

Challenge: +1.5

CSE-S12 HW-5

Name: Venkata Subba Narasa Bharath, Meadam

SBU ID: 112672986

Q1)

a) $y_i = x + z_i$ (x is constant)

$$z_i = y_i - x,$$

Given $z_i \sim N(0, \sigma^2)$

\Rightarrow AS x is constant, $y_i - x$ will

also be a $N(0, \sigma^2)$.

$$\therefore (y_i - x) \sim N(0, \sigma^2)$$

$$\text{Normal distribution, } p(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$(M=0, \sigma=1) \Rightarrow \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

$$\text{Likelihood, } L(y) = \prod_{i=1}^m e^{-\frac{(y_i - x)^2}{2}}$$

Taking log on both sides, we get,

$$\log(L(y)) = \sum_{i=1}^m -\frac{(y_i - x)^2}{2}$$

For maximum, equate derivative to 0.

$$\frac{\partial}{\partial y} (\log(L(y))) = 0$$

$$\Rightarrow \frac{\partial}{\partial y} \left(\sum_{i=1}^m -\frac{(y_i - x)^2}{2} \right) = 0$$

$$\sum_{i=1}^m -2 \frac{(y_i - x)}{2} = 0$$

$$-\left(\sum_{i=1}^m y_i - mx \right) = 0$$

$$x_{MLE} = \frac{\sum_{i=1}^m y_i}{m}$$

$$\text{Bias} = E[\hat{x}] - x$$

$$E[\hat{x}] = E\left[\frac{1}{m} \sum_{i=1}^m y_i\right]$$

$$= \frac{1}{m} E\left[\sum_{i=1}^m y_i\right]$$

$$= \frac{1}{m} \times m \times E[y_i]$$

$$= E[y_i] \quad \left\{ \begin{array}{l} y_i = x + z_i \\ z_i \sim N(0, 1) \end{array} \right. \rightarrow \begin{array}{l} \text{shifted} \\ \text{standard} \\ \text{normal} \end{array}$$

$$= x$$

$$\boxed{\text{Bias} = x - x = 0}$$

$$\text{Variance}, \hat{\sigma}^2$$

It can also be written as

$$\hat{\sigma}^2(\hat{x}) = \sigma^2\left(\frac{1}{m} \sum_{i=1}^m y_i\right)$$

$$= \frac{1}{m^2} \sigma^2\left(\sum_{i=1}^m y_i\right)$$

$$= \frac{1}{m^2} \times m \times 1 = \frac{1}{m}$$

as $m \rightarrow \infty$, variance = 0

Bias: 0

Variance: $\frac{1}{m}$

as $m \rightarrow \infty$

Bias: 0

Variance: 0

$$(b) \text{ minimize } \frac{1}{m} \sum_{i=1}^m (y_i - x)^2 + \frac{\rho}{2} (x - \bar{x})^2$$

for x_{MAP} , take derivative and equate to 0.

$$\Rightarrow \frac{1}{m} * 2 * \sum_{i=1}^m (y_i - x) (-1) + \frac{\rho}{2} * 2 * (x - \bar{x}) (1) = 0$$

$$\Rightarrow \frac{2}{m} \sum_{i=1}^m (y_i - x) = \rho (x - \bar{x})$$

$$\Rightarrow \frac{2}{m} \sum_{i=1}^m y_i - \frac{2}{m} \sum_{i=1}^m x = \rho x - \rho \bar{x}$$

$$\Rightarrow 2 \sum_{i=1}^m y_i - 2m x = \rho m x - \rho \bar{x} m$$

$$x_{MAP} = \frac{2 \sum_{i=1}^m y_i + \rho m \bar{x}}{m(\rho + 2)}$$

$$\text{Bias, } \Rightarrow E[\hat{x}] - x$$

$$= E\left[\frac{2 \sum_{i=1}^m y_i + \rho m \bar{x}}{m(\rho + 2)}\right] - x$$

$$= \frac{\rho m \bar{x}}{m(\rho + 2)} + \frac{2}{m(\rho + 2)} E\left\{\sum_{i=1}^m y_i\right\} - x$$

$$= \frac{\rho m \bar{x}}{m(\rho + 2)} + \frac{2}{m(\rho + 2)} \times m \times E[y_i] - x$$

$$\begin{aligned}
 &= \frac{\beta_0 + \beta_1 \bar{x}}{m(\beta+2)} + \frac{2\gamma_L}{\beta+2} - \gamma_L \\
 &= \frac{\beta_0 \bar{x} + 2\gamma_L - \beta_0 \gamma_L - 2\gamma_L}{\beta+2} \\
 &= \frac{\beta_0 (\bar{x} - \gamma_L)}{\beta+2}
 \end{aligned}$$

$$\therefore \text{Bias} = \frac{\beta_0 (\bar{x} - \gamma_L)}{\beta+2}$$

Variance ,

$$\begin{aligned}
 &= \sigma^2 \left(\frac{2\sum y_i + \beta m \bar{x}}{(\beta+2)m} \right) \\
 &= \sigma^2 \left(\frac{\beta m \bar{x}}{(\beta+2)m} \right) + \sigma^2 \left(\frac{2\sum y_i}{(\beta+2)m} \right) \\
 &= \text{O} \uparrow \text{(constant)} + \frac{4}{(\beta+2)^2 m^2} \sigma^2 (\sum y_i) \\
 &= \frac{4}{(\beta+2)^2 m^2} \quad \text{m.i.} \quad = \frac{4}{(\beta+2)^2 m}
 \end{aligned}$$

$$\text{Bias} = \frac{\beta(\bar{x} - x)}{\beta + 2}$$

as $m \rightarrow \infty$

$$\text{Bias} = \frac{\beta(\bar{x} - x)}{\beta + 2}$$

$$\text{Variance: } \frac{\gamma}{(\beta+2)^2 m}$$

$$\text{Variance: } 0$$

comparison? -0.25

$$\begin{aligned}
 (\text{c}) E[(x - \hat{x})^2] &= E\left[\left(x - E(\hat{x}) + E(\hat{x}) - \hat{x}\right)^2\right] \\
 &= (x - E(\hat{x}))^2 - 2E\left[\left(x - E(\hat{x})\right)\left(E(\hat{x}) - \hat{x}\right)\right] \\
 &\quad + E\left[\left(E(\hat{x}) - \hat{x}\right)^2\right] \\
 &= (x - E(\hat{x}))^2 - 2(x - E(\hat{x})) E\left[E(\hat{x}) - \hat{x}\right] \\
 &\quad + E\left[\left(E(\hat{x}) - \hat{x}\right)^2\right]
 \end{aligned}$$

$$E\left[E(\hat{x}) - \hat{x}\right] = 0$$

$$\begin{aligned}
 &= \underbrace{(x - E(\hat{x}))^2}_{B^2} + E\left[\left(E(\hat{x}) - \hat{x}\right)^2\right] \\
 &= B^2 + V
 \end{aligned}$$

Here, $B = \bar{x} - E(\hat{x}) \rightarrow$ estimator bias

$$V = (E(\hat{x}) - \bar{x})^2$$

$$(2) \text{ Bias}(B) = \frac{s(x - \bar{x})}{\beta + 2}, \text{ variance } (V) = \frac{4}{(\beta + 2)^2 m}$$

$$B^2 = \frac{s^2 (x - \bar{x})^2}{(\beta + 2)^2}, V = \frac{4}{m(\beta + 2)^2}$$

$$\frac{\partial}{\partial \beta} (B^2 + V) = \frac{\partial}{\partial \beta} \left(\frac{s^2 (x - \bar{x})^2}{(\beta + 2)^2} + \frac{4}{m(\beta + 2)^2} \right)$$

↑
Result
from IC

$$\Rightarrow \frac{(\beta + 2)^2 2s(x - \bar{x})^2 - 2(\beta + 2)s^2 (\bar{x} - \bar{\bar{x}})^2}{(\beta + 2)^4} + \frac{-4m^2(\beta + 2)}{m^2(\beta + 2)^4}$$

$$\Rightarrow \frac{4s(x - \bar{x})^2}{(\beta + 2)^3} - \frac{4}{m} \left(\frac{2}{(\beta + 2)^3} \right) = 0$$

$$\Rightarrow \cancel{\frac{4s^2}{(\beta + 2)^3}} = \frac{4}{m} \left(\cancel{\frac{2}{(\beta + 2)^3}} \right)$$

$$s^2 = \frac{2}{m \Delta 2}$$

```
from sklearn.metrics import f1_score
from sklearn import tree
from tabulate import tabulate

In [2]: #load data, in sparse format
def load_data(filename):
    f = open('dorothea/%s.data' % filename)
    I,J = [], []
    for k in range(100000):
        line = f.readline()
        if len(line) == 0:
            print ('done')
            break

        line = [int(x) for x in line.split()]
        J.extend(line)
        I.extend(k for i in range(len(line)))

    return ss.coo_matrix((np.ones(len(I)), (I,J)))

def load_labels(filename):
    f = open('dorothea/%s.labels' % filename)
    y = []
    for k in range(100000):
        line = f.readline()

        if len(line) == 0:
            print ('done')
            break
```

```

Xtrain = load_data('dorothea_train')
ytrain = load_labels('dorothea_train')

Xtest= load_data('dorothea_valid')
ytest = load_labels('dorothea_valid')

print(Xtrain.shape, Xtest.shape, Xtrain.nnz)

done
done
done
done
(800, 100001) (350, 100001) 727760

```

Q1) Sparsity

```

In [3]: xTrainNumpy = Xtrain.todense()
print(ytrain.shape, xTrainNumpy.shape, Xtrain.nnz)

(800,) (800, 100001) 727760

In [4]: ss.coo_matrix(Xtrain)

Out[4]: <800x100001 sparse matrix of type '<class 'numpy.float64'>'  

        with 727760 stored elements in COORDinate format>

In [5]: xTrainNumpy = Xtrain.todense()
sparsity = 1.0 - (Xtrain.nnz / float(xTrainNumpy.size) )
print("Number of Zero Elements: "+ str(xTrainNumpy.size - count_nonzero(xTrainNumpy)) )
print("Number of Non Zero Elements: "+ str(Xtrain.nnz))
print("Sparsity : % 0.3f" %(sparsity))

Number of Zero Elements: 79273040
Number of Non Zero Elements: 727760
Sparsity : 0.991

In [6]: def get_class_balance(y):
    uniqueValues, counts = np.unique(y, return_counts=True)
    total = np.shape(y)[0]
    positiveValues = 0
    negativeValues = 0
    for index, value in enumerate(uniqueValues):
        if value == 1:
            positiveValues = counts[index]
        else:
            negativeValues = counts[index]
    assert(positiveValues+negativeValues == total), "Something wrong please check"
    return positiveValues/total, negativeValues/total

print ('Training class balance is currently %.2f +1, %.2f -1' % get_class_balance(ytrain))
print ('Test class balance is currently %.2f +1, %.2f -1' % get_class_balance(ytest))

Training class balance is currently 0.10 +1, 0.90 -1
Test class balance is currently 0.10 +1, 0.90 -1

```

Q2 comments on class_balance and sparsity

1) dorothea dataset is highly imbalanced dataset with only 10% positives (rare events) and 90% normal events. 0.5

This is the same for both training and test sets.

2) The dataset is highly sparsed, it has 99.1 % of elements as zeros and only 0.9% non-zero elements.

3) The number of training examples in this dataset is almost of the same order as compared to covtype dataset (800 vs 468) but the number of features in dorothea dataset is 100,000 as compared to only 54 features in covtype dataset.

```

In [7]: def get_misclass(y,yhat):
    incorrect = 0
    totalSamples = len(y)
    for i in range(totalSamples):
        if y[i] != yhat[i]:
            incorrect += 1
    return incorrect / totalSamples

def get_f1(y,yhat):
    totalSamples = len(y)
    truePostives = 0
    trueNegatives = 0
    falsePostitives = 0
    falseNegatives = 0

    for i in range(totalSamples):
        if y[i] == 1 and yhat[i] == 1:
            truePostives +=1
        elif y[i] == -1 and yhat[i] == -1:
            trueNegatives +=1
        elif y[i] == -1 and yhat[i] == 1:
            falsePostitives +=1
        elif y[i] == 1 and yhat[i] == -1:
            falseNegatives +=1

    precision = truePostives/(truePostives + falsePostitives)
    recall = truePostives/(truePostives + falseNegatives)

    assert((truePostives + trueNegatives + falsePostitives + falseNegatives) == totalSamples),"something  
is wrong"
    return (2 * (precision* recall))/(precision + recall)

depth = 3
clf = tree.DecisionTreeClassifier(criterion='entropy',
                                   splitter='best',
                                   max_depth=depth,
                                   class_weight='balanced')
clf = clf.fit(Xtrain, ytrain)

print(get_misclass(ytrain, clf.predict(Xtrain)), get_f1(ytrain, clf.predict(Xtrain)))
0.04125 0.7724137931034484

In [8]: yPredTrain = clf.predict(Xtrain)
yPredTest = clf.predict(Xtest)

results = []
results.append(("train",
               get_misclass(ytrain, yPredTrain),
               1 - accuracy_score(ytrain, yPredTrain),
               get_f1(ytrain, yPredTrain),
               f1_score(ytrain, yPredTrain)))

results.append(("test",
               get_misclass(ytest, yPredTest),
               1 - accuracy_score(ytest, yPredTest),
               get_f1(ytest, yPredTest),
               f1_score(ytest, yPredTest)))

columns = ['dataset',
           'missclassification-score (custom)',
```

```
: def getF1Score(depth, results, f1, precision, recall):
```

```

        max_depth=d,
        class_weight='balanced')

clf = clf.fit(Xtrain, ytrain)
yPredTrain = clf.predict(Xtrain)
yPredTest = clf.predict(Xtest)
results.append([d,
                get_f1(ytrain, yPredTrain),
                get_f1(ytest, yPredTest),
                ])

columns = ['depth',
            'F1-score (train)',
            'F1-score (test)',]

pd.set_option("display.max_colwidth",2)
df = pd.DataFrame(results, columns=columns)

print(tabulate(df.set_index('depth'), headers='keys', tablefmt='psql'))

trainF1 = [resultsAtAGivenDepth[1] for resultsAtAGivenDepth in results]
testF1 = [resultsAtAGivenDepth[2] for resultsAtAGivenDepth in results]

plt.figure(figsize=(14,8))
x = depth # Create domain for plot
plt.plot(x, trainF1, label='Training F1 Score') # Plot training error over domain
plt.plot(x, testF1, label='Testing F1 Score') # Plot testing error over domain
plt.xlabel('Maximum Depth', fontsize=12) # Label x-axis
plt.ylabel('F1 Score', fontsize=12) # Label y-axis
plt.title('Depth Vs F1 Score(Test and Train) Without any cross-Validation', fontsize=18) # Label y-axis
plt.legend() # Show plot labels as legend
plt.show()
return df

```

In [10]: testF1ScoresWithoutValidation = getF1ScoreAndPlot(Xtrain,ytrain,Xtest,ytest)

depth	F1-score (train)	F1-score (test)
2	0.713287	0.6
3	0.772414	0.580645
4	0.824324	0.615385
5	0.872483	0.571429
6	0.906667	0.626866
7	0.934211	0.483871
8	0.91358	0.597015
9	0.974359	0.571429
10	0.975	0.575758

0.5

Maximum Depth	Training F1 Score	Testing F1 Score
2	0.72	0.60
3	0.77	0.58
4	0.82	0.62
5	0.87	0.57
6	0.91	0.63
7	0.94	0.49
8	0.93	0.60
9	0.95	0.58
10	0.95	0.58

K-fold cross-validation (k=5)

In [11]:

```

import random
randomIndices = random.sample(range(800), 800)
xSplit = []
ySplit = []
for i in range(5):
    randomRowIndices = randomIndices[(i*160):(i*160)+160]
    xSplit.append(xTrainNumpy[randomRowIndices])
    ySplit.append(ytrain[randomRowIndices])

```

In [12]:

```

def prepareTrainSetForKFoldCV(xTrain,yTrain,indexOfTheValidation):
    assert np.shape(xSplit) == (5,160,100001), "Check xTrain"
    assert np.shape(ySplit) == (5,160), "Check yTrain"
    xTrainWithoutValidationSet = []
    yTrainWithoutValidationSet = []
    for index in range(5):
        if index!=i:
            if len(xTrainWithoutValidationSet) == 0:
                xTrainWithoutValidationSet = xTrain[index]
                yTrainWithoutValidationSet = yTrain[index]
            else:
                xTrainWithoutValidationSet = np.concatenate((xTrainWithoutValidationSet,xTrain[index]),axis=0)
                yTrainWithoutValidationSet = np.concatenate((yTrainWithoutValidationSet,yTrain[index]),axis=0)

```

```
: def buildTreeAndgetF1Score(Xtr  
    XVa  
    i,  
  
    clf = tree.DecisionTreeClass
```

```

clf = clf.fit(Xtrain, ytrain)
yPredTrain = clf.predict(Xtrain)
yPredValidation = clf.predict(XValidation)
return get_f1(ytrain, yPredTrain), get_f1(YValidation, yPredValidation)

```

```

In [14]: depth = list(range(2,11))
allValidationF1Scores = []
averageValidationScores = []
allResults = []
for d in depth:
    results = []
    for i in range(5):
        XTrain,yTrain = prepareTrainSetForKFoldCV(xSplit,ySplit,i)
        XValidation = xSplit[i]
        YValidation = ySplit[i]
        trainF1Score, validationF1Score = buildTreeAndgetF1Score(XTrain,yTrain,
                                                                XValidation,YValidation,
                                                                i,d)
        allValidationF1Scores.append(validationF1Score)
        results.append([d,
                        i+1,
                        trainF1Score,
                        validationF1Score])
    allResults.append([d,
                      i+1,
                      trainF1Score,
                      validationF1Score])
columns = ['depth',
           'iteration in k-fold-cv',
           'F1-score (train)',
           'F1-score (validation)']
df = pd.DataFrame(results, columns=columns)
averageValidationScores.append(df['F1-score (validation)'].mean())
print(tabulate(df.set_index('depth'), headers='keys', tablefmt='psql'))

```

depth	iteration in k-fold-cv	F1-score (train)	F1-score (validation)
2	1	0.730435	0.571429
2	2	0.725664	0.545455
2	3	0.717949	0.608696
2	4	0.736842	0.62069
2	5	0.696429	0.774194
3	1	0.793103	0.666667
3	2	0.810811	0.516129
3	3	0.789916	0.666667
3	4	0.803419	0.551724
3	5	0.739496	0.685714
4	1	0.833333	0.666667
4	2	0.882883	0.606061
4	3	0.826446	0.64
4	4	0.845528	0.451613
4	5	0.806723	0.685714
5	1	0.905983	0.62069
5	2	0.883333	0.588235
5	3	0.873016	0.444444
5	4	0.868217	0.466667
5	5	0.87931	0.810811
6	1	0.918033	0.4
6	2	0.811594	0.5
6	3	0.926829	0.551724
6	4	0.929134	0.451613
6	5	0.915254	0.685714
7	1	0.921875	0.5
7	2	0.966102	0.484848
7	3	0.943089	0.551724
7	4	0.885714	0.473684
7	5	0.897638	0.585366
8	1	0.983871	0.444444
8	2	1	0.540541
8	3	0.818182	0.5
8	4	0.872483	0.378378
8	5	0.983333	0.666667
9	1	0.992	0.571429
9	2	1	0.470588
9	3	1	0.5
9	4	0.992248	0.545455
9	5	1	0.717949
10	1	1	0.571429
10	2	1	0.514286
10	3	0.977444	0.444444
10	4	1	0.5
10	5	1	0.736842

Plot the F1 score over the validation set for all K trials.

```
In [15]: validationScoresByDepth = [allValidationF1Scores[i:i + 5] for i in range(0, len(allValidationF1Scores), 5)]
```

```
plt.figure(figsize=(14,8))
validation = [1,2,3,4,5]
```

```
plt.plot(validation, validationScoresByDepth[0], label='depth = 2')
plt.plot(validation, validationScoresByDepth[1], label='depth = 3')
```

```
plt.plot(validation, validationScoresByDepth[2], label='depth = 4')
```

```
plt.plot(validation, validationScoresByDepth[3], label='depth = 5')
```

```
plt.plot(validation, validationScoresByDepth[4], label='depth = 6')
```

```
plt.xlabel('K-fold cv iteration', fontsize=12) # Label x-axis
```

```
plt.xticks(np.arange(1, 6, step=1))
```

```
plt.ylabel('F1 Score', fontsize=12) # Label y-axis
```

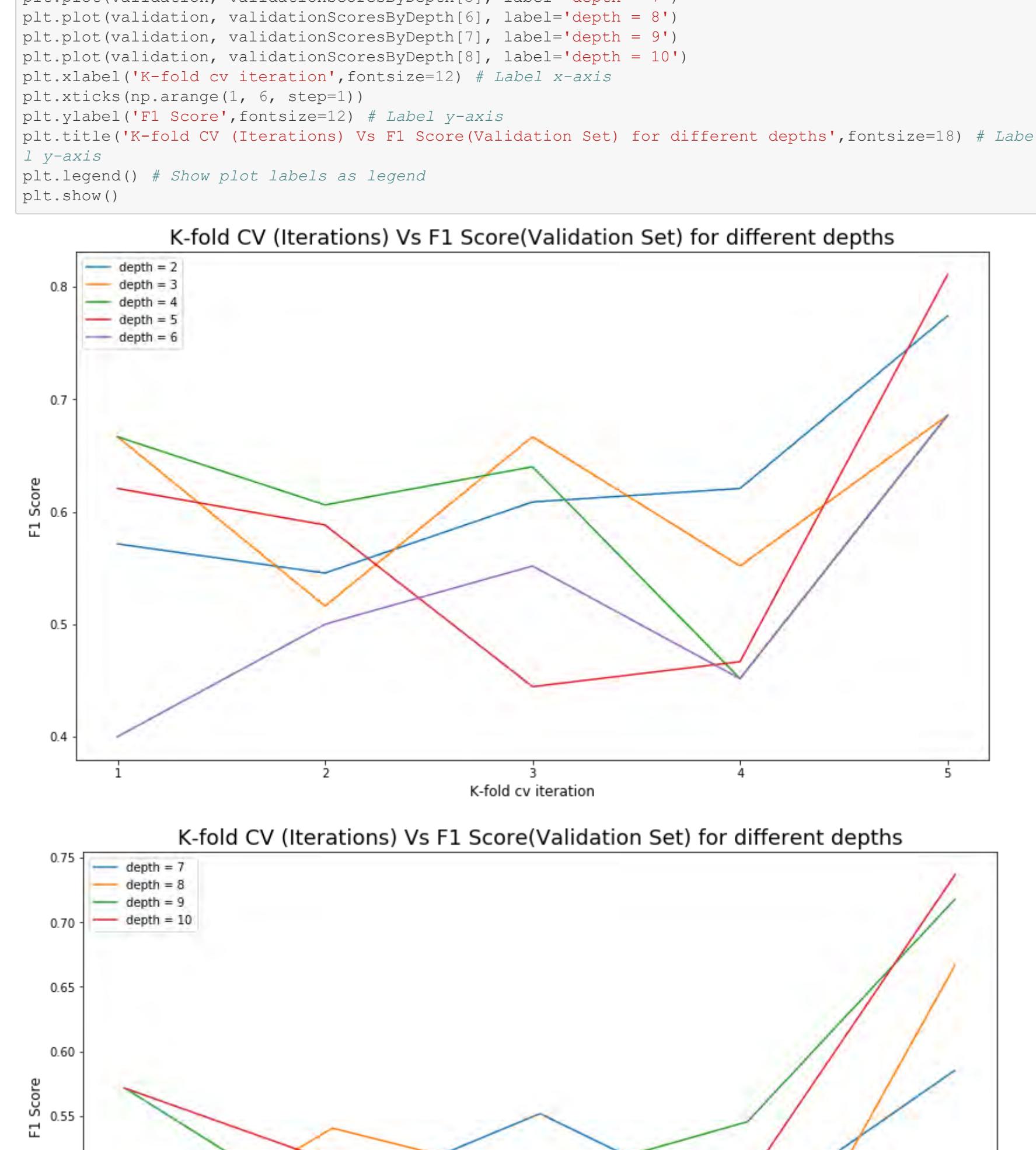
```
plt.title('K-fold CV (Iterations) Vs F1 Score(Validation Set) for different depths', fontsize=18) # Label y-axis
```

```
plt.legend() # Show plot labels as legend
```

```
plt.show()
```

```
plt.figure(figsize=(14,8))
```

```
validation = [1,2,3,4,5]
```

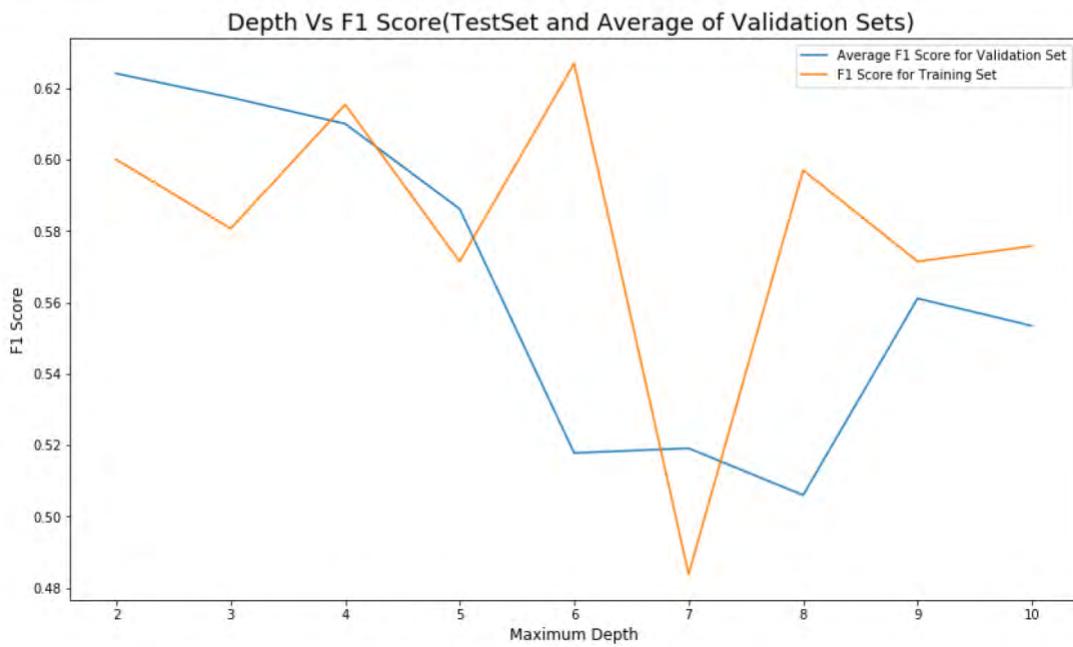


K-fold cv iteration	Blue Model	Green Model	Orange Model
1	0.50	0.50	0.45
2	0.48	0.47	0.52
3	0.52	0.50	0.45
4	0.47	0.50	0.38
5	0.52	0.52	0.52

plot the average validation F1 score against the test F1 score for the tree depth sweep.

```
In [16]: plt.figure(figsize=(14,8))
depth = list(range(2,11))

plt.plot(depth, averageValidationScores, label='Average F1 Score for Validation Set') # Plot training error over domain
testF1Scores = list(testF1ScoresWithoutValidation['F1-score (test)'])
plt.plot(depth, testF1Scores, label='F1 Score for Training Set') # Plot testing error over domain
plt.xlabel('Maximum Depth', fontsize=12) # Label x-axis
plt.ylabel('F1 Score', fontsize=12) # Label y-axis
plt.title('Depth Vs F1 Score(TestSet and Average of Validation Sets)', fontsize=18) # Label y-axis
plt.legend() # Show plot labels as legend
plt.show()
```



0.5

Q3)

(a) $L(y, \hat{y}) = e^{-(y, H_t(x))} = e^{-y_i H_t(x_i)}$

$$\hat{y}_i = H_t(x_i)$$

$$\begin{aligned}H_t(x) &= \sum_{i=1}^t \alpha_i h_i(x) \\&= \sum_{i=1}^{t-1} \alpha_i h_i(x) + \alpha_t h_t(x)\end{aligned}$$

$$H_t(x) = H_{t-1}(x) + \alpha_t h_t(x)$$

$$\begin{aligned}L(y, \hat{y}) &= \frac{1}{n} \sum_{i=1}^n e^{-y_i H(x_i)} \\&= \frac{1}{n} \sum_{i=1}^n e^{-y_i (H_{t-1} + \alpha_t h_t)} \\&= \frac{1}{n} \sum_{i=1}^n e^{-y_i H_{t-1}} e^{-y_i \alpha_t h_t} \\&= \frac{1}{n} \sum_{i=1}^n w_i^+ e^{-y_i \alpha_t h_t}\end{aligned}$$

To minimize loss, equate derivative to 0.

$$\frac{\partial L(y, \hat{y})}{\partial \alpha} = \frac{\partial}{\partial \alpha} \left(\frac{1}{n} \sum_{i=1}^n w_i^+ e^{-y_i \alpha_t h_t} \right) = 0$$

$$0 \Rightarrow -\frac{1}{n} \sum_{i=1}^n w_i^T y_i h_t e^{-y_i h_t}$$

$e^{x+y} = e^x e^y$

$$\Rightarrow -\frac{1}{n} \left[\sum_{\substack{i: h(x) = y_i}}^n w_i^T e^{-dt} - \sum_{\substack{i: h(x) \neq y_i}}^n w_i^T e^{dt} \right]$$

$y_i * h_t = 1$ $y_i * h_t = -1$

$$\Rightarrow -\frac{1}{n} \sum_{\substack{i: h(x) = y_i}}^n w_i^T e^{-dt} + \frac{1}{n} \sum_{\substack{i: h(x) \neq y_i}}^n w_i^T e^{dt}$$

$$e_t = \sum_{\substack{i: h(x) \neq y_i}}^n w_i^T$$

$$-e_t = \sum_{\substack{i: h(x) = y_i}}^n w_i^T$$

$$\Rightarrow 0 = (-e_t) e^{-dt} - e_t e^{dt}$$

$$\Rightarrow (-e_t) e^{-dt} = e_t e^{dt}$$

$$\Rightarrow \frac{1-e_t}{e^{dt}} = e_t e^{dt}$$

$$\Rightarrow \frac{1-e_t}{e_t} = e^{2dt}$$

$$\Rightarrow dt = \frac{1}{2} \ln \left(\frac{1-e_t}{e_t} \right)$$

In this we proved that the dt for a given iteration is $\frac{1}{2} \ln \left(\frac{1-e_t}{e_t} \right)$ and we proved that this dt minimizes the loss function.

So, an update, i.e. at the next step the empirical risk will indeed be reduced.

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
from sklearn import tree
import scipy.io as sio
import pandas as pd
from numpy import count_nonzero
import matplotlib.pyplot as plt
import scipy.sparse as ss
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn import tree
from tabulate import tabulate
```

```
In [2]: data = sio.loadmat('mnist.mat')

Xtrain = data['trainX'][10000, :].astype(int)
Xtest = data['testX'].astype(int)
ytrain = data['trainY'][0, :10000].astype(int)
ytest = data['testY'][0, :].astype(int)

idx = np.logical_or(np.equal(ytrain, 4), np.equal(ytrain, 9))
Xtrain = Xtrain[idx, :]
ytrain = ytrain[idx]
ytrain[np.equal(ytrain, 4)] = 1
ytrain[np.equal(ytrain, 9)] = -1

idx = np.logical_or(np.equal(ytest, 4), np.equal(ytest, 9))
Xtest = Xtest[idx, :]
ytest = ytest[idx]
ytest[ytest == 4] = 1
ytest[ytest == 9] = -1

sio.savemat('mnist_binary_small.mat', {'Xtrain': Xtrain, 'ytrain': ytrain, 'Xtest': Xtest, 'ytest': ytest})

data = sio.loadmat('mnist_binary_small.mat')

print(Xtrain.shape, Xtest.shape, ytrain.shape, ytest.shape)
(1958, 784) (1991, 784) (1958,) (1991,)
```

Helper Functions

```
In [3]: def get_misclass(y,yhat):
    incorrect = 0
    totalSamples = len(y)
    for i in range(totalSamples):
        if y[i] != yhat[i]:
            incorrect += 1
    return incorrect / totalSamples

def getExponentialLoss(y, yhat):
    return np.sum(np.exp(-y * yhat)) / len(y)

def plot(y, yLabel, title):
    x = list(range(len(y)))
    plt.figure(figsize=(14, 8))
    plt.plot(x, y)
    plt.xlabel('Iterations', fontsize=12)
    plt.ylabel(yLabel, fontsize=12)
    plt.title(title, fontsize=18)
    plt.show()
```

Without AdaBoost

```
In [4]: clf = tree.DecisionTreeClassifier(criterion='entropy',
                                      splitter='best',
                                      max_depth=1,
                                      class_weight='balanced')
w = [1/len(Xtrain)]*len(Xtrain)
clf = clf.fit(Xtrain, ytrain, sample_weight = w)

yPredTrain = clf.predict(Xtrain)
yPredTest = clf.predict(Xtest)
results = []
results.append([
    get_misclass(ytrain, yPredTrain),
    get_misclass(ytest, yPredTest),
])
columns = ['depth',
           'misclassification rate (train)',
           'misclassification rate (test)',]

pd.set_option("display.max_colwidth", 2)
df = pd.DataFrame(results, columns=columns)

print("Train andf test misclassification rates")
print(tabulate(df.set_index('depth'), headers='keys', tablefmt='psql'))
print("\n Train Exponential Loss Value: "+str(getExponentialLoss(ytrain, yPredTrain)))
```

With AdaBoost

```
In [5]: def predictAdaboost(X, stumps, stump_weights):
    # Stumps will be of size = (iterations, ) -> For every iteration we get 1 stump
    # Stump weights will be of size = (iterations, ) -> For every iteration we get 1 stump

    allStumpPredictions = []
    for i in range(len(stumps)):
        allStumpPredictions.append((stumps[i].predict(X)))

    weightedStumpPredictions = allStumpPredictions * stump_weights[:, None]
    sumOfWeightedStumpPredictions = weightedStumpPredictions.sum(axis=0)
    finalStumpPrediction = np.sign(sumOfWeightedStumpPredictions)

    return finalStumpPrediction

def getErrorAndAlpha(stumpWeight, stumpPrediction, y):
    error = np.sum(stumpWeight[(stumpPrediction != y)])
    alpha = 1 / 2 * np.log((1 - error) / error)
    return error, alpha

def fitAdaboost(XTrain, yTrain, XTest, yTest, iterations):
    # Total samples
    numberOfSamples = XTrain.shape[0]

    # Initialization
    exponentialLossOnlyForTheCurrentStump = np.zeros(shape=iterations)
    exponentialLossConsideringAllStumps = np.zeros(shape=iterations)
    trainMisclassificationRate = np.zeros(shape=iterations)
    testMisclassificationRate = np.zeros(shape=iterations)
    trainMisclassificationRateContinuous = np.zeros(shape=iterations)
    testMisclassificationRateContinuous = np.zeros(shape=iterations)
    weights = np.ones(shape=(iterations, numberOfSamples))
    stumps = np.zeros(shape=(iterations, ), dtype=object)
    stump_weights = np.zeros(shape=iterations)
    errors = np.zeros(shape=iterations)
    alphas = np.zeros(shape=iterations)

    # Initialize weights uniformly
    # Initial weights will be 1/n
    # n -> number of samples
    weights[0] = np.ones(shape=numberOfSamples) / numberOfSamples

    for index in range(iterations):
        # We will create new stump for every iteration
        currentWeights = weights[index]
        stump = tree.DecisionTreeClassifier(criterion='entropy',
                                            splitter='best',
                                            max_depth=1,
                                            max_leaf_nodes=2,
                                            class_weight='balanced')
        stump = stump.fit(XTrain, yTrain, sample_weight=currentWeights)

        # Stump Predictions for Test Set and Train Set
        stumpTrainPrediction = stump.predict(XTrain)
        stumpTestPrediction = stump.predict(XTest)

        # Stump Train and Test Misclassification rates
        trainMisclassificationRate[index] = get_misclass(yTrain, stumpTrainPrediction)
        testMisclassificationRate[index] = get_misclass(yTest, stumpTestPrediction)
        exponentialLossOnlyForTheCurrentStump[index] = getExponentialLoss(stumpTrainPrediction, yTrain)

        error, alpha = getErrorAndAlpha(currentWeights, stumpTrainPrediction, yTrain)

        updatedSampleWeights = currentWeights * np.exp(-alpha * yTrain * stumpTrainPrediction)
        # Normalizing new sample weights
        updatedSampleWeights /= updatedSampleWeights.sum()

        # If not final iteration, update sample weights for t+1
        if index != iterations - 1:
            weights[index + 1] = updatedSampleWeights

        stumps[index] = stump
        stump_weights[index] = alpha

        # All the required results for plotting
        cumulative_stump_train_prediction = predictAdaboost(XTrain, stumps[:index + 1], stump_weights[:index + 1])
        cumulative_stump_test_prediction = predictAdaboost(XTest, stumps[:index + 1], stump_weights[:index + 1])

        errors[index] = error
        alphas[index] = alpha
        exponentialLossConsideringAllStumps[index] = getExponentialLoss(cumulative_stump_train_prediction, yTrain)
        trainMisclassificationRateContinuous[index] = get_misclass(yTrain, cumulative_stump_train_prediction)
        testMisclassificationRateContinuous[index] = get_misclass(yTest, cumulative_stump_test_prediction)

    output = [stumps,
              stump_weights,
              exponentialLossOnlyForTheCurrentStump,
              trainMisclassificationRate,
              testMisclassificationRate,
              exponentialLossConsideringAllStumps,
              trainMisclassificationRateContinuous,
              testMisclassificationRateContinuous,
              errors,
              alphas]
    return output
```

```
In [6]: iterations = 100
modelParameters = fitAdaboost(Xtrain, ytrain, Xtest, ytest, iterations)
stumps = modelParameters[0]
stump_weights = modelParameters[1]
exponentialLossOnlyForTheCurrentStump = modelParameters[2]
trainMisclassificationRate = modelParameters[3]
testMisclassificationRate = modelParameters[4]
exponentialLossConsideringAllStumps = modelParameters[5]
trainMisclassificationRateContinuous = modelParameters[6]
testMisclassificationRateContinuous = modelParameters[7]
errors = modelParameters[8]
alphas = modelParameters[9]
yPredTrain = predictAdaboost(Xtrain, stumps, stump_weights)
yPredTest = predictAdaboost(Xtest, stumps, stump_weights)
results = [(get_misclass(ytrain, yPredTrain),
            get_misclass(ytest, yPredTest),
            1)]
columns = ['misclassification rate (train)',
           'misclassification rate (test)',]
```

```
pd.set_option("display.max_colwidth", 2)
df = pd.DataFrame(results, columns=columns)

print("Train andf test misclassification rates after " + str(iterations) + " iterations")
print(tabulate(df, headers='keys', tablefmt='psql'))
```

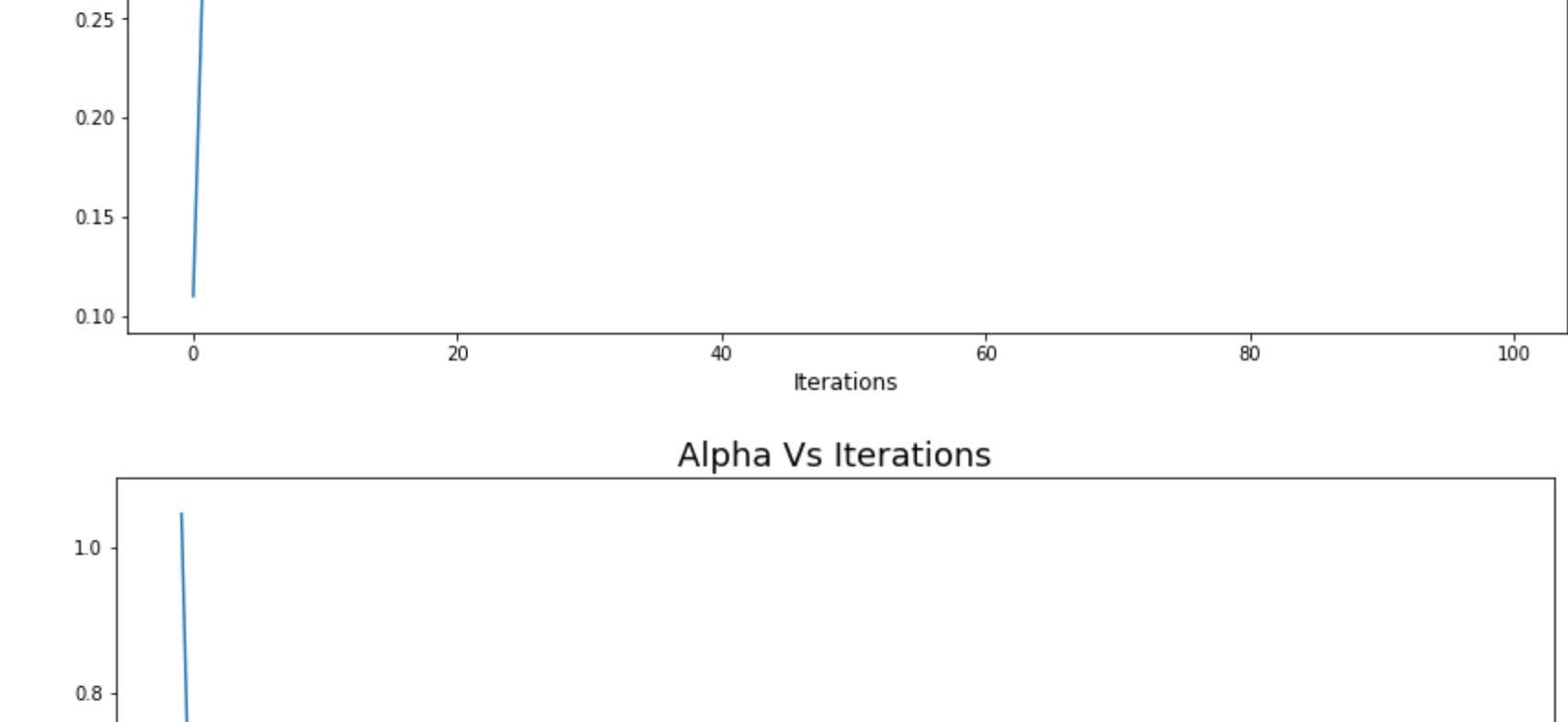
```
print("\n Train Exponential Loss Value after " + str(iterations) + " iterations: " + str(getExponentialLoss(ytrain, yPredTrain)))
```

```
Train andf test misclassification rates after 100 iterations
+---+-----+
| ! misclassification rate (train) | misclassification rate (test) |
+---+-----+
| 0 | 0.014811 | 0.048217 |
+---+-----+
```

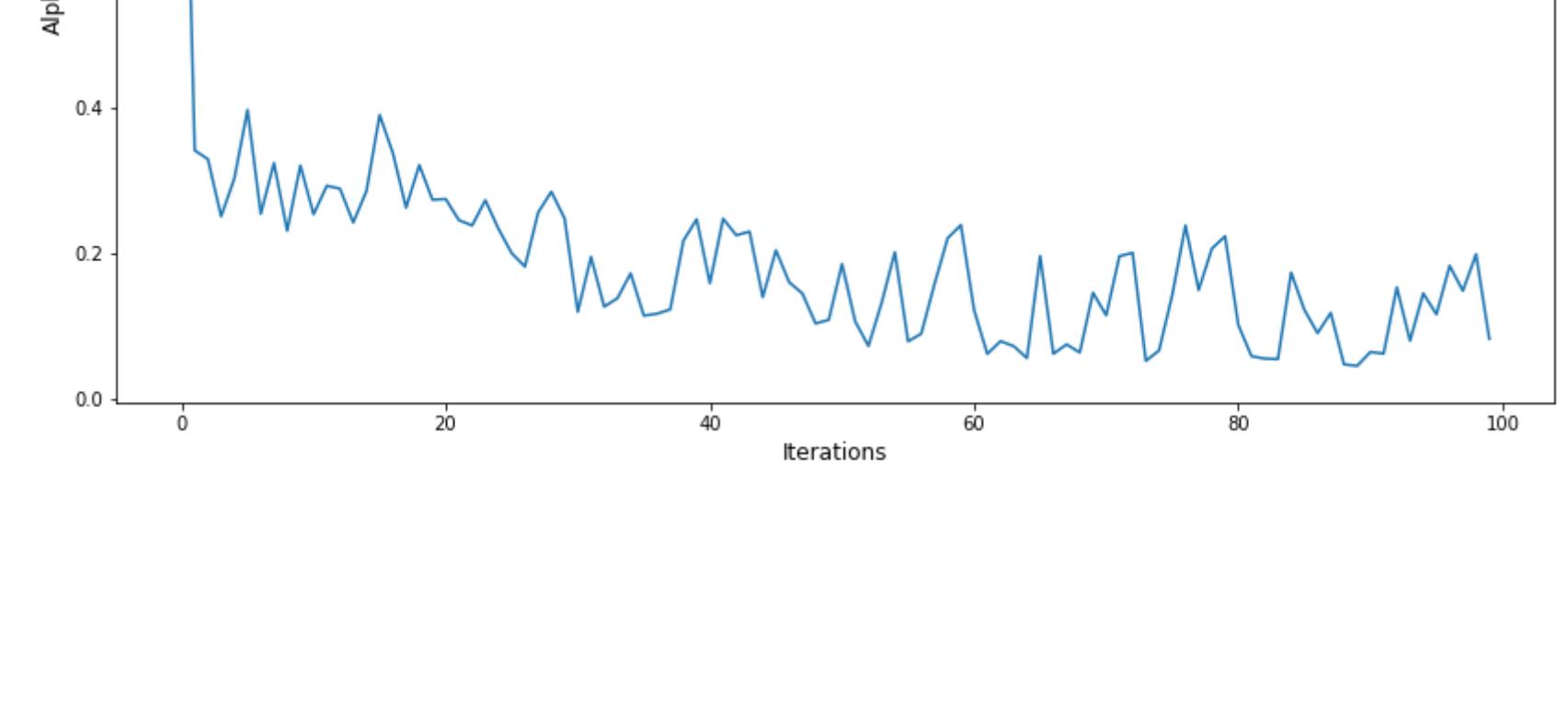
```
Train Exponential Loss Value after 100 iterations: 0.4026913253549667
```

```
In [7]: plot(trainMisclassificationRate, 'Train Misclassification Rate', 'Train Misclassification Rate Vs Iterations (After every stump)')
plot(testMisclassificationRate, 'Test Misclassification Rate', 'Test Misclassification Rate Vs Iterations (After every stump)')
plot(exponentialLossOnlyForTheCurrentStump, 'Exponential Loss', 'Exponential Loss Vs Iterations (After every stump)')
```

Train Misclassification Rate Vs Iterations (After every stump)



Test Misclassification Rate Vs Iterations (After every stump)



Exponential Loss Vs Iterations (Considering All Stumps)



```
In [8]: plot(trainMisclassificationRateContinuous, 'Train Misclassification Rate', 'Train Misclassification Rate Vs Iterations (Considering all Stumps)')
plot(testMisclassificationRateContinuous, 'Test Misclassification Rate', 'Test Misclassification Rate Vs Iterations (Considering All Stumps)')
plot(exponentialLossConsideringAllStumps, 'Exponential Loss', 'Exponential Loss Vs Iterations (Considering All Stumps)')
```

Train Misclassification Rate Vs Iterations (Considering All Stumps)

Test Misclassification Rate Vs Iterations (Considering All Stumps)

Exponential Loss Vs Iterations (Considering All Stumps)


```
In [9]: plot(errors, 'Error', 'Error Vs Iterations')
plot(alphas, 'Alpha', 'Alpha Vs Iterations')
```

Error Vs Iterations

Alpha Vs Iterations

Comments

1) Total error takes a value between 0 and 1.

0-> very good stump

1-> very bad stump

2) Alpha, or the amount of say a stump has is inversely proportional to the error.

More, the error, less alpha and vice-versa.

When total error is small, i.e. if a stump is good then alpha will be a large positive value.

Interpretations (in terms of weighted performance):

1) If a stump has a high weight and high alpha value, it's say in the final prediction would be high.

2) If a stump has a low alpha value, it's say would be very less.

3) If error is 0.5 (i.e. just a random guess), it's alpha value would be 0, i.e that stump would mean nothing in the final prediction.

challenge

+0.75

$$(a) H(x) = -p_{\text{red}} \log_2 p_{\text{red}} - p_{\text{black}} \log_2 p_{\text{black}}$$

For maximum entropy, $H'(x) = 0$

We know that,

$$p_{\text{red}} + p_{\text{black}} = 1$$

$$p_{\text{black}} = 1 - p_{\text{red}}$$

$$H(x) = -p_{\text{red}} \log_2 p_{\text{red}} - (1 - p_{\text{red}}) \log_2 (1 - p_{\text{red}})$$

$$H'(x) = -\log p - \frac{P}{P} - \left(-\log(1-P) - \frac{(1-P)}{1-P} \right) = 0$$

$$\Rightarrow -\log P - 1 + \log(1-P) + 1 = 0$$

$$\Rightarrow \log \left(\frac{1-P}{P} \right) = 0$$

$$1-P = P \Rightarrow 2P = 1 \Rightarrow P = 1/2$$

To maximize entropy, buy 5 red socks and 5 black socks

$$(b) \underset{P}{\text{maximize}} f(p) := -P \log_2(P) - (1-P) \log_2(1-P)$$

$$f'(p) = -\log P - \frac{P}{P} - \left[-\log(1-P) + \frac{(1-P)}{1-P} \right] = 0$$

$$\Rightarrow -\log P - 1 + \log(1-P) + 1 = 0$$

$$\Rightarrow \log\left(\frac{c-p}{p}\right) = 0$$

$$\Rightarrow c-p = p \Rightarrow c = 2p \quad p = c/2$$

$$\therefore p = c/2 \quad \text{for optimum value}$$



(c) Assume we have 3 different colours,

$$P(\text{colour 1}) = x$$

$$P(\text{colour 2}) = y$$

$$P(\text{colour 3}) = z = 1-x-y$$

$$H(x,y) = -x \log x - y \log y - (1-x-y) \log(1-x-y)$$

$$\frac{\partial H(x,y)}{\partial x} = 0 \quad , \quad \frac{\partial H(x,y)}{\partial y} = 0 \quad \left. \begin{array}{l} \text{for maximum} \\ \text{entropy} \end{array} \right\}$$

$$\frac{\partial H(x,y)}{\partial x} = 0, \quad \text{we get} \quad 2x + y = 1 \quad \text{--- } \textcircled{1}$$

$$\frac{\partial H(x,y)}{\partial y} = 0, \quad \text{we get} \quad 2y + x = 1 \quad \text{--- } \textcircled{2}$$

$$\text{Solving } \textcircled{1} \text{ & } \textcircled{2}, \text{ we get } x = \frac{1}{3}, y = \frac{1}{3}, z = \frac{1}{3}$$



we can extend this to n -colours,

where $P(i) = \frac{1}{n}$ ($i = 1 \dots n$) how?

Part (a) of the question is when $n = 2$

Here, $n = 100$

$$P(i) = \frac{1}{100}$$

Total socks = 10,000

\therefore we need to buy $\frac{10,000}{100} = 100$ socks

of each color.

+0.75

(Q2) $P_A(Y_i=1) = P$,
 $P_A(Y_i=1 | Y_{i-1}=1) = cP - CP$
 $P_A(Y_i=-1 | Y_{i-1}=-1) = CP - P + 1$

(a) we will prove this with induction.

for $k=1$:- $P_A(Y_1=1) = P \rightarrow$ given

for $k=n-1$:- $P_A(Y_{n-1}=1) = P \rightarrow$ assumption

for $k=n$:- $P_A(Y_n=1) = P_A(Y_n=1 | Y_{n-1}=1) P_A(Y_{n-1}=1)$
 $+ P_A(Y_n=1 | Y_{n-1}=-1) P_A(Y_{n-1}=-1)$

$$\begin{aligned} &= (cP - CP)(P) + (1 - (cP - CP + 1))(1-P) \\ &= (cP - CP)(P) + (P - CP)(1-P) \\ &= P \end{aligned}$$

$\therefore P_A(Y_i=1) = P \text{ for all } i$

(b) $\text{corr}(u, v) = \frac{E[uv] - E[u]E[v]}{\sqrt{\text{var}(u) \text{ var}(v)}}$

(Y_i, Y_{i-1})	$Y_i * Y_{i-1}$	$P(Y_i, Y_{i-1}) = P(Y_i Y_{i-1}) P(Y_{i-1})$
$(1, 1)$	1	$((1+p - (p)) (p))$
$(1, -1)$	-1	$(p - (p) (1-p))$
$(-1, 1)$	-1	$((1-p - p + (p)) (p))$
$(-1, -1)$	1	$((p - p + 1) (1-p))$

$$E[uv] = \sum_{i=1}^n (u_i v_i) P_{XY}(u_i, v_i)$$

$$= 4p^2 - 4p^2(1-p) + 4p(1-p) + 1 - 4p$$

$$E[v] = 1 \cdot p - 1 \cdot (1-p) = 2p - 1$$

$$E[u] = E[v] = 2p - 1$$

$$E[uv] - E[u]E[v] = 4p(1-p)$$

$$E[v^2] = 1^2 p + (-1)^2 (1-p) = p + 1 - p = 1$$

$$\text{var}(u) = E[u^2] - E[u]^2 = 1 - (2p-1)^2 = 4p(1-p)$$

$$\text{var}(v) = \text{var}(u) = 4p(1-p)$$

$$\text{corr}(u, v) = \frac{uPC(1-P)}{\sqrt{uP(1-P) * vP(1-P)}} = C$$

$\therefore \text{corr}(u, v) = C$



c) For $m=3$, majority votes (at least 2 1's)
are possible in the following scenario

y_1	y_2	y_3
1	1	1
1	1	-1
1	-1	1
-1	1	1

$$Pr(y_i=1 | y_{i-1}=1) = P_{11}$$

$$Pr(y_i=-1 | y_{i-1}=-1) = P_{00}$$

$$Pr(y_i=-1 | y_{i-1}=1) = 1-P_{11}$$

$$Pr(y_i=1 | y_{i-1}=1) = 1-P_{00}$$

$$= P * P_{11} * P_{11}$$

$$+ P * P_{11} * (1-P_{11})$$

$$+ P * (1-P_{11}) * (1-P_{00})$$

$$\begin{aligned}
& + (-P) * (-P_{00}) * P_{11} \\
= & P * \cancel{(P_{11})^2} + P * \cancel{P_{11}} - P * \cancel{(P_{11})^2} \\
& + P - P P_{00} - \cancel{P * P_{11}} + P * P_{11} * P_{00} \\
& + P_{11} - P_{00} P_{11} - P P_{11} + P P_{00} P_{11} \\
= & P - P P_{00} + 2(P P_{00} P_{11}) + P_{11} - P_{00} P_{11} \\
& - P P_{11} \\
= & P + P_{11} - P P_{11} - P_{00} P_{11} - P P_{00} + 2(P P_{00} P_{11}) \\
& \quad \checkmark
\end{aligned}$$

Challenge Q2

```
In [1]: import numpy as np
import random
import itertools
from collections import Counter
from tabulate import tabulate
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

probability that I will buy a yacht, for m = 10

```
In [2]: def getProbabilityFromPreviousConsultant(current, previous, p, c):
    if current == 1 and previous == 1: ## 1->1
        return c + p -(c*p)
    elif current == -1 and previous == 1: ## 1-1->0
        return 1 - (c + p -(c*p))
    elif current == -1 and previous == -1: #-1-1->1
        return (c*p) -p + 1
    elif current == 1 and previous == -1:
        return p -(c*p)
```

```
In [3]: m = 10
allPermutations = list(itertools.product([-1, 1], repeat=m))

cpValues = [
    [0,0],
    [-0.25, 0.25],
    [0.25,0.25],
    [0.5,0.25],
    [0.75,0.25],
    [1,0.25],
    [-0.25, 0.5],
    [0.25,0.5],
    [0.5,0.5],
    [0.75,0.5],
    [1,0.5],
    [-0.25, 0.75],
    [0.25,0.75],
    [0.5,0.75],
    [0.75,0.75],
    [1,1],
]
results = []
for cpValue in cpValues:
    c,p = cpValue[0], cpValue[1]
    ## checking if probabilities lie in the range 0-1
    if 0<=c + p -(c*p)<=1 and 0<=(c + p -(c*p))<=1 and 0<=(c*p) -p + 1<=1 and 0<=p -(c*p)<=1:

        total = 0
        for permutation in allPermutations: #[1,1,1]
            frequencyMap = Counter(permutation)
            totalProbabilityForThisPermutation = 0
            if frequencyMap[1]>(m/2):
                for index in range(len(permutation)):
                    # For 1st advisor
                    if index == 0:
                        if permutation[index] == 1:
                            totalProbabilityForThisPermutation += p
                        continue
                    else:
                        totalProbabilityForThisPermutation += (1-p)
                    continue
                # For the rest
                totalProbabilityForThisPermutation *= getProbabilityFromPreviousConsultant(permutation[index],
ion[index-1],
permutation[index-1],p,c)

            total += (totalProbabilityForThisPermutation)
        results.append([c,
                      p,
                      m,
                      total])
columns = ['c',
           'p',
           'm',
           'probability']
df = pd.DataFrame(results, columns=columns)
print(tabulate(df.set_index('c'), headers='keys', tablefmt='psql'))
```

c	p	m	probability
0	0	10	0
-0.25	0.25	10	0.00194145
0.25	0.25	10	0.0553765
0.5	0.25	10	0.108958
0.75	0.25	10	0.179941
1	0.25	10	0.25
-0.25	0.5	10	0.34174
0.25	0.5	10	0.405044
0.5	0.5	10	0.430332
0.75	0.5	10	0.457382
1	0.5	10	0.5
-0.25	0.75	10	0.970681
0.25	0.75	10	0.869105
0.5	0.75	10	0.813797
0.75	0.75	10	0.76134
1	1	10	1

```
In [4]: plt.figure(figsize=(14,8))
validation = [1,2,3,4,5]

plt.plot(list(df.loc[df['c'] == -0.25]['p']), list(df.loc[df['c'] == -0.25]['probability']), label='c = -0.25')
plt.plot(list(df.loc[df['c'] == 0.25]['p']), list(df.loc[df['c'] == 0.25]['probability']), label='c = 0.25')
plt.plot(list(df.loc[df['c'] == 0.5]['p']), list(df.loc[df['c'] == 0.5]['probability']), label='c = 0.5')
plt.plot(list(df.loc[df['c'] == 0.75]['p']), list(df.loc[df['c'] == 0.75]['probability']), label='c = 0.75')
plt.xlabel('p values', fontsize=12) # Label x-axis
plt.xticks(np.arange(0, 1, step=0.1))
plt.ylabel('Total probability', fontsize=12) # Label y-axis
plt.title('p-value vs probability (for a fixed c)', fontsize=18) # Label y-axis
plt.legend() # Show plot labels as legend
plt.show()
```

Hmm, doesn't look right, see solutions

```
In [5]: plt.figure(figsize=(14,8))
validation = [1,2,3,4,5]

plt.plot(list(df.loc[df['p'] == 0.25]['c']), list(df.loc[df['p'] == 0.25]['probability']), label='p = 0.25')
plt.plot(list(df.loc[df['p'] == 0.5]['c']), list(df.loc[df['p'] == 0.5]['probability']), label='p = 0.5')
plt.plot(list(df.loc[df['p'] == 0.75]['c']), list(df.loc[df['p'] == 0.75]['probability']), label='p = 0.75')
plt.xlabel('p values', fontsize=12) # Label x-axis
plt.xticks(np.arange(0, 1, step=0.1))
plt.ylabel('Total probability', fontsize=12) # Label y-axis
plt.title('c-value vs probability (for a fixed p)', fontsize=18) # Label y-axis
plt.legend() # Show plot labels as legend
plt.show()
```

c-value vs probability (for a fixed p)

Comments

We have generated all the $2^{10} = 1024$ possibilities.

We have only considered cases where there > 5(10/2) 1's.

Probability increases as p increases for a fixed c

Simulate the sequential advisers,

```
In [6]: # We will run the experiment for 10,000 times
```

```
In [7]: def getNextConsultantValue(previousConsultant,
                                p,
                                c):
    if previousConsultant == 1:
        pValue = c + p -(c*p) # This is the probability for (1,1)
        return np.random.choice([-1, 1], p=[1-pValue,pValue])
    else:
        pValue = (c*p) -p + 1 # This is the probability for (-1,-1)
        return np.random.choice([-1, 1], p=[pValue,1-pValue])
```

```
In [8]: numberoftrials = 10000

cpValues = [
    [0,0],
    [0.25,0.25],
    [0.5,0.5],
    [0.75,0.25],
    [0.75,0.75],
    [1,1],
]

mValues = [25,100]

for m in mValues:
    results = []
    for cpValue in cpValues:
        cValue,pValue = cpValue[0], cpValue[1]
        if (0<= cValue+pValue-(cValue*pValue)<=1 and
            0<= 1-(cValue+pValue-(cValue*pValue))<=1 and
            0<= (cValue*pValue)-pValue+1 <=1 and
            0<= pValue-(cValue*pValue)<= 1):

            permutations = []
            buyAYacht = 0

            for i in range(numberoftrials):

                permutations.append([()])
                for index in range(m):
                    if index == 0:
                        firstConsultant = np.random.choice([-1, 1], p=[1-pValue,pValue])
                        permutations[i].append(firstConsultant)
                    else:
                        previousConsultant = permutations[i][index-1]
                        nextConsultant = getNextConsultantValue(previousConsultant,
                                                                pValue,
                                                                cValue)
                        permutations[i].append(nextConsultant)
                frequencyMap = Counter(permutations[i])
                if frequencyMap[1] >= m/2:
                    buyAYacht += 1
                results.append([cValue,
                                pValue,
                                m,
                                numberoftrials,
                                buyAYacht/numberoftrials])

columns = ['c',
           'p',
           'm',
           'numberOfTrials',
           'probability of buying a yacht'
          ]
df = pd.DataFrame(results, columns=columns)
print(tabulate(df.set_index('c'), headers='keys', tablefmt='psql'))
```

c	p	m	numberOfTrials	probability of buying a yacht
0	0	25	10000	0
0.25	0.25	25	10000	0.0168
0.5	0.5	25	10000	0.5019
0.75	0.25	25	10000	0.1399
0.75	0.75	25	10000	0.8638
1	1	25	10000	1

c	p	m	numberOfTrials	probability of buying a yacht
0	0	100	10000	0
0.25	0.25	100	10000	0
0.5	0.5	100	10000	0.5233
0.75	0.25	100	10000	0.0252
0.75	0.75	100	10000	0.9789
1	1	100	10000	1

Resources:

1. <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>
2. https://mitpress.mit.edu/sites/default/files/titles/content/boosting_foundations_algorithms/chapter007.html
3. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html>
4. <https://geoffriddock.com/adaboost-from-scratch-in-python/>
5. <https://www.cs.toronto.edu/~mbrubake/teaching/C11/Handouts/AdaBoost.pdf>
6. <https://jeremykun.com/2015/05/18/boosting-census/>
7. <https://xavierbourretsicotte.github.io/AdaBoost.html>
8. https://www.youtube.com/watch?v=LsK-xG1cLYA&list=PLblh5JKOoLUICTaGLRoHQDuF_7q2GfuJF&index=42&ab_channel=StatQuestwithJoshStarmer