PRML Assignment I

2020 年 4 月 13 日

1 Dataset

Click here to visit the dataset.

The dataset is created using the ***create_dataset*** function, who writes the samples generated by the function ***generate_gaussian_distributed_samples*** to a text file. You can load the dataset by calling the function ***load_dataset*** and split it into training set and testing set by calling the function ***split_dataset***.

2 Description of models

A super class called ***LinearModel*** is defined in **source.py** for the two models and it contains attributes and method for plotting the classification result.

The linear discriminative model is defined by Python class ***LinearDiscriminativeModel*** in **source.py**. On the linear classification of n dimensional C classes, an instance of the model has C n × 1 weight vectors, i.e. an n × C weight matrix. The weight matrix is specified on calling the method ***train*** to train the model. The method train uses **gradient descent strategy** and you can specify the training options. Once the model is trained, you can classify samples by calling the ***classify*** method, who uses softmax function to generate the probability that samples belong to each class.

The linear generative model is defined by Python class ***LinearGenerativeModel*** in **source.py**. On the linear classification of n dimensional C classes, an instance of the model has C n × 1 weight vectors, C biases and one covariance matrix, i.e. an n × C weight matrix, a C × 1 bias vector and an n × n covariance matrix. The weight matrix, bias vector and covariance matrix are specified on calling the method ***train*** to train the model. The method train calculates mathematical expectations and covariance matrices of each class using maximum likelihood estimation. Once the model is trained, you can classify samples by calling the ***classify*** method, who uses a non overflow and underflow ***softmax*** function to generate the probability that samples belong to each class.

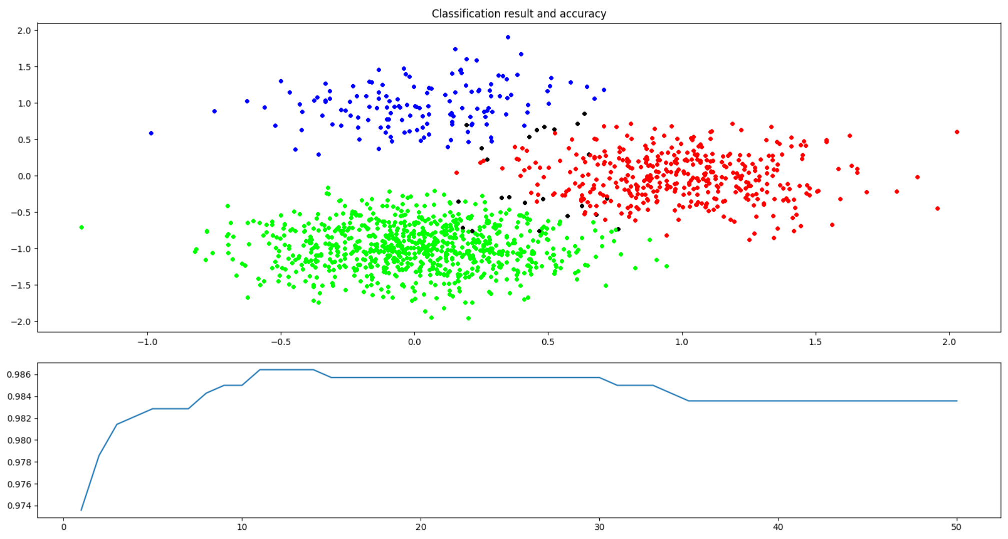
The **major differences** between these two models are the **training method** and **classification method**. On the **training method**, the **discriminative model** uses gradient descent strategy, i.e. an **optimization method**, to approach an ideal weight matrix, while the **generative model** tries to generalize the statistical properties of given samples using **statistical method**. As a result, the **generative model** doesn't need to train the model by iteration and can generate new samples from its estimation of samples' mathematical expectations and covariances while the

discriminative model have to train the model by means of iteration or other approaches and can not generate new samples. On the **classification method**, the **discriminative model** assumes the probability that samples belong to each class has the form of $softmax(w^T x)$, which comes from the **linear property** of the classifier. However, the **generative model** adopts $softmax(w^T x + b)$ as the form of probability that samples belong to each class because it assumes that **samples are compiled to Gaussian distributions**. Thus the **generative model** is extremely relied on its assumption of samples' statistical distribution and **is sensitive to outliers or samples compiled to multi-center Gaussian distribution** while the **discriminative models** can distinguish samples compiled to any distribution as long as they are linearly separable. The performance of two models is similar.

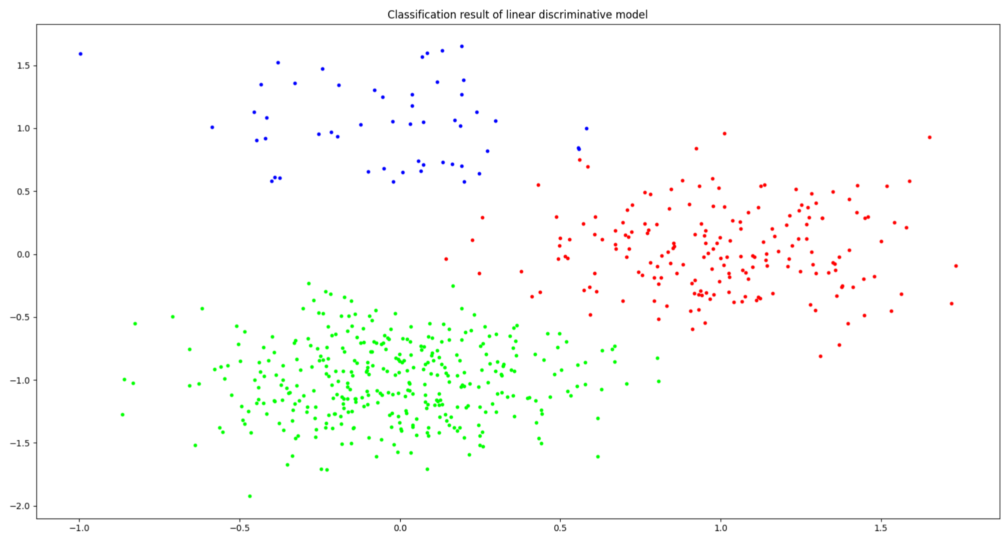
3 Performance of models

3.1 Normal dataset

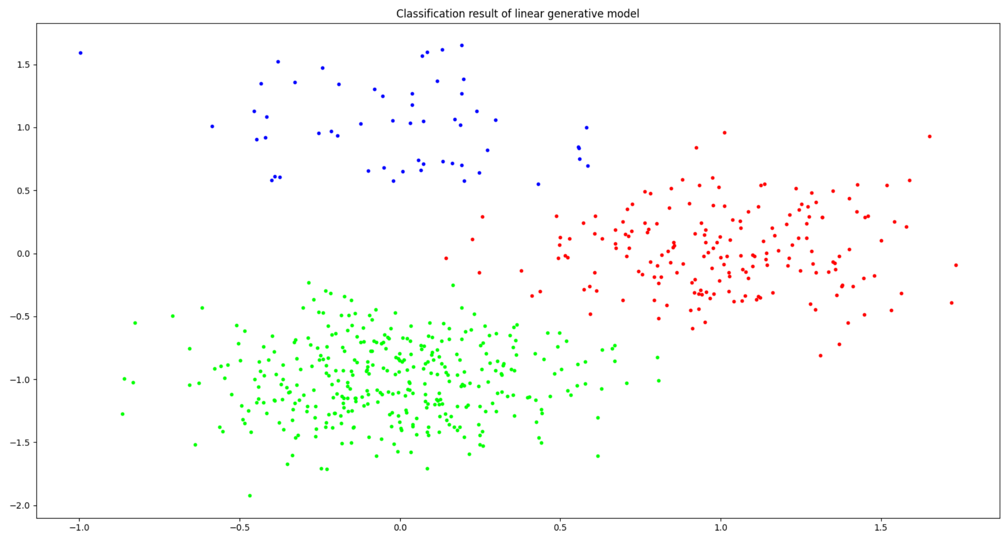
Use samples to train the **generative model**, the training process is shown in this picture. **The misclassified samples are shown in black**. It's evident that the model reach a high accuracy according to the lower part of the picture.



Then test the model on the testing dataset, the model can identified 98% of samples correctly. Note that the misclassified samples are **not** shown in the following picture since we won't tell the model the correct labels when calling the ***classify*** method.

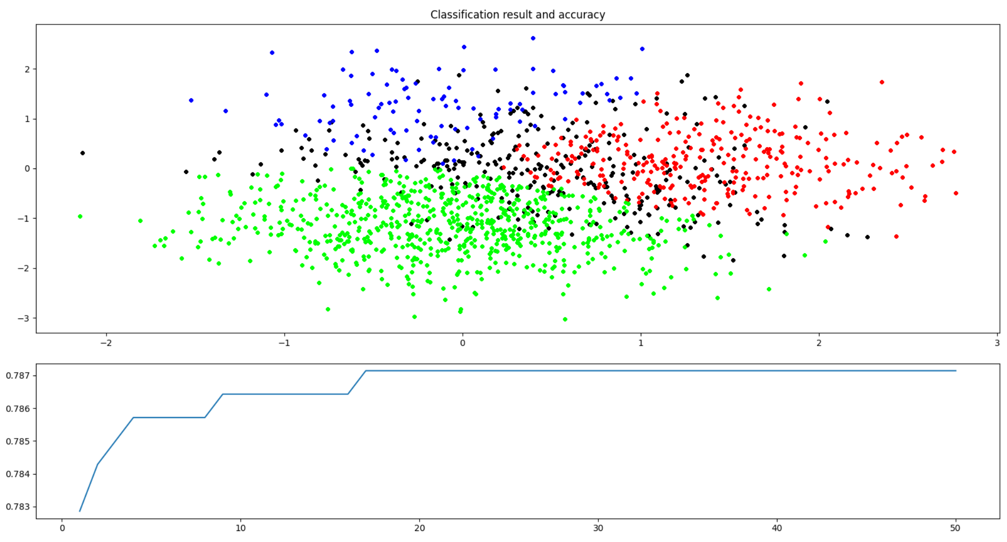


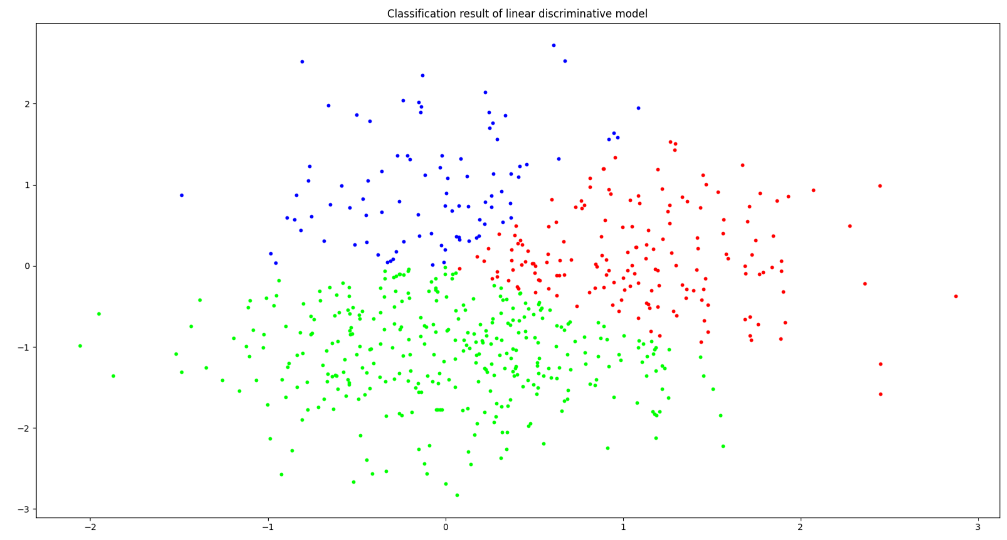
Apply the **generative model** to the same training set and testing set, the model can also identified 97.8% percent of samples correctly.

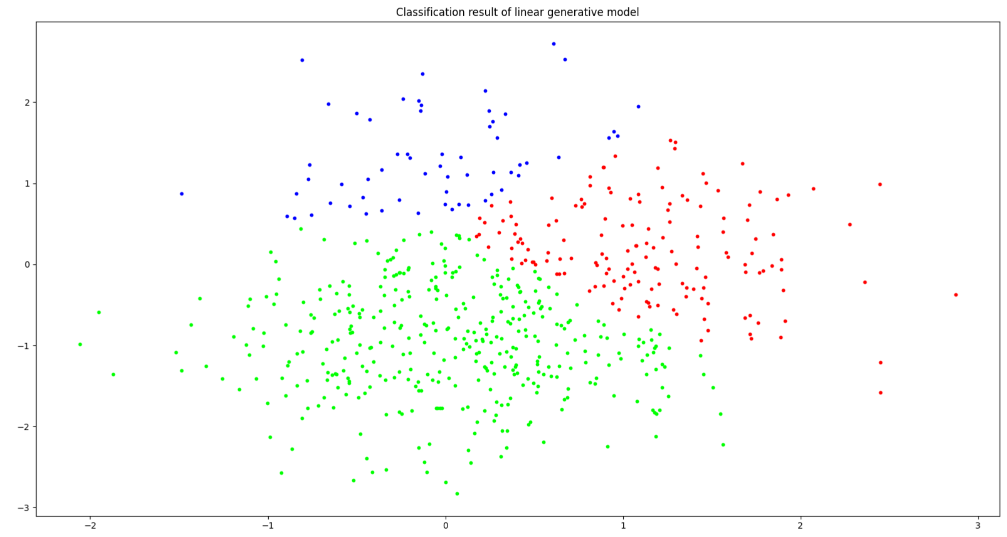


3.2 Dataset with greater covariance

Increase the covariance of Gaussian distributions from $0.1I$ to $0.5I$ and repeat above. Performance of **generative model** and **discriminative model** deteriorates and the proportion of black points in the classification result increases rapidly. This is because linear models can only learn a linear decision boundary and when samples of different classes start to mix together they become **linearly unseparable**. These two models can identify 80% and 80.6% of samples respectively, which implies that **their ability to adapt to linearly unseparable samples are similar**.


Classification result and accuracy
2
1
0
−1
−2
−3
−2
−1
0
1
2
3
0.787
0.786
0.785
0.784
0.783
0
10
20
30
40
50

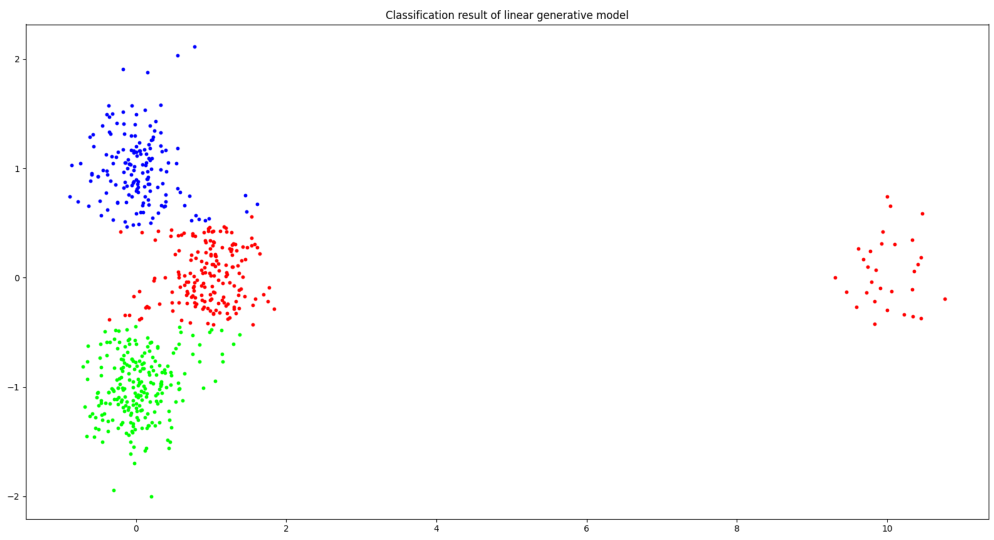

Classification result of linear discriminative model
2
1
0
−1
−2
−3
−2
−1
0
1
2
3


Classification result of linear generative model
2
1
0
−1
−2
−3
−2
−1
0
1
2
3

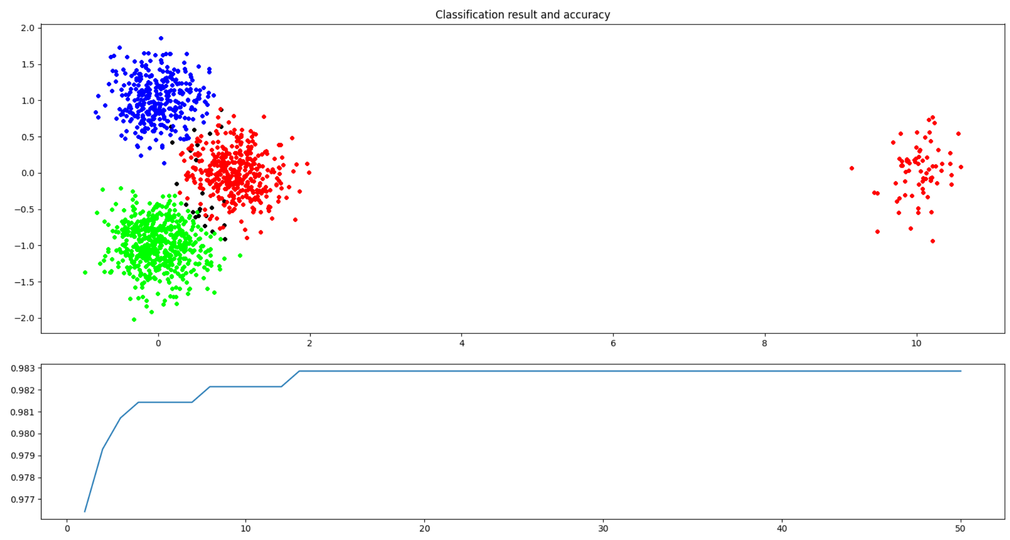
3.3 Dataset with outliers

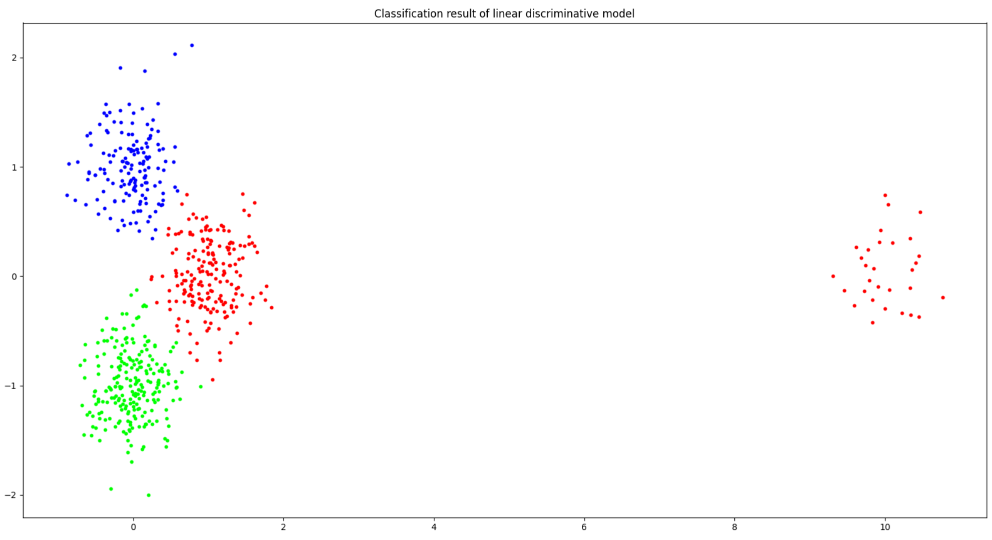
Add some outliers to the dataset and evaluate these two model. There are 15% of outliers in the dataset and they belong to the red class.

Performance of **generative model** deteriorates a lot. This is because the **generative model** calculates each class's mean value and uses **a shared covariance** to generate the probability that samples belong to each class. Outliers will **shift the mean value of red class and increase the shared covariance**. The **shift of the mean value** results in **the merge of the Gaussian distribution centers of red class**, which should be **seperated**, while **the increase the shared covariance** results in **the probability of samples belong to other classes in the vicinity of red class's Gaussian distribution center increases**. This problem **cannot** be solved by **calculating the covariance of each class** since **the mean value of red class is still shifted**. I think that **the clustering method** can handle the outliers easily. The generative model can identify 93.3% of samples.



Performance of **discriminative model** doesn't deteriorate greatly since it learn the linear decision boundary by minimizing the cost function and **the outliers does not generative more cost than normal samples** as long as they are correctly classified. The discriminative model can identify 97.7% of samples.





4 Demonstration

Run this demonstration in **Python Console** or copy these commands to the ***main*** function in **source.py** and run it. This demo is also at the beginning of **source.py**.

```
>>> from source import *
>>> create_dataset()
>>> samples, labels = load_dataset()
>>> set_of_samples, set_of_labels = split_dataset(samples, labels, [0.7])
>>> training_samples, testing_samples = set_of_samples
>>> training_labels, testing_labels = set_of_labels
>>> model1 = LinearDiscriminativeModel()
>>> model1.train(training_samples, training_labels, learning_rate=.9, max_epochs=20,
    plot_training_process=False)
Epoch          Accuracy
1              0.9714285714285714
2              0.9785714285714285
3              0.9814285714285714
4              0.9828571428571429
5              0.9828571428571429
6              0.9828571428571429
7              0.9828571428571429
8              0.9814285714285714
9              0.9814285714285714
10             0.9842857142857143
11             0.9842857142857143
12             0.9842857142857143
13             0.9842857142857143
14             0.9842857142857143
15             0.9842857142857143
16             0.9842857142857143
17             0.9842857142857143
18             0.9842857142857143
19             0.9842857142857143
20             0.9842857142857143
>>> labels1 = model1.classify(testing_samples)
>>> acc = sum(testing_labels == labels1) / testing_labels.size
>>> acc[0, 0]
0.98
>>> model1.w
matrix([[ 2.89275597, −1.44460552, −1.44815044],
[ 0.77766718, −4.04375914, 3.26609195]])
>>> model2 = LinearGenerativeModel()
>>> model2.train(training_samples, training_labels)
>>> labels2 = model2.classify(testing_samples)
>>> acc = sum(testing_labels == labels2) / testing_labels.size
>>> acc[0, 0]
0.98
>>> model2.w
matrix([[   9.57567184,    0.06218358,    0.07589503],
[ −0.11014837, −10.67945524, 11.09769756]])
>>> model2.b
matrix([[−5.85848281],
[−5.9195002 ],
[−8.16539317]])
```