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: $K(x_i,x_q)$ = $exp(-\gamma|$

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

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]:

```
import numpy as np
import pandas as pd
import numpy as np

from sklearn.svm import SVC
from tqdm.notebook import tqdm
from matplotlib import pyplot as plt

from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
```

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)
```

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 [3]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
# https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

# 1. Split the data into Xtrain(60), Xcv(20), Xtest(20)
x_tr, x_test, y_tr, y_test = train_test_split(X, y, test_size = 0.2,
                                              stratify = y, random_state = 2)
x_train, x_cv, y_train , y_cv = train_test_split(x_tr, y_tr, test_size = 0.25,
                                             stratify = y_tr, random_state = 2)

print('X_Train shape',x_train.shape )
print('X_Test shape',x_test.shape )
print('X_Cv shape',x_cv.shape )

# 2. Train SVC(gamma=0.001,C=100.) on the (Xtrain, ytrain)

gamma_ = 0.001
svc_clf = SVC(gamma = gamma_, C = 100)
svc_clf.fit(x_train, y_train)
```

```
X_Train shape (3000, 5)
X_Test shape (1000, 5)
X_Cv shape (1000, 5)
```

Out[3]:
```
SVC(C=100, gamma=0.001)
```

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*

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:
$K(x_i, x_q)$
=
$exp(-\gamma|$

In [4]:
```
# https://towardsdatascience.com/radial-basis-function-rbf-kernel-the-go-to-kernel-acf0d22c798a
# https://towardsdatascience.com/support-vector-machines-learning-data-science-step-by-step-f2a569d90f76
# https://github.com/eriklindernoren/ML-From-Scratch/blob/master/
#         mlfromscratch/supervised_learning/support_vector_machine.py

def decision_function(x, intercept, coeff, support_vector, gamma ):

#     RBF kernel is defined as: K(xi,xq) = exp(−γ||xi−xq||2)
    kernel = np.zeros((x.shape[0], support_vector.shape[0]))
    for id_x, pt in enumerate(x):
        for id_y, vec in enumerate(support_vector):
            k_value = np.exp(-gamma * np.sum((pt- vec)**2))
            kernel[id_x][id_y] = k_value

#     yi*αi*K(xi,xq)) + intercept
    custom_decision = np.sum(coeff * kernel, axis = 1) + intercept

    return custom_decision
```

In [5]:
```
fcv = decision_function(x_cv, svc_clf.intercept_, svc_clf.dual_coef_,
                                                  svc_clf.support_vectors_, gamma_ )
```

Comparing Custom implementation and Native SVC implementation

In [6]:
```
print(f'Shape at Native SVC implementation\t : {fcv.shape}')
print(f'Shape at Custom implementation\t\t : {fcv.shape}')
```

```
Shape at Native SVC implementation       : (1000,)
Shape at Custom implementation           : (1000,)
```

In [7]:
```
# https://numpy.org/doc/stable/reference/generated/numpy.around.html

result_ = all(np.round(svc_clf.decision_function(x_cv), 7) == np.round(fcv, 7))
print(f"'True' if all values are same, other-wise 'False'\t: {result_}")

n_ = 180

print(f'\nComparison of 1st {n_} values :\
            \n{np.round(svc_clf.decision_function(x_cv)[:n_], 7) == np.round(fcv[:n_], 7)}')

fcv[:20]
```

```
'True' if all values are same, other-wise 'False'       : True

Comparison of 1st 180 values :
[ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True]
```

Out[7]:
```
array([-3.2630509 ,  1.84661142, -3.92647752, -1.67949529, -2.14324374,
       -3.05654121, -3.31298576, -1.56365973, -3.76088812, -3.70935314,
        1.71459596, -2.87275849, -2.57540088, -3.01488941, -3.46797186,
       -0.73400885, -1.33553508,  0.24029827, -1.53850604, -1.13269479])
```

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

> 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} , $P(Y = 1|$ where f_{test} `= decision_function(` X_{test} `)` , W and b will be learned as metioned in the above step

In [8]:

```python
# https://www.delftstack.com/howto/numpy/numpy-count-zero/
# https://numpy.org/doc/stable/reference/generated/numpy.count_nonzero.html

n_pos = np.count_nonzero(y_cv)
print(f'Positive counts : {n_pos}')

n_neg = len(y_cv) - n_pos
print(f'Negative counts : {n_neg}')

calibrated_y_pos = (n_pos + 1) / (n_pos + 2)
calibrated_y_neg = 1 / (n_neg + 2)

print(f"\nCalibrated 'y' positives : {round(calibrated_y_pos, 4)}")
print(f"Calibrated 'y' negatives : {round(calibrated_y_neg, 4)}")
```

```
Positive counts : 303
Negative counts : 697

Calibrated 'y' positives : 0.9967
Calibrated 'y' negatives : 0.0014
```

In [9]:

```
# changing y_cv values

updated_y_cv = []

for p in y_cv:
    if p == 1:
        updated_y_cv.append(calibrated_y_pos)
    else:
        updated_y_cv.append(calibrated_y_neg)
```

In [10]:

```
def sigmoid(w, x, b):
    z = np.dot(w, x) + b
    return (1 / (1 + np.exp(-z)))

def log_loss(w, b, X, Y):

    N = len(X)
    sum_log = 0

    for i in range(N):
        sum_log += Y[i] * np.log10(sigmoid(w, X[i], b)) + \
                            (1 - Y[i] * np.log10(1 - sigmoid(w, X[i], b)))

    return (-1 * sum_log / N)
```

$dw^{(t)} = x_n(y_n - \sigma$

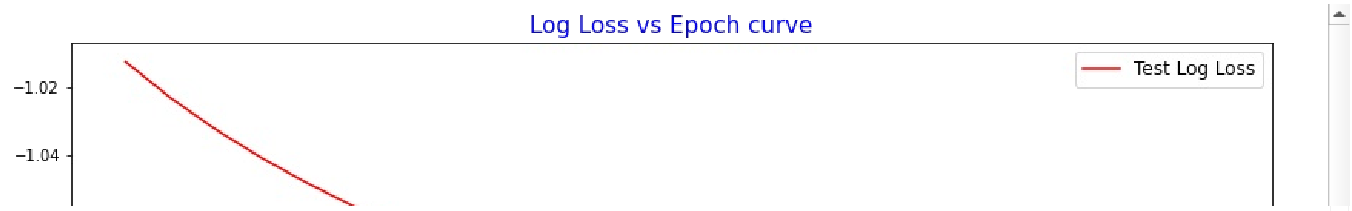
$db^{(t)} = y_n - \sigma$

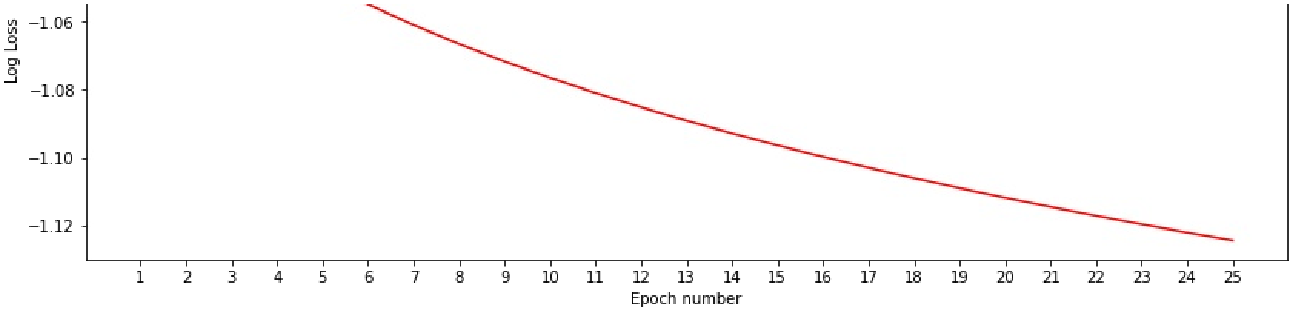
In [11]:

```
N = len(fcv)
w = np.zeros_like(fcv[0])
b = 0

eta0 = 0.0001
alpha = 0.0001
epochs = 25

cv_loss = []

y = updated_y_cv

for epoch in tqdm(range(epochs)):
    for j in range(N):

        dw = fcv[j] * (y[j] -sigmoid(w, fcv[j], b)) - (( alpha / N) * w)
        w = w + (eta0 * dw)

        db = y[j] - sigmoid(w, fcv[j], b)
        b = b + (eta0 * db)

    loss = log_loss(w, b, fcv, y)
    cv_loss.append(loss)
```

In [12]:

```
epoch = np.arange(epochs) + 1
plt.figure(figsize = (14,5))

plt.plot(epoch,cv_loss, c = 'r',label='Test Log Loss')
plt.xticks(epoch)
plt.title('Log Loss vs Epoch curve', fontsize = 15, c = 'b')
plt.xlabel("Epoch number")
plt.ylabel('Log Loss')
plt.legend(fontsize = 12)
plt.show()
```





In [13]:
```
print(f"Optimized 'w' : {w}\nOptimized 'b' : {b}")
```

```
Optimized 'w' : 0.8930445253560461
Optimized 'b' : -0.10631255856481144
```

In [14]:
```
f_test = decision_function(x_test, svc_clf.intercept_, svc_clf.dual_coef_,
                           svc_clf.support_vectors_, gamma_ )
```

In [15]:
```
probas = sigmoid(w, f_test, b)

print('Probability scores corresponding to X_test :\n')

for i in range(0, len(probas),2):
    print(f'{i+1} : {round(probas[i], 7)}\t\t{i+2} : {round(probas[i+1],7)}')
```

```
Probability scores corresponding to X_test :

1 : 0.416688		2 : 0.0192823
3 : 0.6265521		4 : 0.1740453
5 : 0.4585416		6 : 0.7987241
7 : 0.0617171		8 : 0.7652264
9 : 0.871245		10 : 0.1903718
11 : 0.0079452		12 : 0.7165572
13 : 0.0643453		14 : 0.3935694
15 : 0.7214826		16 : 0.1935141
17 : 0.6342424		18 : 0.0937576
19 : 0.0621943		20 : 0.0686149
21 : 0.0448959		22 : 0.1413751
23 : 0.0852212		24 : 0.770394
25 : 0.1629849		26 : 0.0488053
27 : 0.0578007		28 : 0.0376143
29 : 0.064047		30 : 0.0454591
31 : 0.0207641		32 : 0.064936
33 : 0.0758397		34 : 0.8291005
35 : 0.8623348		36 : 0.0904771
37 : 0.1696878		38 : 0.0975311
39 : 0.0382135		40 : 0.044395
41 : 0.1693937		42 : 0.5898652
43 : 0.1018816		44 : 0.0368827
45 : 0.104257		46 : 0.0859714
47 : 0.3477797		48 : 0.7336815
49 : 0.8367853		50 : 0.0797621
51 : 0.2386073		52 : 0.1564763
53 : 0.8171789		54 : 0.8573245
55 : 0.0380321		56 : 0.8705612
57 : 0.2174629		58 : 0.4266366
59 : 0.8853907		60 : 0.1092686
61 : 0.8311197		62 : 0.2666246
63 : 0.3009637		64 : 0.1766895
65 : 0.0788019		66 : 0.0847253
67 : 0.1115031		68 : 0.0676126
69 : 0.0324499		70 : 0.125091
71 : 0.3198224		72 : 0.0518647
73 : 0.1236042		74 : 0.2230124
75 : 0.5899637		76 : 0.0276419
77 : 0.0426465		78 : 0.1300676
79 : 0.7915717		80 : 0.7617386
81 : 0.4540425		82 : 0.8094607
83 : 0.0343215		84 : 0.1980267
85 : 0.0606067		86 : 0.5910683
87 : 0.8732757		88 : 0.05082
89 : 0.7115154		90 : 0.755998
91 : 0.0889483		92 : 0.7122565
93 : 0.0609179		94 : 0.0544056
95 : 0.2367443		96 : 0.4145269
97 : 0.0594076		98 : 0.0210503
99 : 0.059749		100 : 0.8436247
```

101 : 0.1343854	102 : 0.7550503
103 : 0.0540898	104 : 0.0901298
105 : 0.1096368	106 : 0.0785074
107 : 0.1486808	108 : 0.4764864
109 : 0.4085436	110 : 0.1953201
111 : 0.075819	112 : 0.073052
113 : 0.0392954	114 : 0.1400272
115 : 0.7427841	116 : 0.8550135
117 : 0.9399219	118 : 0.012057
119 : 0.0391053	120 : 0.7365985
121 : 0.8528265	122 : 0.8269949
123 : 0.7262778	124 : 0.8600211
125 : 0.1163345	126 : 0.0769295
127 : 0.0651601	128 : 0.7799522
129 : 0.1966909	130 : 0.3055396
131 : 0.1402181	132 : 0.0252088
133 : 0.1100535	134 : 0.0803496
135 : 0.0510981	136 : 0.0533917
137 : 0.1610303	138 : 0.1569692
139 : 0.1218936	140 : 0.8551379
141 : 0.0650496	142 : 0.8723668
143 : 0.6210178	144 : 0.6530553
145 : 0.1522633	146 : 0.4719895
147 : 0.5099206	148 : 0.0365077
149 : 0.0656871	150 : 0.0369196
151 : 0.1610482	152 : 0.0778573
153 : 0.4067471	154 : 0.0623637
155 : 0.1086447	156 : 0.8008036
157 : 0.0659766	158 : 0.0871709
159 : 0.0393925	160 : 0.0605108
161 : 0.1074912	162 : 0.7997576
163 : 0.8557173	164 : 0.8582824
165 : 0.8758467	166 : 0.0981632
167 : 0.8381491	168 : 0.8607228
169 : 0.4777841	170 : 0.0112592
171 : 0.7351668	172 : 0.5664729
173 : 0.7024387	174 : 0.0511285
175 : 0.1408591	176 : 0.6587198
177 : 0.1210523	178 : 0.0506065
179 : 0.2494934	180 : 0.8773323
181 : 0.3381833	182 : 0.0475103
183 : 0.3755945	184 : 0.0862817
185 : 0.1891955	186 : 0.6972803
187 : 0.7725712	188 : 0.4481452
189 : 0.1845524	190 : 0.0313618
191 : 0.3254541	192 : 0.0267576
193 : 0.7606647	194 : 0.5132033
195 : 0.0921943	196 : 0.1059181
197 : 0.0882784	198 : 0.0881966
199 : 0.9068643	200 : 0.6576557
201 : 0.089333	202 : 0.8267504
203 : 0.0362481	204 : 0.3077873
205 : 0.0901378	206 : 0.069036
207 : 0.9021697	208 : 0.4793356
209 : 0.0756659	210 : 0.1545746
211 : 0.3487285	212 : 0.6262737
213 : 0.0600545	214 : 0.117989
215 : 0.2160393	216 : 0.107066
217 : 0.673065	218 : 0.8250375
219 : 0.1227067	220 : 0.0769926
221 : 0.0762965	222 : 0.4506047
223 : 0.0084127	224 : 0.0855742
225 : 0.0471799	226 : 0.0570877
227 : 0.3389228	228 : 0.1578069
229 : 0.1106524	230 : 0.7240479
231 : 0.06831	232 : 0.4783849
233 : 0.0611297	234 : 0.8214976
235 : 0.0679524	236 : 0.8092712
237 : 0.0865967	238 : 0.0567997
239 : 0.063947	240 : 0.4029957
241 : 0.0718513	242 : 0.3575394
243 : 0.2534878	244 : 0.161247
245 : 0.2139698	246 : 0.088618
247 : 0.4119216	248 : 0.0692101
249 : 0.8512786	250 : 0.84613
251 : 0.087699	252 : 0.0303256
253 : 0.0688627	254 : 0.0947317
255 : 0.8201501	256 : 0.7530388
257 : 0.0772716	258 : 0.0532142
259 : 0.1254838	260 : 0.0121106
261 : 0.0453512	262 : 0.4207776
263 : 0.1383361	264 : 0.1932395
265 : 0.0546979	266 : 0.2919101
267 : 0.0369849	268 : 0.8750545
269 : 0.3875346	270 : 0.7420072
271 : 0.0520587	272 : 0.0114451
273 : 0.3545062	274 : 0.042666
275 : 0.086917	276 : 0.7886887
277 : 0.2409069	278 : 0.0676356

279 : 0.0774708	280 : 0.9059802
281 : 0.0948023	282 : 0.8279012
283 : 0.7720168	284 : 0.0988732
285 : 0.7706867	286 : 0.7616277
287 : 0.012342	288 : 0.0605001
289 : 0.8427907	290 : 0.875421
291 : 0.3391097	292 : 0.0565028
293 : 0.858073	294 : 0.0645544
295 : 0.1191104	296 : 0.0488483
297 : 0.9636247	298 : 0.0634631
299 : 0.0783448	300 : 0.3734956
301 : 0.1027216	302 : 0.0981074
303 : 0.8918149	304 : 0.0942919
305 : 0.0832435	306 : 0.0483018
307 : 0.1907172	308 : 0.1192871
309 : 0.1602707	310 : 0.2590424
311 : 0.6048052	312 : 0.0368844
313 : 0.1466082	314 : 0.165265
315 : 0.0623107	316 : 0.0481374
317 : 0.0687914	318 : 0.1119139
319 : 0.0571003	320 : 0.0575627
321 : 0.0954562	322 : 0.0600771
323 : 0.0170656	324 : 0.2557295
325 : 0.7663889	326 : 0.0399808
327 : 0.1207285	328 : 0.0414935
329 : 0.7927975	330 : 0.1013444
331 : 0.0888234	332 : 0.0379436
333 : 0.8377125	334 : 0.0371123
335 : 0.1217173	336 : 0.8522484
337 : 0.15146	338 : 0.450862
339 : 0.1588894	340 : 0.1165866
341 : 0.0056141	342 : 0.0312244
343 : 0.4832094	344 : 0.0247187
345 : 0.0682064	346 : 0.0511413
347 : 0.0280433	348 : 0.1060122
349 : 0.1577527	350 : 0.9692459
351 : 0.1626888	352 : 0.3272057
353 : 0.6678571	354 : 0.029795
355 : 0.9254973	356 : 0.573004
357 : 0.854126	358 : 0.7751382
359 : 0.0809962	360 : 0.1596381
361 : 0.1778343	362 : 0.0487713
363 : 0.3259757	364 : 0.7399087
365 : 0.249633	366 : 0.3959965
367 : 0.8522754	368 : 0.5382158
369 : 0.0560989	370 : 0.0742318
371 : 0.082082	372 : 0.7640316
373 : 0.1189966	374 : 0.1057015
375 : 0.1809427	376 : 0.1928139
377 : 0.7014539	378 : 0.0537643
379 : 0.7502586	380 : 0.0644244
381 : 0.9404672	382 : 0.2410723
383 : 0.1136267	384 : 0.2624372
385 : 0.0423627	386 : 0.075319
387 : 0.2564764	388 : 0.0621742
389 : 0.091701	390 : 0.6932281
391 : 0.350643	392 : 0.2141274
393 : 0.0782691	394 : 0.3642119
395 : 0.8513287	396 : 0.0329225
397 : 0.8462035	398 : 0.621531
399 : 0.0092261	400 : 0.1611008
401 : 0.0883043	402 : 0.0196461
403 : 0.2993223	404 : 0.3631868
405 : 0.0683269	406 : 0.0530368
407 : 0.1792909	408 : 0.8034957
409 : 0.8191163	410 : 0.0156676
411 : 0.037354	412 : 0.8012401
413 : 0.0685328	414 : 0.0915571
415 : 0.0175171	416 : 0.2292579
417 : 0.044707	418 : 0.7709473
419 : 0.7983599	420 : 0.0210672
421 : 0.0915917	422 : 0.1083084
423 : 0.6552429	424 : 0.1633837
425 : 0.3357778	426 : 0.1531584
427 : 0.0188269	428 : 0.9264379
429 : 0.1364715	430 : 0.7700998
431 : 0.1815137	432 : 0.0461501
433 : 0.399628	434 : 0.0338289
435 : 0.033568	436 : 0.0877496
437 : 0.1599175	438 : 0.0589157
439 : 0.7609705	440 : 0.0315466
441 : 0.0145542	442 : 0.0899644
443 : 0.2993672	444 : 0.1808722
445 : 0.1255313	446 : 0.5329778
447 : 0.0255323	448 : 0.0303091
449 : 0.4807048	450 : 0.1442192
451 : 0.8252128	452 : 0.0627233
453 : 0.0350227	454 : 0.0267506
455 : 0.0416508	456 : 0.0839061

457 : 0.1707024   458 : 0.0556844
459 : 0.4034751   460 : 0.7540448
461 : 0.8778688   462 : 0.8061194
463 : 0.7334191   464 : 0.8420981
465 : 0.8634283   466 : 0.2018453
467 : 0.1637257   468 : 0.1988014
469 : 0.0503412   470 : 0.5519404
471 : 0.0581242   472 : 0.3183535
473 : 0.7703993   474 : 0.8626518
475 : 0.804896   476 : 0.8788521
477 : 0.4436455   478 : 0.0425607
479 : 0.1730096   480 : 0.0881281
481 : 0.0382338   482 : 0.1537914
483 : 0.8257967   484 : 0.8206953
485 : 0.0188914   486 : 0.6365404
487 : 0.0589705   488 : 0.2209558
489 : 0.1543469   490 : 0.7689988
491 : 0.813572   492 : 0.9693028
493 : 0.0501319   494 : 0.2809195
495 : 0.5643946   496 : 0.131213
497 : 0.8156907   498 : 0.734367
499 : 0.0336235   500 : 0.1459593
501 : 0.1605853   502 : 0.045962
503 : 0.8959871   504 : 0.9696941
505 : 0.1913141   506 : 0.8425999
507 : 0.1179682   508 : 0.0161442
509 : 0.0660731   510 : 0.0351022
511 : 0.0627577   512 : 0.0783969
513 : 0.55854   514 : 0.7207107
515 : 0.1279868   516 : 0.9128115
517 : 0.6911037   518 : 0.9628305
519 : 0.3701673   520 : 0.0095611
521 : 0.1824892   522 : 0.1237805
523 : 0.0617975   524 : 0.2534452
525 : 0.687532   526 : 0.0856188
527 : 0.1198625   528 : 0.1472026
529 : 0.690738   530 : 0.7290946
531 : 0.1773304   532 : 0.1178039
533 : 0.4239684   534 : 0.079347
535 : 0.8768933   536 : 0.0511461
537 : 0.0383306   538 : 0.0308885
539 : 0.2169533   540 : 0.7100097
541 : 0.1551956   542 : 0.0768029
543 : 0.7033314   544 : 0.0665976
545 : 0.7680308   546 : 0.141737
547 : 0.0788629   548 : 0.0297703
549 : 0.399309   550 : 0.1529858
551 : 0.072081   552 : 0.125232
553 : 0.1018504   554 : 0.1707511
555 : 0.0488   556 : 0.773282
557 : 0.6271032   558 : 0.8349432
559 : 0.3203027   560 : 0.1534749
561 : 0.065269   562 : 0.0190987
563 : 0.0331665   564 : 0.1070715
565 : 0.0423627   566 : 0.815368
567 : 0.0438991   568 : 0.0776047
569 : 0.0604015   570 : 0.1608437
571 : 0.6379664   572 : 0.4370184
573 : 0.1040204   574 : 0.9274507
575 : 0.0822877   576 : 0.2879096
577 : 0.0537757   578 : 0.048794
579 : 0.0173333   580 : 0.0733035
581 : 0.1059967   582 : 0.0162774
583 : 0.6546181   584 : 0.7860414
585 : 0.1325872   586 : 0.101496
587 : 0.1179551   588 : 0.8544457
589 : 0.5738542   590 : 0.838728
591 : 0.0905817   592 : 0.967677
593 : 0.0440363   594 : 0.7367456
595 : 0.0614868   596 : 0.7641042
597 : 0.0358395   598 : 0.2658927
599 : 0.7420825   600 : 0.1126494
601 : 0.4000981   602 : 0.042261
603 : 0.0812175   604 : 0.1179823
605 : 0.8035067   606 : 0.2751423
607 : 0.0516236   608 : 0.0594112
609 : 0.1250367   610 : 0.2122422
611 : 0.1021967   612 : 0.6112197
613 : 0.2439678   614 : 0.1272043
615 : 0.0943877   616 : 0.3405616
617 : 0.0283602   618 : 0.7878071
619 : 0.0531915   620 : 0.8578062
621 : 0.1018483   622 : 0.0553666
623 : 0.7966144   624 : 0.2026882
625 : 0.890362   626 : 0.0177971
627 : 0.0404268   628 : 0.8949636
629 : 0.0519455   630 : 0.1432757
631 : 0.5845768   632 : 0.750315
633 : 0.3640526   634 : 0.4941701

635 : 0.6350222
636 : 0.077297
637 : 0.8946953
638 : 0.0367986
639 : 0.0644214
640 : 0.0715172
641 : 0.0236867
642 : 0.6818078
643 : 0.0664861
644 : 0.2738161
645 : 0.0853835
646 : 0.0819009
647 : 0.041282
648 : 0.1300267
649 : 0.0936085
650 : 0.1221218
651 : 0.056023
652 : 0.4390754
653 : 0.7514349
654 : 0.0738501
655 : 0.795578
656 : 0.0727184
657 : 0.1525093
658 : 0.0587678
659 : 0.7523929
660 : 0.1904737
661 : 0.1643414
662 : 0.0844471
663 : 0.1032974
664 : 0.2336578
665 : 0.1034452
666 : 0.2078958
667 : 0.0760673
668 : 0.0144975
669 : 0.0670822
670 : 0.9054419
671 : 0.1141862
672 : 0.732409
673 : 0.8928158
674 : 0.0415476
675 : 0.1150014
676 : 0.1010949
677 : 0.0222538
678 : 0.6473896
679 : 0.2660882
680 : 0.0757506
681 : 0.7614262
682 : 0.0451101
683 : 0.87787
684 : 0.0219126
685 : 0.2132107
686 : 0.1777663
687 : 0.889775
688 : 0.2663
689 : 0.2797105
690 : 0.1070868
691 : 0.0247296
692 : 0.0616184
693 : 0.2035038
694 : 0.0789244
695 : 0.6963869
696 : 0.1643096
697 : 0.7837796
698 : 0.8005332
699 : 0.0832146
700 : 0.816511
701 : 0.0301341
702 : 0.7809375
703 : 0.2960902
704 : 0.8368717
705 : 0.0198522
706 : 0.1852637
707 : 0.087278
708 : 0.0318634
709 : 0.0289221
710 : 0.0591253
711 : 0.0435458
712 : 0.8299324
713 : 0.0417511
714 : 0.7823835
715 : 0.1081027
716 : 0.0375323
717 : 0.7904461
718 : 0.0680384
719 : 0.054145
720 : 0.0276548
721 : 0.130388
722 : 0.5175327
723 : 0.131203
724 : 0.0335861
725 : 0.0255795
726 : 0.8614914
727 : 0.0297619
728 : 0.6181718
729 : 0.0291766
730 : 0.0359278
731 : 0.8466925
732 : 0.7361048
733 : 0.0213922
734 : 0.8612928
735 : 0.0533212
736 : 0.0101461
737 : 0.0942334
738 : 0.53485
739 : 0.2855977
740 : 0.1549274
741 : 0.0376507
742 : 0.0504642
743 : 0.7760163
744 : 0.7895039
745 : 0.5824131
746 : 0.1241387
747 : 0.2028762
748 : 0.0268847
749 : 0.0908678
750 : 0.0372297
751 : 0.1514957
752 : 0.7124393
753 : 0.8002739
754 : 0.1014527
755 : 0.0302275
756 : 0.0496272
757 : 0.6464639
758 : 0.5917737
759 : 0.8327225
760 : 0.0517724
761 : 0.0522881
762 : 0.6822165
763 : 0.1524498
764 : 0.9062716
765 : 0.1361774
766 : 0.0631379
767 : 0.0430701
768 : 0.1143372
769 : 0.077699
770 : 0.9169214
771 : 0.0432671
772 : 0.4521099
773 : 0.1623578
774 : 0.2189171
775 : 0.2111922
776 : 0.0169555
777 : 0.0226291
778 : 0.8178337
779 : 0.8221195
780 : 0.9343268
781 : 0.2680564
782 : 0.8152278
783 : 0.188072
784 : 0.0391432
785 : 0.060801
786 : 0.1022189
787 : 0.8454154
788 : 0.0540629
789 : 0.0904179
790 : 0.1310639
791 : 0.8427371
792 : 0.0704222
793 : 0.8888192
794 : 0.8830725
795 : 0.0419272
796 : 0.0339343
797 : 0.0520878
798 : 0.0428591
799 : 0.0203122
800 : 0.0491558
801 : 0.8804063
802 : 0.0378555
803 : 0.0428426
804 : 0.8546994
805 : 0.0496954
806 : 0.1365084
807 : 0.0485144
808 : 0.082903
809 : 0.8045618
810 : 0.1486564
811 : 0.0422332
812 : 0.3001433

813 : 0.0938954	814 : 0.0459262
815 : 0.1892287	816 : 0.7947175
817 : 0.2965024	818 : 0.0701891
819 : 0.0975662	820 : 0.0966868
821 : 0.1376837	822 : 0.5975689
823 : 0.0142017	824 : 0.0368412
825 : 0.8394893	826 : 0.4394743
827 : 0.7757441	828 : 0.5950147
829 : 0.9434304	830 : 0.1485824
831 : 0.1461005	832 : 0.7104747
833 : 0.2404953	834 : 0.0344447
835 : 0.0198453	836 : 0.0550875
837 : 0.0630081	838 : 0.093362
839 : 0.2691874	840 : 0.053199
841 : 0.0815102	842 : 0.2768791
843 : 0.366625	844 : 0.0201481
845 : 0.1030155	846 : 0.0723966
847 : 0.6870788	848 : 0.1094895
849 : 0.7961429	850 : 0.8164461
851 : 0.1073864	852 : 0.7810504
853 : 0.0764873	854 : 0.7191076
855 : 0.69069	856 : 0.0936894
857 : 0.0496897	858 : 0.4314621
859 : 0.6522448	860 : 0.2488364
861 : 0.77822	862 : 0.5157698
863 : 0.1037493	864 : 0.5500519
865 : 0.5130052	866 : 0.0564945
867 : 0.098334	868 : 0.0491772
869 : 0.153361	870 : 0.0502346
871 : 0.0679408	872 : 0.7212709
873 : 0.0448638	874 : 0.1989078
875 : 0.1089928	876 : 0.1922292
877 : 0.8892273	878 : 0.3156098
879 : 0.0188765	880 : 0.0967628
881 : 0.0606198	882 : 0.3863181
883 : 0.2460286	884 : 0.0501304
885 : 0.0406828	886 : 0.0527619
887 : 0.7066607	888 : 0.0449113
889 : 0.0886032	890 : 0.5610581
891 : 0.8518798	892 : 0.6696197
893 : 0.4965377	894 : 0.8326462
895 : 0.0401742	896 : 0.2192309
897 : 0.7809972	898 : 0.7770518
899 : 0.064357	900 : 0.0420659
901 : 0.07914	902 : 0.0794905
903 : 0.5072591	904 : 0.6403751
905 : 0.2316229	906 : 0.796806
907 : 0.1480889	908 : 0.1031244
909 : 0.8137061	910 : 0.7604287
911 : 0.7942385	912 : 0.817155
913 : 0.1623854	914 : 0.5843713
915 : 0.1477117	916 : 0.0650371
917 : 0.5284937	918 : 0.1973547
919 : 0.6106552	920 : 0.0376721
921 : 0.0905451	922 : 0.0425649
923 : 0.0165778	924 : 0.6183185
925 : 0.0881024	926 : 0.0777
927 : 0.0423625	928 : 0.275026
929 : 0.5122969	930 : 0.0496239
931 : 0.7880252	932 : 0.1799675
933 : 0.9067496	934 : 0.662789
935 : 0.0766545	936 : 0.2327331
937 : 0.4111024	938 : 0.7579886
939 : 0.0298829	940 : 0.0799507
941 : 0.0846479	942 : 0.7610764
943 : 0.1414355	944 : 0.360887
945 : 0.0399165	946 : 0.4998434
947 : 0.789158	948 : 0.8630445
949 : 0.1945163	950 : 0.1044701
951 : 0.0621772	952 : 0.1162623
953 : 0.6734552	954 : 0.8307558
955 : 0.1043717	956 : 0.0291928
957 : 0.153192	958 : 0.1485263
959 : 0.1643407	960 : 0.0793981
961 : 0.1293424	962 : 0.0387043
963 : 0.8533068	964 : 0.0991921
965 : 0.0519285	966 : 0.1439571
967 : 0.0986599	968 : 0.1019222
969 : 0.1333638	970 : 0.0419041
971 : 0.7554275	972 : 0.0528733
973 : 0.2136461	974 : 0.0114781
975 : 0.8046484	976 : 0.06125
977 : 0.1641421	978 : 0.525577
979 : 0.1471527	980 : 0.3234363
981 : 0.0943487	982 : 0.5129799
983 : 0.4086777	984 : 0.8158244
985 : 0.8355591	986 : 0.4393871
987 : 0.0336269	988 : 0.0207137
989 : 0.210839	990 : 0.9138068

```
991 : 0.6061039        992 : 0.6096628
993 : 0.3086643        994 : 0.8554815
995 : 0.0680337        996 : 0.0294318
997 : 0.0302578        998 : 0.8583048
999 : 0.5757179        1000 : 0.8828848
```

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