Q5

April 11, 2022

[14]:
```python
#Importing necessary library
import numpy as np
from sklearn.decomposition import SparsePCA
import matplotlib.pyplot as plt
import random
from sklearn.feature_extraction.text import TfidfVectorizer
```

[15]:
```python
file = open('DataQ5.txt','r+')
reviews = file.read()
file.close()
```

[16]:
```python
reviewsList = reviews.split("\n")
```

[17]:
```python
onlyReviewsList = []
positiveNegativeList = []
for i in reviewsList:
    onlyReviewsList.append(i[0:len(i)-4].lower())
    positiveNegativeList.append(i[len(i)-1:])

N = len(onlyReviewsList)
trainSize = int(3/4 * N)
testSize = int(1/4 * N)
```

[18]:
```python
onlyReviewsList = np.array(onlyReviewsList)
positiveNegativeList = np.array(positiveNegativeList)
Tn = positiveNegativeList.astype(int)
```

[19]:
```python
vectorizer = TfidfVectorizer()
TF_IDF = vectorizer.fit_transform(onlyReviewsList)
```

1 Part A

[20]:
```python
trainTF_IDF = TF_IDF[:trainSize,]
testTF_IDF = TF_IDF[trainSize:,]
```

2 Part B

[21]:
```
# Reducing the dimension to 30 using PCA    TRAIN DATA
temp = SparsePCA(n_components = 30)                             #Error Need a␣
 ↪dense matrix
reducedTrainTF_IDF = temp.fit_transform(trainTF_IDF.toarray())
print(reducedTrainTF_IDF.shape)

# Reducing the dimension to 30 using PCA   TEST DATA
temp = SparsePCA(n_components = 30)                             #Error Need a␣
 ↪dense matrix
testReducedTF_IDF = temp.fit_transform(testTF_IDF.toarray())
print(testReducedTF_IDF.shape)
```

```
(750, 30)
(250, 30)
```

[22]:
```
#Splitting into Train and Validation
trainSize = int(5/6 * 3/4 *N)
validSize = int(1/6 * 3/4 *N)

trainReducedTF_IDF = reducedTrainTF_IDF[:trainSize,]
validReducedTF_IDF = reducedTrainTF_IDF[trainSize:,]
trainTn = Tn[:trainSize]
validTn = Tn[trainSize:trainSize + validSize]
testTn = Tn[trainSize + validSize:]

trainReducedTF_IDF = trainReducedTF_IDF.T
validReducedTF_IDF = validReducedTF_IDF.T
testReducedTF_IDF = testReducedTF_IDF.T
```

3 Part C

[23]:
```
class LogisticRegession:
    def __init__(self, nIter, batchSize, lr, regularizationCoefficient = 0):
        self.nIter = nIter
        self.batchSize = batchSize
        self.lr = lr
        self.regularizationCoefficient = regularizationCoefficient

        #Initialize W and b
        self.W = np.random.rand(30,1)
        self.b = np.random.rand(1,1)

        #Loss list
        self.lossTrain = []
        self.lossValid = []
```

```
    def fit(self,rcFlag=0):
        for j in range(self.nIter):
            i = 0
            while(True):
                flag = 0
                batchStartIndex = i * self.batchSize
                batchEndIndex = (i+1) * self.batchSize
                if (batchEndIndex>=trainSize):
                    batchEndIndex = trainSize
                    flag = 1
                #Forward pass
                Y = np.matmul((self.W).T, trainReducedTF_IDF) + self.b
                Y = np.exp((-1) * Y)
                Y = np.reciprocal(1 + Y)

                #Backward pass
                temp1 = Y.reshape(1,625)-trainTn.reshape(1,625)     #temp1 = yn-tn
                temp1 = temp1[0][batchStartIndex:batchEndIndex]
                temp1 = temp1.reshape(1,batchEndIndex-batchStartIndex)

                temp2 = trainReducedTF_IDF.T[batchStartIndex:batchEndIndex,]     #temp2 = x[batchSize]

                if(rcFlag == 0):
                    self.W = self.W - self.lr * (np.matmul(temp1, temp2)).T
                else:
                    self.W = self.W - self.lr * ((np.matmul(temp1, temp2)).T + self.regularizationCoefficient * self.W)
                self.b = self.b - self.lr * np.sum(temp1)

                i += 1
                if(flag==1):
                    break

            #Calculating Loss on Train Data
            temp3 = trainTn.reshape(1,trainSize)
            temp4 = Y.reshape(trainSize,1)
            temp5 = np.matmul(temp3,np.log(temp4))

            temp6 = np.full((1,trainSize),1) - temp3
            temp7 = np.full((trainSize,1),1) - temp4
            temp8 = np.matmul(temp6,np.log(temp7))

            calcLoss = -1 * (temp5 + temp8)
            self.lossTrain.append(int(calcLoss))
```

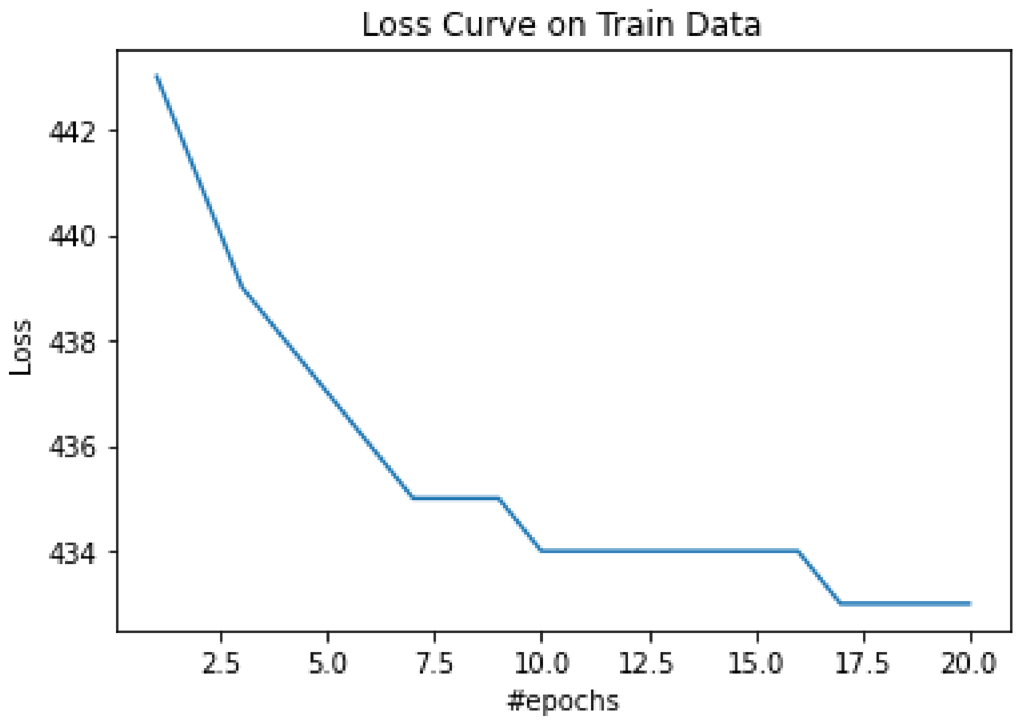
```python
            #Calculating Loss on Validation Data
            #Forming Y
            Y_valid = np.matmul((self.W).T, validReducedTF_IDF) + self.b
            Y_valid = np.exp((-1) * Y_valid)
            Y_valid = np.reciprocal(1 + Y_valid)

            temp3 = validTn.reshape(1,validSize)
            temp4 = Y_valid.reshape(validSize,1)
            temp5 = np.matmul(temp3,np.log(temp4))

            temp6 = np.full((1,validSize),1) - temp3
            temp7 = np.full((validSize,1),1) - temp4
            temp8 = np.matmul(temp6,np.log(temp7))

            calcLoss = -1 * (temp5 + temp8)
            self.lossValid.append(int(calcLoss))

    def predict(self):
        Y = np.matmul((self.W).T, testReducedTF_IDF) + self.b
        Y = np.exp((-1) * Y)
        Y = np.reciprocal(1 + Y)

        correct = 0
        total = Y.shape[1]
        predicted = []
        for i in range(total):
            if (Y[0][i]<0.5):
                predicted.append(1)
            else:
                predicted.append(0)

        for i in range(total):
            if(predicted[i] == int(testTn[i])):
                correct += 1
        accuracy = correct/total * 100
        print('\n'+'\033[1m' + "Review Classification Accuracy on Test Data = ", "{:.2f}".format(accuracy), " %" + '\033[0m' )

    def plotLossCurve(self,y,flag):
        x = [(i+1) for i in range(self.nIter)]

        plt.xlabel('#epochs')
        plt.ylabel('Loss')
        s = ""
        if (flag == 0):
            s = 'Train Data'
```

```
        else:
            s = 'Valid Data'
        plt.title('Loss Curve on ' + s)
        plt.plot(x, y)
        plt.show()

    def showLossValues(self,flag):
        x =[]
        if (flag == 0):
            x = self.lossTrain
            print('\033[1m' + "Loss values at each epoch for Train Data" +␣
 ↪'\033[0m')
        else:
            x = self.lossValid
            print('\033[1m' + "Loss values at each epoch for Valid Data" +␣
 ↪'\033[0m')

        for i in range(self.nIter):
            print((i+1),"th Iteration->LossValue = ", x[i])
```

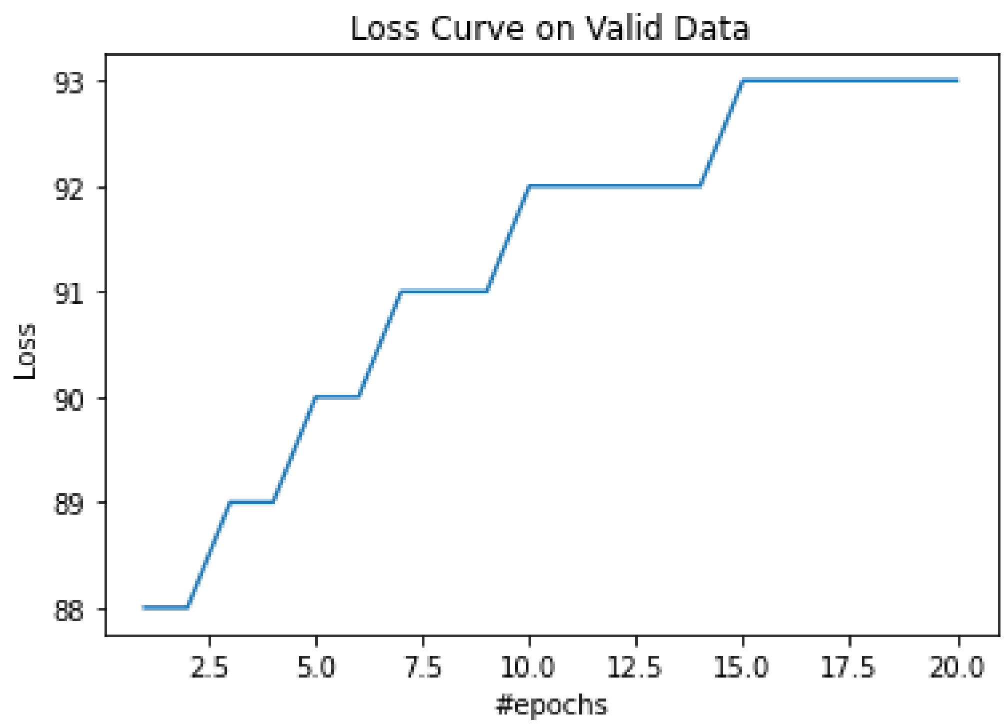
[24]:
```
epochs = 20
batchSizeList = [32, 64, 128]
LRList = [1e-3, 1e-2, 1e-1 ]
```

[25]:
```
k = 0
for bs in batchSizeList:
    for lr in LRList:
        k += 1
        print("Case " + str(k) + ": BatchSize = " + str(bs) +" and Learning␣
 ↪Rate = " + str(lr))
        tempObj = LogisticRegession(epochs,bs,lr)
        tempObj.fit()
        tempObj.plotLossCurve(tempObj.lossTrain,0)
        # tempObj.showLossValues(0)
        tempObj.plotLossCurve(tempObj.lossValid,1)
        # tempObj.showLossValues(1)
        tempObj.predict()
       ␣
 ↪print("------------------------------------------------------------------------------")
```

```
Case 1: BatchSize = 32 and Learning Rate = 0.001
```



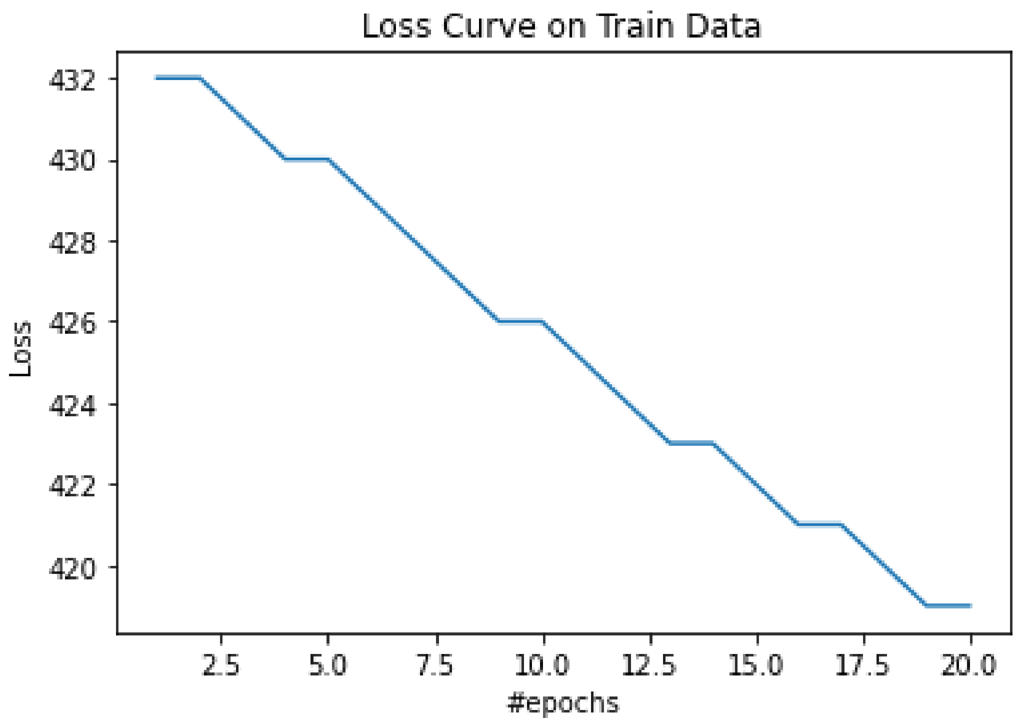
Loss Curve on Train Data
442
440
438
436
434
Loss
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
#epochs

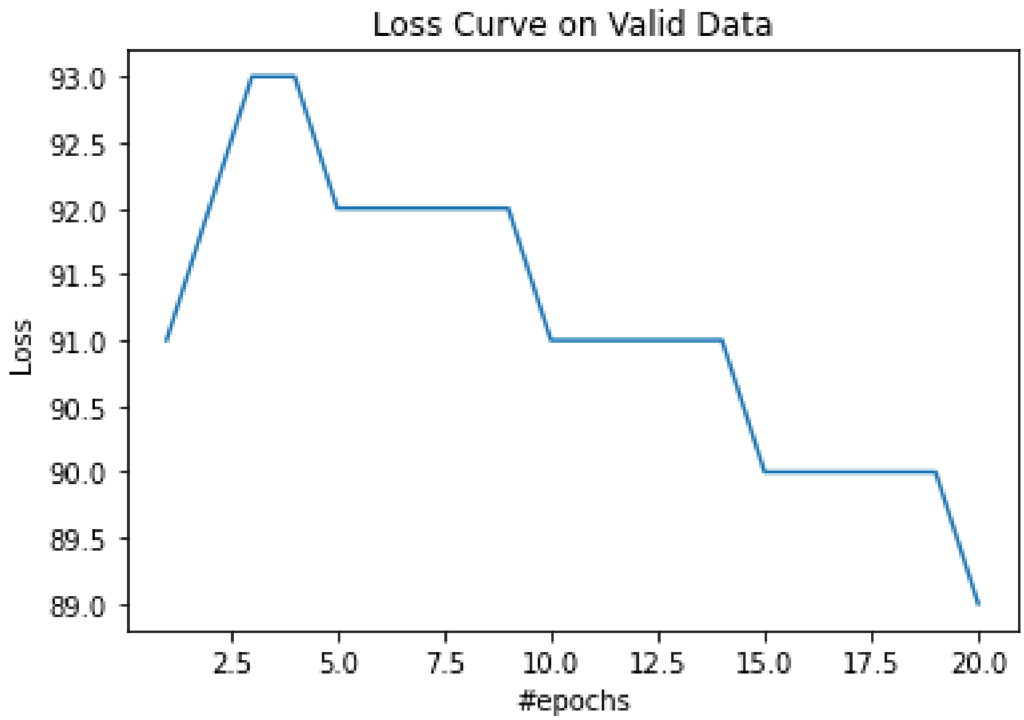


Loss Curve on Valid Data
93
92
91
90
89
88
Loss
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
#epochs

Review Classification Accuracy on Test Data = 60.80 %

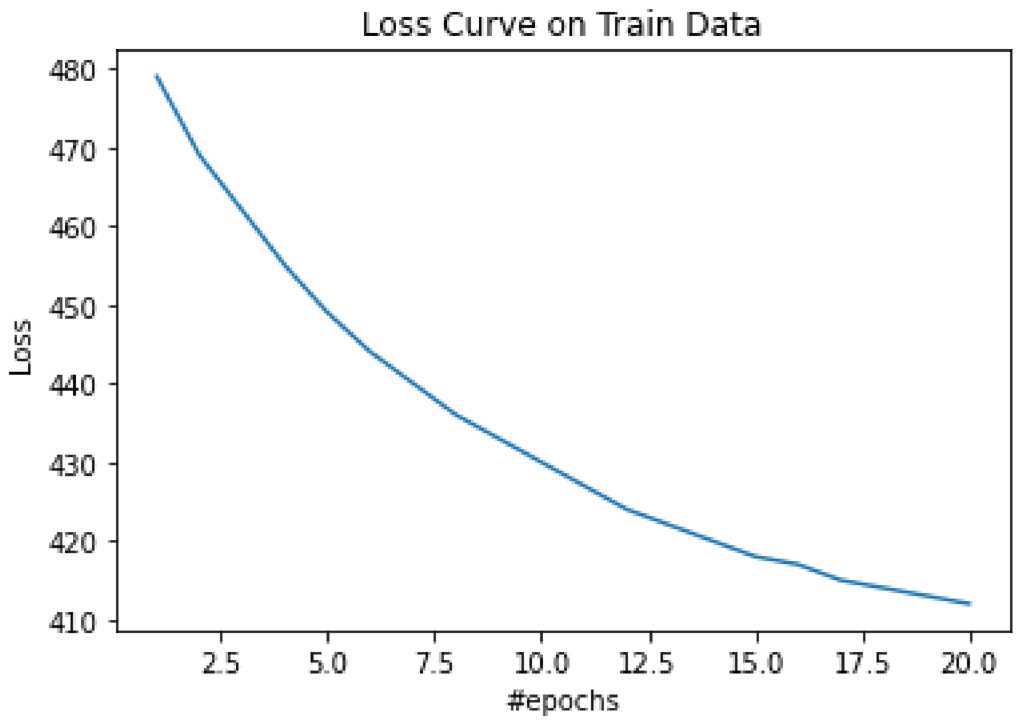
```
-------------------------------------------------------------------------------
Case 2: BatchSize = 32 and Learning Rate = 0.01
```



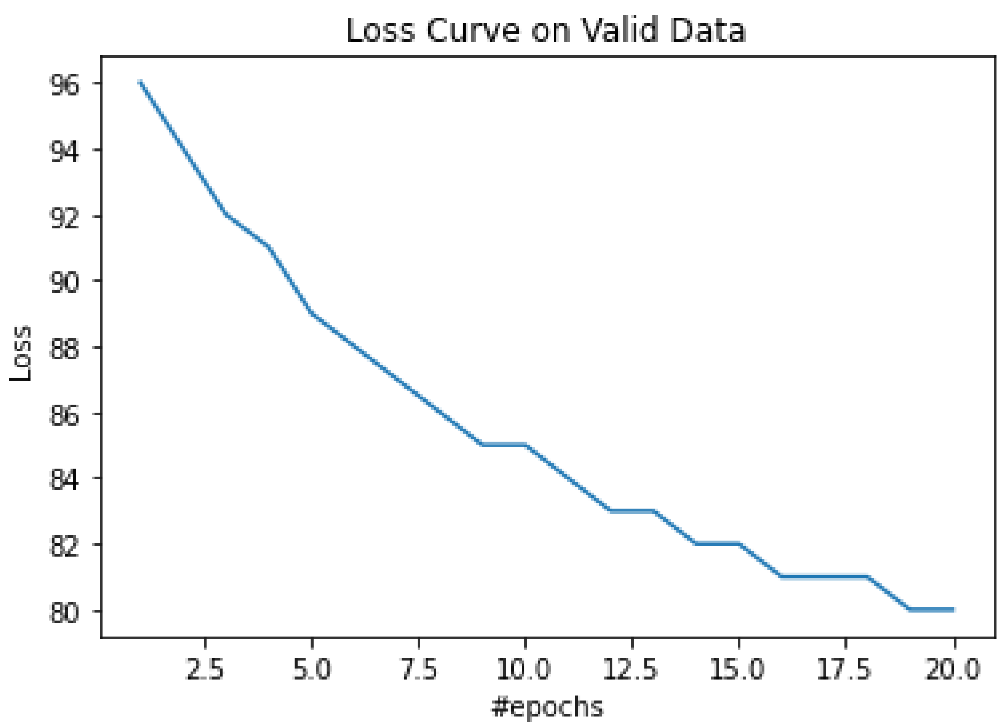


Review Classification Accuracy on Test Data = 61.20 %

Case 3: BatchSize = 32 and Learning Rate = 0.1



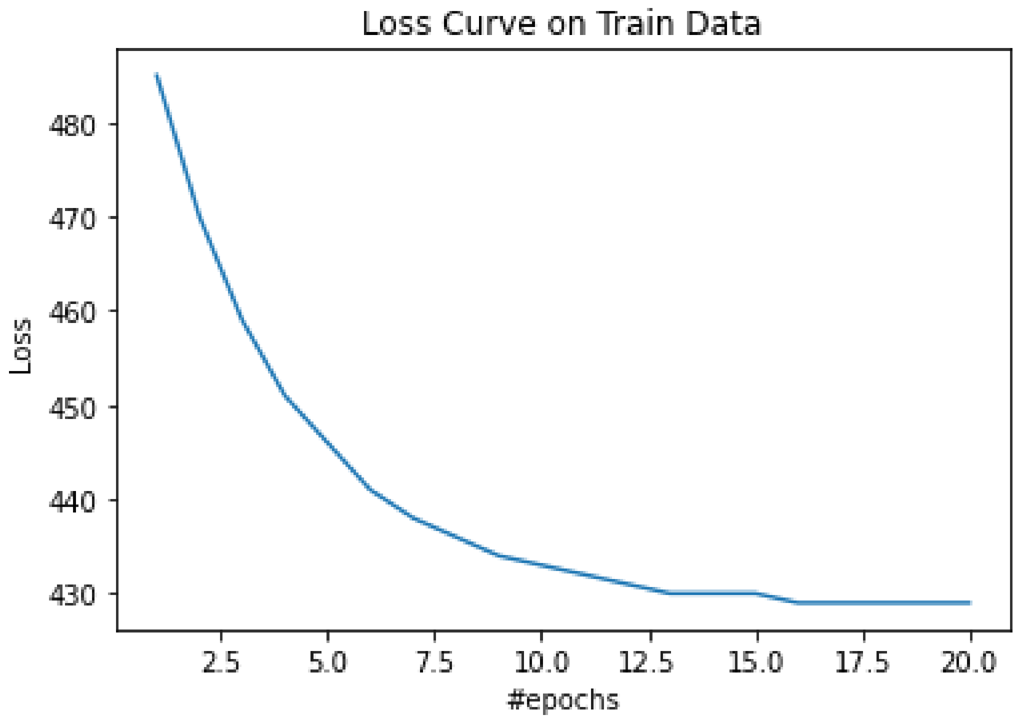
Loss Curve on Train Data
480
470
460
450
440
430
420
410
Loss
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
#epochs

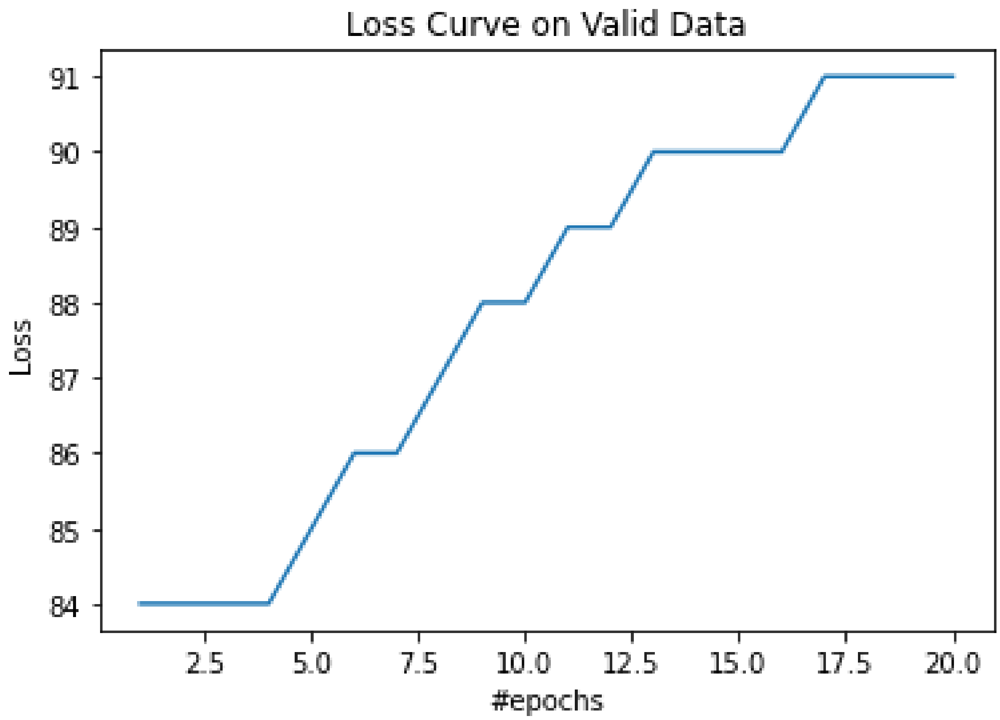


Loss Curve on Valid Data
96
94
92
90
88
86
84
82
80
Loss
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
#epochs

Review Classification Accuracy on Test Data = 61.20 %

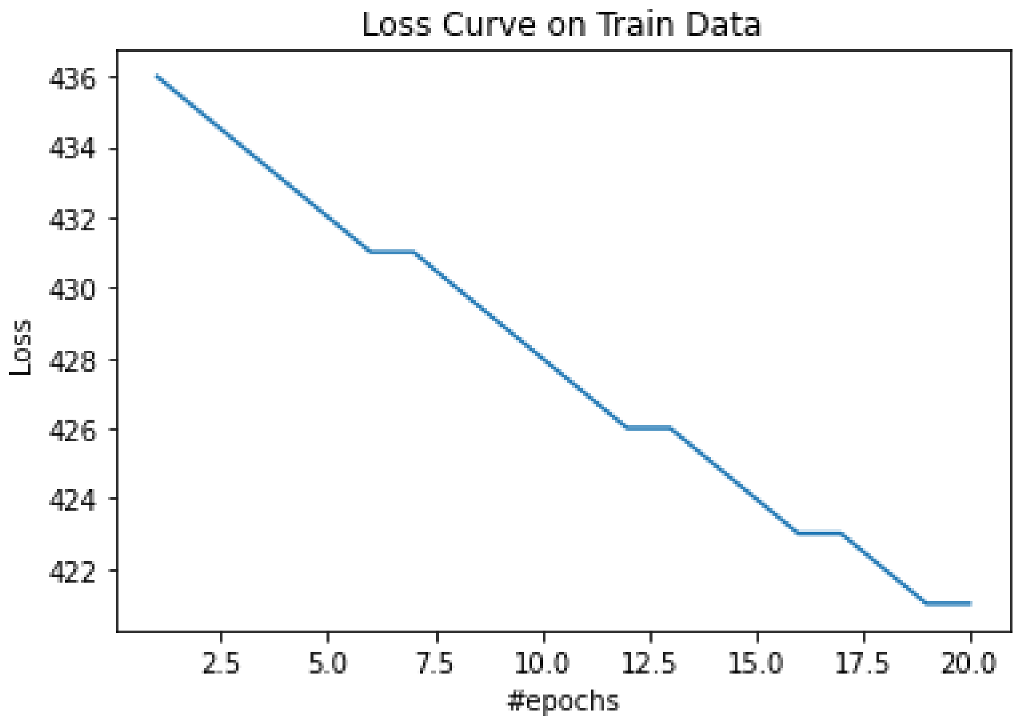
Case 4: BatchSize = 64 and Learning Rate = 0.001

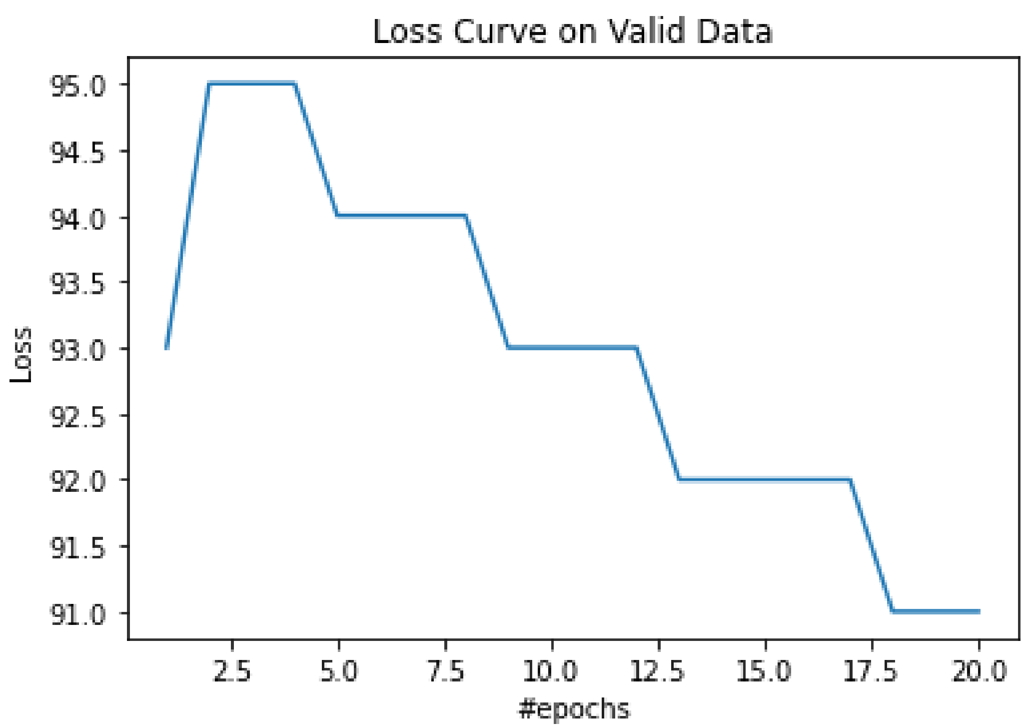




Review Classification Accuracy on Test Data = 60.80 %

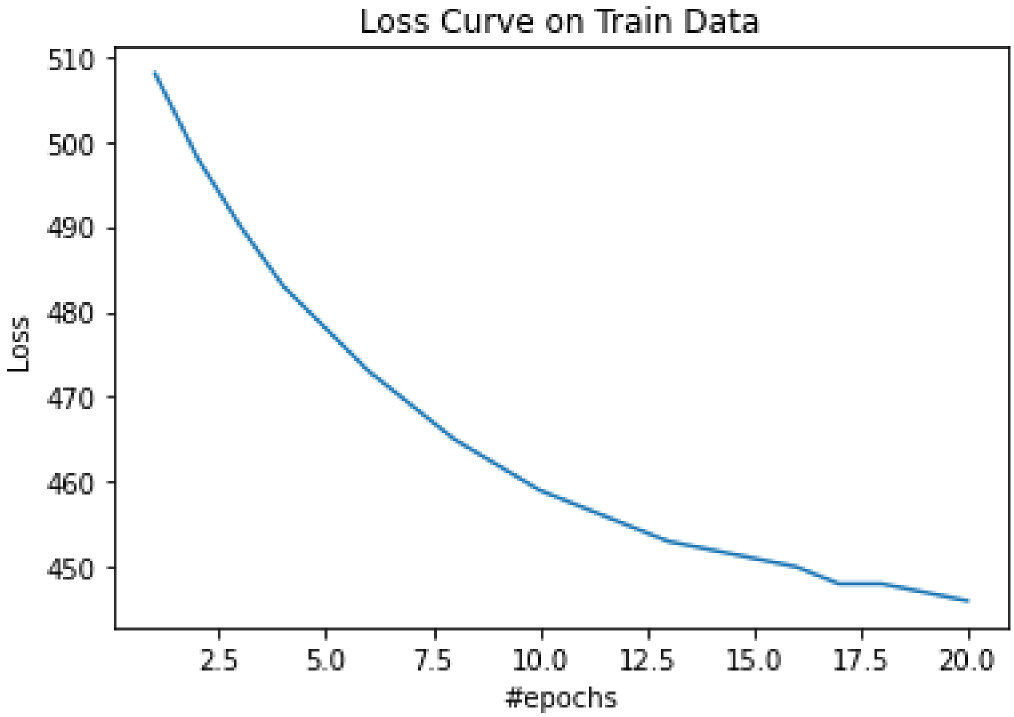
Case 5: BatchSize = 64 and Learning Rate = 0.01

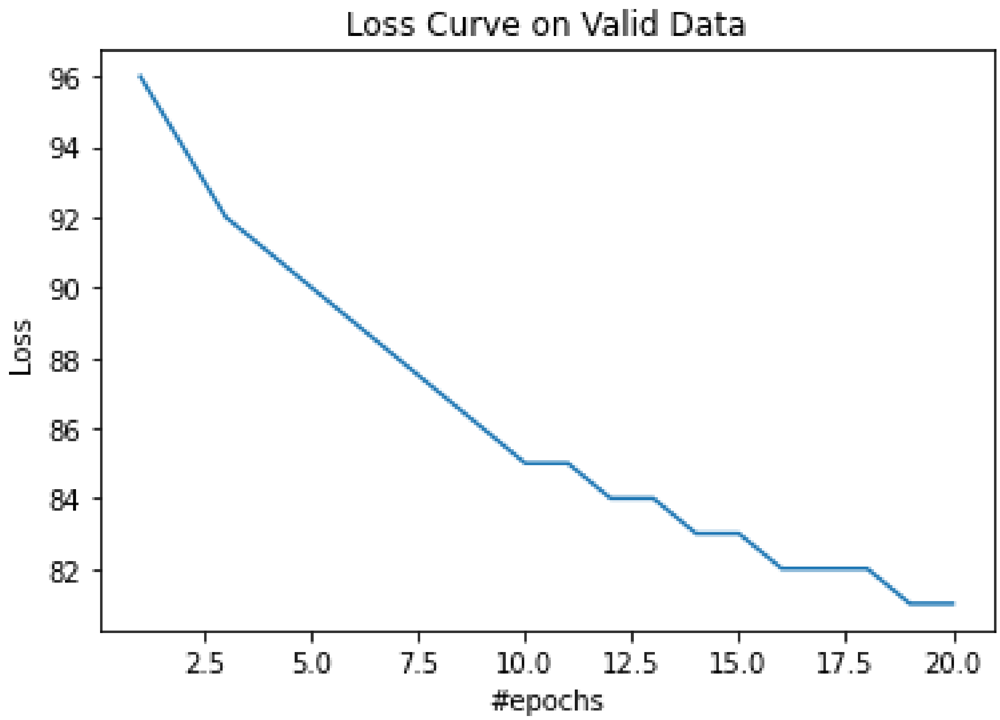

Loss Curve on Train Data
436
434
432
430
428
426
424
422
Loss
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
#epochs


Loss Curve on Valid Data
95.0
94.5
94.0
93.5
93.0
92.5
92.0
91.5
91.0
Loss
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
#epochs

Review Classification Accuracy on Test Data = 61.20 %

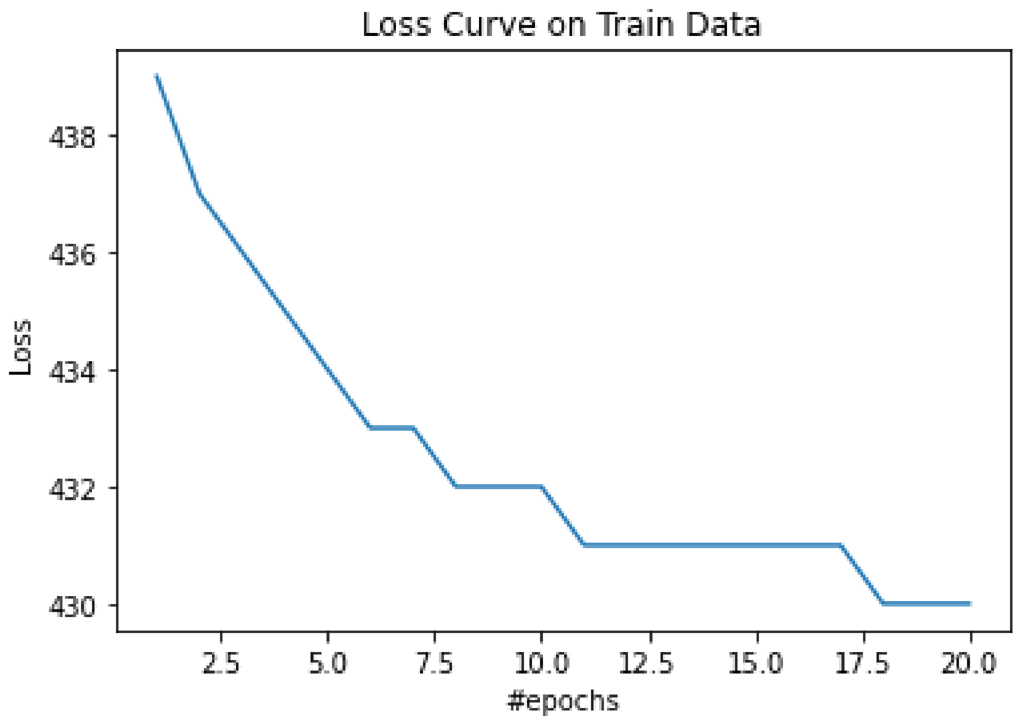
Case 6: BatchSize = 64 and Learning Rate = 0.1

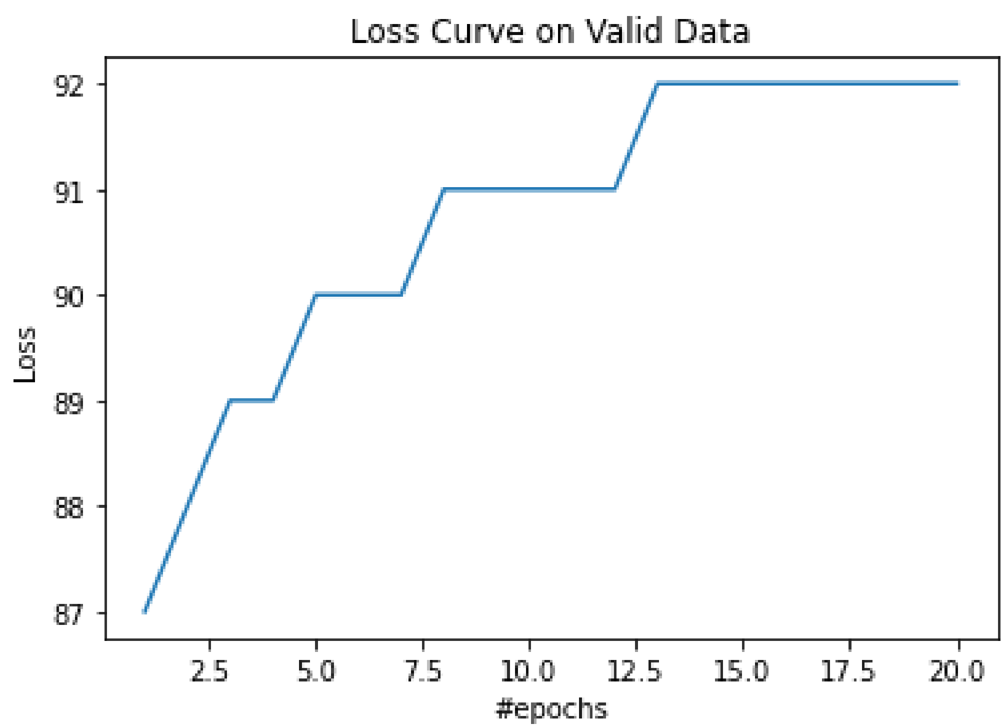




Review Classification Accuracy on Test Data = 61.20 %

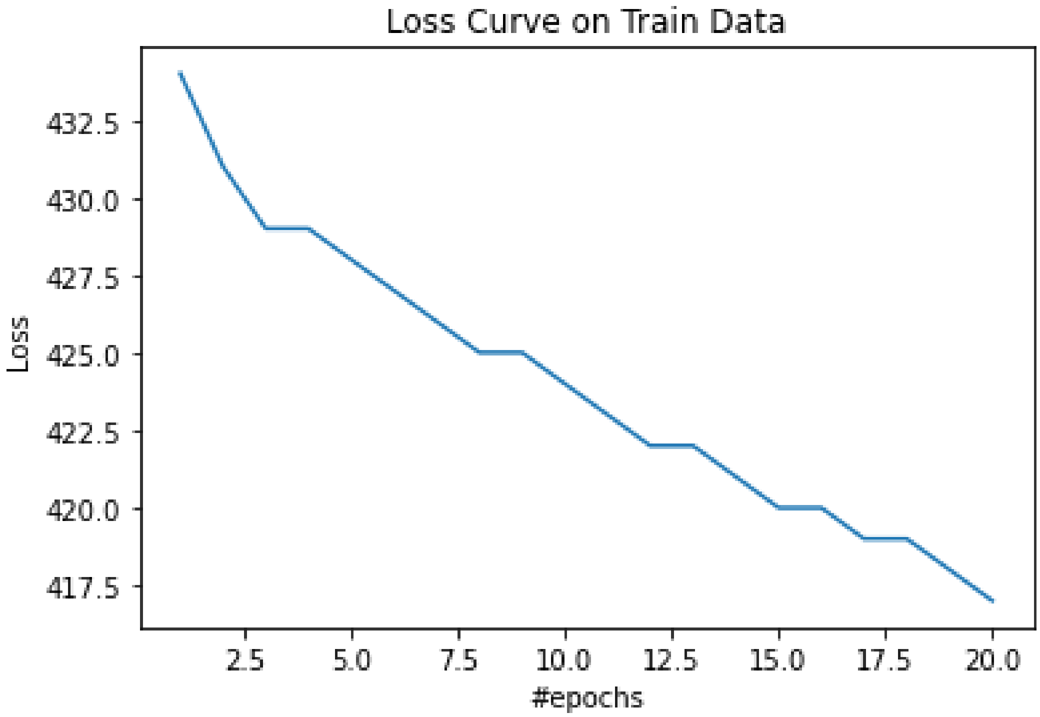
```
------------------------------------------------------------------------------
Case 7: BatchSize = 128 and Learning Rate = 0.001
```

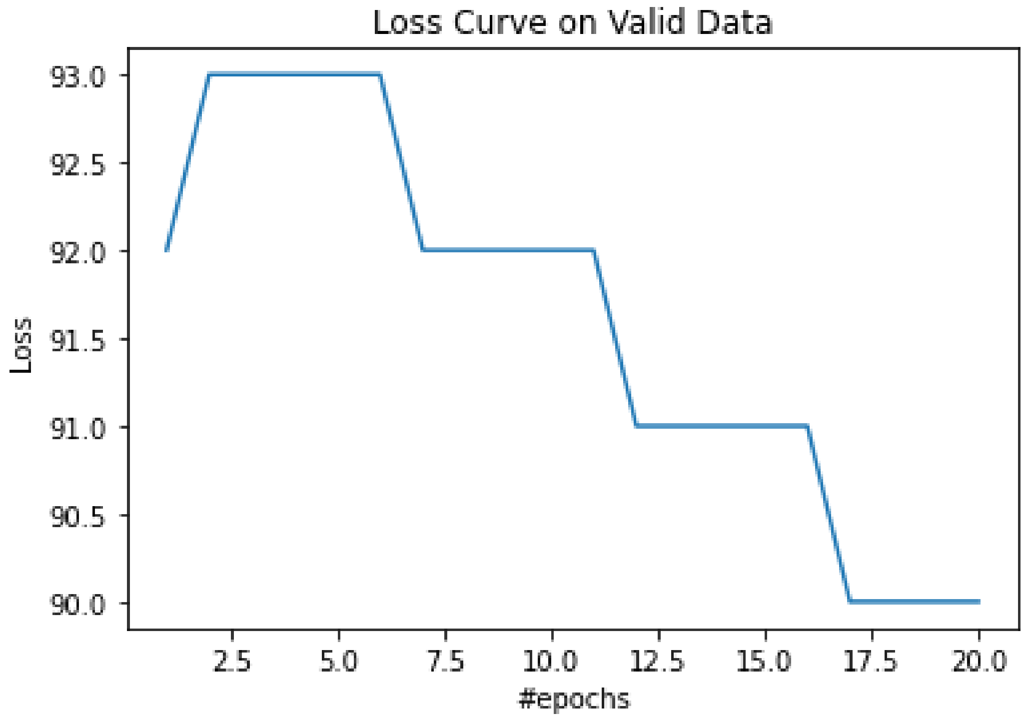

Loss Curve on Train Data
Loss
438
436
434
432
430
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
#epochs


Loss Curve on Valid Data
Loss
92
91
90
89
88
87
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
#epochs

Review Classification Accuracy on Test Data = 61.20 %

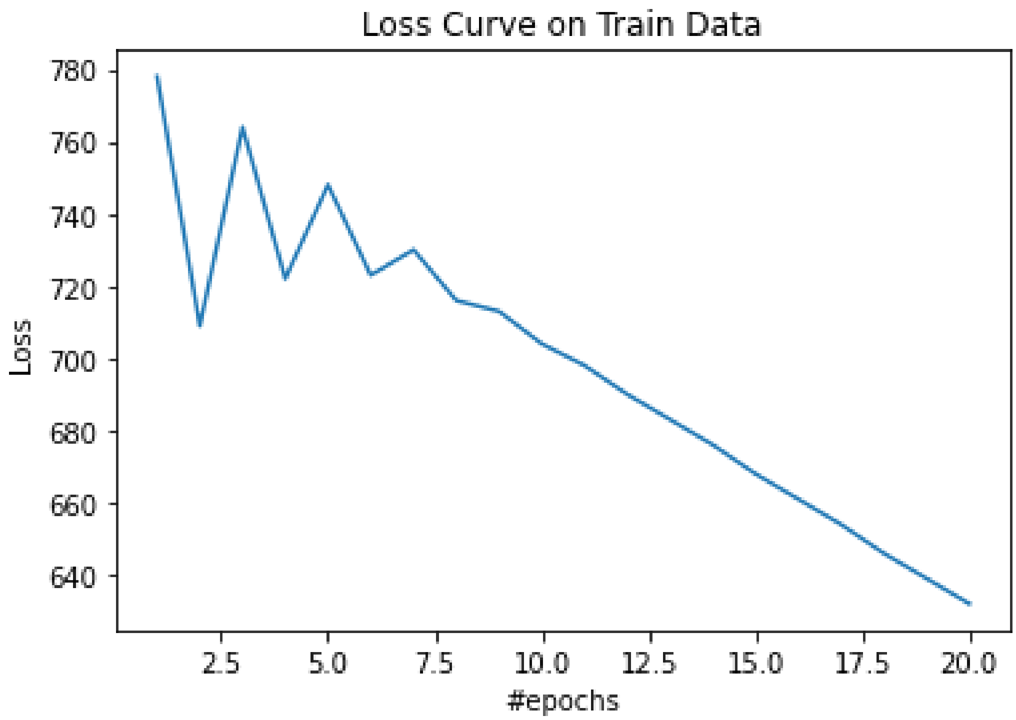
```
--------------------------------------------------------------------------------
Case 8: BatchSize = 128 and Learning Rate = 0.01
```

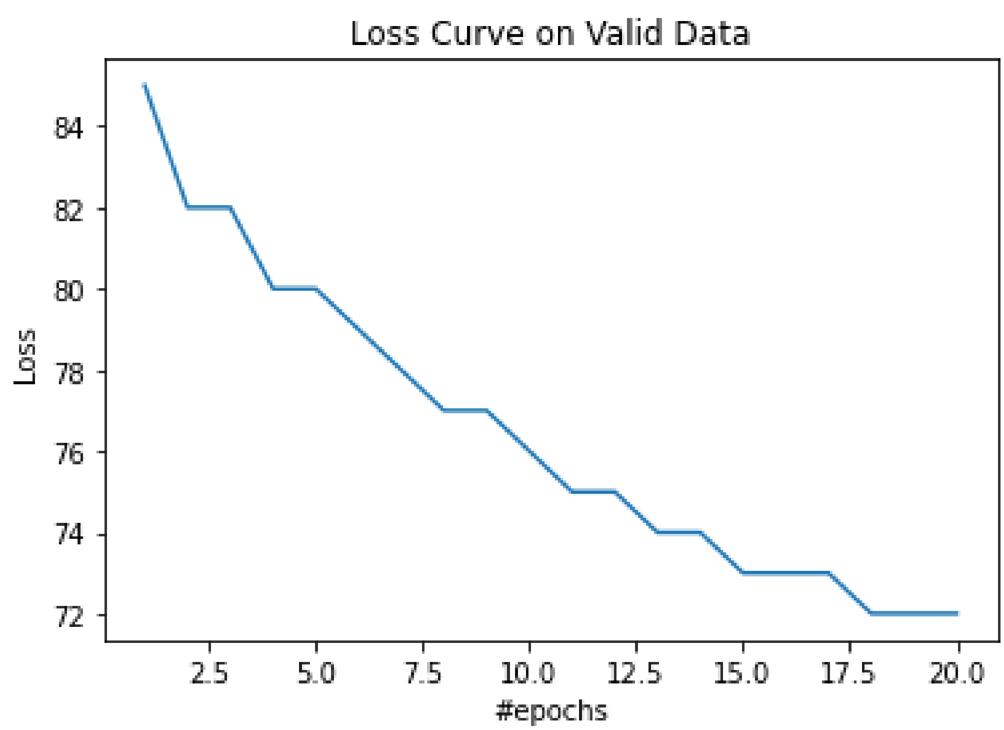




Review Classification Accuracy on Test Data = 61.20 %

Case 9: BatchSize = 128 and Learning Rate = 0.1


Loss Curve on Train Data
780
760
740
720
700
680
660
640
Loss
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
#epochs


Loss Curve on Valid Data
84
82
80
78
76
74
72
Loss
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
#epochs

```
Review Classification Accuracy on Test Data =  56.00  %
--------------------------------------------------------------------------------
```

Review classification accuracy for the nine cases is given below each cell. According to my result, I'm getting best accuracy of 61.2

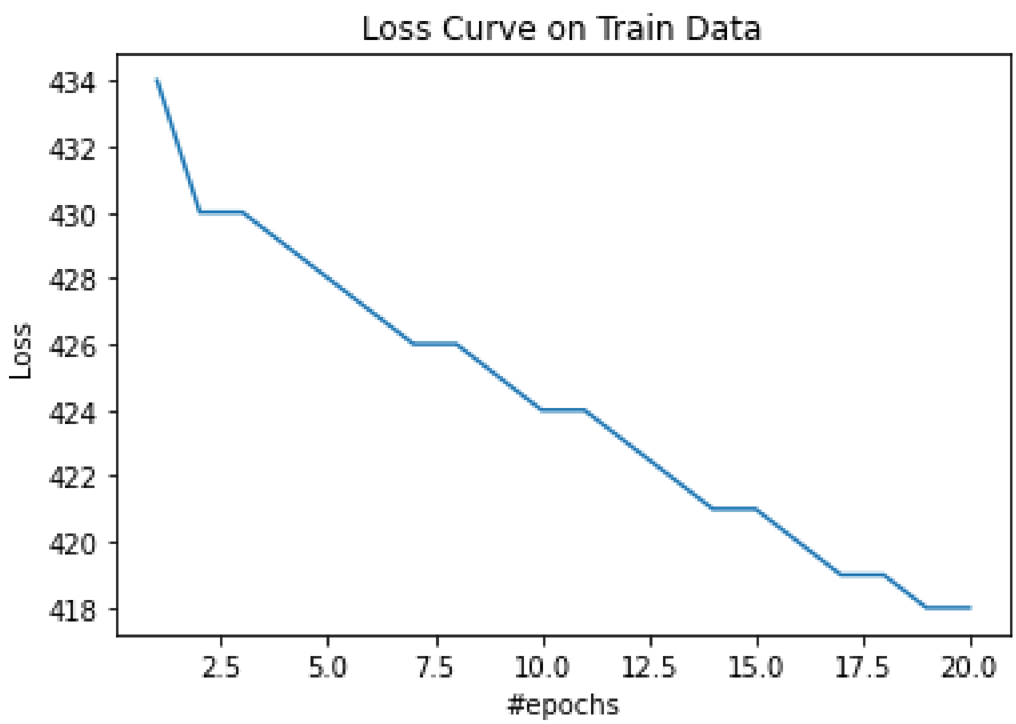
Case 3: BatchSize = 32 and Learning Rate = 0.1 Case 6: BatchSize = 64 and Learning Rate = 0.1 Case 8: BatchSize = 128 and Learning Rate = 0.01

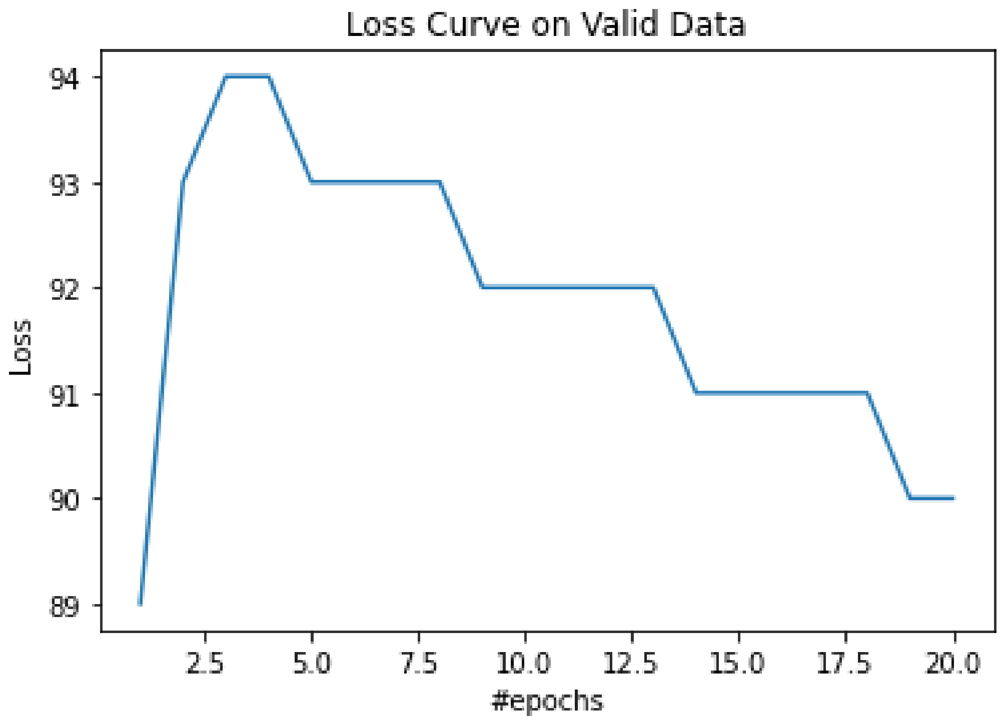
4 Part D

Out of 9 cases in Part C, only choosing a single case with the best performance for Part D. Best Case BatchSize = 64, Learning Rate = 1e-2

Case 1 : Regularization Coefficient = 1e-2

[26]:
```
Dcase1 = LogisticRegession(epochs,64,1e-2,1e-2)
Dcase1.fit(1)
Dcase1.plotLossCurve(Dcase1.lossTrain,0)
Dcase1.plotLossCurve(Dcase1.lossValid,1)
Dcase1.predict()
```

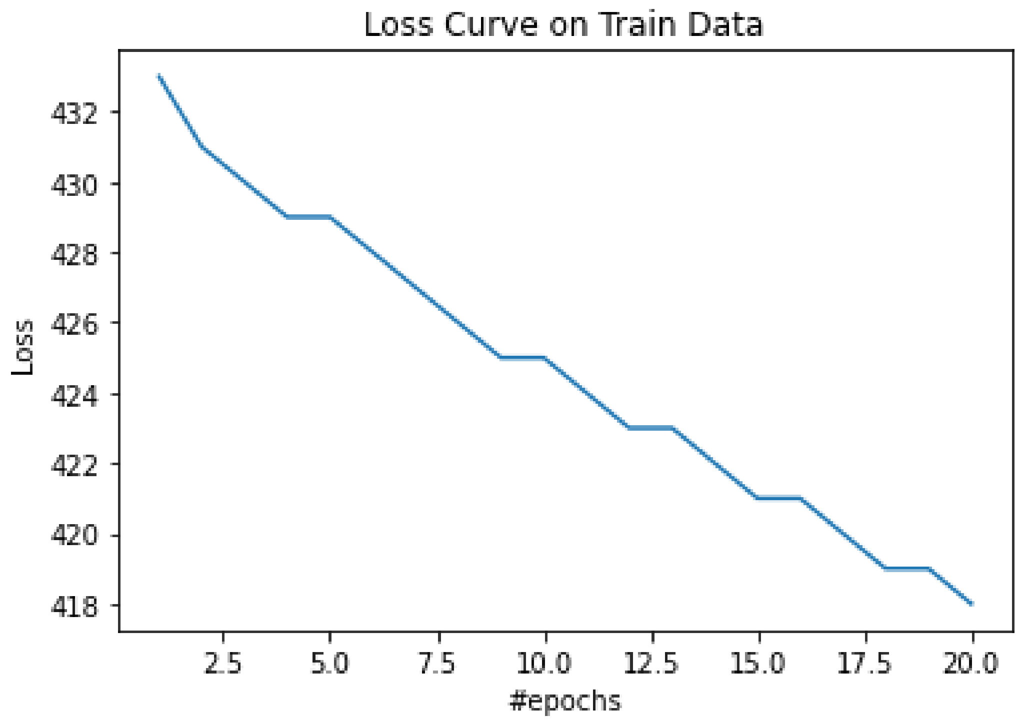


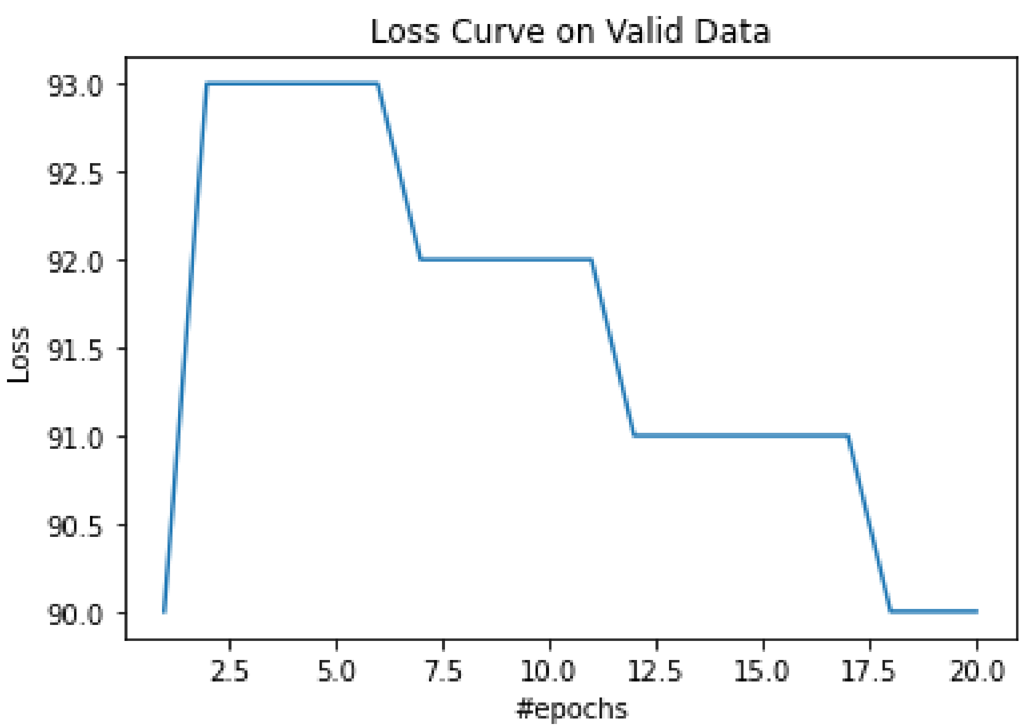



```
Review Classification Accuracy on Test Data =  61.20  %
```

Case 2 : Regularization Coefficient = 1e-1

[27]:
```
Dcase2 = LogisticRegession(epochs,64,1e-2,1e-1)
Dcase2.fit(1)
Dcase2.plotLossCurve(Dcase2.lossTrain,0)
Dcase2.plotLossCurve(Dcase2.lossValid,1)
Dcase2.predict()
```

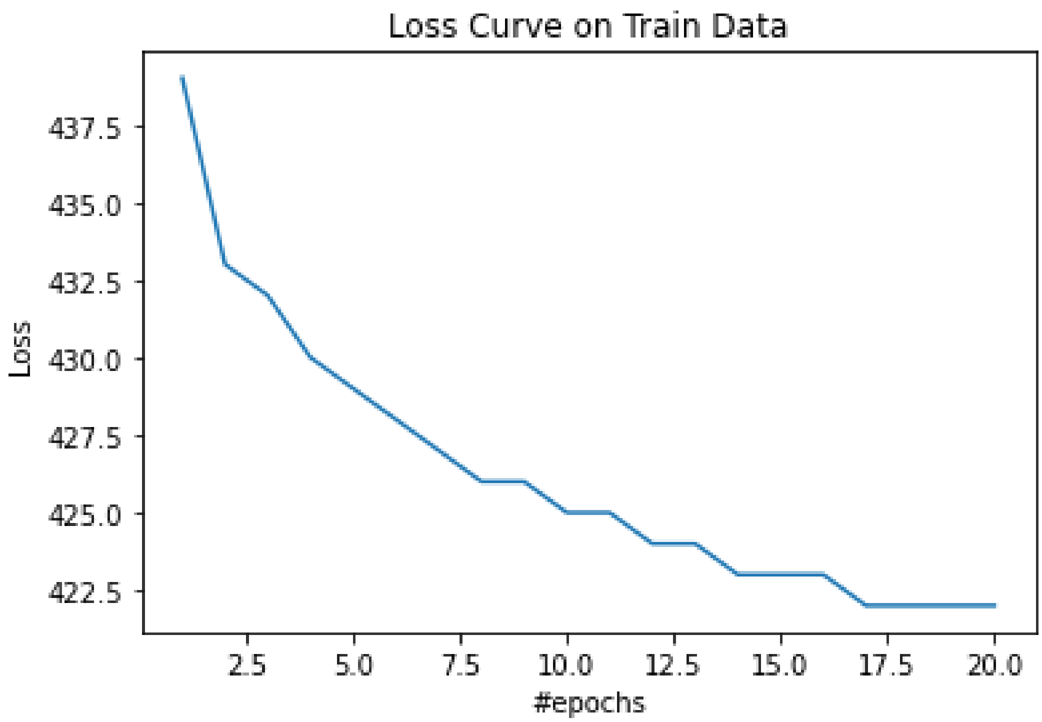

Loss Curve on Train Data
432
430
428
426
424
422
420
418
Loss
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
#epochs

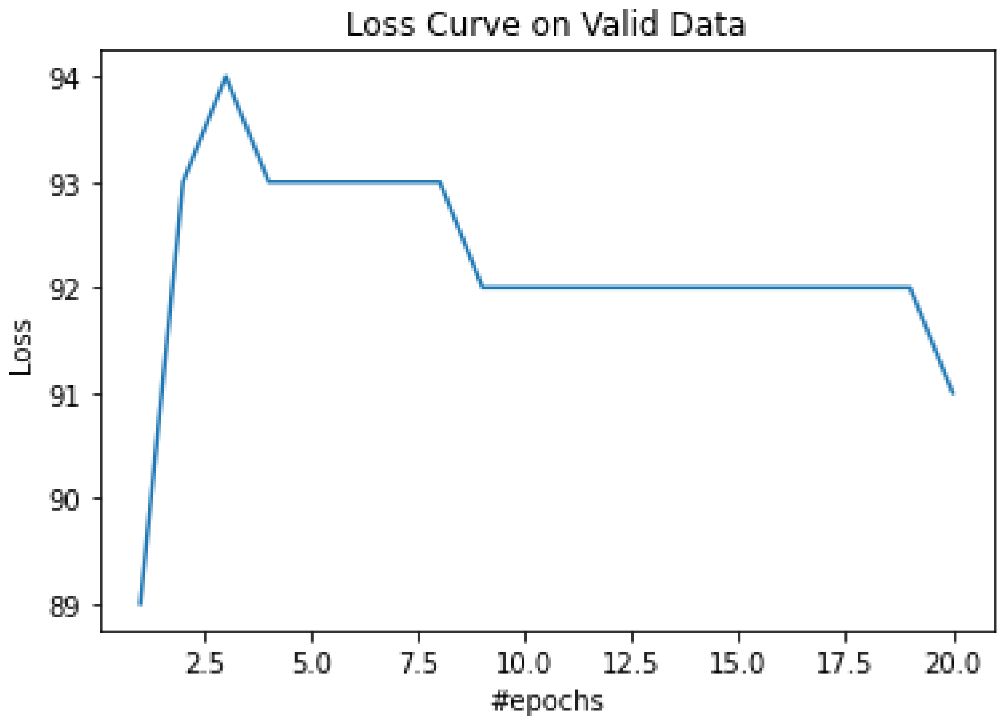

Loss Curve on Valid Data
93.0
92.5
92.0
91.5
91.0
90.5
90.0
Loss
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
#epochs

```
Review Classification Accuracy on Test Data =  61.20  %
```

Case 3 : Regularization Coefficient = 1

```
[28]: Dcase3 = LogisticRegession(epochs,64,1e-2,1)
      Dcase3.fit(1)
      Dcase3.plotLossCurve(Dcase3.lossTrain,0)
      Dcase3.plotLossCurve(Dcase3.lossValid,1)
      Dcase3.predict()
```





`Review Classification Accuracy on Test Data = 61.20 %`

When regularization coefficient is 1, overfitting is observed.