기계학습 13주차 과제 보고서

12191656 이채연

P1. 제공된 Fashion-MNIST 데이터 셋을 gaussian naive bayesian classifier를 통해 분류 수행

모든 image 파일을 불러오고 끝의 label은 끝에서 5번째에 위치하므로 이를 indexing하여 label로 지정해 주었다.

In [1]:
```
import os
import cv2
import numpy as np
import csv
import math
```

In [2]:
```
def data_loader(path, data_size):

    file_list = os.listdir(path)
    feature_info = list()
    label_info = list()

    for idx in range(data_size):
        img_path = os.path.join(path, file_list[idx])

        feature = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        feature = list(map(float,feature.flatten()))
        label = int(file_list[idx][-5])

        feature_info.append(feature)
        label_info.append(label)

    return feature_info, label_info
```

총 10개의 class가 있으므로 10개의 class로 분류하였고, conditional independence를 적용하기 위해 class가 주어졌을 때 class 별 각 feature들의 평균과 표준 편차를 계산하였다.

In [3]:
```
def get_GaussianNBC(train_samples, train_labels):

    fashion_class_0_samples = []
    fashion_class_1_samples = []
    fashion_class_2_samples = []
    fashion_class_3_samples = []
    fashion_class_4_samples = []
    fashion_class_5_samples = []
    fashion_class_6_samples = []
    fashion_class_7_samples = []
    fashion_class_8_samples = []
    fashion_class_9_samples = []

    for k in range(len(train_samples)):
        sample = train_samples[k]
        label = train_labels[k]

        if label == 0:
            fashion_class_0_samples.append(sample)
        elif label == 1:
            fashion_class_1_samples.append(sample)
        elif label == 2:
            fashion_class_2_samples.append(sample)
        elif label == 3:
            fashion_class_3_samples.append(sample)
        elif label == 4:
            fashion_class_4_samples.append(sample)
        elif label == 5:
            fashion_class_5_samples.append(sample)
        elif label == 6:
            fashion_class_6_samples.append(sample)
        elif label == 7:
            fashion_class_7_samples.append(sample)
        elif label == 8:
            fashion_class_8_samples.append(sample)
        elif label == 9:
            fashion_class_9_samples.append(sample)
```

```
    samples_by_classes = [
        fashion_class_0_samples,
        fashion_class_1_samples,
        fashion_class_2_samples,
        fashion_class_3_samples,
        fashion_class_4_samples,
        fashion_class_5_samples,
        fashion_class_6_samples,
        fashion_class_7_samples,
        fashion_class_8_samples,
        fashion_class_9_samples
    ]

    numOf_classes = 10
    means_by_classes = []
    stdevs_by_classes = []

    for C in range(numOf_classes):
        means = []
        stdevs = []
        for features in zip(*samples_by_classes[C]):
            means.append(np.mean(features))
            stdevs.append(np.std(features))
        means_by_classes.append(means)
        stdevs_by_classes.append(stdevs)

    return means_by_classes, stdevs_by_classes
```

Gaussian Probability Density Function은 다음과 같다.

In [4]:
```
def Gaussian_PDF(x, mean, stdev):
    if stdev == 0.0:
        if x == mean:
            return 1.0
        else:
            return 0.0
    return (1/(math.sqrt(2 * math.pi)*stdev)) *(math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2)))))
```

Predict 할 때에는 conditional independence 공식을 적용하였다.

```
In [5]: def predict(means, stdevs, test_samples):

            pred_classes = []
            numOf_classes = 10
            numOf_features = 784

            for i in range(len(test_samples)):
                prob_by_classes = []
                for C in range(numOf_classes):
                    prob = 0
                    for j in range(numOf_features):
                        mean = means[C][j]
                        stdev = stdevs[C][j]
                        x = test_samples[i][j]
                        gaussian_pdf = Gaussian_PDF(x, mean, stdev)
                        if gaussian_pdf == 0 :
                            prob = -math.inf
                            break
                        else :
                            prob += math.log(Gaussian_PDF(x, mean, stdev))
                    prob_by_classes.append(prob)
                bestProb = -math.inf
                pred_Label = 0
                for C in range(numOf_classes):
                    if prob_by_classes[C] > bestProb:
                        bestProb = prob_by_classes[C]
                        pred_Label = C
                pred_classes.append(pred_Label)

            return pred_classes
```

Predict한 것에 대해 정확도를 측정하는 코드를 다음과 같이 구현하였다. 함수를 호출하면 전체 예측 결과에 대해 예측에 성공한 것의 비율(정확도)이 리턴된다.

```
In [6]: def get_Acc(pred_classes_of_testset, gt_of_testset):
            accuracy = np.equal(pred_classes_of_testset, gt_of_testset)
            return list(accuracy).count(True)/len(accuracy)*100
```

Fashion_MNIST 데이터 셋의 path를 인자로 넣어 data_loader를 호출하였고, 이렇게 해서 얻어진 Fashion_MNIST train dataset으로 gaussian naive bayesian classifier를 훈련하였다. 그리고 test dataset으로 모델의 정확도를 측정하였다.

P1. 제공된 Fashion-MNIST 데이터 셋을 gaussian naive bayesian classifier를 통해 분류 수행

```
In [7]: train_set_path = 'Fashion-MNIST/train'
        test_set_path = 'Fashion-MNIST/test'

        train_samples, train_labels = data_loader(train_set_path, 60000)
        test_samples, test_labels = data_loader(test_set_path, 10000)

        means_by_classes, stdevs_by_classes = get_GaussianNBC(train_samples, train_labels)

        pred_classes = predict(means_by_classes, stdevs_by_classes, test_samples)

        acc = get_Acc(pred_classes, test_labels)

        print('Acc: %s' %acc)

        Acc: 63.09
```

모델의 정확도는 63.09%로 나왔다.

처음에는 predict함수에서 conditional independence 공식을 그대로 이용하여 클래스가 주어졌을 때 각 feature의 probability를 곱해서 probability를 계산하였다.

```
In [14]: pred_classes_no_log = predict_no_log(means_by_classes, stdevs_by_classes, test_samples)

         acc_no_log = get_Acc(pred_classes_no_log, test_labels)

         print('Acc: %s' %acc_no_log)

         Acc: 10.0
```

그 결과 모델의 정확도는 10%가 나왔고, 모델이 잘 작동하지 않는 문제가 무엇인지 생각하였다. 모델이 잘 작동하지 않았던 이유는 강의 이론 시간에 배웠던 내용인, predict함수에서 작은 값들을 계속 곱하여 컴퓨터가 계산하기에 너무 작은 값이 되어버려 0으로 처리하기 때문이라는 것을 알았다. 그래서 강의 이론 시간에 배운 대로 log의 특성을 활용하여 각 gaussian distribution값에 log를 취한 뒤, 더해주는 방식으로 바꿔보았다. 이렇게 하여도 각 class에 대한 probability의 대소

관계는 변하지 않는다.
그 결과, 모델의 정확도가 63.09%로 모델의 성능이 크게 향상하였다.

P2. P1에서 만든 모델에 min-max 정규화 방법을 적용해 적용 전후의 성능 비교 분석 수행

train_samples와 test_samples에 min-max normalization 방법을 적용하였다.
각 feature는 픽셀로, 0~255의 값을 가진다. 그래서 min값을 0으로, max값을 1로 하여 data에 min-max normalization을 적용하였다.

P2. P1에서 만든 모델에 min-max 정규화 방법을 적용해 적용 전후의 성능 비교 분석 수행

In [8]:
```
train_samples_minmax = train_samples
test_samples_minmax = test_samples

for i in range(len(train_samples)):
    for j in range(784) :
        train_samples_minmax[i][j] = train_samples_minmax[i][j]/255

for i in range(len(test_samples)):
    for j in range(784) :
        test_samples_minmax[i][j] = test_samples_minmax[i][j]/255

means_by_classes_minmax, stdevs_by_classes_minmax = get_GaussianNBC(train_samples_minmax, train_labels)

pred_classes_minmax = predict(means_by_classes_minmax, stdevs_by_classes_minmax, test_samples_minmax)

acc_minmax = get_Acc(pred_classes_minmax, test_labels)

print('Before min-max normalization) Acc: %s' %acc)
print('After min-max normalization) Acc: %s' %acc_minmax)
```

```
Before min-max normalization) Acc: 63.09
After min-max normalization) Acc: 63.62
```

그 결과 정확도는 63.62%로 min-max normalization을 하기 전과 비교했을 때 미세하게 모델의 성능이 좋아졌다.

참고로 predict함수에서 conditional independence 공식을 그대로 이용하여 클래스가 주어졌을 때 각 feature의 probability를 곱해서 probability를 계산했을 때 raw data에 대해서 10%의 정확도가 나왔다고 하였는데, min-max normalization을 적용한 data에 대해서는 정확도가 54.87%로 raw data와 비교했을 때 매우 큰 성능 향상을 보았다.

In [15]:
```
pred_classes_minmax_no_log = predict_no_log(means_by_classes_minmax, stdevs_by_classes_minmax, test_samples_minmax)

acc_minmax_no_log = get_Acc(pred_classes_minmax_no_log, test_labels)

print('Before min-max normalization) Acc: %s' %acc_no_log)
print('After min-max normalization) Acc: %s' %acc_minmax_no_log)
```

```
/var/folders/9q/sg0csh752098vchj19dv46mw0000gn/T/ipykernel_5669/3159619399.py:15: RuntimeWarning: overflow encounte
red in double_scalars
  prob *= Gaussian_PDF(x, mean, stdev)
/var/folders/9q/sg0csh752098vchj19dv46mw0000gn/T/ipykernel_5669/3159619399.py:15: RuntimeWarning: invalid value enc
ountered in double_scalars
  prob *= Gaussian_PDF(x, mean, stdev)
```

```
Before min-max normalization) Acc: 10.0
After min-max normalization) Acc: 54.87
```

이와 비교하여 log 특성을 이용했을 때는 min-max normalization의 효과는 미미하지만, 정확도는 log 특성을 이용하기 전인 54.87%보다 높은 63.62%로 나온다는 것을 알 수 있다.

P3. P1에서 만든 모델을 변경하여 온라인 학습을 수행할 수 있도록 하고, 기존 배치 학습 방법과 비교

기존의 배치 학습 방법에서는 가용 가능한 데이터를 모두 사용하여 모델 학습을 수행하였다. 이와 다르게 온라인 학습 방법은 가용 가능한 데이터를 미니 배치 단위로 사용하여 점진적으로 모델 학습을 수행한다.

구현 코드는 다음과 같다. 전체 데이터에 대해 평균과 표준편차를 한꺼번에 계산하지 않고, 미니 배치 단위를 1로 하여 점진적으로 평균과 표준편차를 계산하였다.

P3. P1에서 만든 모델을 변경하여 온라인 학습을 수행할 수 있도록 하고, 기존 배치 학습 방법과 비교

In [9]:
```
def get_GaussianNBC_online(train_samples, train_labels):

    fashion_class_0_samples = []
    fashion_class_1_samples = []
    fashion_class_2_samples = []
    fashion_class_3_samples = []
    fashion_class_4_samples = []
    fashion_class_5_samples = []
    fashion_class_6_samples = []
    fashion_class_7_samples = []
    fashion_class_8_samples = []
    fashion_class_9_samples = []

    for k in range(len(train_samples)):
        sample = train_samples[k]
        label = train_labels[k]

        if label == 0:
            fashion_class_0_samples.append(sample)
        elif label == 1:
            fashion_class_1_samples.append(sample)
        elif label == 2:
            fashion_class_2_samples.append(sample)
        elif label == 3:
            fashion_class_3_samples.append(sample)
        elif label == 4:
            fashion_class_4_samples.append(sample)
        elif label == 5:
            fashion_class_5_samples.append(sample)
        elif label == 6:
            fashion_class_6_samples.append(sample)
        elif label == 7:
            fashion_class_7_samples.append(sample)
        elif label == 8:
            fashion_class_8_samples.append(sample)
        elif label == 9:
            fashion_class_9_samples.append(sample)

    samples_by_classes = [
        fashion_class_0_samples,
        fashion_class_1_samples,
        fashion_class_2_samples,
        fashion_class_3_samples,
        fashion_class_4_samples,
        fashion_class_5_samples,
        fashion_class_6_samples,
        fashion_class_7_samples,
        fashion_class_8_samples,
        fashion_class_9_samples
    ]

    numOf_classes = 10
    means_by_classes = []
    stdevs_by_classes = []

    for C in range(numOf_classes):
        means = []
        stdevs = []
        for features in zip(*samples_by_classes[C]):
            features = list(features)
            mean = features[0]
            var = 0
            for i in range(1, len(features)) :
                old_mean = mean;
                old_var = var;
                mean = old_mean + ((features[i]-old_mean)/(i+1))
                var = old_var + ((((features[i]-old_mean)*(features[i]-mean))-old_var)/(i+1))

            means.append(mean)
            stdevs.append(math.sqrt(var))

        means_by_classes.append(means)
        stdevs_by_classes.append(stdevs)

    return means_by_classes, stdevs_by_classes
```

이렇게 온라인 학습을 수행한 함수 이름을 get_GaussianNBC_online으로 하였고, 이

get_GaussianNBC_online 함수를 호출함으로써 온라인 학습을 수행하였다.

```
In [10]: means_by_classes_online, stdevs_by_classes_online = get_GaussianNBC_online(train_samples, train_labels)

         pred_classes_online = predict(means_by_classes_online, stdevs_by_classes_online, test_samples)

         acc_online = get_Acc(pred_classes_online, test_labels)

         print('Online learning - before min-max normalization) Acc: %s' %acc_online)

         means_by_classes_minmax_online, stdevs_by_classes_minmax_online = get_GaussianNBC_online(train_samples_minmax, train

         pred_classes_minmax_online = predict(means_by_classes_minmax_online, stdevs_by_classes_minmax_online, test_samples_m

         acc_minmax_online = get_Acc(pred_classes_minmax_online, test_labels)

         print('Online learning - after min-max normalization) Acc: %s' %acc_minmax_online)
```

```
Online learning - before min-max normalization) Acc: 63.62
Online learning - after min-max normalization) Acc: 63.62
```

모델의 정확도는 min-max normalization 전후 모두 63.62%가 나왔다.
기존의 배치 학습 방법과 온라인 학습 방법은 학습하는 과정은 차이가 있지만, 결과적으로 모델의 성능에는 차이가 없음을 알 수 있다.

참고로 predict함수에서 conditional independence 공식을 그대로 이용하여 클래스가 주어졌을 때 각 feature의 probability를 곱해서 probability를 계산한 경우, 온라인 학습을 수행했을 때 모델의 정확도는 min-max normalization 전후 모두 54.87%가 나왔다.

```
In [16]: pred_classes_online_no_log = predict_no_log(means_by_classes_online, stdevs_by_classes_online, test_samples)

         acc_online_no_log = get_Acc(pred_classes_online_no_log, test_labels)

         print('Online learning - before min-max normalization) Acc: %s' %acc_online_no_log)

         pred_classes_minmax_online_no_log = predict_no_log(means_by_classes_minmax_online, stdevs_by_classes_minmax_online,

         acc_minmax_online_no_log = get_Acc(pred_classes_minmax_online_no_log, test_labels)

         print('Online learning - after min-max normalization) Acc: %s' %acc_minmax_online_no_log)
```

```
Online learning - before min-max normalization) Acc: 54.87
Online learning - after min-max normalization) Acc: 54.87
```

온라인 학습을 수행할 때에도 log 특성을 이용하지 않았을 때의 모델이 log 특성을 이용했을 때의 모델보다 성능이 떨어지는 것을 알 수 있다.