0.007957737004217298, 0.8260651596144029, 0.4759844352483603, 0.005493828555075233, 0.20344716266041377, 0.315851
95965507534, 0.9871403701683222, 0.9067551266543815, 0.011440086556955265, 0.17670241546978677, 0.0076375043562122188, 0.9484613862203669, 0.03763541866824508,
0.10389149303766163, 0.14343254930391894, 0.8821258113230699, 0.0230080983776724, 0.022458199579819104, 0.6452778883857878, 0.2269955543381045, 0.060387831793
59782, 0.03973977950674534, 0.009981121455773286, 0.5317631884913141, 0.023135437569218115, 0.005752028728514195, 0.004448346823424566, 0.005684123549974109,
0.9334539593091518, 0.12445541459445243, 0.034851719377517784, 0.11878512517990605, 0.8890974398936752, 0.6613871187527062, 0.002155361718730909, 0.0248415651
1710386, 0.006433222801387303, 0.07524221522734724, 0.8554451862726993, 0.9065341777922162, 0.14760399751175507, 0.8622749722801535, 0.011310636132740196, 0.8
948170447600673, 0.9622146410889729, 0.5193516010215815, 0.17428101213977082, 0.8439194767357363, 0.8459059600738495, 0.008382478191375872, 0.0200789292689400
36, 0.24599213020816207, 0.0030244162334840453, 0.013263046492265094, 0.014042190564353987, 0.9518891183208489, 0.012120061586774322, 0.07411487304537728, 0.9
029268028549483, 0.013203763078525594, 0.016897584629914315, 0.9683245235928053, 0.0007975520479262399, 0.021967835039204544, 0.01756316362703198, 0.923944831
5175238, 0.042274868618081336, 0.049846317025661425, 0.010002191236835893, 0.8416134186215078, 0.864423775656, 0.0026532549650931845, 0.05102197698433873, 0.0
03131039922159221, 0.850313041472486, 0.8716846824165186, 0.4412786086822296, 0.10351699430370714, 0.9196232183587024, 0.0035334217722255545, 0.03618321147642
307, 0.5806427663912433, 0.012980823460820908, 0.48966021674155225, 0.5983973980520698, 0.11600097494268943, 0.1653631157173856, 0.001327635845819174, 0.97601
70186735415, 0.15613789431108666, 0.9532657878948447, 0.012755032161757753, 0.004444890861787377, 0.012088352350468647, 0.9143051919914392, 0.0413942986314555
2, 0.002765828354813565, 0.8992980934018229, 0.01031393975987417, 0.9895901482203027, 0.03556765696384337, 0.9071146373132947, 0.9591846355909776, 0.003890057
5676323675, 0.8452267026530209, 0.9018928827902665, 0.032869141048295435, 0.0571992384523265, 0.18507598024031977, 0.009206086843662158, 0.20160350391118922,
0.06699131245382793, 0.04019860261823433, 0.021561988463000933, 0.049038475664102445, 0.008017087795046694, 0.019504427547933142, 0.012129213461018041, 0.06647
133008464381, 0.01766441544216768, 0.6334523323878094, 0.08071451935947646, 0.023438944764285517, 0.03159225999300921, 0.05174920708364413, 0.947337855981220
1, 0.815290671832673, 0.006591476942438054, 0.0009657819306343415, 0.13298301296470835, 0.8256118760367751, 0.40811987510150066, 0.027782308058782134, 0.01372
4556280730544, 0.0341393163373114, 0.02200096382620749, 0.03999942036736662, 0.22361499674939364, 0.30622220332844435, 0.0035523044769656476, 0.90097595651420
53, 0.9421585221985304, 0.0029663448505672634, 0.883632505228418, 0.49088348274834104, 0.9004728112572871, 0.4563621274998662, 0.6649914590466938, 0.004923276
4868577545, 0.1051007014958135, 0.006289540746846559, 0.23137679873779693, 0.10635793591525722, 0.9033650068167899, 0.007430106825292272, 0.04502282560153768,
0.005919406176566327, 0.21662905359838133, 0.9748427812821264, 0.024980307018034047, 0.012695896845469111, 0.38790398043983765, 0.12517934586510243, 0.0641725
0342815611, 0.10416044806242727, 0.013339458384492811, 0.3164543792401975, 0.05654619948917097, 0.02718233265970043, 0.010892943653481526, 0.0123274200014644
9, 0.030169442687219977, 0.8849448892431654, 0.05099470782307198, 0.16098733052177758, 0.07874055175932534, 0.011504080530474442, 0.9466758030796915, 0.019455
515056851436, 0.013498867002000165, 0.008643669602428142, 0.02338471173817452, 0.013319960516538745, 0.9280546571835809, 0.005358179151434211, 0.0021746540720
897936, 0.8872282396662406, 0.03042378759553967, 0.9499197571394152, 0.016187809653018952, 0.9175728488476436, 0.8971484393203376, 0.9210042203193037, 0.02028
372847601688, 0.01975449872999804, 0.10404414569873884, 0.014908736338101605, 0.8544657583869762, 0.03562298139977468, 0.04360820906189882, 0.709318952471091
7, 0.013786930062980268, 0.1867957336381786, 0.006929643611400418, 0.3802110567790626, 0.507948783701398, 0.04428398609241807, 0.9573522552259038, 0.670809372
6168366, 0.015250607835016852, 0.30871986811200813, 0.05650616936854968, 0.674411271338204, 0.7796600510242325, 0.009024976942831622, 0.00913929455070158, 0.0
31520081118525674, 0.0392908461590666, 0.03878263525277148, 0.012835422469578861, 0.8599488000758413, 0.9504408373839542, 0.007085082093750249, 0.002619772491
497284, 0.013351528883867583, 0.0038944592671321873, 0.012634325926598843, 0.014704950917206408, 0.037079434654996093, 0.14035674313496754, 0.0179034943876870
8, 0.006540237611462195, 0.001421276081724003, 0.8796245044557817, 0.021575417579724483, 0.0912404492606723, 0.003812637609558308, 0.019647117670580392, 0.059
71493605031084, 0.024742924735497978, 0.030848724225876343, 0.011438104635767875, 0.03085705375939177, 0.7838758698436921, 0.288799813597528, 0.98367013161068
32, 0.8813830413583778, 0.8737880704411254, 0.03868799735631634, 0.011796265392701177, 0.16767266030366795, 0.8431464028203274, 0.023671285440176888, 0.896325
3216082507, 0.025679078772543808, 0.9299345518993907, 0.0819624509581768, 0.025298920597021848, 0.5943828031902237, 0.03522422263113214, 0.013726617427364822,
0.003425073615616315, 0.35117174925963834, 0.05158145707433851, 0.8130221478548042, 0.05060690915214314, 0.9615179020912212, 0.09920498279554311, 0.0226766398
93045254, 0.2554300685757442, 0.006817175923268844, 0.08591775701317345, 0.0026809869685376925, 0.9856877278007821, 0.1783741451159112, 0.006680501438531301,
0.0037118513646032403, 0.00816687861083103, 0.14745797306362615, 0.29741923544897997, 0.004064232034985723, 0.9359169589042367, 0.006091212560470382, 0.016710
196107522897, 0.025478674368288403, 0.020048409920860256, 0.9642024677435483, 0.08419739977307612, 0.014127874479255599, 0.6009519513196845, 0.014160282034100
78, 0.004581904326785543, 0.09244052071536543, 0.008933881548571724, 0.011007721461224685, 0.4613970286657202, 0.6901600720965295, 0.0031193401638770486, 0.02
0660253239829528, 0.005771860729190929, 0.9269014670290131, 0.005086111483081345, 0.9695798811835467, 0.16145665742492143, 0.026885399743094156, 0.01252573819
5562254, 0.9146122028506933, 0.010530643706610882, 0.6084446903121608, 0.02387938029232971, 0.31740082293097716, 0.0391085135396463, 0.26661880153708695, 0.17
425509772695602, 0.5486176388159295, 0.05336729987117464, 0.2539698656856387, 0.007030603547993043, 0.020052611583447462, 0.06650590890298948, 0.0228116622181
14204, 0.021482974752282717, 0.017621106275502025, 0.015350157595153478, 0.03296992384373664, 0.008972533397667244, 0.9230413903771499, 0.1706559260900858, 0.
9277899845556469, 0.007219408124422209, 0.7316874534858469, 0.16668905275020796, 0.012084121690021336, 0.03570852349550301, 0.9049256462338913, 0.0173850980326
25577, 0.0834229060689596, 0.010886175177360301, 0.9398837542766053, 0.022734684024799397, 0.17842440421891617, 0.5507601504354224, 0.07769274617935601, 0.004
7019916132491416, 0.05748617486891963, 0.002560733409678765, 0.09833851804314793, 0.9158921733509944, 0.6732704113203809, 0.8469927194411899, 0.01090614097514
6368, 0.7711380524411919, 0.1719186563902077, 0.014597140651016981, 0.9179392140254585, 0.05479196324803622, 0.010708973534896154, 0.006179754881761183, 0.0485
9755002280579, 0.2771923319999139, 0.0021365423676156414, 0.037243562185695495, 0.0335624413734091, 0.0038476466633211963, 0.006530485159845803, 0.01125916054
4640873, 0.21517370767695482, 0.05316813192692454, 0.8861796749244987, 0.029321836739419092, 0.0022745912677754475, 0.28830994307037755, 0.01276947531501263
3, 0.8585358719075178, 0.06762661784621822, 0.9357945877186702, 0.9155072612844136, 0.29810960395890934, 0.08305086021353059, 0.7927617956593797, 0.0166720249
42644626, 0.023792887288902935, 0.8990633662028513, 0.9692372349822839, 0.025010287196864888, 0.9166863038274449, 0.05815154193610908, 0.009018983245797007,
0.7023122882474042, 0.0056859607415298465, 0.043592097798880014, 0.2663285270148487, 0.05810440753248776, 0.130101638020477, 0.03511727117097223, 0.0762030911
8819343, 0.18714282542714866, 0.9194360159114736, 0.03359403649757745, 0.8511469688312853, 0.022856672881365222, 0.750834235553276, 0.015382652336346642, 0.00
7165546074266068, 0.036930214742453636, 0.0026838805160135817, 0.0038359006302866442, 0.915356666841629, 0.218150172100777, 0.9049433299100443, 0.011750920123
961695, 0.010023893262591023, 0.8934257140182094, 0.8029618278603653, 0.747008421060196, 0.008593764979204284, 0.02996907921746544, 0.6155475337375084, 0.0211
763212254634, 0.009912740467532235, 0.018439362842612113, 0.017305169232395442, 0.9416052796939323, 0.1380082655917911, 0.001135950516825506, 0.00171369031100
65087, 0.014486499688404614, 0.059424139241715586, 0.8448774929936464, 0.8892990916898387, 0.006144656387913832, 0.017550240315264573, 0.009326198918709344,
0.029899133580421726, 0.02686076338921893, 0.908417515526719, 0.6400207083563656, 0.7876153433511827, 0.8974987644250132, 0.9147832886549552, 0.00152207829170
49998, 0.008119529452721543, 0.331211301989774, 0.7488180716581544, 0.9265947367580925, 0.7267677588254423, 0.9419262202713251, 0.05981088461270495, 0.2329567
1961178513, 0.008886082162941669, 0.012342076450067095, 0.020892375838459782, 0.935858635614539, 0.6579054254683824, 0.012396186542942172, 0.0208188031041492
3, 0.017801901127801948, 0.18780667442840111, 0.010177641076990616, 0.03890445720080009, 0.01765232728499047, 0.06502926706122208, 0.8765215092402678, 0.739761
5608370037, 0.27486109605606335, 0.9479916955738351, 0.02465342565787844, 0.8326859736589576, 0.7182272855530825, 0.08382999603575807, 0.03239958073538869, 0.
7920483261970865, 0.015507175951933736, 0.03731840018645909, 0.004867466086170625, 0.8708423819500741, 0.39499504286763615, 0.06401099276450813, 0.5949630776
308059, 0.004586335828173447, 0.0020985934856826687, 0.8895962287797359, 0.9417913173302764, 0.1605388898605259, 0.7887246643407232, 0.012308527131255012, 0.3
6009235228617786, 0.9290619271464052, 0.03686087112240463, 0.09781701470486533, 0.010370883483724181, 0.012372824162919294, 0.008499231181105899, 0.9296902269
361884, 0.002352179054386855, 0.03369684659425547, 0.15996007729890913, 0.7174928956269603, 0.10198160804423961, 0.15105760466582585, 0.9243855253336528, 0.05
244429944016827, 0.0096314493894438126, 0.0010247674931590648, 0.012180549695337682, 0.004254121318351341, 0.05169412886062728, 0.9025840647018418, 0.015292623
047793746, 0.007881661448321134, 0.05240888344598458, 0.005533226261239993, 0.020240145633752954, 0.02190723643038961, 0.008071051616611193, 0.0031513751239677
606, 0.6278077919469351, 0.0024500158939048267, 0.03401920687864047, 0.14249622772201398, 0.9138636847109306, 0.03638388841973371, 0.035625890473509246, 0.018
992197687039497, 0.6765867889284505, 0.9724358089239492, 0.014870964002812733, 0.05897775348723221, 0.03367497843227707, 0.014532753904144283, 0.69377121973299
8, 0.012736123255336124, 0.01009224163565739, 0.3035347837302824, 0.05578103359547396, 0.8954802968402146, 0.910114260422436, 0.0694316841148188, 0.0605033395
87433674, 0.9426315419850522, 0.6341560374389287, 0.0038182005299674, 0.013924738876956846, 0.6867944702993711, 0.8164955755077037, 0.015647430124640396, 0.01
5977454902225567, 0.22176045138882952, 0.009427555776346734, 0.004210367974297324, 0.22227465585690614, 0.03549594089390186, 0.9837665080048047, 0.91408664652
06904, 0.12512095233978346, 0.017153944913095555, 0.8876968617244396, 0.0014278756671650837, 0.04338894251039712, 0.8113457986503221, 0.948326335831227, 0.090
31417640910575, 0.9287223732067597, 0.010451443911545273, 0.7849842561487705, 0.9200655467385225, 0.0014121331287695353, 0.03203763123237309, 0.00812630529064
4613, 0.917070461739644, 0.7177726670716597, 0.05621505473951251, 0.9710198252120391, 0.020972439953739683, 0.0011301808995184545, 0.0766409345246784, 0.04443
282902876281, 0.019746747861353, 0.9177768550977199, 0.8716673130464043, 0.818357112447188, 0.020868074579247112, 0.03612567908680196, 0.7868447637057217, 0.
960943622792097, 0.23817467992663066, 0.9298253260959451, 0.0253083877486775, 0.5831317126668926, 0.006853071416702282, 0.0388177059385466, 0.904142066999919
3, 0.007908025004451653, 0.10310326213799531, 0.0307817708064836, 0.04377526665088593, 0.005943186887325765, 0.35187812635262256, 0.012365600463797675, 0.0447
0435221465508, 0.05375496155892317, 0.005483950176306995, 0.01685209417995656, 0.5301854499695864, 0.9196219791673442, 0.016475663537916786, 0.0106482313711324
146, 0.8302216767295716, 0.006357235206929645, 0.03548663383354546, 0.5570264515066163, 0.4340720495071271, 0.006906605817688499, 0.012857362206669167, 0.029
240981561905167, 0.934676538123794, 0.3331579367580997, 0.24760123285058952, 0.029528506495365918, 0.004797641580616669, 0.0027215327293875755, 0.005714108363
574435, 0.008305600063265694, 0.013100271192047068, 0.9669014607094286, 0.9196246014250599, 0.005397675295964074, 0.007267347284055392, 0.7083580770204444, 0.9
285201107744 63, 0.6605455077287288, 0.8418044139951117, 0.006161550128980891, 0.00485785092685666, 0.025163857010942897, 0.39312376612330774, 0.20682434178872
866, 0.007013593450977939, 0.017321707140522898, 0.9259300162002735, 0.012806002828201657, 0.0538891350620549, 0.02858261430883169 6, 0.5356806814285673, 0.022
094083833176632, 0.027290191242150993, 0.9025115434158038, 0.09779135862893151, 0.6972968658763775, 0.8604371371158968, 0.9141138279260204, 0.4942707446901395,
0.014470775990579167, 0.11712324204337933, 0.9274402542578638, 0.049917387124594 86, 0.002917098550891854, 0.0444989181034 00086, 0.9325735277368814, 0.01626612
4978093374, 0.08257386753154072, 0.008171623671903232, 0.03141899759129818, 0.0799377316948801, 0.017593747771009145, 0.030328581887067 41, 0.20481241021888 75,
0.7083434199189101, 0.0023517877401 75269, 0.17287885215496823, 0.03974443891231009, 0.9932047416692786, 0.5257483612462376, 0.9873965917078832, 0.851358093539
3782, 0.17350346993042481, 0.6291549626714529, 0.03047927616291394, 0.8972729995884771, 0.152905719511103898, 0.04169936808547081, 0.8963953841027106, 0.015041
642465423038, 0.8248730331924743, 0.21594127766676025, 0.21609580409652762, 0.8201854772354967, 0.8661741157159882, 0.8989912314716288, 0.005434565152639596,
0.021670930117685536, 0.032604355335719186, 0.2185048491379299, 0.042088633200409992, 0.8580943734055915, 0.014807543265967128, 0.004003860434128941, 0.0119434
58043127908, 0.943578915360485, 0.31818394672342853, 0.05598653375767826 5, 0.016981112310441058, 0.014117699756059539, 0.28984024779568 84, 0.85849092580455 54,
0.22724881145595824, 0.007255466942573851, 0.07615545479677402, 0.002877487422903117, 0.7575088931260698, 0.9684576104696321, 0.058873640186337 45, 0.022646349
93884578, 0.021348349993282997, 0.019503674408582 48, 0.018110053179802334, 0.02383904761334543, 0.41856373807235 13, 0.01851174763730869, 0.011001208928550523]
```

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