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

August 16, 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 numpy as np
import matplotlib.pyplot as plt
```

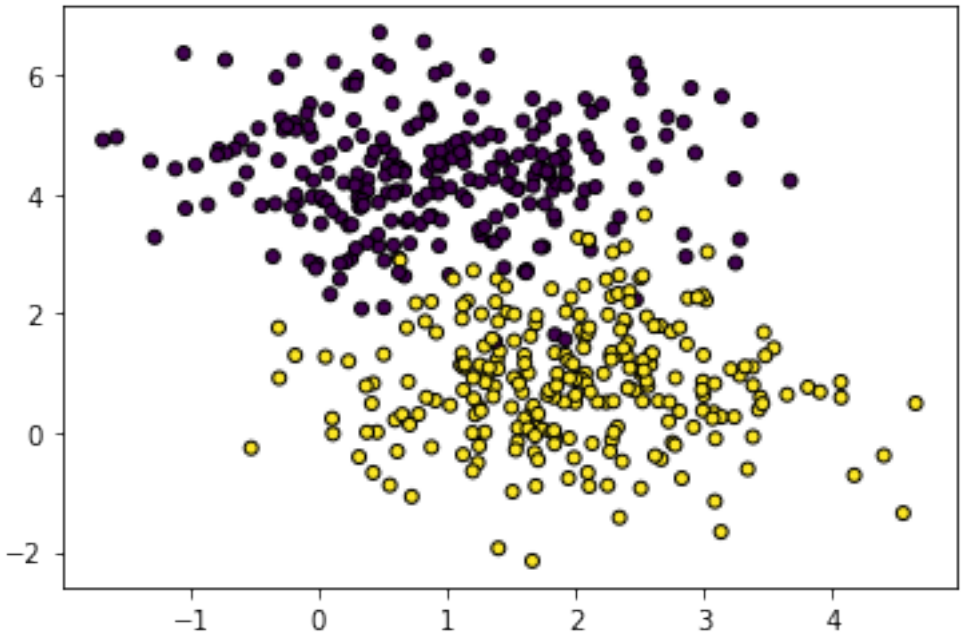
[2]:
```
from sklearn import linear_model
from sklearn.datasets import make_classification, make_blobs
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

2.2 Implement Given Classification Dataset

[3]:
```
X, y = make_blobs(n_samples=500, centers=2, n_features=2, random_state=0)

plt.scatter(X[:, 0], X[:, 1], marker='o', c=y, s=25, edgecolor='k')
```

[3]:
```
<matplotlib.collections.PathCollection at 0x7efe26ccb9d0>
```



2.3 Feature Scaling

[4]:
```
# feature scaling helps reaching convergence faster
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

2.4 Train Test Splitting

[5]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

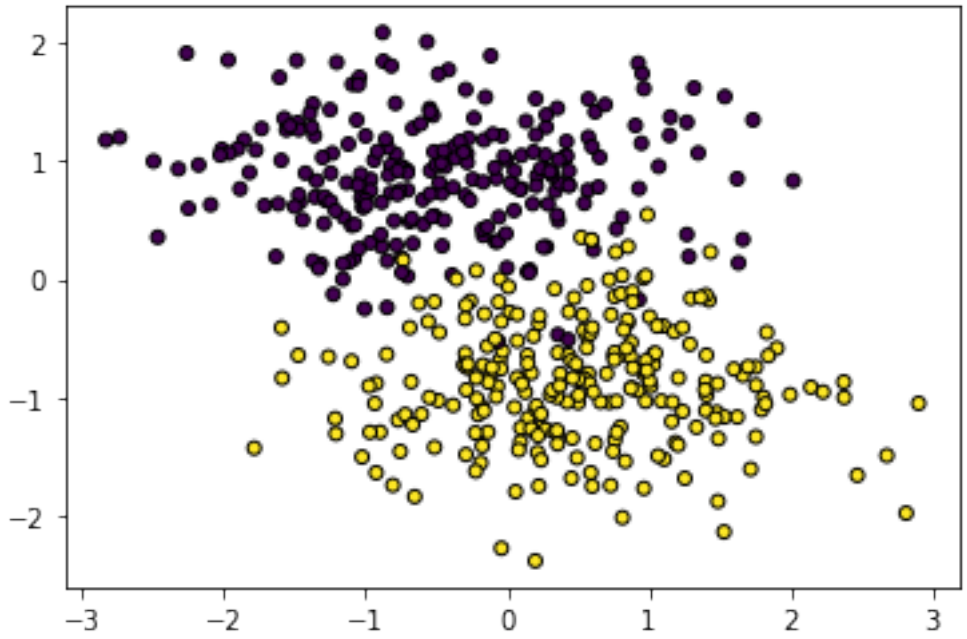
2.5 Add Intercept terms for each train and test data

[6]:
```
intercept = np.ones((X_train.shape[0], 1))
X_train   = np.concatenate((intercept, X_train), axis=1)  # add intercept
intercept = np.ones((X_test.shape[0], 1))
X_test    = np.concatenate((intercept, X_test), axis=1)  # add intercept
```

2.6 Show training data after feature scaling

[7]:
```
plt.scatter(X[:, 0], X[:, 1], marker='o', c=y,
            s=25, edgecolor='k')
```

[7]:
```
<matplotlib.collections.PathCollection at 0x7efe253f2df0>
```



2.7 Task 1: Create LogisticRegression class

[8]:
```
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
        self.epoch_to_print = print_loss_every
        self.epoch_to_record_history = record_history_every
        self.training_history = []


    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 = []
        self.training_history = []

        # 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
                # 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:
                self.training_history.append(current_loss)

            if i % self.epoch_to_print == 0:
                print(f'loss at epoch {i}: {current_loss}')

        self.stop_epoch = i
        print(f'fitting model completed by loss: {current_loss}')

    def show_history(self):

        if len(self.training_history) == 0:
            print('hitory is empty!, fit model before!')
        else:
            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)

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

    # 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]))

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

    def round_pred(self, pred):
        return np.round(pred)
```

2.8 Task 2: Perform classification

[41]:
```
# selective methods are 'batch', 'mini-batch', 'stochastic'

model = LogisticRegression(method='mini-batch', max_iterations=50000,␣
↪early_stopping=True,
                           alpha=.0001, tol=.00001, mini_batch_size=100,␣
↪record_history_every=50,print_loss_every=100)
```

[42]:
```
model.fit(X_train, y_train)
```

```
loss at epoch 0: 69.31471805599453
loss at epoch 100: 53.47436325334273
loss at epoch 200: 42.081800508401855
loss at epoch 300: 37.27659869825479
loss at epoch 400: 33.01829288909449
loss at epoch 500: 27.583240825701957
loss at epoch 600: 28.45279712965094
loss at epoch 700: 25.476963283362384
loss at epoch 800: 23.29986118311235
loss at epoch 900: 21.698323987787774
loss at epoch 1000: 22.072387778164376
loss at epoch 1100: 20.163416969116227
```

```
loss at epoch 1200: 20.779383392235847
loss at epoch 1300: 18.563370062766335
loss at epoch 1400: 19.52587106780651
loss at epoch 1500: 18.381175677065258
loss at epoch 1600: 15.169948480500334
loss at epoch 1700: 17.090892477643205
loss at epoch 1800: 16.676012884167207
loss at epoch 1900: 17.765446332139074
loss at epoch 2000: 16.20006402779587
loss at epoch 2100: 16.573010826331494
loss at epoch 2200: 13.530249038402197
loss at epoch 2300: 14.779381221352516
loss at epoch 2400: 15.742454472880713
loss at epoch 2500: 14.212706857438851
loss at epoch 2600: 15.984135519420915
loss at epoch 2700: 15.562360678152276
loss at epoch 2800: 14.680548692245537
loss at epoch 2900: 14.167814574108277
loss at epoch 3000: 15.057672202513864
loss at epoch 3100: 15.124239185970756
loss at epoch 3200: 11.283367688587722
loss at epoch 3300: 14.010950138133193
loss at epoch 3400: 12.601146831016816
loss at epoch 3500: 14.239743252199707
loss at epoch 3600: 14.254432381367975
loss at epoch 3700: 10.865182845174427
loss at epoch 3800: 12.476699985879025
loss at epoch 3900: 12.309793159269416
loss at epoch 4000: 15.32224647910998
loss at epoch 4100: 12.568804620292164
loss at epoch 4200: 13.078230694057286
loss at epoch 4300: 11.111502130605588
loss at epoch 4400: 12.649181803514056
loss at epoch 4500: 13.846514889678412
loss at epoch 4600: 12.38262294444844
loss at epoch 4700: 9.836507896126003
loss at epoch 4800: 11.739564084611018
loss at epoch 4900: 11.399885755662279
loss at epoch 5000: 12.458111224369189
loss at epoch 5100: 11.827113712232183
loss at epoch 5200: 13.90349163715451
loss at epoch 5300: 9.465181726115409
loss at epoch 5400: 10.248790747929235
loss at epoch 5500: 12.305143851523198
loss at epoch 5600: 9.824856512258403
loss at epoch 5700: 11.404220245180124
loss at epoch 5800: 9.700145437907766
loss at epoch 5900: 12.415125315331222
```

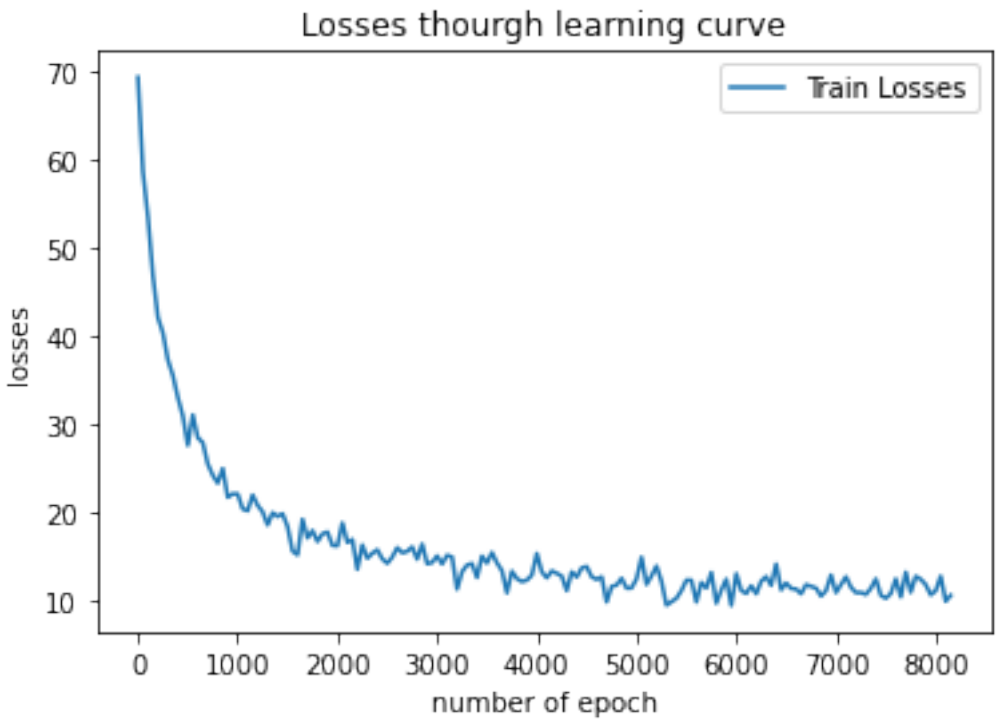
```
loss at epoch 6000: 13.040217002133543
loss at epoch 6100: 10.793556065831229
loss at epoch 6200: 10.76165736539987
loss at epoch 6300: 12.704458964860669
loss at epoch 6400: 14.128966299284887
loss at epoch 6500: 11.975357885129245
loss at epoch 6600: 11.297965695872477
loss at epoch 6700: 11.788122046948752
loss at epoch 6800: 11.400726676453026
loss at epoch 6900: 11.11543469986471
loss at epoch 7000: 10.903011729836823
loss at epoch 7100: 12.686759892251862
loss at epoch 7200: 10.889428586159719
loss at epoch 7300: 10.687186517883912
loss at epoch 7400: 12.44386871035238
loss at epoch 7500: 10.257389366578805
loss at epoch 7600: 12.444296829470208
loss at epoch 7700: 13.212047145847036
loss at epoch 7800: 12.741261073014261
loss at epoch 7900: 11.790582063645523
loss at epoch 8000: 11.109223187090466
loss at epoch 8100: 9.907791464776038
early_stopped at epoch: 8168
fitting model completed by loss: 11.757320096436894
```

[43]:
```
y_pred = model.predict(X_test)
loss = model.loss(y_pred, y_test)
print(f'Testing loss: {loss}')
```

```
Testing loss: 16.67744603101461
```

2.9 Task 3: Plot Learning

[44]:
```
model.show_history()
```



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

[48]:
```
class classification_report():

    def __init__(self, actual, predict):

        self.actual = actual
        self.predict = predict

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

[49]:
```
report = classification_report(y_test, model.round_pred(y_pred))

print('Model Evaluation')
print(f'Evaluation - Accuracy : {report.accuracy()} %')
print(f'Evaluation - Precision : {report.precision()} %')
print(f'Evaluation - Recall : {report.recall()} %')
print(f'Evaluation - F1 : {report.f1()} %')
```

```
Model Evaluation
Evaluation - Accuracy : 96.0 %
Evaluation - Precision : 97.29729729729729 %
Evaluation - Recall : 94.73684210526316 %
Evaluation - F1 : 95.99999999999999 %
```

2.10.1 Case of using classificaiton report from sklearn.metrics

[50]:
```
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       0.95      0.97      0.96        74
           1       0.97      0.95      0.96        76

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

```
weighted avg       0.96      0.96      0.96       150
```

[ ]: