기계학습 10주차 과제 보고서

12191656 이채연

P1. MNIST 데이터를 활용하여 multi-class logistic regression 코드 작성

Training data의 feature vector의 shape이 (60000, 784)이고, Training data의 label y는 shape이 (60000,1)이었지만 이를 one hot vector로 변환하여 shape을 (60000,10)으로 바꾸었다. 따라서 label y는 각 행에 대하여 정답 레이블 숫자에 해당하는 index만 1이고 나머지는 0인 형태를 띈다.

```
X, y = fetch_openml("mnist_784", version=1, return_X_y=True, as_frame=False)

train_X = np.array(X[:-10000])
test_X = np.array(X[-10000:])

## one hot encoding
train_y = y[:-10000].astype(np.int)
test_y = y[-10000:].astype(np.int)

n_values = np.max(train_y) + 1
train_y = np.eye(n_values)[train_y]
test_y = np.eye(n_values)[test_y]
```

따라서 가중치 w의 shape은 (784,10), 편향 b의 shape은 (1,10)이 된다.

MultiClassLogisticRegression 클래스의 객체를 인스턴스화하고, 객체의 train_model method를 호출하면 MNIST training dataset에 대한 Multi Class Logistic Regression model의 훈련이 시작된다.

```
##Train

MCLR_cls = MultiClassLogisticRegression(epoch=20000)
result_dict = MCLR_cls.train_model(train_normalized_X, train_y, test_normalized_X, test_y)
```

train_model method의 내부 코드 구현은 다음과 같이 했다.

```
def train_model(self, X_train, Y_train, X_test, Y_test, print_cost = False):
    dim = np.shape(X_train)[1]
    w, b = self.initialize_weight(dim)
    parameters, costs = self.gradient_descent(w, b, X_train, Y_train, print_cost = False)

    self.w = parameters["w"]
    self.b = parameters["b"]

    Y_prediction_test = self.predict(X_test)
    Y_prediction_train = self.predict(X_train)

    train_score = 100-np.mean((np.sum(np.abs(Y_prediction_train - Y_train), axis = 1))/2) * 100
    test_score = 100-np.mean((np.sum(np.abs(Y_prediction_test - Y_test), axis = 1))/2) * 100
    print(train_score)
    print(test_score)
    print(100-np.mean((np.sum(np.abs(Y_prediction_test - Y_test), axis = 1))/2) * 100)
    print("test accuracy: {} %".format(100-np.mean((np.sum(np.abs(Y_prediction_test - Y_test), axis = 1))/2) * 100))
    result_dict = {"costs": costs,
                   "Y_prediction_test": Y_prediction_test,
                   "Y_prediction_train": Y_prediction_train,
                   "w": self.w,
                   "b": self.b,
                   "learning_rate": self.learning_rate,
                   "num_iterations": self.epoch,
                   "train accuracy": train_score,
                   "test accuracy": test_score}
    return result_dict
```

먼저, 가중치와 편향을 초기화 해준다. 가중치 w는 정규분포로부터 (784, 10) shape의 numpy array

형태로 무작위 샘플링 하였다. 편향은 각 요소가 0~1의 균일분포 표준정규분포 난수인 (1, 10) shape의 numpy array형태로 초기화하였다.

```
def initialize_weight(self,dim):
    w = np.random.normal(0,1,(dim,10))
    b = np.random.rand(10)
    return w,b
```

그 다음으로 경사하강법을 이용해서 가중치와 편향을 optimization하였다.

```
def cost(self, y_hat, y, N):
    cost = -(1/N)*np.sum(y*np.log(y_hat)+(1-y)*np.log(1-y_hat), axis = 0)
    cost = np.squeeze(cost)
    return cost

def cal_gradient(self, w, y_hat, X, y):
    N = X.shape[0] # 60000
    # 784 * 60000 * 60000 * 10 = 784 * 10
    delta_w = (1/N)*np.matmul(X.T, (y_hat-y))

    # 1 * 10
    delta_b = (1/N)*np.sum(y_hat-y, axis = 0)
    grads = {"delta_w": delta_w,
             "delta_b": delta_b}
    return grads

def gradient_position(self, w, b, X, y):
    N = X.shape[0]
    y_hat = self.hypothesis(w, X, b)
    cost = self.cost(y_hat, y, N)
    grads = self.cal_gradient(w, y_hat, X, y)
    return grads, cost

def gradient_descent(self, w, b, X, y, print_cost = False):

    costs = []

    for i in range(self.epoch):
        grads, cost = self.gradient_position(w, b, X, y)

        delta_w = grads["delta_w"]
        delta_b = grads["delta_b"]

        delta_w = delta_w.reshape(784, 10)
        delta_b = delta_b.reshape(1, 10)

        w = w -(self.learning_rate * delta_w)
        b = b -(self.learning_rate * delta_b)
        if i % 100 == 0:
            costs.append(cost)

        if print_cost and i % 100 == 0:
            print("Cost after iteration %i: %f" %(i, cost))

        params = {"w" : w,
                  "b" : b}

        grads = {"delta_w": delta_w,
                 "delta_b": delta_b}

    return params, costs
```

Epoch를 20000으로 해서 w와 b의 gradient를 계산하고 cost가 감소하는 방향으로 w와 b를 업데이트해주었다.

이렇게 optimization된 w와 b를 이용해서 모델의 각각 train dataset과 test dataset에 대한 예측값을 계산하는 predict함수를 호출하였다.
Predict method 내부 구현 코드는 다음과 같다.

```
def predict(self,X):
    X = np.array(X)
    N = X.shape[0]

    Y_prediction = np.zeros(N)

    w = self.w.reshape(784, 10)
    b = self.b.reshape(1, 10)

    # 60000 * 10
    y_hat = self.hypothesis(w, X, b)

    for i in range(len(y_hat)):
        idx = np.argmax(y_hat[i])
        Y_prediction[i] = idx

    Y_prediction = Y_prediction.astype(np.int)
    n_values = np.max(Y_prediction) + 1
    Y_prediction = np.eye(n_values)[Y_prediction]

    return Y_prediction
```

모델의 예측 값 y_hat은 shape이 (60000,10)이다. 각 행에서 각 열(인덱스 0부터 9)의 원소는 해당 행의 데이터가 각각 인덱스 0~9 일 확률 값을 나타낸다. 10개의 확률 값 중 가장 큰 값의 인덱스가 model이 예측한 해당 데이터의 숫자가 된다. 이렇게 60000개의 예측 값이 나오면, 이것을 (60000, 10)의 shape을 가지는 one hot vector로 변환해주고 이 one hot vector를 return해준다.

train_model method에서 test dataset에 대한 예측값 Y_prediction_test와 train dataset에 대한 예측값 Y_prediction_train을 얻은 뒤, 예측값과 실제값의 차이를 통해 train_score와 test_score를 계산하고 test accuracy를 출력해주었다.

이렇게 모델을 train dataset에 대해 train한 결과 테스트 정확도 84.3%를 달성하였다.

P2. MNIST data-loader 작성 시 Z-score normalization을 활용하여 데이터 정규화 수행

```
train_normalized_X = (train_X - np.mean(train_X, axis = 0)) / np.std(train_X, axis = 0)
train_normalized_X[np.isnan(train_normalized_X)] = train_X[np.isnan(train_normalized_X)]
test_normalized_X = (test_X - np.mean(test_X, axis = 0)) / np.std(test_X, axis = 0)
test_normalized_X[np.isnan(test_normalized_X)] = test_X[np.isnan(test_normalized_X)]
```

각 feature(인덱스 0~783)별로 평균과 편차를 계산하여 z-score normalization을 해주었다.
그리고 표준편차가 0인 column에 대해서는 원래 feature값을 넣어주었다.

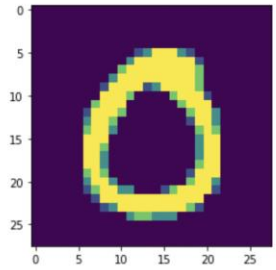
P3. MNIST 테스트 결과 시각화

랜덤한 10개의 test set의 이미지에 대해 모델이 예측한 클래스와 실제 데이터의 클래스를 출력한 결과를 시각화하였다.

test_y 와 Y_prediction_test가 one hot vector이므로 argmax함수를 이용하여 원소의 값이 가장 큰 (1인) 인덱스를 각각 label값과 prediction값으로 지정하였다.

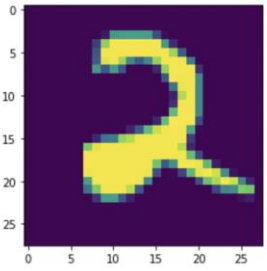
```
plt.imshow(test_X[407].reshape(28,28))
label = np.argmax(test_y[407])
prediction = np.argmax(result_dict['Y_prediction_test'][407])
print("Label: ",label)
print("Prediction: ",prediction)
```

```
Label:  0
Prediction:  0
```



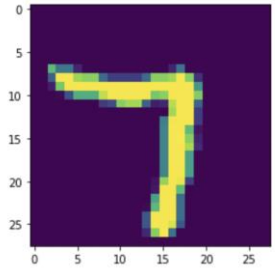
```
plt.imshow(test_X[1407].reshape(28,28))
label = np.argmax(test_y[1407])
prediction = np.argmax(result_dict['Y_prediction_test'][1407])
print("Label: ",label)
print("Prediction: ",prediction)
```

```
Label:  2
Prediction:  2
```



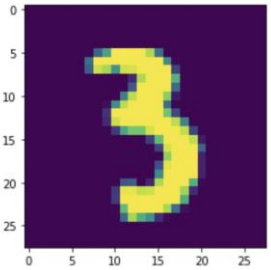
```
plt.imshow(test_X[2407].reshape(28,28))
label = np.argmax(test_y[2407])
prediction = np.argmax(result_dict['Y_prediction_test'][2407])
print("Label: ",label)
print("Prediction: ",prediction)
```

```
Label:  7
Prediction:  7
```



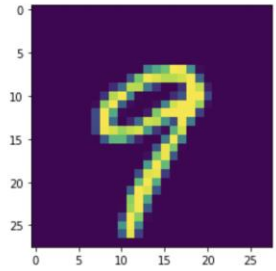
```
plt.imshow(test_X[3407].reshape(28,28))
label = np.argmax(test_y[3407])
prediction = np.argmax(result_dict['Y_prediction_test'][3407])
print("Label: ",label)
print("Prediction: ",prediction)
```

```
Label:  3
Prediction:  3
```



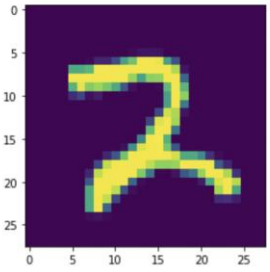
```
plt.imshow(test_X[4407].reshape(28,28))
label = np.argmax(test_y[4407])
prediction = np.argmax(result_dict['Y_prediction_test'][4407])
print("Label: ",label)
print("Prediction: ",prediction)
```

```
Label:  9
Prediction:  9
```



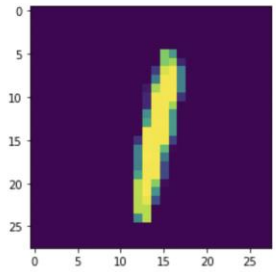
```
plt.imshow(test_X[5407].reshape(28,28))
label = np.argmax(test_y[5407])
prediction = np.argmax(result_dict['Y_prediction_test'][5407])
print("Label: ",label)
print("Prediction: ",prediction)
```

```
Label:  2
Prediction:  2
```



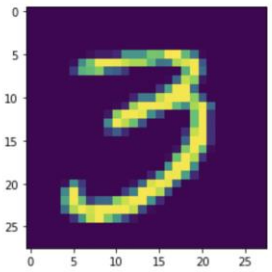
```
plt.imshow(test_X[6407].reshape(28,28))
label = np.argmax(test_y[6407])
prediction = np.argmax(result_dict['Y_prediction_test'][6407])
print("Label: ",label)
print("Prediction: ",prediction)
```

```
Label:  1
Prediction:  1
```



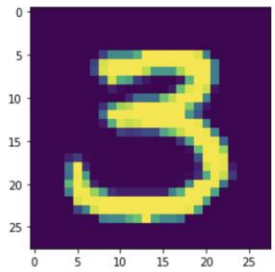
```
plt.imshow(test_X[7407].reshape(28,28))
label = np.argmax(test_y[7407])
prediction = np.argmax(result_dict['Y_prediction_test'][7407])
print("Label: ",label)
print("Prediction: ",prediction)
```

```
Label:  3
Prediction:  3
```



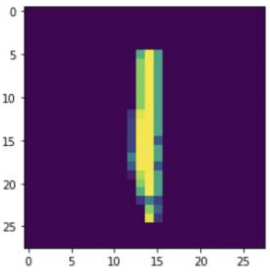
```
plt.imshow(test_X[8407].reshape(28,28))
label = np.argmax(test_y[8407])
prediction = np.argmax(result_dict['Y_prediction_test'][8407])
print("Label: ",label)
print("Prediction: ",prediction)
```

```
Label:  3
Prediction:  3
```



```
plt.imshow(test_X[9407].reshape(28,28))
label = np.argmax(test_y[9407])
prediction = np.argmax(result_dict['Y_prediction_test'][9407])
print("Label: ",label)
print("Prediction: ",prediction)
```

```
Label:  1
Prediction:  1
```



모델은 10개의 test set의 이미지에 대해 10번 옳게 예측하였다. 즉, 모델은 이미지를 10/10의 확률로 예측에 성공하고, 0/10의 확률로 예측에 실패했음을 알 수 있다.