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 import tqdm
from matplotlib import pyplot as plt

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


plt.style.use('fivethirtyeight')
```

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 $(\sum_{i=1}^{\text{all the support vectors}}(y_i\alpha_iK(x_i, x_q)) + intercept)$, 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
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

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).

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]:

```
# 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_train)
print(f'Positive counts : {n_pos}')

n_neg = len(y_train) - 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 : 908
Negative counts : 2092

Calibrated 'y' positives : 0.9989
Calibrated 'y' negatives : 0.0005
```

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)
```

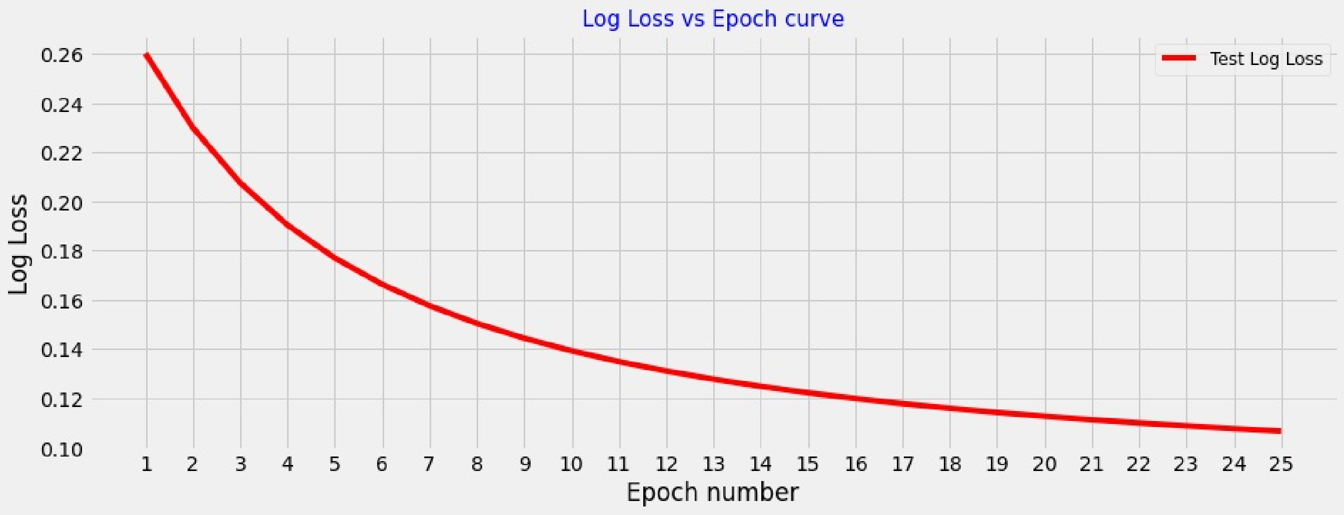
```
100%|████████████████████████████████████████████| 25/25 [00:00<00:00, 64.59it/s]
```

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' : 0.8964154031692937
Optimized 'b' : -0.1059103765770649
```

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 : 0.4165747           2 : 0.0190188
3 : 0.6271969           4 : 0.1733171
5 : 0.4585853           6 : 0.799688
7 : 0.061171            8 : 0.7661707
9 : 0.872142            10 : 0.1896545
11 : 0.007809           12 : 0.7174306
13 : 0.0637876          14 : 0.3933717
15 : 0.7223651          16 : 0.1927996
17 : 0.6349105          18 : 0.0931003
19 : 0.0616461          20 : 0.0680393
21 : 0.0444377          22 : 0.1406477
23 : 0.0845875          24 : 0.771343
25 : 0.1622533          26 : 0.0483245
27 : 0.0572729          28 : 0.0372025
29 : 0.0634906          30 : 0.0449975
31 : 0.0204865          32 : 0.0643757
33 : 0.0752368          34 : 0.8300568
35 : 0.8632498          36 : 0.0898285
37 : 0.1689578          38 : 0.0968646
39 : 0.0377977          40 : 0.0439399
41 : 0.1686636          42 : 0.5903914
43 : 0.1012054          44 : 0.0364759
45 : 0.1035759          46 : 0.0853356
47 : 0.3474237          48 : 0.734585
49 : 0.8377354          50 : 0.0791457
51 : 0.2379581          52 : 0.1557444
53 : 0.8181413          54 : 0.8582482
55 : 0.0376176          56 : 0.8714598
57 : 0.2167779          58 : 0.4265603
59 : 0.8862525          60 : 0.1085779
61 : 0.8320746          62 : 0.2660353
63 : 0.3004638          64 : 0.1759626
65 : 0.0781887          66 : 0.0840931
67 : 0.1108084          68 : 0.0670411
69 : 0.0320749          70 : 0.1243772
71 : 0.3193777          72 : 0.0513671
73 : 0.1228921          74 : 0.222336
75 : 0.5904902          76 : 0.0273043
77 : 0.0422021          78 : 0.1293486
79 : 0.7925336          80 : 0.7626793
```

81 : 0.4540692 82 : 0.8104249
83 : 0.0339327 84 : 0.1973169
85 : 0.0600657 86 : 0.5915985
87 : 0.8741682 88 : 0.050328
89 : 0.712379 90 : 0.7569324
91 : 0.0883039 92 : 0.7131215
93 : 0.0603754 94 : 0.0538948
95 : 0.2360916 96 : 0.4144056
97 : 0.0588722 98 : 0.0207701
99 : 0.059212 100 : 0.8445676
101 : 0.1336626 102 : 0.7559836
103 : 0.0535805 104 : 0.0894821
105 : 0.1089454 106 : 0.0778952
107 : 0.1479503 108 : 0.4765982
109 : 0.4084003 110 : 0.1946074
111 : 0.0752162 112 : 0.0724592
113 : 0.0388725 114 : 0.1393005
115 : 0.7437013 116 : 0.8559409
117 : 0.9405504 118 : 0.01187
119 : 0.0386836 120 : 0.7375065
121 : 0.8537572 122 : 0.8279527
123 : 0.7271689 124 : 0.8609402
125 : 0.1156322 126 : 0.0763227
127 : 0.0645989 128 : 0.7809083
129 : 0.1959797 130 : 0.3050527
131 : 0.1394914 132 : 0.0248915
133 : 0.1093614 134 : 0.0797313
135 : 0.0506046 136 : 0.052886
137 : 0.1602985 138 : 0.1562373
139 : 0.1211836 140 : 0.8560652
141 : 0.0644889 142 : 0.8732613
143 : 0.6216455 144 : 0.6537779
145 : 0.1515319 146 : 0.4720842
147 : 0.5101588 148 : 0.0361035
149 : 0.0651236 150 : 0.0365127
151 : 0.1603164 152 : 0.0772473
153 : 0.4065972 154 : 0.0618147
155 : 0.1079551 156 : 0.8017678
157 : 0.0654119 158 : 0.0865315
159 : 0.0389689 160 : 0.0599703
161 : 0.1068038 162 : 0.8007216
163 : 0.8566436 164 : 0.8592045
165 : 0.8767332 166 : 0.0974953
167 : 0.8390979 168 : 0.8616407
169 : 0.4779008 170 : 0.0110815
171 : 0.7360725 172 : 0.5669181
173 : 0.7032836 174 : 0.0506348
175 : 0.140132 176 : 0.6594581
177 : 0.1203434 178 : 0.0501157
179 : 0.2488659 180 : 0.8782152
181 : 0.337796 182 : 0.0470368
183 : 0.375333 184 : 0.0856449
185 : 0.1884772 186 : 0.6981141
187 : 0.773522 188 : 0.4481496
189 : 0.1838304 190 : 0.0309949
191 : 0.3250267 192 : 0.0264273
193 : 0.7616043 194 : 0.5134539
195 : 0.0915411 196 : 0.1052337
197 : 0.0876359 198 : 0.0875543
199 : 0.9076548 200 : 0.658391
201 : 0.0886875 202 : 0.8277083
203 : 0.0358457 204 : 0.3073069
205 : 0.0894901 206 : 0.0684587
207 : 0.9029777 208 : 0.4794582
209 : 0.0750636 210 : 0.1538429
211 : 0.3483755 212 : 0.6269176
213 : 0.0595161 214 : 0.1172843
215 : 0.2153521 216 : 0.1063794
217 : 0.6738411 218 : 0.8259964
219 : 0.1219957 220 : 0.0763856
221 : 0.0756919 222 : 0.4506184
223 : 0.0082704 224 : 0.0849395
225 : 0.0467083 226 : 0.0565634
227 : 0.3385379 228 : 0.157075
229 : 0.1099592 230 : 0.724935
231 : 0.0677357 232 : 0.4785039
233 : 0.0605863 234 : 0.8224583
235 : 0.0673795 236 : 0.8102354
237 : 0.085959 238 : 0.0562769
239 : 0.063391 240 : 0.4028321
241 : 0.0712631 242 : 0.3572159
243 : 0.2528688 244 : 0.1605152
245 : 0.2132797 246 : 0.0879745
247 : 0.4117907 248 : 0.0686321
249 : 0.8522116 250 : 0.8470699
251 : 0.0870581 252 : 0.0299667
253 : 0.0682861 254 : 0.094072
255 : 0.8211114 256 : 0.7539697
257 : 0.0766636 258 : 0.0527095

259 : 0.1247695	260 : 0.0119229
261 : 0.0448903	262 : 0.4206795
263 : 0.1376104	264 : 0.1925248
265 : 0.0541856	266 : 0.2913851
267 : 0.0365775	268 : 0.8759429
269 : 0.3873152	270 : 0.7429233
271 : 0.0515601	272 : 0.0112652
273 : 0.3541725	274 : 0.0422214
275 : 0.0862784	276 : 0.7896495
277 : 0.2402622	278 : 0.067064
279 : 0.0768621	280 : 0.906774
281 : 0.0941425	282 : 0.8288583
283 : 0.7729672	284 : 0.0982036
285 : 0.771636	286 : 0.7625684
287 : 0.0121517	288 : 0.0599596
289 : 0.8437346	290 : 0.8763085
291 : 0.3387254	292 : 0.0559814
293 : 0.8589955	294 : 0.0639958
295 : 0.1184041	296 : 0.0483672
297 : 0.9640836	298 : 0.0629093
299 : 0.0777332	300 : 0.3732268
301 : 0.1020436	302 : 0.0974396
303 : 0.8926577	304 : 0.0936333
305 : 0.0826159	306 : 0.0478238
307 : 0.1900002	308 : 0.1185805
309 : 0.1595389	310 : 0.2584356
311 : 0.605381	312 : 0.0364777
313 : 0.1458784	314 : 0.1645338
315 : 0.0617619	316 : 0.0476603
317 : 0.0682152	318 : 0.1112186
319 : 0.056576	320 : 0.0570361
321 : 0.0947948	322 : 0.0595385
323 : 0.0168241	324 : 0.2551154
325 : 0.7673343	326 : 0.0395534
327 : 0.12002	328 : 0.0410563
329 : 0.7937599	330 : 0.1006693
331 : 0.0881793	332 : 0.0375297
333 : 0.8386617	334 : 0.036704
335 : 0.1210075	336 : 0.8531801
337 : 0.1507288	338 : 0.4508767
339 : 0.1581574	340 : 0.1158839
341 : 0.0055104	342 : 0.0308586
343 : 0.4833467	344 : 0.0244056
345 : 0.0676325	346 : 0.0506476
347 : 0.0277025	348 : 0.1053276
349 : 0.1570208	350 : 0.9696555
351 : 0.1619572	352 : 0.3267837
353 : 0.6686197	354 : 0.0294403
355 : 0.9262053	356 : 0.5734721
357 : 0.8550548	358 : 0.776091
359 : 0.0803757	360 : 0.1589062
361 : 0.177108	362 : 0.0482906
363 : 0.3255499	364 : 0.7408217
365 : 0.2490059	366 : 0.3958075
367 : 0.853207	368 : 0.5385592
369 : 0.0555795	370 : 0.0736347
371 : 0.081458	372 : 0.7649747
373 : 0.1182905	374 : 0.1050175
375 : 0.1802182	376 : 0.1920988
377 : 0.7022967	378 : 0.0532568
379 : 0.7511859	380 : 0.0638663
381 : 0.9410924	382 : 0.2404279
383 : 0.1129285	384 : 0.2618382
385 : 0.0419201	386 : 0.0747179
387 : 0.255864	388 : 0.0616261
389 : 0.0910491	390 : 0.6940527
391 : 0.3502964	392 : 0.2134376
393 : 0.0776577	394 : 0.363911
395 : 0.8522617	396 : 0.0325439
397 : 0.8471433	398 : 0.6221602
399 : 0.0090734	400 : 0.1603689
401 : 0.0876617	402 : 0.0193792
403 : 0.2988178	404 : 0.3628824
405 : 0.0677525	406 : 0.052533
407 : 0.1785654	408 : 0.8044601
409 : 0.820078	410 : 0.0154406
411 : 0.0369441	412 : 0.8022043
413 : 0.0679575	414 : 0.0909056
415 : 0.0172711	416 : 0.2285919
417 : 0.04425	418 : 0.7718968
419 : 0.7993237	420 : 0.0207868
421 : 0.0909401	422 : 0.1076195
423 : 0.6559716	424 : 0.1626522
425 : 0.3353829	426 : 0.1524268
427 : 0.0185679	428 : 0.9271412
429 : 0.1357472	430 : 0.7710485
431 : 0.1807895	432 : 0.0456845
433 : 0.3994522	434 : 0.0334438
435 : 0.0331847	436 : 0.0871085

437 : 0.1591856
438 : 0.0583826
439 : 0.7619104
440 : 0.0311784
441 : 0.0143391
442 : 0.0893172
443 : 0.2988628
444 : 0.1801477
445 : 0.124817
446 : 0.5333019
447 : 0.0252122
448 : 0.0299504
449 : 0.4808326
450 : 0.1434904
451 : 0.8261716
452 : 0.0621727
453 : 0.0346289
454 : 0.0264203
455 : 0.0412126
456 : 0.0832765
457 : 0.1699727
458 : 0.0551671
459 : 0.4033133
460 : 0.7549769
461 : 0.8787504
462 : 0.8070838
463 : 0.7343221
464 : 0.8430428
465 : 0.8643413
466 : 0.2011396
467 : 0.1629943
468 : 0.1980924
469 : 0.0498518
470 : 0.5523337
471 : 0.0575949
472 : 0.3179044
473 : 0.7713483
474 : 0.8635662
475 : 0.8058604
476 : 0.8797313
477 : 0.4436329
478 : 0.0421168
479 : 0.1722809
480 : 0.087486
481 : 0.0378179
482 : 0.1530598
483 : 0.8267552
484 : 0.8216564
485 : 0.0186317
486 : 0.6372154
487 : 0.0584371
488 : 0.2202761
489 : 0.1536152
490 : 0.7699465
491 : 0.8145354
492 : 0.9697118
493 : 0.0496437
494 : 0.2803654
495 : 0.5648325
496 : 0.1304929
497 : 0.8166536
498 : 0.7352715
499 : 0.0332399
500 : 0.1452297
501 : 0.1598534
502 : 0.0454975
503 : 0.8968166
504 : 0.9700995
505 : 0.1905976
506 : 0.843544
507 : 0.1172635
508 : 0.0159122
509 : 0.065508
510 : 0.0347079
511 : 0.062207
512 : 0.0777851
513 : 0.5589571
514 : 0.7215919
515 : 0.1272698
516 : 0.9135779
517 : 0.6919235
518 : 0.963296
519 : 0.3698869
520 : 0.0094041
521 : 0.1817658
522 : 0.1230682
523 : 0.0612511
524 : 0.2528261
525 : 0.6883435
526 : 0.084984
527 : 0.1191552
528 : 0.1464726
529 : 0.691557
530 : 0.7299905
531 : 0.1766038
532 : 0.1170994
533 : 0.4238821
534 : 0.078732
535 : 0.8777773
536 : 0.0506523
537 : 0.0379141
538 : 0.0305253
539 : 0.2162676
540 : 0.7108703
541 : 0.1544638
542 : 0.0761965
543 : 0.7041782
544 : 0.0660303
545 : 0.7689777
546 : 0.1410094
547 : 0.0782495
548 : 0.0294157
549 : 0.399132
550 : 0.1522543
551 : 0.0714919
552 : 0.124518
553 : 0.1011743
554 : 0.1700215
555 : 0.0483192
556 : 0.7742334
557 : 0.6277497
558 : 0.835895
559 : 0.3198595
560 : 0.1527433
561 : 0.0647073
562 : 0.018837
563 : 0.0327862
564 : 0.1063848
565 : 0.04192
566 : 0.816331
567 : 0.0434469
568 : 0.0769956
569 : 0.0598615
570 : 0.1601118
571 : 0.6386456
572 : 0.4369809
573 : 0.1033398
574 : 0.9281488
575 : 0.0816631
576 : 0.2873739
577 : 0.0532681
578 : 0.0483132
579 : 0.0170891
580 : 0.0727098
581 : 0.1053121
582 : 0.016044
583 : 0.6553451
584 : 0.787001
585 : 0.1318659
586 : 0.1008206
587 : 0.1172505
588 : 0.855374
589 : 0.5743253
590 : 0.8396762
591 : 0.0899328
592 : 0.9681008
593 : 0.0435833
594 : 0.7376538
595 : 0.0609418
596 : 0.7650473
597 : 0.03544
598 : 0.2653017
599 : 0.7429987
600 : 0.1119529
601 : 0.399924
602 : 0.041819
603 : 0.0805964
604 : 0.1172775
605 : 0.8044711
606 : 0.2745737
607 : 0.0511273
608 : 0.0588758
609 : 0.1243229
610 : 0.2115497
611 : 0.1015199
612 : 0.6118163
613 : 0.2433291
614 : 0.1264882

615 : 0.0937288
616 : 0.340182
617 : 0.0280168
618 : 0.7887675
619 : 0.052687
620 : 0.8587291
621 : 0.1011721
622 : 0.0548508
623 : 0.7975779
624 : 0.2019835
625 : 0.8912092
626 : 0.0175482
627 : 0.0399965
628 : 0.8957964
629 : 0.0514475
630 : 0.1425473
631 : 0.5850849
632 : 0.7512424
633 : 0.3637512
634 : 0.4943489
635 : 0.6356927
636 : 0.0766889
637 : 0.895529
638 : 0.0363925
639 : 0.0638634
640 : 0.0709302
641 : 0.0233826
642 : 0.6826057
643 : 0.0659193
644 : 0.2732442
645 : 0.0847494
646 : 0.0812775
647 : 0.0408462
648 : 0.1293077
649 : 0.0929516
650 : 0.1214115
651 : 0.055504
652 : 0.4390456
653 : 0.7523637
654 : 0.0732544
655 : 0.7965412
656 : 0.0721268
657 : 0.1517779
658 : 0.0582354
659 : 0.753323
660 : 0.1897565
661 : 0.16361
662 : 0.0838158
663 : 0.1026182
664 : 0.2329995
665 : 0.1027657
666 : 0.2071975
667 : 0.0754635
668 : 0.014283
669 : 0.0665129
670 : 0.9062378
671 : 0.1134871
672 : 0.7333104
673 : 0.8936555
674 : 0.0411101
675 : 0.1143011
676 : 0.1004204
677 : 0.0219624
678 : 0.6480962
679 : 0.2654976
680 : 0.0751479
681 : 0.7623666
682 : 0.0446507
683 : 0.8787516
684 : 0.0216244
685 : 0.2125196
686 : 0.17704
687 : 0.8906241
688 : 0.26571
689 : 0.2791534
690 : 0.1064001
691 : 0.0244164
692 : 0.0610728
693 : 0.2028001
694 : 0.0783108
695 : 0.6972186
696 : 0.1635782
697 : 0.7847381
698 : 0.8014974
699 : 0.0825871
700 : 0.8174737
701 : 0.0297767
702 : 0.7818943
703 : 0.2955767
704 : 0.8378217
705 : 0.0195833
706 : 0.1845422
707 : 0.0866383
708 : 0.0314928
709 : 0.0285742
710 : 0.0585912
711 : 0.0430958
712 : 0.8308881
713 : 0.0413123
714 : 0.7833412
715 : 0.1074141
716 : 0.0371211
717 : 0.7914077
718 : 0.0674652
719 : 0.0536355
720 : 0.0273171
721 : 0.1296687
722 : 0.5177994
723 : 0.1304829
724 : 0.0332028
725 : 0.025259
726 : 0.8624079
727 : 0.0294074
728 : 0.6187905
729 : 0.0288267
730 : 0.0355277
731 : 0.8476317
732 : 0.7370121
733 : 0.0211088
734 : 0.8622097
735 : 0.0528159
736 : 0.0099818
737 : 0.093575
738 : 0.535181
739 : 0.2850558
740 : 0.1541956
741 : 0.0372388
742 : 0.0499742
743 : 0.7769698
744 : 0.7904651
745 : 0.5829138
746 : 0.123426
747 : 0.2021718
748 : 0.0265533
749 : 0.0902181
750 : 0.0368206
751 : 0.1507644
752 : 0.7133047
753 : 0.801238
754 : 0.1007774
755 : 0.0298694
756 : 0.0491418
757 : 0.6471679
758 : 0.5923063
759 : 0.8336761
760 : 0.0512753
761 : 0.0517883
762 : 0.6830154
763 : 0.1517183
764 : 0.9070643
765 : 0.1354533
766 : 0.0625855
767 : 0.042623
768 : 0.1136379
769 : 0.0770896
770 : 0.91767
771 : 0.0428188
772 : 0.4521292
773 : 0.1616261
774 : 0.2182342
775 : 0.2104983
776 : 0.0167152
777 : 0.0223343
778 : 0.818796
779 : 0.82308
780 : 0.934988
781 : 0.2674705
782 : 0.8161909
783 : 0.1873528
784 : 0.0387212
785 : 0.0602591
786 : 0.101542
787 : 0.8463562
788 : 0.0535538
789 : 0.0897694
790 : 0.1303439
791 : 0.8436811
792 : 0.0698394

793 : 0.8896711	794 : 0.8839406
795 : 0.0414873	796 : 0.0335484
797 : 0.051589	798 : 0.0424134
799 : 0.0200389	800 : 0.048673
801 : 0.8812815	802 : 0.0374421
803 : 0.0423969	804 : 0.8556273
805 : 0.0492096	806 : 0.135784
807 : 0.0480351	808 : 0.0822764
809 : 0.8055262	810 : 0.1479259
811 : 0.0417913	812 : 0.2996411
813 : 0.0932378	814 : 0.0454619
815 : 0.1885104	816 : 0.7956804
817 : 0.29599	818 : 0.0696073
819 : 0.0968996	820 : 0.0960223
821 : 0.1369585	822 : 0.5981209
823 : 0.0139904	824 : 0.0364348
825 : 0.8404368	826 : 0.439446
827 : 0.7766974	828 : 0.5955581
829 : 0.944037	830 : 0.1478519
831 : 0.1453709	832 : 0.7113362
833 : 0.2398498	834 : 0.0340551
835 : 0.0195764	836 : 0.0545732
837 : 0.0624563	838 : 0.0927057
839 : 0.2686042	840 : 0.0526944
841 : 0.0808881	842 : 0.2763148
843 : 0.3663324	844 : 0.0198763
845 : 0.1023369	846 : 0.0718063
847 : 0.6878892	848 : 0.1087984
849 : 0.7971062	850 : 0.8174088
851 : 0.1066992	852 : 0.7820072
853 : 0.0758821	854 : 0.7199858
855 : 0.6915088	856 : 0.0930323
857 : 0.0492039	858 : 0.4314037
859 : 0.6529652	860 : 0.2482077
861 : 0.779175	862 : 0.5160299
863 : 0.1030692	864 : 0.5504384
865 : 0.513255	866 : 0.0559732
867 : 0.0976657	868 : 0.0486942
869 : 0.1526295	870 : 0.0497458
871 : 0.067368	872 : 0.722153
873 : 0.0444058	874 : 0.1981989
875 : 0.1083025	876 : 0.1915136
877 : 0.8900779	878 : 0.3151525
879 : 0.018617	880 : 0.0960981
881 : 0.0600787	882 : 0.3860945
883 : 0.2453941	884 : 0.0496422
885 : 0.0402508	886 : 0.0522596
887 : 0.7075144	888 : 0.0444531
889 : 0.0879598	890 : 0.5614841
891 : 0.852812	892 : 0.6703869
893 : 0.4967255	894 : 0.8335999
895 : 0.0397455	896 : 0.2185486
897 : 0.781954	898 : 0.778006
899 : 0.0637992	900 : 0.0416251
901 : 0.0785257	902 : 0.078875
903 : 0.5074873	904 : 0.6410614
905 : 0.230961	906 : 0.7977694
907 : 0.1473585	908 : 0.1024456
909 : 0.8146695	910 : 0.761368
911 : 0.7952013	912 : 0.8181175
913 : 0.1616537	914 : 0.5848788
915 : 0.1469815	916 : 0.0644765
917 : 0.5288012	918 : 0.1966441
919 : 0.61125	920 : 0.03726
921 : 0.0898963	922 : 0.0421209
923 : 0.0163413	924 : 0.6189377
925 : 0.0874603	926 : 0.0770906
927 : 0.0419198	928 : 0.2744571
929 : 0.512544	930 : 0.0491385
931 : 0.7889858	932 : 0.1792424
933 : 0.9075405	934 : 0.6635382
935 : 0.0760487	936 : 0.2320731
937 : 0.4109685	938 : 0.7589252
939 : 0.0295275	940 : 0.0793336
941 : 0.084016	942 : 0.7620164
943 : 0.140708	944 : 0.3605749
945 : 0.0394895	946 : 0.5000437
947 : 0.790119	948 : 0.8639582
949 : 0.1938028	950 : 0.1037885
951 : 0.0616291	952 : 0.1155601
953 : 0.6742322	954 : 0.831711
955 : 0.1036903	956 : 0.0288428
957 : 0.1524605	958 : 0.1477958
959 : 0.1636094	960 : 0.0787829
961 : 0.1286241	962 : 0.0382853
963 : 0.8542369	964 : 0.0985218
965 : 0.0514305	966 : 0.1432284
967 : 0.0979908	968 : 0.1012459
969 : 0.1326418	970 : 0.0414643

```
971 : 0.7563613          972 : 0.0523704
973 : 0.2129555          974 : 0.0112978
975 : 0.8056128          976 : 0.060706
977 : 0.1634107          978 : 0.5258737
979 : 0.1464227          980 : 0.3230027
981 : 0.09369            982 : 0.5132296
983 : 0.4085349          984 : 0.8167872
985 : 0.8365103          986 : 0.4393584
987 : 0.0332432          988 : 0.0204366
989 : 0.2101446          990 : 0.914569
991 : 0.6066839          992 : 0.6102544
993 : 0.3081865          994 : 0.8564082
995 : 0.0674605          996 : 0.0290799
997 : 0.0298995          998 : 0.8592269
999 : 0.5761955          1000 : 0.8837534
```

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