EEL5840 Fundamental Machine Learning Homework 7

Hudanyun Sheng

All the code used are attached at the end of this report.

Problem 1

(1) When the Fisher discriminant ratio is used as the performance metric for the feature selection method, the related definitions and calculations are listed here:

the **within-class** scatter matrix:

$$S_w = \sum_{i=1}^{M} P_i \Sigma_i$$

where $\Sigma_i = E[(\mathbf{x} - \mu_\mathbf{i})(\mathbf{x} - \mu_\mathbf{i})^T]$ is the covariance for class ω_i, and P_i is the *a priori* probability of class ω_i.

the **between-class** scatter matrix:

$$S_b = \sum_{i=1}^{M} P_i(\mu_i - \mu_0)(\mu_i - \mu_0)^T$$

where $\mu_0 = \sum_{i=1}^{M} P_i \mu_i$ is the global mean vector.

One definition of the Fisher ratio is:

$$J = trace\{S_w^{-1} S_b\}$$

which is that we want to maximize the between class variance and minimize the within class variance at the same time. Diagonal loading is used sometimes to make the matrix S_b is invertible, the fisher ration can also be written as:

$$J = trace\{(S_w + \lambda I)^{-1} S_b\}$$

Forward feature selection(FFS) is to sequentially add features, evaluating performance with some performance metric, for here the Fisher ratio, with each potential next feature, adding the feature that improves results.

Backward feature selection(BFS) is to sequentially remove features, evaluating performance with the performance metric, removing the feature that does most reduces the performance metric when removed.

Based on all discussed above, both forward feature selection and backward selection methods are performed, and their performances are recorded and compared.

Some preprocessing procedures are also performed on the data set to compare the result. The method I used to normalize the data is

$$\frac{X - X_{min}}{X_{max} - X_{min}}$$

- **FFS**

(a) The matrix below shows the ratio of adding every feature **without** normalization:

iteration	feature1	feature2	feature3	feature4	feature5	feature6	feature7	feature8	feature9	feature10	feature11	feature12	feature13
1	0.3105	0.7435	0.0364	0.1385	0.3566	0.2861	0.9627	0.1801	0.1215	0.3113	0.0011	0.0144	0.0009
2	1.2944	1.8521	0.9730	0.9627	1.4963	0.9716	-	1.9544	1.0763	1.4045	1.1841	1.0514	0.9630
3	2.0053	2.8735	2.0267	3.3270	2.2836	1.9735	-	-	2.1920	2.3429	2.0265	1.9592	1.9547
4	3.3655	4.3476	3.4884	-	3.4685	3.4430	-	-	3.4576	3.5340	4.1669	3.3783	3.3282
5	4.3495	-	4.3498	-	4.4429	4.6051	-	-	4.4325	4.7155	5.2261	4.6164	4.3482
6	5.5285	-	5.3272	-	5.4111	5.5624	-	-	5.2533	5.3954	-	5.2927	5.2266
7	5.8454	-	5.6565	-	5.7403	-	-	-	5.5795	5.7289	-	5.6069	5.5632
8	-	-	5.8608	-	6.0263	-	-	-	5.9721	5.9648	-	5.8920	5.8457
9	-	-	6.0713	-	-	-	-	-	6.1024	6.0958	-	6.0768	6.0264
10	-	-	6.1582	-	-	-	-	-	-	6.1624	-	6.1610	6.1026
11	-	-	6.2245	-	-	-	-	-	-	-	-	6.2086	6.1626
12	-	-	-	-	-	-	-	-	-	-	-	6.2696	6.2245
13	-	-	-	-	-	-	-	-	-	-	-	-	6.2697

The sequence of adding a new feature is 7-8-4-2-11-6-1-5-9-10-3-12-13. Below is the curve showing how Fisher ratio changes with every new feature added in:

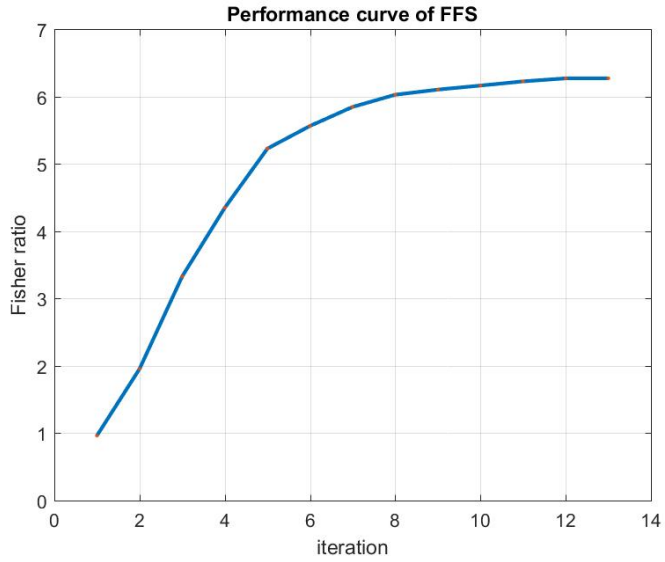


Figure 1: The changing of Fisher ratio with every feature added when using Forward feature selection, without feature normalization

We learn from this figure that if we keep adding every another feature, the ratio keep increasing fast until the 8th feature is added in, and the ratio is approaching stable (i.e. not changing any more) after 10-th

feature is added in. Both situations would be performed experiment on, i.e. combination of features {7, 8, 4, 2, 11, 6, 1, 5} as well as combination of features {7, 8, 4, 2, 11, 6, 1, 5, 9, 10} would be tested on.

(b) The matrix below shows the ratio of adding every feature **with** normalization:

iteration	feature1	feature2	feature3	feature4	feature5	feature6	feature7	feature8	feature9	feature10	feature11	feature12	feature13
1	0.0216	0.1211	0.0675	0.0148	0.0113	0.2029	0.4302	3.2685	0.0060	0.1614	0.0454	0.0184	0.0009
2	4.0463	3.3621	3.3504	4.4849	3.3542	3.3569	4.7945	-	3.4267	3.5860	4.3628	3.5855	3.2689
3	6.0263	4.9614	4.8878	5.5238	4.8261	6.4961	-	-	4.9922	5.0003	5.3303	4.9697	4.7945
4	7.4719	6.8331	6.5621	7.5148	6.5154	-	-	-	6.6678	6.6823	6.9431	6.5854	6.4961
5	9.0794	7.8939	7.6201	-	7.5619	-	-	-	7.6180	7.8076	8.0773	7.6791	7.5150
6	-	9.2633	9.1517	-	9.1769	-	-	-	9.2073	9.3881	10.4178	9.4163	9.0798
7	-	10.5207	10.4325	-	10.4226	-	-	-	10.8678	10.8960	-	10.4549	10.4200
8	-	10.9558	10.9375	-	10.8961	-	-	-	11.2763	-	-	10.9720	10.8987
9	-	11.3212	11.3000	-	11.2846	-	-	-	-	-	-	11.3623	11.2795
10	-	11.4534	11.3782	-	11.3679	-	-	-	-	-	-	-	11.3648
11	-	-	11.5119	-	11.4621	-	-	-	-	-	-	-	11.4559
12	-	-	-	-	11.5319	-	-	-	-	-	-	-	11.5141
13	-	-	-	-	-	-	-	-	-	-	-	-	11.5339

The sequence of adding a new feature is 8-7-6-4-1-11-10-9-12-2-3-5-13. Below is the curve showing how Fisher ratio changes with every new feature added in:

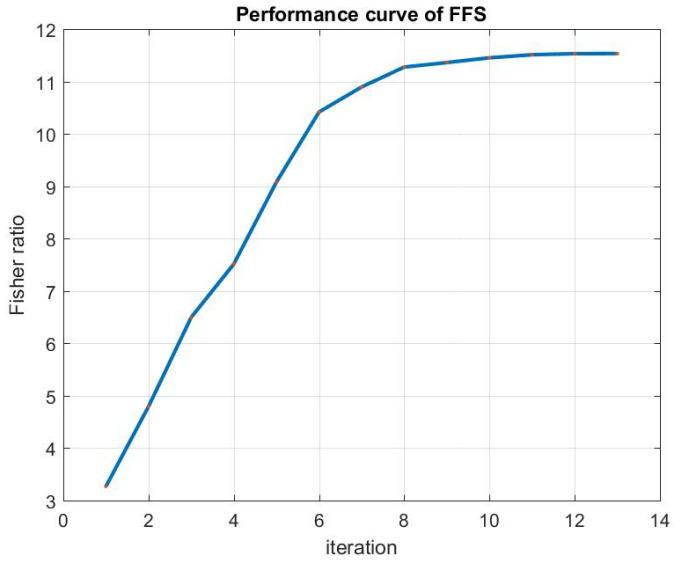


Figure 2: The changing of Fisher ratio with every feature added when using Forward feature selection, with feature normalization

We can learn from this figure that if we keep adding every another feature, the ratio keep increasing fast until the 8-th feature is added in, and the ratio is approaching stable (i.e. not changing any more) after 10-th feature is added in. Both situations would be performed experiment on, i.e. combination of features {8, 7, 6, 4, 1, 11, 10, 9} as well as combination of features {8, 7, 6, 4, 1, 11, 10, 9, 12, 2} would be tested

on.

- **BFS**

(a) The matrix below shows the ratio of adding every feature **without** normalization:

iteration	feature1	feature2	feature3	feature4	feature5	feature6	feature7	feature8	feature9	feature10	feature11	feature12	feature13
1	6.0407	5.4959	6.2089	4.4220	6.1466	5.9668	4.3286	4.1543	6.1851	6.2164	5.2861	6.2245	6.2696
2	6.0405	5.4959	6.2086	4.4219	6.1463	5.9668	4.3283	4.1540	6.1850	6.2163	5.2854	6.2245	-
3	6.0078	5.4956	6.1624	4.4206	6.1052	5.9006	4.2455	4.1472	6.1477	6.1582	5.1508	-	-
4	5.8811	5.1753	-	4.3754	6.0674	5.8366	4.0977	4.1202	6.0958	6.1024	5.1352	-	-
5	5.7780	5.1705	-	4.0433	5.9721	5.7758	4.0797	3.7330	6.0263	-	4.8475	-	-
6	5.7403	5.0728	-	3.4722	5.8454	5.7164	4.0366	3.0088	-	-	4.6933	-	-
7	5.5624	4.8308	-	3.1573	-	5.5285	3.8657	2.7311	-	-	4.6053	-	-
8	-	4.3325	-	3.0230	-	5.2261	3.6864	2.1873	-	-	4.6051	-	-
9	-	4.1669	-	2.9446	-	-	3.6023	2.1115	-	-	4.3476	-	-
10	-	3.3270	-	2.8735	-	-	2.3933	1.8525	-	-	-	-	-
11	-	-	-	1.9544	-	-	1.5743	0.9627	-	-	-	-	-
12	-	-	-	-	-	-	0.1801	0.9627	-	-	-	-	-

The sequence of dropping another feature is 13-12-3-10-9-5-1-6-11-2-4-8, finally if only one feature is kept, that would be feature 7. Below is the curve showing how Fisher ratio changes with every new feature dropped from the total feature set:

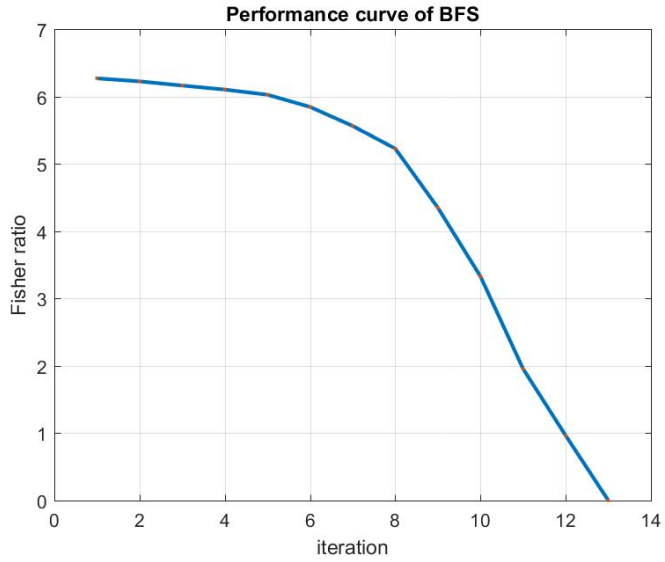


Figure 3: The changing of Fisher ratio with every feature dropped from the total feature set when using Backward feature selection, without feature normalization

We can learn from this figure that if we keep dropping every another feature, the ratio keep decreasing slowly (barely not changing) until the 8-th feature is dropped, the ratio starts decreasing quickly. So the result would be features {6, 11, 2, 4, 8, 7} should be kept at least, and another test on keeping features {

5, 1, 6, 11, 2, 4, 8, 7 } is also performed.

(b) The matrix below shows the ratio of adding every feature **with** normalization:

iteration	feature1	feature2	feature3	feature4	feature5	feature6	feature7	feature8	feature9	feature10	feature11	feature12	feature13
1	9.0094	11.3898	11.4646	8.8772	11.5141	10.0034	9.9347	3.0162	11.1820	11.1178	10.5592	11.4068	11.5319
2	9.0094	11.3875	11.4621	8.8752	11.5119	10.0003	9.9331	3.0145	11.1803	11.1159	10.5590	11.4041	-
3	9.0071	11.3782	11.4534	8.8751	-	9.9835	9.9138	3.0145	11.1748	11.0741	10.4401	11.3802	-
4	8.9628	11.3623	-	8.8172	-	9.9559	9.9085	2.6545	11.0817	11.0409	10.2613	11.3212	-
5	8.6221	-	-	8.7114	-	9.9378	9.8947	2.2184	10.9720	10.9153	9.9777	11.2763	-
6	8.5768	-	-	8.6960	-	9.7891	9.7379	2.1904	10.8960	10.8678	9.4782	-	-
7	8.4663	-	-	8.6548	-	9.3799	9.2695	2.0892	-	10.4178	9.3881	-	-
8	8.0773	-	-	8.3911	-	8.8792	8.5474	2.0125	-	-	9.0794	-	-
9	7.5148	-	-	7.4719	-	7.3465	6.1271	1.7184	-	-	-	-	-
10	-	-	-	6.4961	-	5.5238	4.7918	1.6456	-	-	-	-	-
11	-	-	-	-	-	4.7945	3.3569	1.5388	-	-	-	-	-
12	-	-	-	-	-	-	3.2685	0.4302	-	-	-	-	-

The sequence of dropping another feature is 13-5-3-2-12-9-10-11-1-4-6-7-8, if finally only one feature is kept, that would be feature 8. Below is the curve showing how Fisher ratio changes with every new feature dropped from the total feature set:

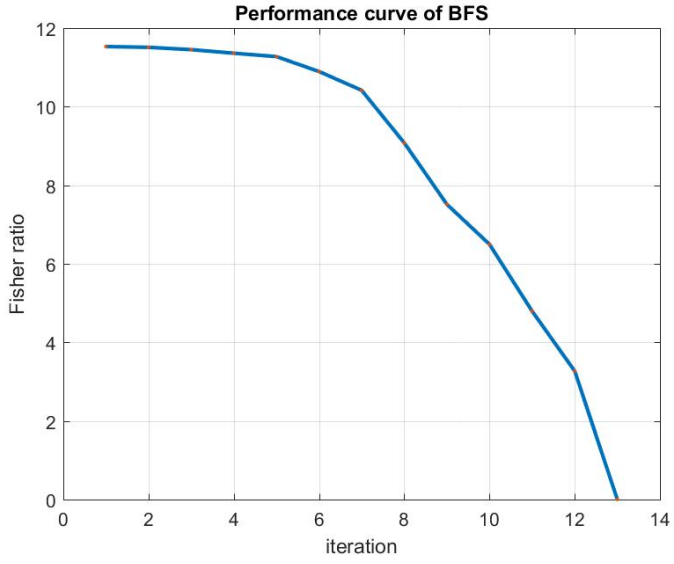


Figure 4: The changing of Fisher ratio with every feature dropped from the total feature set when using Backward feature selection, with feature normalization

We can learn from this figure that if we keep dropping every another feature, the ratio keep decreasing slowly (barely not changing) until the 7th feature is dropped, the ratio starts decreasing quickly. So the result would be features {10, 11, 1, 4, 6, 7, 8} being kept at least, and another test on keeping features { 12, 9,10, 11, 1, 4, 6, 7, 8 } is also performed.

Based on the discussions above, the number of features to be kept are listed in the table below:

	no normalization	with normalization
FFS	8	8
BFS	6	6

(2) Theoretically, I do not think the classification performance using the Bayes Classifier have best performance using the feature set I determined using FFS and BFS. The reasons are listed below:

- The original data set has the dimension of 13, and with **forward feature selection** we may finally keep 8 features or so, and with **backward feature selection** we may finally keep 6 features or so. But no matter the dimension of the data being 13, 8 or 6, the number of the instances is only 6497. Based on *the Curse of dimensionality* the number of the instances is far from enough to densely populate the space of dimension of 6, 8, or 13.
- Either **forward feature selection** or **backward feature selection** is performed in a greedy way, i.e. the local optima is found every step, but there is no guarantee for global optimal. And this is also the key reason I insist the feature set I determined either using **FFS** or **BFS** will not perform better than the full feature set.
- We choose the features by observing the curve decide when it becomes changing slowly, but actually it is still changing- the Fisher ratio is still increasing after 8-th feature being added in the feature set, and the Fisher ratio is still decreasing after 6-th feature being dropped from the feature set.

Experiments are also performed to compare the performance using Bayes Classifier. I use "0" to represent red wine, and "1" to represent white wine. 70% of the data are used as training set, the the remaining 30% of the data are used as test set.

First set of experiments compared the performance with or without data normalization. The confusion matrices and the correct classification ratios are calculated. Based on the results I learned the importance of data normalization (though not that obvious in this data set). Later experiments are based normalized data and measured by the correct classification ratio. Later experiments are based normalized data and measured by the correct classification ratio. As stated before, with **FFS**, 8 features are kept, they are { 8, 7, 6, 4, 1, 11, 10, 9}, and with **BFS**, 6 features are kept, they are are {8, 7, 6, 4, 1, 11}. The table below summaries my results:

		correct classification rate on train set		correct classification rate on test set	
		red wine	white wine	red wine	white wine
All features	no normalize	0.9946	0.9907	0.9778	0.9890
	with normalize	1.0000	0.9994	1.0000	0.9986
Selected featues	FFS	1.0000	0.9991	1.0000	0.9986
	BFS	1.0000	0.9988	0.9939	0.9986

Based on the results above, we would see that the performance with data normalization would be better (though not super obvious in this data set); and on this data set, the performance with feature selected by either FFS or BFS is not better than with full features set. Though conclusions like "with fewer feature we still get similar results" may be made, but I stick to the comment I made before "the number of samples is not enough based on the dimension of the given data set for us to make a conclusion".

Problem 2

Below is a summary of how EM learn the parameters: Initialize mixture component mean μ_i, variance Σ_i, and proportion π_i.

The "responsibility" of every component for every data point is calculated $C_{ik} = p(z_i = k|x_i, \theta) = \frac{\pi_i^t p_{z_i}(x_i|\theta_{z_i}^t, z_i)}{\sum_{k=1}^{K} \pi_i^t p_{z_i}(x_i|\theta_{z_i}^t, z_i)}$.

Then μ, Σ, and π are updated iteratively.

(1) Since the **EM algorithm** need the input of the number of mixture component, before EM is implemented to determine the mixture proportion, PCA is performed to get some initial guess of number of mixture components. The original dimension of the data set is 5, so I tried to project it into dimension of 3. The visualizations are shown below:

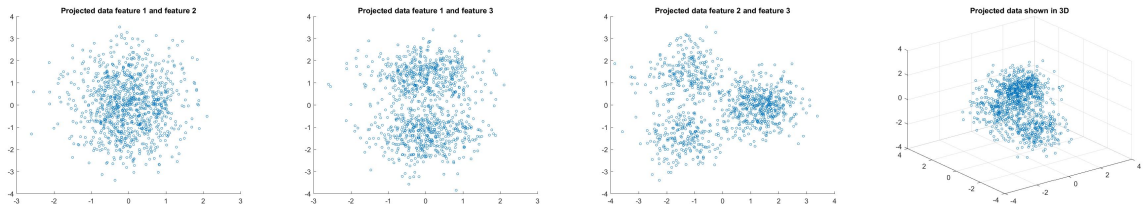
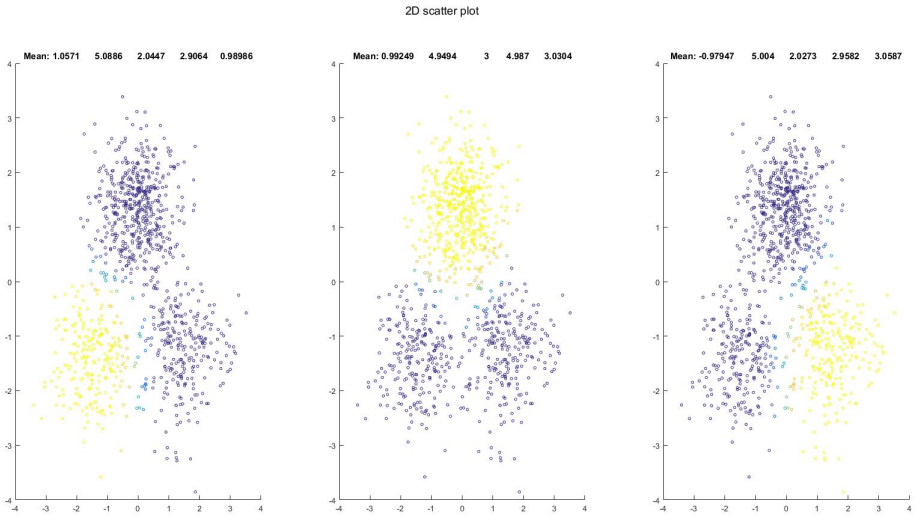
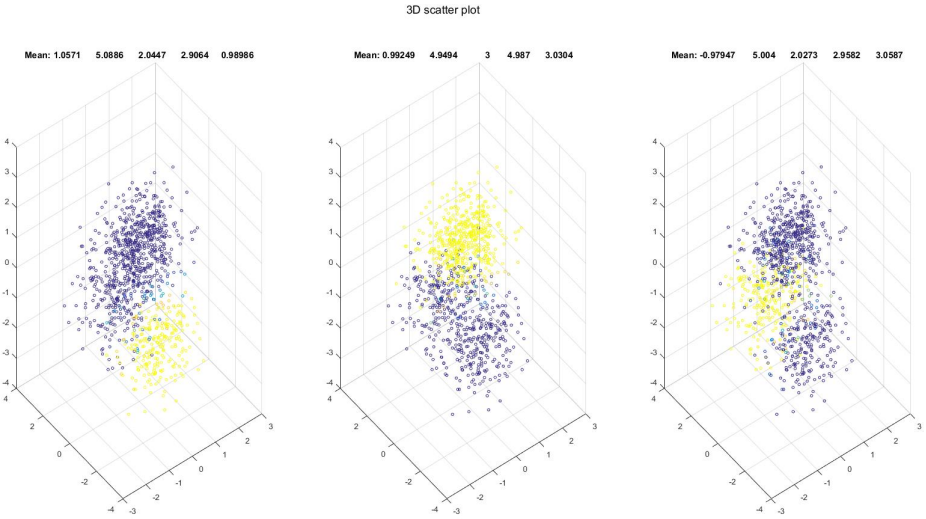


Figure 5: From left to right: the scatter plot with feature 1 and 2 after performing PCA, the scatter plot with feature 1 and 3 after performing PCA, the scatter plot with feature 2 and 3 after performing PCA, the 3D scatter plot with all 3 features after performing PCA.

With the help of principal component analysis and the visualization, I go the initial guess of the number of mixture components may be 3. It can also be greater than 3.

Then **EM algorithm** is implemented on the data set. I started with setting the "number of mixture components" to 3. The result of EM is shown below. In this figure, the colors are considered as "pseudo color", i.e. for a given data point, different "responsibilities" of different mixture component would result in different color.





The results of experiments I did are shown below, the parameters learned are pretty consistent:

	Experiment 1					Experiment 2					Experiment 3					Experiment 4					Experiment 5				
pi	0.2589	0.5014	0.2397			0.2397	0.5014	0.2589			0.5014	0.2397	0.2589			0.2589	0.2397	0.5014			0.2589	0.2397	0.5014		
mu	-0.9795	5.0040	2.0273	2.9582	3.0587	1.0571	5.0886	2.0447	2.9064	0.9898	0.9925	4.9494	3.0000	4.9870	3.0304	-0.9795	5.0040	2.0273	2.9582	3.0587	-0.9795	5.0040	2.0273	2.9582	3.0587
	0.9925	4.9494	3.0000	4.9870	3.0304	0.9925	4.9494	3.0000	4.9870	3.0304	1.0571	5.0886	2.0447	2.9064	0.9898	1.0571	5.0886	2.0447	2.9064	0.9899	1.0571	5.0886	2.0447	2.9064	0.9899
	1.0571	5.0886	2.0447	2.9064	0.9899	-0.9794	5.0040	2.0273	2.9582	3.0587	-0.9794	5.0040	2.0273	2.9582	3.0587	0.9925	4.9494	3.0000	4.9870	3.0304	0.9925	4.9494	3.0000	4.9870	3.0304
sigma	0.4782	0.5377	0.5336	0.5612	0.5520	0.4340	0.4662	0.4143	0.4238	0.4690	0.4678	0.5159	0.2524	0.5360	0.4245	0.4782	0.5377	0.5335	0.5612	0.5520	0.4782	0.5377	0.5335	0.5612	0.5520
	0.4678	0.5159	0.2524	0.5360	0.4245	0.4678	0.5159	0.2524	0.5360	0.4245	0.4340	0.4662	0.4143	0.4238	0.4690	0.4340	0.4662	0.4143	0.4238	0.4690	0.4340	0.4662	0.4143	0.4238	0.4690
	0.4340	0.4662	0.4143	0.4238	0.4690	0.4783	0.5377	0.5336	0.5612	0.5520	0.4783	0.5377	0.5336	0.5612	0.5520	0.4678	0.5159	0.2524	0.5360	0.4245	0.4678	0.5159	0.2524	0.5360	0.4245
	Experiment 6					Experiment 7					Experiment 8					Experiment 9					Experiment 10				
pi	0.2589	0.2397	0.5014			0.2589	0.5014	0.2397			0.5014	0.2397	0.2589			0.2397	0.5014	0.2589			0.5014	0.2589	0.2397		
mu	-0.9795	5.0040	2.0273	2.9582	3.0587	-0.9795	5.0040	2.0273	2.9582	3.0587	0.9925	4.9494	3.0000	4.9870	3.0304	1.0571	5.0886	2.0447	2.9064	0.9899	0.9925	4.9494	3.0000	4.9870	3.0304
	1.0571	5.0886	2.0447	2.9064	0.9899	0.9925	4.9494	3.0000	4.9870	3.0304	1.0571	5.0886	2.0447	2.9064	0.9899	0.9925	4.9494	3.0000	4.9870	3.0304	-0.9794	5.0040	2.0273	2.9582	3.0587
	0.9925	4.9494	3.0000	4.9870	3.0304	1.0571	5.0886	2.0447	2.9064	0.9899	-0.9795	5.0040	2.0273	2.9582	3.0587	-0.9795	5.0040	2.0273	2.9582	3.0587	1.0571	5.0886	2.0447	2.9064	0.9898
sigma	0.4782	0.5377	0.5336	0.5612	0.5520	0.4782	0.5377	0.5336	0.5612	0.5520	0.4678	0.5159	0.2524	0.5360	0.4245	0.4340	0.4662	0.4143	0.4238	0.4690	0.4678	0.5159	0.2524	0.5360	0.4245
	0.4340	0.4662	0.4143	0.4238	0.4690	0.4678	0.5159	0.2524	0.5360	0.4245	0.4340	0.4662	0.4143	0.4238	0.4690	0.4678	0.5159	0.2524	0.5360	0.4245	0.4783	0.5377	0.5336	0.5612	0.5520
	0.4678	0.5159	0.2524	0.5360	0.4245	0.4340	0.4662	0.4143	0.4238	0.4690	0.4782	0.5377	0.5336	0.5612	0.5520	0.4782	0.5377	0.5335	0.5612	0.5520	0.4340	0.4662	0.4143	0.4238	0.4690

From the figures above, I concluded that if assuming 3 mixture components, the clusters are pretty consistent. And also, by performing the EM algorithm several times, the mixture proportions as well as the mean and covariance

for every cluster are also pretty consistent.

If assuming 4 mixture components, the mixture proportions as well as the mean and covariance for every cluster are not consistent. The results of experiments I did are shown below, the parameters learned are not consistent:

	Experiment 1					Experiment 2					Experiment 3					Experiment 4					Experiment 5				
pi	0.2405	0.2374	0.2583	0.2638		0.2367	0.2402	0.5023	0.0209		0.2218	0.2793	0.2404	0.2585		0.2442	0.0127	0.2419	0.5012		0.2583	0.2660	0.2405	0.2352	
mu	1.0579	5.0889	2.0472	2.9075	0.9946	1.0560	5.0844	2.0539	2.9099	0.9655	0.9507	5.0213	3.2576	4.8141	3.0834	-0.9504	4.9945	2.0287	3.0116	3.1248	-0.9788	5.0019	2.0215	2.9568	3.0594
	0.9502	5.0145	3.2372	4.8277	3.0739	-1.0204	4.9497	2.0510	2.9223	3.0725	1.0205	4.8931	2.7991	5.1266	2.9889	-1.6956	5.1701	2.0131	1.9755	2.0067	1.0237	4.8922	2.7915	5.1313	2.9917
	-0.9789	5.0019	2.0214	2.9567	3.0594	0.9920	4.9485	2.9994	4.9841	3.0304	1.0580	5.0893	2.0470	2.9070	0.9945	1.0482	5.0888	2.0440	2.9044	0.9973	1.0579	5.0890	2.0472	2.9074	0.9946
	1.0244	4.8922	2.7906	5.1321	2.9923	-0.2750	5.7118	1.6218	3.2992	2.8754	-0.9785	5.0021	2.0219	2.9571	3.0593	0.9925	4.9493	3.0002	4.9876	3.0302	0.9505	5.0156	3.2403	4.8259	3.0753
sigma	0.4345	0.4677	0.4154	0.4228	0.4746	0.4365	0.4673	0.4060	0.4210	0.4286	0.5875	0.6644	0.1726	0.5574	0.3181	0.4525	0.5535	0.5155	0.5114	0.5045	0.4806	0.5370	0.5252	0.5605	0.5519
	0.5853	0.6645	0.1790	0.5692	0.3260	0.4488	0.5159	0.5298	0.5479	0.5885	0.3779	0.3903	0.2226	0.4722	0.5065	0.3041	0.2437	0.8743	0.4847	0.0240	0.3697	0.3771	0.2235	0.4617	0.5100
	0.4806	0.5370	0.5251	0.5604	0.5519	0.4687	0.5161	0.2532	0.5388	0.4253	0.4346	0.4674	0.4155	0.4222	0.4747	0.4428	0.4650	0.4152	0.4242	0.4750	0.4345	0.4676	0.4154	0.4227	0.4746
	0.3682	0.3748	0.2238	0.4598	0.5106	0.7765	0.2350	0.4525	0.6251	0.0060	0.4807	0.5372	0.5253	0.5605	0.5518	0.4679	0.5157	0.2524	0.5354	0.4247	0.5856	0.6644	0.1780	0.5676	0.3249

	Experiment 6					Experiment 7					Experiment 8					Experiment 9					Experiment 10				
pi	0.2402	0.0209	0.5023	0.2367		0.2611	0.0221	0.2148	0.5020		0.2397	0.4149	0.0870	0.2584		0.2478	0.2581	0.2535	0.2406		0.5008	0.2471	0.2396	0.0125	
mu	-1.0204	4.9497	2.0510	2.9223	3.0725	-0.9680	5.0073	2.0256	2.9563	3.0545	1.0563	5.0880	2.0449	2.9061	0.9898	0.9481	5.0086	3.2209	4.8354	3.0671	0.9939	4.9492	2.9997	4.9880	3.0311
	-0.2750	5.7118	1.6218	3.2992	2.8754	1.4200	5.1836	2.0925	2.4652	1.0041	1.1457	4.9814	2.9803	4.9777	3.0287	-0.9793	5.0016	2.0209	2.9564	3.0596	-0.9276	4.9923	2.0331	2.9591	3.0035
	0.9920	4.9485	2.9994	4.9841	3.0304	1.0233	5.0737	2.0410	2.9527	0.9670	0.2570	4.7998	3.0962	5.0212	3.0369	1.0287	4.8936	2.7882	5.1362	2.9960	1.0588	5.0889	2.0442	2.9067	0.9899
	1.0560	5.0844	2.0539	2.9099	0.9655	0.9940	4.9504	2.9992	4.9846	3.0301	-0.9807	5.0036	2.0245	2.9580	3.0594	1.0577	5.0886	2.0473	2.9080	0.9947	-1.9869	5.2349	1.9724	2.9782	4.0949
sigma	0.4488	0.5159	0.5298	0.5479	0.5885	0.4909	0.5387	0.5344	0.5630	0.5496	0.4343	0.4677	0.4144	0.4237	0.4692	0.5849	0.6656	0.1843	0.5775	0.3306	0.4662	0.5158	0.2526	0.5352	0.4249
	0.7765	0.2350	0.4525	0.6251	0.0060	0.0874	0.1356	0.4522	0.0119	1.5065	0.3757	0.4439	0.2631	0.5085	0.4371	0.4804	0.5369	0.5250	0.5604	0.5520	0.4424	0.5410	0.5584	0.5834	0.5219
	0.4687	0.5161	0.2532	0.5388	0.4253	0.4536	0.4993	0.4084	0.4435	0.3373	0.2603	0.8338	0.1883	0.6805	0.3636	0.3604	0.3627	0.2262	0.4495	0.5135	0.4322	0.4664	0.4142	0.4236	0.4703
	0.4365	0.4673	0.4060	0.4210	0.4286	0.4680	0.5154	0.2526	0.5390	0.4223	0.4782	0.5361	0.5312	0.5625	0.5522	0.4345	0.4679	0.4153	0.4234	0.4745	0.0695	0.4117	0.0768	0.1643	0.0279

One of the scatter plots are shown below:

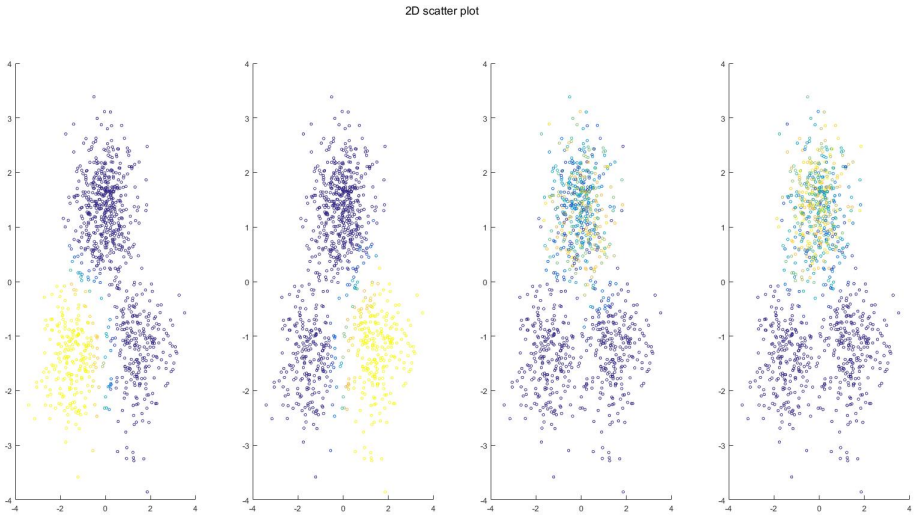


Figure 6: 2-d scatter plot assuming 4 component

From the figures we can also see that there are two clusters which are pretty consistent, but the other two are always mixing together. And also from the experiment results, by run the EM code several, the parameters changed a lot and are not consistent. Sometimes the mixture proportion for one cluster is tend to be really small, which is a sign to merge with other cluster(s).

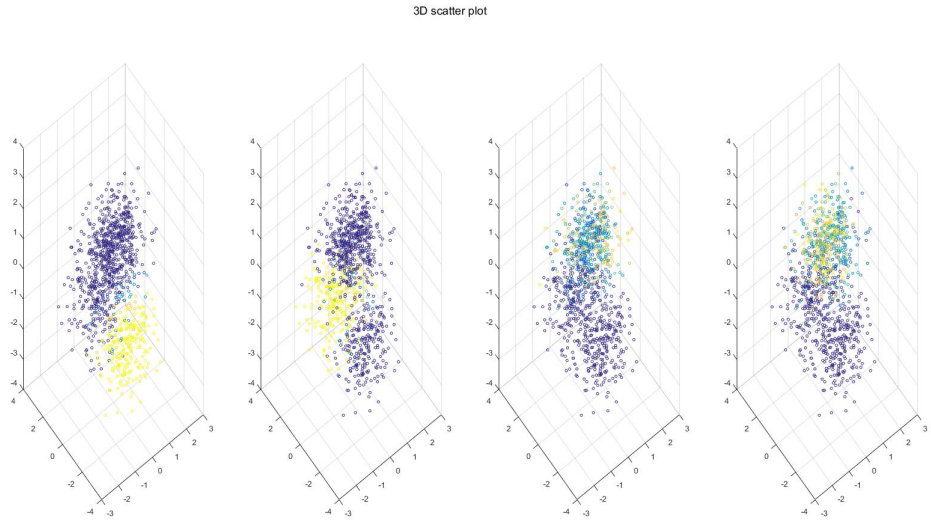


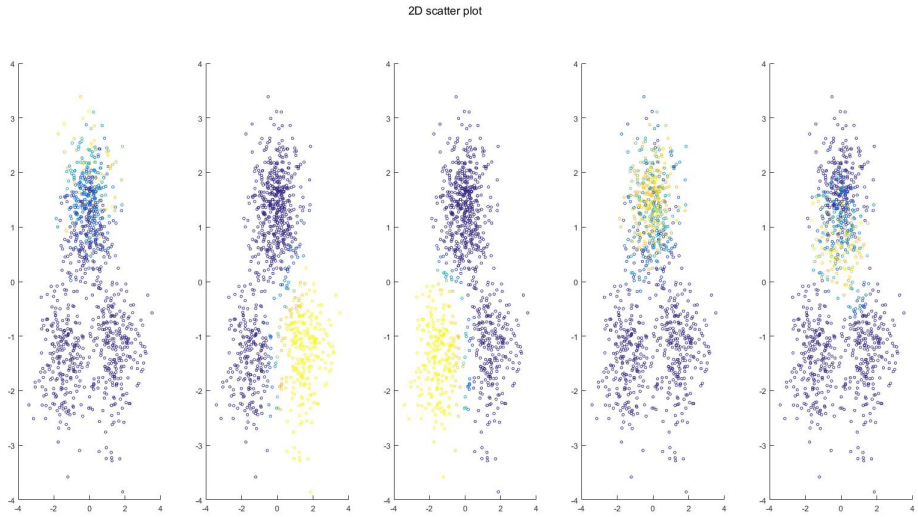
Figure 7: 3-d scatter plot assuming 4 component

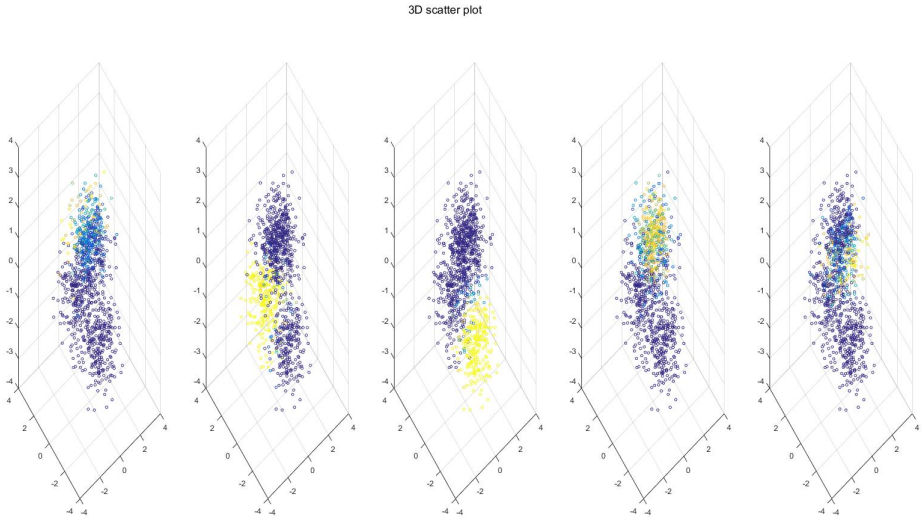
If assuming 5 mixture components, the mixture proportions as well as the mean and covariance for every cluster are not consistent. The results of experiments I did are shown below, the parameters learned are not consistent:

	Experiment 1					Experiment 2					Experiment 3					Experiment 4					Experiment 5				
pi	0.2438	0.2419	0.0127	0.4129	0.0887	0.1675	0.2596	0.2404	0.2796	0.0530	0.2738	0.2272	0.0128	0.2427	0.2435	0.0143	0.0471	0.2534	0.5024	0.1828	0.2594	0.0220	0.2159	0.2248	0.2778
	-0.9511	4.9941	2.0258	3.0118	3.1259	0.9694	5.0647	3.2974	4.7008	3.1233	1.0216	4.8926	2.7943	5.1293	2.9907	0.7487	5.8304	1.6880	2.7467	2.1348	-0.9700	5.0041	2.0167	2.9527	3.0561
	1.0476	5.0884	2.0441	2.9040	0.9973	-0.9744	5.0026	2.0247	2.9602	3.0586	0.9504	5.0191	3.2531	4.8192	3.0786	1.0154	5.0703	1.8601	2.4386	0.8117	1.4185	5.1830	2.0893	2.4650	1.0013
mu	-1.6968	5.1683	2.0195	1.9810	2.0062	1.0591	5.0905	2.0469	2.9062	0.9956	-1.6997	5.1651	2.0298	1.9886	2.0053	-1.0046	4.9880	2.0311	2.9617	3.0691	1.0235	5.0723	2.0444	2.9559	0.9718
	1.1462	4.9835	2.9808	4.9765	3.0316	1.1293	4.9539	2.8422	5.0941	3.0165	1.0489	5.0894	2.0464	2.9052	1.0022	0.9928	4.9508	2.9985	4.9834	3.0323	1.0574	4.9151	2.8100	5.1452	3.0236
	0.2729	4.7929	3.0931	5.0312	3.0206	0.3343	4.5569	2.9191	5.3687	2.8094	-0.9489	4.9924	2.0215	3.0103	3.1266	1.0650	5.0516	2.1128	3.0336	0.9767	0.9327	4.9831	3.1559	4.8508	3.0381
	0.4526	0.5518	0.5128	0.5129	0.5040	0.5932	0.6308	0.1625	0.4892	0.2726	0.3748	0.3851	0.2221	0.4670	0.5075	0.6391	0.1941	0.7815	0.0545	0.3905	0.4925	0.5373	0.5253	0.5617	0.5504
	0.4430	0.4662	0.4152	0.4239	0.4753	0.4835	0.5368	0.5262	0.5619	0.5513	0.5874	0.6640	0.1738	0.5618	0.3220	0.5377	0.2093	0.3808	0.1477	0.9358	0.0879	0.1351	0.4522	0.0119	1.5053
sigma	0.3030	0.2443	0.8798	0.4887	0.0239	0.4344	0.4661	0.4153	0.4200	0.4760	0.2997	0.2452	0.8898	0.4950	0.0238	0.4496	0.5304	0.5303	0.5679	0.5501	0.4538	0.5018	0.4095	0.4454	0.3416
	0.3769	0.4402	0.2637	0.5078	0.4337	0.3280	0.3501	0.2444	0.4616	0.4852	0.4436	0.4663	0.4163	0.4229	0.4808	0.4688	0.5151	0.2536	0.5398	0.4192	0.3311	0.3177	0.2425	0.4118	0.5170
	0.2682	0.8398	0.1876	0.6736	0.3838	0.2940	0.8349	0.1578	0.5577	0.5327	0.4548	0.5530	0.5052	0.5109	0.5034	0.4039	0.5204	0.3787	0.4481	0.2867	0.5861	0.6735	0.2067	0.6091	0.3415

	Experiment 6					Experiment 7					Experiment 8					Experiment 9					Experiment 10				
pi	0.2606	0.0221	0.4181	0.0844	0.2147	0.0220	0.2159	0.2594	0.2801	0.2226	0.0207	0.0124	0.2285	0.5021	0.2363	0.1366	0.2611	0.0219	0.2147	0.3658	0.0132	0.0125	0.2465	0.4878	0.2400
	-0.9691	5.0069	2.0228	2.9562	3.0549	1.4184	5.1830	2.0893	2.4650	1.0013	-0.2970	5.7133	1.5933	3.2833	2.8752	0.8109	4.4777	2.8857	5.0273	2.9779	1.8850	5.4601	3.1461	5.3640	2.8260
	1.4200	5.1838	2.0927	2.4652	1.0045	1.0234	5.0722	2.0444	2.9560	0.9718	-1.9900	5.2405	1.9697	2.9820	4.0930	-0.9693	5.0100	2.0279	2.9560	3.0537	-1.9872	5.2345	1.9725	2.9786	4.0946
mu	1.1450	4.9809	2.9789	4.9755	3.0254	-0.9701	5.0041	2.0166	2.9526	3.0561	-0.9637	4.9337	2.0598	2.9214	3.0132	1.4209	5.1829	2.0925	2.4651	1.0044	-0.9306	4.9924	2.0321	2.9575	3.0036
	0.2419	4.8032	3.1019	5.0187	3.0520	0.9327	4.9821	3.1532	4.8522	3.0370	0.9930	4.9488	2.9990	4.9847	3.0308	1.0248	5.0778	2.0400	2.9488	0.9677	0.9687	4.9353	2.9954	4.9771	3.0378
	1.0222	5.0727	2.0412	2.9526	0.9665	1.0586	4.9156	2.8098	5.1463	3.0249	1.0583	5.0847	2.0537	2.9104	0.9644	1.0625	5.1228	3.0397	4.9693	3.0481	1.0589	5.0888	2.0452	2.9082	0.9913
	0.4910	0.5371	0.5320	0.5643	0.5499	0.0879	0.1351	0.4522	0.0119	1.5053	0.8118	0.2339	0.4278	0.6227	0.0061	0.4493	0.4646	0.3145	0.5099	0.5671	0.0745	0.0086	0.0360	0.1032	0.8616
	0.0875	0.1356	0.4521	0.0119	1.5046	0.4537	0.5018	0.4095	0.4455	0.3415	0.0704	0.4082	0.0749	0.1624	0.0281	0.4880	0.5389	0.5354	0.5613	0.5484	0.0696	0.4115	0.0767	0.1641	0.0279
sigma	0.3752	0.4466	0.2627	0.5130	0.4376	0.4925	0.5372	0.5254	0.5616	0.5504	0.4142	0.5164	0.5562	0.5709	0.5610	0.0869	0.1354	0.4525	0.0119	1.5139	0.4395	0.5410	0.5579	0.5827	0.5224
	0.2544	0.8305	0.1883	0.6824	0.3447	0.5852	0.6722	0.2075	0.6099	0.3421	0.4674	0.5161	0.2534	0.5384	0.4255	0.4536	0.4954	0.4083	0.4384	0.3387	0.4552	0.5222	0.2583	0.5443	0.4103
	0.4538	0.5013	0.4086	0.4436	0.3369	0.3296	0.3158	0.2428	0.4093	0.5179	0.4342	0.4676	0.4057	0.4208	0.4284	0.4560	0.4224	0.2253	0.5470	0.3722	0.4322	0.4667	0.4148	0.4250	0.4712

The proportion of one or two component(s) are tend to be small, which indicates merging with other components. One of the scatter plots are shown below:





From the figures we can also see that there are two clusters which are pretty consistent, but the other three are always mixing together. And by run the EM code several, the parameters changed a lot and are not consistent.

(2) My estimate for the mixture proportions are $p_1 = 0.2397$, $p_2 = 0.5014$, $p_3 = 0.2589$.

My estimate for the means are:

$$\mu_1 = \begin{bmatrix} 1.0571 & 5.0886 & 2.0447 & 2.9064 & 0.9899 \end{bmatrix},$$

$$\mu_2 = \begin{bmatrix} 0.9925 & 4.9494 & 3.0000 & 4.9870 & 3.0304 \end{bmatrix},$$

$$\mu_3 = \begin{bmatrix} -0.9795 & 5.0040 & 2.0273 & 2.9582 & 3.0587 \end{bmatrix}$$

My estimate for the covariances are:

$$\Sigma_1 = \begin{bmatrix} 0.4340 & 0 & 0 & 0 & 0 \\ 0 & 0.4662 & 0 & 0 & 0 \\ 0 & 0 & 0.4143 & 0 & 0 \\ 0 & 0 & 0 & 0.4238 & 0 \\ 0 & 0 & 0 & 0 & 0.4690 \end{bmatrix},$$

$$\Sigma_2 = \begin{bmatrix} 0.4678 & 0 & 0 & 0 & 0 \\ 0 & 0.5159 & 0 & 0 & 0 \\ 0 & 0 & 0.2524 & 0 & 0 \\ 0 & 0 & 0 & 0.5360 & 0 \\ 0 & 0 & 0 & 0 & 0.4245 \end{bmatrix},$$

$$\Sigma_3 = \begin{bmatrix} 0.4782 & 0 & 0 & 0 & 0 \\ 0 & 0.5377 & 0 & 0 & 0 \\ 0 & 0 & 0.5336 & 0 & 0 \\ 0 & 0 & 0 & 0.5612 & 0 \\ 0 & 0 & 0 & 0 & 0.5520 \end{bmatrix}$$

MATLAB code used in this report

The **main script** for **Problem 1**

```
dbstop if error

close all

clear

clc

data1 = importdata('RedWine_HW7.txt');

redWine = data1.data;

data2 = importdata('WhiteWine_HW7.txt');

whiteWine = data2.data;

d0 = size(redWine,2); % get the original dimension of the data set

%% Preprocessing

redW = (redWine-min(redWine))./(max(redWine)-min(redWine)); % normalize the data

whiteW = (whiteWine-min(whiteWine))./(max(whiteWine)-min(whiteWine)); % normalize the data

% redW = redWine; % not normalize the data

% whiteW = whiteWine; % not normalize the data

N1 = size(redW,1);

N2 = size(whiteW,1);

%% Forward Feature Selection (FFS)

d_list = 1:d0;

d_seqFFS = [];

for ii = 1:d0

        for jj = 1:length(d_list)

        ratio_FFS(ii,jj) = calculateFisherRatio(redW(:,[d_seqFFS, d_list(jj)]), ...

        whiteW(:,[d_seqFFS, d_list(jj)]));

        end

        d_seqFFS(ii) = find(ratio_FFS(ii,:) == max(ratio_FFS(ii,:)));

end

for i = 1:d0
```

```
    temp(i) = ratio_FFS(i,d_seqFFS(i));
end
figure, plot(temp, 'Linewidth', 2), hold on, plot(temp, '.', 'Linewidth', 8)
grid on, xlabel('iteration'), ylabel('Fisher ratio'), title('Performance curve of FFS')
%% Backward Feature Selection (BFS)
d_list = 1:d0;
d_seqBFS = [];
for ii = 1:(d0-1)
    for jj = 1:length(d_list)
        idx_list = d_list;
        idx_list(jj) = [];
        ratio_BFS(ii,d_list(jj)) = calculateFisherRatio(redW(:,idx_list), ...
        whiteW(:,idx_list));
    end
    temp = ratio_BFS(ii,:);
    d_seqBFS(ii) = find(ratio_BFS(ii,:) == max(temp(temp>0)));
    d_list(d_list == d_seqBFS(ii)) = [];
end
for i = 1:(d0-1)
    temp(i) = ratio_BFS(i,d_seqBFS(i));
end
figure, plot(temp, 'Linewidth', 2), hold on, plot(temp, '.', 'Linewidth', 8)
grid on, xlabel('iteration'), ylabel('Fisher ratio'), title('Performance curve of BFS')
```

The function **"calculateFisherRatio"** used in the main script

```
function ratio = calculateFisherRatio(D1, D2 )
%The function to calculate the Fisher Ratio
%   Input:
%       D1: data from class 1
%       D2: data from class 2
```

```
% lambda: parameter for diagonal loading
D = [D1;D2];
d = size(D,2);
mu0 = mean(D,1);
n1 = size(D1,1);
n2 = size(D2,1);
pi1 = n1/(n1+n2);
pi2 = n2/(n1+n2);
mu1 = mean(D1,1);
mu2 = mean(D2,1);
Sw = pi1*(D1-mu1)'*(D1-mu1)/n1 + pi2*(D2-mu2)'*(D2-mu2)/n2;
Sm = (D-mu0)'*(D-mu0)/(n1+n2);
Sb = pi1*(mu1-mu0)'*(mu1-mu0) + pi2*(mu2-mu0)'*(mu2-mu0);
ratio = trace(Sb*pinv(Sw));
end
```

The **main script** for **Problem 2**

```
dbstop if error
close all
clear
clc
% GMM = importdata('GMDataSet_HW7.txt');
% save('GMM.mat', 'GMM')
load('GMM.mat') %GMM
%% PCA
projected_GMM = PCA( GMM, 3);
%% main part: EM-GMM
numCom = 3;
[ mu, sigma, pi, C ] = EM_GMM( GMM, numCom );
for i = 1:numCom
```

```
    figure(101), subplot(1,numCom, i)
    scatter3(projected_GMM(:,1), projected_GMM(:,2), projected_GMM(:,3), 10, C(:,i))
%     title(['Mean: ', num2str(mu(i,:))])
    figure(102), subplot(1,numCom, i)
    scatter(projected_GMM(:,2), projected_GMM(:,3), 10, C(:,i))
%     title(['Mean: ', num2str(mu(i,:))])
end
figure(101), suptitle('3D scatter plot'), saveas(gcf, 'scatter3d.jpg')
figure(102), suptitle('2D scatter plot'), saveas(gcf, 'scatter2d.jpg')
```

The function "PCA" used is the same that I used in homework 2, coded on my own.

The function **EM_GMM** for **Problem 2**

```
function [ mu, sigma, pi, C ] = EMGMM( data, num_com )
%function of expectation maximization for Gaussian Mixture Model
%   Input:
%    data: input data
% num_com: number of components
%  Output:
%      mu: means learned by EM
%   sigma: variances learned by EM (only consider diagonal covariance matrix)
%      pi: probabilities of a data coming from a component

% initialization
maxIter = 600;
thres = 0.0001;
d = size(data,2); % dimension of data
N = size(data,1); % # of data points
for i = 1:num_com
    mu(i,:) = data(max(1,round(N*rand)),:);
    sigma(i,:) = ones(1,d);
```

```
    pi(i) = 1/num_com;
end
% E-step
for k = 1:num_com
    C(:,k) = mvnpdf(data, mu(k,:), sigma(i)*eye(d))*pi(k);
end
C = C./repmat(sum(C,2),[1,num_com]);
% M-step
diff = inf;
numIter = 1;
while (diff>thres && numIter <= maxIter)
    mu0 = mu;
    sigma0 = sigma;
    pi0 = pi;
    temp = sum(sum(C));
    for k = 1:num_com
        temp1 = sum(C(:,k),1); % sum_i(C_ik)
        % update mu
        mu(k,:) = sum(data.*repmat(C(:,k),[1,d]),1)/temp1;
    % update sigma
    temp2 =  (data-repmat(mu(k,:), [N, 1]));
    J = zeros(size(sigma(k,:)));
    for i = 1:size(data,1)
        J = J+C(i,k)*temp2(i,:).^2;
    end
    sigma(k,:) = J/temp1;
        %update pi
        pi(k) = temp1/temp;
    end
```

```
    %update C_ik
    for k = 1:num_com
%           C(:,k) = mvnpdf(data, mu(k,:), sigma(:,:,k))*pi(k);
        C(:,k) = mvnpdf(data, mu(k,:), sigma(k,:)*eye(d))*pi(k);
    end
    C = C./repmat(sum(C,2),[1,num_com]);
    diff = sum(sum(abs(mu0-mu))) + sum(sum(abs(sigma0-sigma))) + sum(abs(pi0-pi));
    numIter = numIter+1;
end
end
```