01- Supervised Learning - Classification - Logistic Regression -Binary(Solution)_st122097_thantham

August 20, 2021

1 Lab Work 01 Supervised-Learning Logistic Regression - Binary

1.1 Name: Thantham Khamyai

1.2 Student ID: 122097

2 Tasks Completed

- Create LogisticRegression Class, and set default method as 'mini-batch'
- Perform classification using given dataset creation
- Plot learning curve through epochs
- Create 'classification_report' containing 4 functions of each metric (accuracy, precision, recall, f1)

2.1 Import Neccessary Packages

[1]:
```
# Import Basic packages
import numpy as np
import matplotlib.pyplot as plt
```

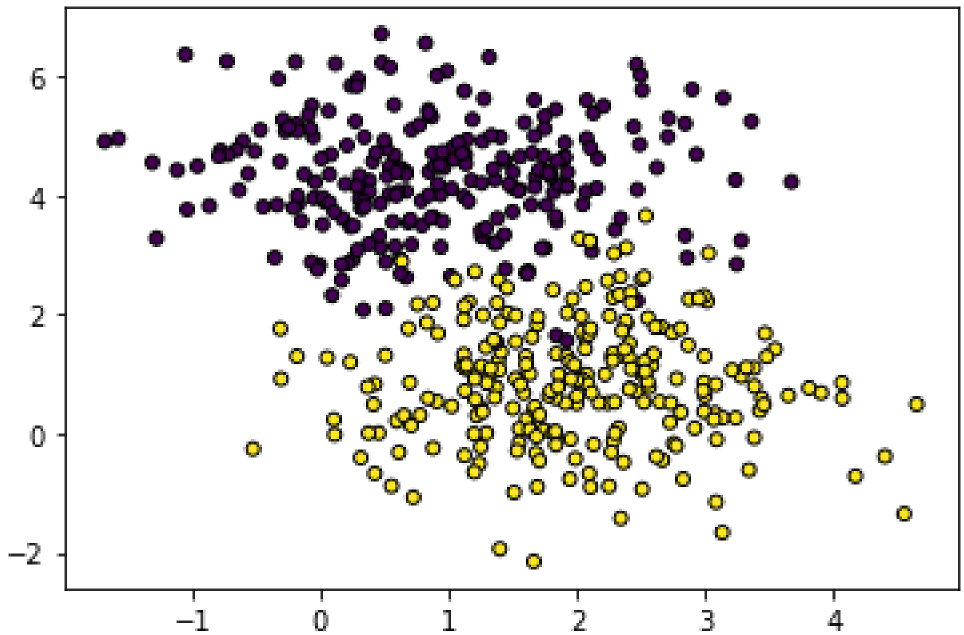
[2]:
```
# Import sklaern packages and neccessary functions
from sklearn import linear_model
from sklearn.datasets import make_classification, make_blobs
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

2.2 Implement Given Classification Dataset

[3]:
```
X, y = make_blobs(n_samples=500, centers=2, n_features=2, random_state=0) #␣
 ↪Generate isotropic Gaussian blobs for clustering

plt.scatter(X[:, 0], X[:, 1], marker='o', c=y, s=25, edgecolor='k') # show␣
 ↪scatter plot of them
```

[3]:
```
<matplotlib.collections.PathCollection at 0x7f2e34574880>
```



2.3 Feature Scaling

[4]:
```
# feature scaling helps reaching convergence faster
scaler = StandardScaler() # create Scaler instance
X = scaler.fit_transform(X) # fit and transform X data for standradization
```

2.4 Train Test Splitting

[81]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) #␣
↪split training and testing data with 70/30 ratio randomly
```

[82]:
```
X.shape
```

[82]: (500, 2)

2.5 Add Intercept terms for each train and test data

[83]:
```
# for avoiding repeatitive step of intercepts insertion, make function to do␣
↪that
def add_intercept(X):
    return np.insert(X, 0, 1, axis=1) # insert 1 at index 0 of axis 1 (column)
```

[84]:
```
X_train = add_intercept(X_train) # add intercept
X_test = add_intercept(X_test) # add intercept
```

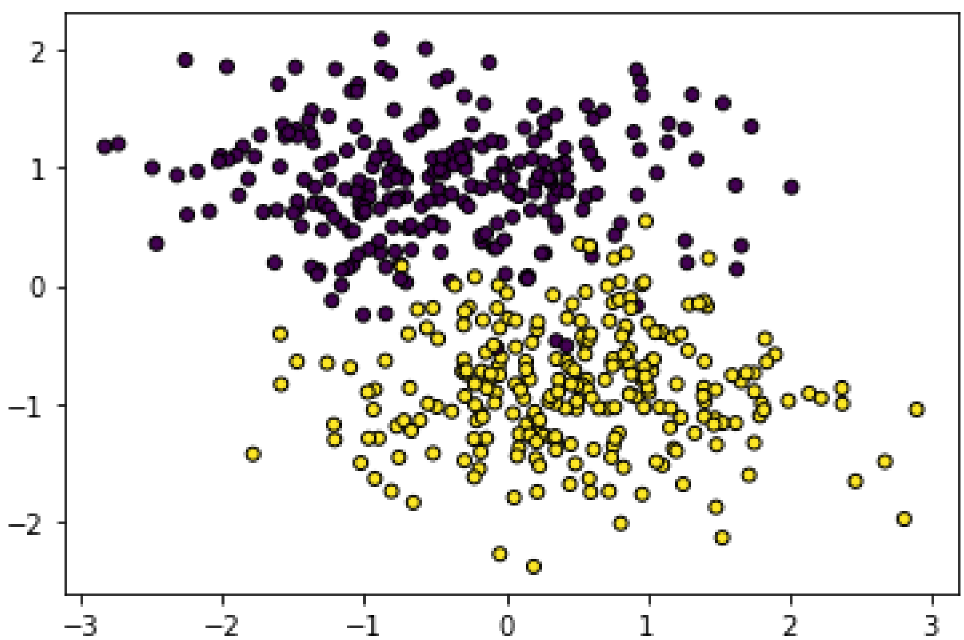
```python
print(f'shape of X_train: {X_train.shape}')
print(f'shape of X_test: {X_test.shape}')
print(f'shape of y_train: {y_train.shape}')
print(f'shape of y_test: {y_test.shape}')
```

```
shape of X_train: (350, 3)
shape of X_test: (150, 3)
shape of y_train: (350,)
shape of y_test: (150,)
```

2.6 Show training data after feature scaling

[85]:
```python
# show scatter plot to invualize data after scaling
plt.scatter(X[:, 0], X[:, 1], marker='o', c=y,
            s=25, edgecolor='k')
```

[85]:
```
<matplotlib.collections.PathCollection at 0x7f2e327f2490>
```



2.7 Task 1: Create LogisticRegression class

[86]:
```python
class LogisticRegression:

    def __init__(self, method='mini-batch', max_iterations=10000,␣
↪early_stopping=False,
```

```
                     alpha=.0001, tol=.00001, mini_batch_size=100,␣
↪previous_loss=10000,
                     record_history_every=100,  print_loss_every=1000):
        self.method = method
        self.max_iterations = max_iterations
        self.early_stopping = early_stopping
        self.alpha = alpha
        self.tol = tol
        self.mini_batch_size = mini_batch_size
        self.previous_loss = previous_loss # initial loss to investigate tol␣
↪threshold for early stopping
        self.epoch_to_print = print_loss_every # print current loss for every ..
↪. interation
        self.epoch_to_record_history = record_history_every # record loss for␣
↪every ... iteration
        self.training_history = [] # list to keep loss values from fitting


    def fit(self, X, y):
        # 1. initalize theta
        self.theta = self.init_theta(X)

        # init blank used idx list for check repeatitive idx of stochastiac␣
↪method
        idx_used = [] # list to record used idx for stochastic method
        self.training_history = [] # lise to record loss values through epochs

        # 2. loop along predefined n iterations
        for i in range(self.max_iterations):

            # 2.1 condition to choose method
            if self.method=='batch':
                # pass all samples
                x_to_train = X # dump all x
                y_to_train = y # dump sll y

            elif self.method=='stochastic': # <= With Replacement
                # randomly select 1 sample
                select_idx = np.random.randint(X.shape[0])# random idx
                while select_idx in idx_used:
                    select_idx = np.random.randint(X.shape[0])# random idx

                x_to_train = np.array([X[select_idx, :]]) # extract one X by␣
↪idx
                y_to_train = np.array([y[select_idx]]) # extract one y by idx
```

```
                idx_used.append(select_idx)

                if len(idx_used) == X.shape[0]:
                    idx_used = []

            elif self.method=='mini-batch':
                # randomly select portion of samples following predefined mini␣
↪batch size
                select_start_idx = np.random.randint(X.shape[0] - self.
↪mini_batch_size) # random starting idx
                x_to_train = X[select_start_idx:select_start_idx + self.
↪mini_batch_size, :] # extract portion of X
                y_to_train = y[select_start_idx:select_start_idx + self.
↪mini_batch_size] # extract portion of y

            else:
                print('''wrong method defined 'batch','stochastic','mini-batch'␣
↪only''')
                return

            # 2.2 predict y hat by dot x_to_train with theta
            yhat = self.predict(x_to_train)

            # 2.3 calculate error by minus yhst with y_to_train
            error = yhat - y_to_train

            # 2.4 calculate current mse to detect early stopping
            current_loss = self.loss(yhat, y_to_train)

            # 2.5 if early stopping set as True & difference of current and␣
↪previous loss is less than threshold
            if self.early_stopping & (np.abs(self.previous_loss - current_loss)␣
↪< self.tol):
                self.stop_epoch = i # keep early stopping iteration in␣
↪stop_epoch variable
                # print early stopped epoch and exit loop
                print(f'early_stopped at epoch: {i+1}')
                break

            # 2.6 if not early stop or set False, update previous loss
            self.previous_loss = current_loss

            # 2.7 calculate gradient of trainingdata
            grad = self.gradient(x_to_train, error)

            # 2.8 update theta
```

```
            self.theta = self.theta - self.alpha * grad

            # add history loss
            if i % self.epoch_to_record_history ==0: # if this iteration is every ... for recording loss
                self.training_history.append(current_loss) # save this loss

            # print current loss
            if i % self.epoch_to_print == 0: # if this iteration is every ... for printing loss
                print(f'loss at epoch {i}: {current_loss}') # print current iteration loss

        self.stop_epoch = i # if no early stopping -> keep last iteration number to stop_epoch
        print(f'fitting model completed by loss: {current_loss}')

    def show_history(self):

        if len(self.training_history) == 0: # if no loss in history list
            print('hitory is empty!, fit model before!')
        else: # else show learning curve
            plt.plot(np.arange(start = 1, stop = self.stop_epoch, step=self.epoch_to_record_history) , self.training_history, label = "Train Losses")
            plt.title("Losses thourgh learning curve")
            plt.xlabel("number of epoch")
            plt.ylabel("losses")
            plt.legend()

    # function to predict yhat
    def predict(self, X):
        return self.sigmoid(X @ self.theta) # put h in sigmoid function

    # function to calculate loss
    def loss(self, yhat, y):
        return - np.sum(y * np.log(yhat) + (1 - y) * np.log(1 - yhat)) # losgistic loss function

    # function to calculate gradient
    def gradient(self, X, error):
        return X.T @ error

    # function to create initial theta
    def init_theta(self, X):
        return np.zeros((X.shape[1])) # fill all theta with 0
```

```
    # function to return sigmoid
    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x)) # sigmoid function

    def round_pred(self, pred):
        return np.round(pred) # use for rounding predicted y for classification␣
 ↪report check
```

2.8 Task 2: Perform classification

2.8.1 2.1 Create model instance ('mini-batch' set as task defined)

[91]:
```
# selective methods are 'batch', 'mini-batch', 'stochastic'

model = LogisticRegression(method='mini-batch', max_iterations=30000,␣
 ↪early_stopping=True,
                           alpha=.001, tol=.00001, mini_batch_size=100,␣
 ↪record_history_every=200,print_loss_every=500)
```

2.8.2 2.2 perform classification (implementing early stopping also)

[92]:
```
model.fit(X_train, y_train) # ftting model
```

```
loss at epoch 0: 69.31471805599453
loss at epoch 500: 12.490181569275114
loss at epoch 1000: 9.308949423589029
loss at epoch 1500: 12.882894509350729
loss at epoch 2000: 10.798859219659597
loss at epoch 2500: 8.817050424532924
loss at epoch 3000: 12.628672346094012
loss at epoch 3500: 10.303887546243198
loss at epoch 4000: 8.247411809339916
loss at epoch 4500: 10.792251243907183
loss at epoch 5000: 7.63648071660539
loss at epoch 5500: 7.864112148402541
loss at epoch 6000: 7.86053658204818
loss at epoch 6500: 9.95143979551892
loss at epoch 7000: 8.263332322376481
loss at epoch 7500: 7.619907835357193
loss at epoch 8000: 6.588487481137533
loss at epoch 8500: 3.0993788031224634
loss at epoch 9000: 6.531574059067772
loss at epoch 9500: 9.39159704858699
loss at epoch 10000: 9.483202765036902
loss at epoch 10500: 12.820689460535139
loss at epoch 11000: 2.797131171520912
loss at epoch 11500: 10.66697202249538
```

```
loss at epoch 12000: 10.269834608055282
loss at epoch 12500: 7.668267733948334
loss at epoch 13000: 2.772471248452014
loss at epoch 13500: 8.832524574402784
loss at epoch 14000: 9.306665159190828
loss at epoch 14500: 9.755115325110703
loss at epoch 15000: 7.578097167925002
loss at epoch 15500: 8.97887372237708
loss at epoch 16000: 10.846363007916008
loss at epoch 16500: 10.081343777413188
loss at epoch 17000: 8.341425260519697
loss at epoch 17500: 7.5602229366335
loss at epoch 18000: 10.435349878404482
loss at epoch 18500: 8.049475222684938
loss at epoch 19000: 8.356370267693352
loss at epoch 19500: 9.990008914787651
loss at epoch 20000: 9.590866755991033
loss at epoch 20500: 7.617735085421128
loss at epoch 21000: 2.946431296704531
loss at epoch 21500: 8.985264792538459
loss at epoch 22000: 2.7833942752567618
early_stopped at epoch: 22417
fitting model completed by loss: 9.56030741698966
```

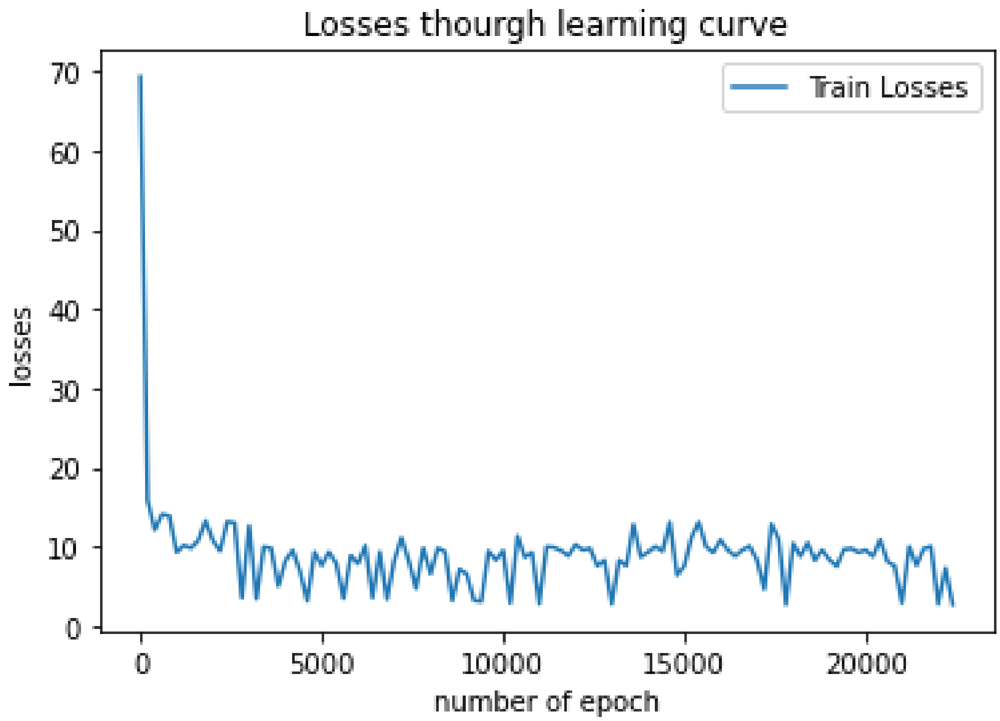
2.8.3 2.3 predicting y by x_test and show training loss

[93]:
```python
y_pred = model.predict(X_test)
loss = model.loss(y_pred, y_test)
print(f'Testing loss: {loss}')
```

```
Testing loss: 19.85861791732553
```

2.9 Task 3: Plot Learning

[94]:
```python
# just use show history functionfrom modelinstance
model.show_history()
```



2.10 Task 4: Create classification_report class and evaluate model using created class

2.10.1 4.1 Create class of classification_report

[95]:
```
class classification_report2(): #<== add '2' after classname because of
↪preventing conflict with sklearn's classification_report

    def __init__(self, actual, predict):

        self.actual = actual
        self.predict = predict

        self.TP = ((self.actual == 1) & (self.predict == 1)).sum() # True
↪Positive (correct prediction)
        self.TN = ((self.actual == 0) & (self.predict == 0)).sum() # True
↪Negative (correct prediction)
        self.FN = ((self.actual == 1) & (self.predict == 0)).sum() # False
↪Negative (Predict as No, but actually Yes)
        self.FP = ((self.actual == 0) & (self.predict == 1)).sum() # False
↪Positive (Predict as Yes, but actually No)

    def accuracy(self):
```

```
        # Accuracy = (TP+TN)/(TP+TN+FN+FP)
        self.acc = 100 * (self.TP + self.TN)/ float( self.TP + self.TN + self.
↪FN + self.FP)
        return self.acc

    def precision(self):
        # Precision = (TP)/(TP+FP)
        self.precision = 100* (self.TP)/ float(self.TP + self.FP)
        return self.precision

    def recall(self):
        # Recall = (TP)/(TP+FN)
        self.recall = (100* self.TP)/ float(self.TP + self.FN)
        return self.recall

    def f1(self):
        # F1 = 2 * (Precision * Recall) / (Precision + Recall)
        self.f1 = 2 * self.precision * self.recall / (self.precision + self.
↪recall)
        return self.f1

    # ADDITIONAL FUNCTION TO SIMPLY PRINTING METRICS
    def show_metrics(self):
        print('Model Evaluation')
        print(f'Evaluation - Accuracy : {self.accuracy()} %')
        print(f'Evaluation - Precision : {self.precision()} %')
        print(f'Evaluation - Recall : {self.recall()} %')
        print(f'Evaluation - F1 : {self.f1()} %')
```

2.10.2 4.2 Implement created classificaiton report class

[96]:
```
report = classification_report2(y_test, model.round_pred(y_pred)) # <==

report.show_metrics()
```

```
Model Evaluation
Evaluation - Accuracy : 96.66666666666667 %
Evaluation - Precision : 96.15384615384616 %
Evaluation - Recall : 97.40259740259741 %
Evaluation - F1 : 96.7741935483871 %
```

2.10.3 ! Case of using classificaiton report from sklearn.metrics

[97]:
```
from sklearn.metrics import classification_report

print('Scikit-learn Classification Report: \n{}'.
↪format(classification_report(y_test, model.round_pred(y_pred))))
```

```
Scikit-learn Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.96      0.97        73
           1       0.96      0.97      0.97        77

    accuracy                           0.97       150
   macro avg       0.97      0.97      0.97       150
weighted avg       0.97      0.97      0.97       150
```

3 Additional Tasks

3.1 Implement 'batch' gradient descent training

[98]:
```
# Instantiate logistic regression model with 'batch' method
model_batch = LogisticRegression(method='batch', max_iterations=30000,␣
 ↪early_stopping=True,
                           alpha=.0001, tol=.00001,␣
 ↪record_history_every=200,print_loss_every=1000)

# fitting model
model_batch.fit(X_train, y_train) # fitting
y_batch_pred = model_batch.predict(X_test) # predict y_pred

# Show results
print('\n Scratch Classification report:') # show header of classification␣
 ↪report from scratch
classification_report2(y_test, model_batch.round_pred(y_batch_pred)).
 ↪show_metrics() # print classification report from scratch
print('\n Scikit-learn Classification Report: \n{} \n'.
 ↪format(classification_report(y_test, model_batch.round_pred(y_batch_pred)))␣
 ↪# print sklearn's classification report
model_batch.show_history() # show training history
```

```
loss at epoch 0: 242.60151319598086
loss at epoch 1000: 42.896470076003475
loss at epoch 2000: 35.463100423079446
loss at epoch 3000: 32.70368701442028
loss at epoch 4000: 31.27052093272336
loss at epoch 5000: 30.40369877227689
loss at epoch 6000: 29.831200882055178
loss at epoch 7000: 29.430733833939716
loss at epoch 8000: 29.139034729382814
loss at epoch 9000: 28.92008424107431
loss at epoch 10000: 28.751890396158636
loss at epoch 11000: 28.620291851537267
```

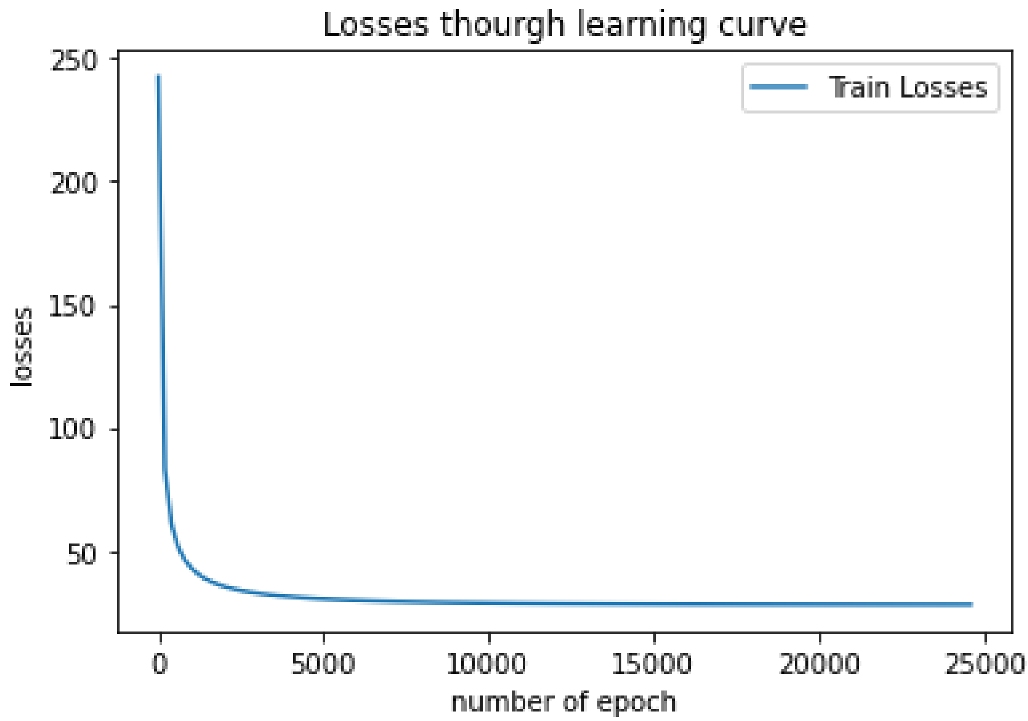
```
loss at epoch 12000: 28.51577965385196
loss at epoch 13000: 28.43174933555552
loss at epoch 14000: 28.36348434688063
loss at epoch 15000: 28.307537076294626
loss at epoch 16000: 28.26133723511853
loss at epoch 17000: 28.222935904661412
loss at epoch 18000: 28.19083351368634
loss at epoch 19000: 28.16386138805337
loss at epoch 20000: 28.14109843859212
loss at epoch 21000: 28.121811454597818
loss at epoch 22000: 28.105411595420904
loss at epoch 23000: 28.091422209529306
loss at epoch 24000: 28.079454710725756
early_stopped at epoch: 24667
fitting model completed by loss: 28.072446445403322

 Scratch Classification report:
Model Evaluation
Evaluation - Accuracy : 96.66666666666667 %
Evaluation - Precision : 96.15384615384616 %
Evaluation - Recall : 97.40259740259741 %
Evaluation - F1 : 96.7741935483871 %

 Scikit-learn Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.96      0.97        73
           1       0.96      0.97      0.97        77

    accuracy                           0.97       150
   macro avg       0.97      0.97      0.97       150
weighted avg       0.97      0.97      0.97       150
```



3.2 Implement 'stochastic' gradient descent training

[102]:
```
# Instantiate logistic regression model with 'stochastic' method
model_sto = LogisticRegression(method='stochastic', max_iterations=30000,␣
↪early_stopping=False,
                              alpha=.003, tol=.00001,␣
↪record_history_every=200,print_loss_every=1000)

# fitting model
model_sto.fit(X_train, y_train) # fitting
y_sto_pred = model_sto.predict(X_test) # predict y_pred

print('\n Scratch Classification report:') # show header of classification␣
↪report from scratch
classification_report2(y_test, model_sto.round_pred(y_sto_pred)).show_metrics()␣
↪# print classification report from scratch
print('\n Scikit-learn Classification Report: \n{}'.
↪format(classification_report(y_test, model_sto.round_pred(y_sto_pred)))) #␣
↪print sklearn's classification report
model_sto.show_history() # show training history
```

```
loss at epoch 0: 0.6931471805599453
loss at epoch 1000: 0.35571692007468564
```

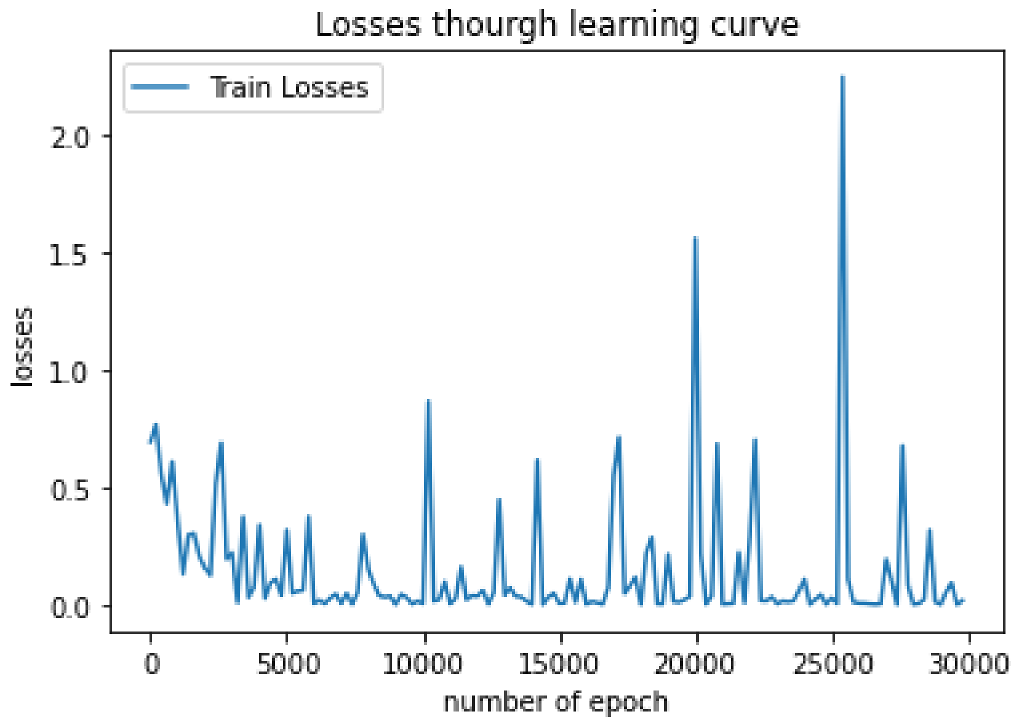
```
loss at epoch 2000: 0.15541742036559283
loss at epoch 3000: 0.221507601199212
loss at epoch 4000: 0.3389651340681303
loss at epoch 5000: 0.318973008293097
loss at epoch 6000: 0.00483230182729728
loss at epoch 7000: 0.00429480337617706
loss at epoch 8000: 0.14677768501300537
loss at epoch 9000: 0.0005378604076623165
loss at epoch 10000: 0.004835249496437901
loss at epoch 11000: 0.0035639750626063555
loss at epoch 12000: 0.03561496997749992
loss at epoch 13000: 0.04073750647505176
loss at epoch 14000: 0.001126540595838642
loss at epoch 15000: 0.0057130932276017255
loss at epoch 16000: 0.0019803619916849294
loss at epoch 17000: 0.5517707994730998
loss at epoch 18000: 0.0010688550434514975
loss at epoch 19000: 0.21486445665177917
loss at epoch 20000: 1.559956016236564
loss at epoch 21000: 0.0022147408070564024
loss at epoch 22000: 0.2884858587184741
loss at epoch 23000: 0.0037173522061345696
loss at epoch 24000: 0.10850048765894664
loss at epoch 25000: 0.02919791604255811
loss at epoch 26000: 0.006382853433155805
loss at epoch 27000: 0.19254989429177008
loss at epoch 28000: 0.0012949292066469639
loss at epoch 29000: 0.0006699425096270181
fitting model completed by loss: 0.056806342660580066

 Scratch Classification report:
Model Evaluation
Evaluation - Accuracy : 96.0 %
Evaluation - Precision : 96.1038961038961 %
Evaluation - Recall : 96.1038961038961 %
Evaluation - F1 : 96.10389610389609 %

 Scikit-learn Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        73
           1       0.96      0.96      0.96        77

    accuracy                           0.96       150
   macro avg       0.96      0.96      0.96       150
weighted avg       0.96      0.96      0.96       150
```


Losses thourgh learning curve
Train Losses
losses
number of epoch
0.0
0.5
1.0
1.5
2.0
0
5000
10000
15000
20000
25000
30000

[ ]: