* Problem 2: Gradient Descent Algo. and Logistic Regression.

(1) Likelihood function with respect to parameter w is given by,

$$L(w) = \prod_{n=1}^{N} p(C_1 | x_n)^{y_n} (1 - p(C_1 | x_n))^{1-y_n}$$

→ Applying negative logarithm on both sides,

$$\therefore -\ln(L(w)) = -\ln\left[\prod_{n=1}^{N} p(C_1 | x_n)^{y_n} (1-p(C_1|x_n))^{1-y_n}\right]$$

$$= -\sum_{n=1}^{N} \left[(y_n \ln(p(C_1|x_n)) + (1-y_n)\ln(1-p(C_1|x_n))\right] \quad \ldots\ldots (i)$$

→ Now, $p(C_1|x) = \sigma(w^T x + w_0) = f(x)$

$$\therefore p(C_1|x_n) = \sigma(w^T x_n + w_0) = f(x_n) \quad \ldots\ldots (ii)$$

→ Also, $\sigma(a) = \dfrac{1}{(1+\exp(-a))}$ and $a(x) = w^T x + w_0$

$$\therefore \sigma(a) = \frac{1}{1+\exp(-(w^T + w_0))}$$

→ Taking derivation of above function with respect to ω,

$$\frac{\partial(\sigma(a))}{\partial\omega} = \frac{\partial}{\partial\omega}\left(\frac{1}{1+(\exp(-a))}\right)$$

$$= \frac{\partial}{\partial a}\left(\frac{1}{1+\exp(-a)}\right) \times \frac{\partial a}{\partial\omega}$$

↳ (Partial derivatives)

$$= \frac{1}{(1+\exp(-a))^2} \cdot \exp(-a) \cdot \frac{\partial a}{\partial\omega}$$

$$= \cancel{\frac{\exp(-(a))}{(1+\exp(-a))^2}}$$

$$= \frac{1}{1+\exp(-a)} \times \underbrace{\frac{(1+\exp(-a)-1)}{1+\exp(-a)}}_{} \cdot \frac{\partial a}{\partial\omega}$$

↳ Adding and Subtracting 1

$$= \frac{1}{1+\exp(-a)} \cdot \left(1 - \frac{1}{1+\exp(-a)}\right) \cdot \frac{\partial a}{\partial\omega}$$

$$= \sigma(a)\left(1-\sigma(a)\right) \cdot \frac{\partial a}{\partial\omega}$$

$$\therefore \frac{\partial}{\partial\omega}(\sigma(a)) = \sigma(a)\left(1-\sigma(a)\right) \cdot \frac{\partial a}{\partial\omega} \quad \text{.....(iii)}$$

→ Now, $-\ln L(w) = \mathcal{E}(w)$

∴ from equations (i) and (ii)

$$-\ln L(w) = -\sum_{n=1}^{N}\left(y_n \ln(\sigma(a)) + (1-y_n)\ln(1-\sigma(a))\right)$$

$$= \mathcal{E}(w)$$

→ Now taking derivation of above equation with request to w,

$$\therefore \frac{\partial \mathcal{E}(w)}{\partial w} = -\sum_{n=1}^{N}\left(y_n \frac{\partial}{\partial w}\ln(\sigma(a)) + (1-y_n)\frac{\partial}{\partial w}\ln(1-\sigma(a))\right)$$

$$= -\sum_{n=1}^{N}\left(y_n \frac{1}{\sigma(a)}\frac{\partial}{\partial w}(\sigma(a)) + (1-y_n)\left(\frac{1}{\sigma(a)-1}\right)\frac{\partial}{\partial w}(\sigma(a))\right)$$

$$= -\sum_{n=1}^{N}\left(y_n \frac{1}{\sigma(a)}\frac{\partial}{\partial w}(\sigma(a)) + (1-y_n)\left(\frac{1}{\sigma(a)-1}\right)\frac{\partial}{\partial w}(\sigma(a))\right)$$

→ From equation (iii), we get,

$$\frac{\partial \mathcal{E}(w)}{\partial w} = -\sum_{n=1}^{N}\left(y_n \frac{1}{\sigma(a)}\sigma(a)(1-\sigma(a))\frac{\partial a}{\partial w} + \frac{(1-y_n)}{\sigma(a)-1}\sigma(a)(1-\sigma(a))\frac{\partial a}{\partial w}\right.$$

→ According to $a(x) = w^T x + w_0$

we get $\frac{\partial a}{\partial w} = x$

∴ from above value, we get,

$$\rightarrow \frac{\partial(\mathcal{E}(w))}{\partial w} = -\sum_{n=1}^{N} \left(y_n(1-\sigma(a))x_n + (y_n - 1)\cdot\sigma(a)\cdot x_n \right)$$

$$= -\sum_{n=1}^{N} \left(y_n(1-f(x_n))x_n + (y_n-1)\cdot(f(x_n)\cdot x_n) \right)$$

↳ from (ii)

$$= -\sum_{n=1}^{N} y_n x_n - y_n f(x_n)\cdot x_n + y_n f(x_n)\cdot x_n - f(x_n)\cdot x_n$$

$$= -\sum_{n=1}^{N} x_n (y_n - f(x_n))$$

→ Thus we have obtained following result,

$$\nabla_w \mathcal{E}(w) = \sum_{n=1}^{N} (f(x_n) - y_n) x_n$$

Problem 2. Gradient Descent Algorithm and Logistic Regression

Importing Libraries

In [118…

```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import cross_val_score
from numpy import log, dot, e
from sklearn.model_selection import KFold
import random
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.preprocessing import MinMaxScaler
from sklearn.datasets import load_breast_cancer
```

Reading the dataset

In [119…

```
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cance
headers = ['id', 'type', 'mean radius', 'mean texture', 'mean perimeter', 'me
dataset = pd.read_csv(url,names = headers)

dataset
```

Out[119…

	id	type	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mear concavity
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001(
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869(
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974(
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414(
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980(
...	...	...	...	...	...	...	...	...	..
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.2439(
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.1440(
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.0925'
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.3514(
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.0000(

569 rows × 32 columns

Changes to dataset for ease of use

In [120…

```
# Replacing M with 1 and B with 0
dataset = dataset.replace('M',1)
dataset = dataset.replace('B',0)

# Converting to array
y = dataset[dataset.columns[1:2]]
y = y.to_numpy().reshape(len(temp))
X = dataset[dataset.columns[2:]]
X = X.to_numpy()

dataset
```

Out[120…

	id	type	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mear concavity
0	842302	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001(
1	842517	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869(
2	84300903	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974(
3	84348301	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414(
4	84358402	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980(
...	...	...	...	...	...	...	...	...	..
564	926424	1	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.2439(
565	926682	1	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.1440(
566	926954	1	16.60	28.08	108.30	858.1	0.08455	0.10230	0.0925
567	927241	1	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.3514(
568	92751	0	7.76	24.54	47.92	181.0	0.05263	0.04362	0.0000(

569 rows × 32 columns

Stochastic Gradient Descent

In [121…

```
class LR_Stochastic:

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

    # instance variables
    def __init__(self, lr = 0.0002, iters = 10000):
        self.lr = lr
        self.iters = iters
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        # initialize parameters
        samples, features = X.shape
        # set weights to zero
        self.weights = np.zeros(features)
        self.bias = 0
        costs = []
        epochs = []

        # stochastic gradient descent algorithm
```

```
        for i in range(self.iters):
            r_index = random.randint(0,samples - 1)

            sample_x = X[r_index]
            sample_y = y[r_index]
            # logistic regression equation values
            linear_model = np.dot(sample_x, self.weights) + self.bias
            # sigmoid values
            y_pred = self._sigmoid(linear_model)
            # derivativation
            dw = (1/samples) * np.dot(sample_x.T,y_pred - sample_y)
            db = (1/samples) * np.sum(y_pred - sample_y)
            # updating the weights
            self.weights -= self.lr*dw
            self.bias -= self.lr*db
            # cost function
            cost = np.square(sample_y-y_pred)

            if i%100==0: # at every 100th iteration record the cost and iters
                costs.append(cost)
                epochs.append(i)


        return costs,epochs

    # Testing the model
    def predict(self, X):
        linear_model = np.dot(X, self.weights) + self.bias
        y_pred = self._sigmoid(linear_model)
        y_pred_cls = [1 if i > 0.5 else 0 for i in y_pred]
        return y_pred_cls
```

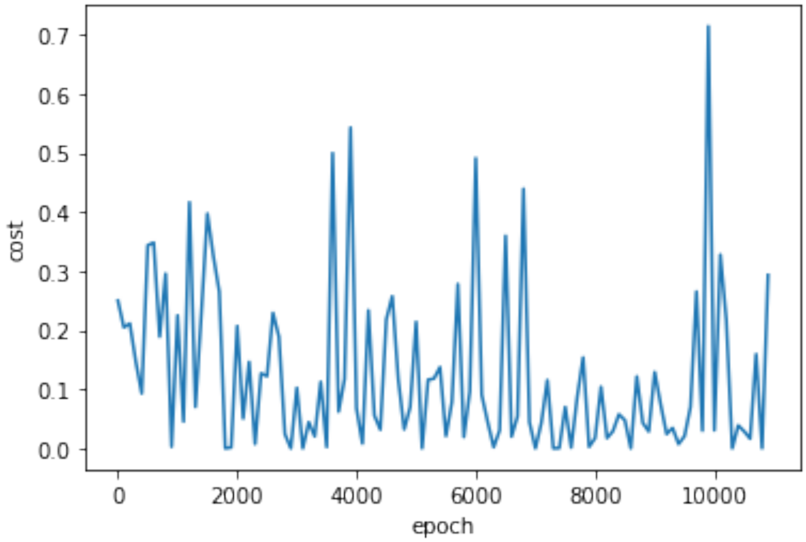
Implementing, training, testing, evaluating, recall, precision, and accuracy

In [122…

```
if __name__ == "__main__":

    # accuracy of the predictions
    def accuracy(y_true, y_pred):
        accuracy = np.sum(y_true == y_pred) / len(y_true)
        return accuracy

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,

    regressor = LR_Stochastic(lr = 0.0002, iters=11000)
    axisx,axisy = regressor.fit(X_train, y_train)
    predictions = regressor.predict(X_test)

    print("Prediction accuracy:", accuracy(y_test, predictions))
```

```
# Confusion matrix and different scores
confusionM = [[0,0],[0,0]]

for i in range(len(y_test)):
    if(y_test[i] == 0 and predictions[i] == 0):
        confusionM[0][0] += 1
    if(y_test[i] == 0 and predictions[i] == 1):
        confusionM[0][1] += 1
    if(y_test[i] == 1 and predictions[i] == 0):
        confusionM[1][0] += 1
    if(y_test[i] == 1 and predictions[i] == 1):
        confusionM[1][1] += 1

# true negative, false positive, false negative, true positive
tn = confusionM[0][0]
fp = confusionM[0][1]
fn = confusionM[1][0]
tp = confusionM[1][1]

precision_score = tp/(fp + tp)
recall_score = tp/(fn + tp)
accuracy_score = (tp + tn)/(tp + fn + tn + fp)
print('Precision: %.3f' % precision_score)
print('Recall: %.3f' % recall_score)
print('Accuracy: %.3f' % accuracy_score)


plt.xlabel("epoch")
plt.ylabel("cost")
plt.plot(axisy,axisx)

#cross-validation
kFold = KFold(n_splits=3, random_state=None)

acc_score = []

for train , test in kFold.split(X):
    X_train , X_test = X[train,:],X[test,:]
    y_train , y_test = y[train] , y[test]

    regressor.fit(X_train,y_train)
    pred_values = regressor.predict(X_test)

    acc = accuracy(pred_values, y_test)
    acc_score.append(acc)

avg_acc_score = sum(acc_score)/3

print('accuracy of each fold - {}'.format(acc_score))
print('Avg accuracy : {}'.format(avg_acc_score))
```

```
Prediction accuracy: 0.8951048951048951
Precision: 0.975
Recall: 0.736
Accuracy: 0.895
accuracy of each fold - [0.9157894736842105, 0.8894736842105263, 0.93121693121
69312]
Avg accuracy : 0.9121600297038893
```



Mini-Batch gradient descent

In [123…

```
class LR_Mini_Batch:

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

    # defining instance variables
    def __init__(self, lr = 0.0001, iters = 200, batch_size = 20):
        self.lr = lr
        self.iters = iters
        self.weights = None
        self.bias = None
        self.batch_size = batch_size

    def fit(self, X, y):
        # initialize paramenters
        samples, features = X.shape
        # set all weights equal to 0 and bias equal to 0
        self.weights = np.zeros(features)
        self.bias = 0
        costs = []
        epochs = []

        # mini-batch gradient descent algorithm
        for i in range(self.iters):
```

```
            r_indices = np.random.permutation(samples)
            sample_x = X[r_indices]
            sample_y = y[r_indices]

            # iterations in batches
            for j in range(0,samples,self.batch_size):
                Xt = sample_x[j:j+self.batch_size]
                yt = sample_y[j:j+self.batch_size]

                # linear logistic regression equation
                linear_model = np.dot(Xt, self.weights) + self.bias

                # sigmoid values
                y_pred = self._sigmoid(linear_model)

                # derivatation
                dw = (1/samples) * np.dot(Xt.T,y_pred - yt)
                db = (1/samples) * np.sum(y_pred - yt)

                # updating the weights
                self.weights -= self.lr*dw
                self.bias -= self.lr*db

                # MeanSquaredError of costs
                cost = np.mean(np.square(yt-y_pred))

            if i%10==0: # at every 10th iteration record the cost and iters v
                costs.append(cost)
                epochs.append(i)


        return costs,epochs

    # Testing of model
    def predict(self, X):
        linear_model = np.dot(X, self.weights) + self.bias
        y_pred = self._sigmoid(linear_model)
        y_pred_cls = [1 if i > 0.5 else 0 for i in y_pred]
        return y_pred_cls
```

Implementing, training, testing, evaluating, recall, precision, and accuracy

In [124…

```
if __name__ == "__main__":

    # accuracy of predictions
    def accuracy(y_true, y_pred):
        accuracy = np.sum(y_true == y_pred) / len(y_true)
        return accuracy
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,

regressor = LR_Mini_Batch(lr = 0.0001, iters=100, batch_size = 10)
axisx,axisy = regressor.fit(X_train, y_train)
predictions = regressor.predict(X_test)

print("Prediction accuracy:", accuracy(y_test, predictions))

plt.xlabel("epoch")
plt.ylabel("cost")
plt.plot(axisy,axisx)

# Confusion matrix and different scores
confusionM = [[0,0],[0,0]]

for i in range(len(y_test)):
    if(y_test[i] == 0 and predictions[i] == 0):
        confusionM[0][0] += 1
    if(y_test[i] == 0 and predictions[i] == 1):
        confusionM[0][1] += 1
    if(y_test[i] == 1 and predictions[i] == 0):
        confusionM[1][0] += 1
    if(y_test[i] == 1 and predictions[i] == 1):
        confusionM[1][1] += 1

# true negative, false positive, false negative, true positive
tn = confusionM[0][0]
fp = confusionM[0][1]
fn = confusionM[1][0]
tp = confusionM[1][1]

precision_score = tp/(fp + tp)
recall_score = tp/(fn + tp)
accuracy_score = (tp + tn)/(tp + fn + tn + fp)
print('Precision: %.3f' % precision_score)
print('Recall: %.3f' % recall_score)
print('Accuracy: %.3f' % accuracy_score)


# cross-validation
kFold = KFold(n_splits=3, random_state=None)

acc_score = []

for train , test in kFold.split(X):
    X_train , X_test = X[train,:],X[test,:]
    y_train , y_test = y[train] , y[test]

    regressor.fit(X_train,y_train)
    pred_values = regressor.predict(X_test)

    acc = accuracy(pred_values , y_test)
    acc_score.append(acc)
```

```
    avg_acc_score = sum(acc_score)/3

    print('accuracy of each fold - {}'.format(acc_score))
    print('Avg accuracy : {}'.format(avg_acc_score))
```

```
Prediction accuracy: 0.8951048951048951
Precision: 0.975
Recall: 0.736
Accuracy: 0.895
accuracy of each fold - [0.8894736842105263, 0.9052631578947369, 0.92592592592
59259]
Avg accuracy : 0.9068875893437297
```