Tarbalho 2 - Análise de Agrupamentos

Claudson Oliveira - 200822007
Adriele Alvarenga Secundino - 200822002

Outubro de 2012

1 Problema de Classificação - Base de Dados Wine

O problema consiste em classificar tipos de vinho usando 13 atributos. Para este trabalho usamos o método de agrupamento *K-means*

Segue abaixo as informação sobre a base de dado *Wine*
Número de exemplares: 178
Atributos: 13 contínuos

- *1* Alcalinity of ash
- *2* Achool
- *3* Ash
- *4* Color intensity
- *5* OD280/OD315 of diluted wines
- *6* Flavanoids
- *7* Hue
- *8* Magnesium
- *9* Malic acid Alcohol
- *10* Nonflavanoid phenols
- *11* Proline
- *12* Total phenols
- *13* Proanthocyanins

Número de saídas: 3
Codificação de saída:

- Saída 1 = Wine1
- Saída 2 = Wine2
- Saída 3 = Wine3

Tipo de problema: Classificação
Fonte problema: banco de dados da UCI Machine learning ftp://ftp.ics.uci.edu/pub/machine-learning-databases/wine
Discussão problema: ftp://ftp.ics.uci.edu/pub/machine-learning- bases de dados / vinho

2 Listagem das amostras da base dados

Listagem das amostras da base de dados, acompanhadas da respectiva classe real e da classe calculada pelo k-means, somente para a execução que apresentar o maior percentual de acertos.:

1	2	3	4	5	6	7	8	9	10	11	12	13	Classe	CK
12.53	5.51	2.64	25.0	96.0	1.79	0.6	0.63	1.1	5.0	0.82	1.69	515.0	1.0	2.0
13.52	3.17	2.72	23.5	97.0	1.55	0.52	0.5	0.55	4.35	0.89	2.06	520.0	1.0	2.0
13.88	5.04	2.23	20.0	80.0	0.98	0.34	0.4	0.68	4.9	0.58	1.33	415.0	1.0	2.0
13.5	3.12	2.62	24.0	123.0	1.4	1.57	0.22	1.25	8.6	0.59	1.3	500.0	1.0	2.0
12.79	2.67	2.48	22.0	112.0	1.48	1.36	0.24	1.26	10.8	0.48	1.47	480.0	1.0	2.0
13.11	1.9	2.75	25.5	116.0	2.2	1.28	0.26	1.56	7.1	0.61	1.33	425.0	1.0	2.0
13.84	4.12	2.38	19.5	89.0	1.8	0.83	0.48	1.56	9.01	0.57	1.64	480.0	1.0	2.0
12.36	3.83	2.38	21.0	88.0	2.3	0.92	0.5	1.04	7.65	0.56	1.58	520.0	1.0	2.0
13.73	4.36	2.26	22.5	88.0	1.28	0.47	0.52	1.15	6.62	0.78	1.75	520.0	1.0	2.0
12.2	3.03	2.32	19.0	96.0	1.25	0.49	0.4	0.73	5.5	0.66	1.83	510.0	1.0	2.0
12.77	2.39	2.28	19.5	86.0	1.39	0.51	0.48	0.64	9.899999	0.57	1.63	470.0	1.0	2.0
12.7	3.87	2.4	23.0	101.0	2.83	2.55	0.43	1.95	2.57	1.19	3.13	463.0	2.0	2.0
12.0	0.92	2.0	19.0	86.0	2.42	2.26	0.3	1.43	2.5	1.38	3.12	278.0	2.0	2.0
13.05	3.86	2.32	22.5	85.0	1.65	1.59	0.61	1.62	4.8	0.84	2.01	515.0	2.0	2.0
11.84	0.89	2.58	18.0	94.0	2.2	2.21	0.22	2.35	3.05	0.79	3.08	520.0	2.0	2.0
12.67	0.98	2.24	18.0	99.0	2.2	1.94	0.3	1.46	2.62	1.23	3.16	450.0	2.0	2.0
12.16	1.61	2.31	22.8	90.0	1.78	1.69	0.43	1.56	2.45	1.33	2.26	495.0	2.0	2.0
12.08	1.83	2.32	18.5	81.0	1.6	1.5	0.52	1.64	2.4	1.08	2.27	480.0	2.0	2.0
12.0	1.51	2.42	22.0	86.0	1.45	1.25	0.5	1.63	3.6	1.05	2.65	450.0	2.0	2.0
12.69	1.53	2.26	20.7	80.0	1.38	1.46	0.58	1.62	3.05	0.96	2.06	495.0	2.0	2.0
12.29	2.83	2.22	18.0	88.0	2.45	2.25	0.25	1.99	2.15	1.15	3.3	290.0	2.0	2.0
11.62	1.99	2.28	18.0	98.0	3.02	2.26	0.17	1.35	3.25	1.16	2.96	345.0	2.0	2.0
12.29	1.41	1.98	16.0	85.0	2.55	2.5	0.29	1.77	2.9	1.23	2.74	428.0	2.0	2.0
12.29	3.17	2.21	18.0	88.0	2.85	2.99	0.45	2.81	2.3	1.42	2.83	406.0	2.0	2.0
12.34	2.45	2.46	21.0	98.0	2.56	2.11	0.34	1.31	2.8	0.8	3.38	438.0	2.0	2.0
11.82	1.72	1.88	19.5	86.0	2.5	1.64	0.37	1.42	2.06	0.94	2.44	415.0	2.0	2.0
12.42	2.55	2.27	22.0	90.0	1.68	1.84	0.66	1.42	2.7	0.86	3.3	315.0	2.0	2.0
12.25	1.73	2.12	19.0	80.0	1.65	2.03	0.37	1.63	3.4	1.0	3.17	510.0	2.0	2.0
12.72	1.75	2.28	22.5	84.0	1.38	1.76	0.48	1.63	3.3	0.88	2.42	488.0	2.0	2.0
12.22	1.29	1.94	19.0	92.0	2.36	2.04	0.39	2.08	2.7	0.86	3.02	312.0	2.0	2.0
12.52	2.43	2.17	21.0	88.0	2.55	2.27	0.26	1.22	2.0	0.9	2.78	325.0	2.0	2.0
11.41	0.74	2.5	21.0	88.0	2.48	2.01	0.42	1.44	3.08	1.1	2.31	434.0	2.0	2.0
12.08	1.39	2.5	22.5	84.0	2.56	2.29	0.43	1.04	2.9	0.93	3.19	385.0	2.0	2.0
11.03	1.51	2.2	21.5	85.0	2.46	2.17	0.52	2.01	1.9	1.71	2.87	407.0	2.0	2.0
11.82	1.47	1.99	20.8	86.0	1.98	1.6	0.3	1.53	1.95	0.95	3.33	495.0	2.0	2.0
12.42	1.61	2.19	22.5	108.0	2.0	2.09	0.34	1.61	2.06	1.06	2.96	345.0	2.0	2.0
12.77	3.43	1.98	16.0	80.0	1.63	1.25	0.43	0.83	3.4	0.7	2.12	372.0	2.0	2.0
11.56	2.05	3.23	28.5	119.0	3.18	5.08	0.47	1.87	6.0	0.93	3.69	465.0	2.0	2.0
12.42	4.43	2.73	26.5	102.0	2.2	2.13	0.43	1.71	2.08	0.92	3.12	365.0	2.0	2.0
13.05	5.8	2.13	21.5	86.0	2.62	2.65	0.3	2.01	2.6	0.73	3.1	380.0	2.0	2.0
11.87	4.31	2.39	21.0	82.0	2.86	3.03	0.21	2.91	2.8	0.75	3.64	380.0	2.0	2.0
12.07	2.16	2.17	21.0	85.0	2.6	2.65	0.37	1.35	2.76	0.86	3.28	378.0	2.0	2.0
12.43	1.53	2.29	21.5	86.0	2.74	3.15	0.39	1.77	3.94	0.69	2.84	352.0	2.0	2.0
11.79	2.13	2.78	28.5	92.0	2.13	2.24	0.58	1.76	3.0	0.97	2.44	466.0	2.0	2.0

12.37	1.63	2.3	24.5	88.0	2.22	2.45	0.4	1.9	2.12	0.89	2.78	342.0	2.0	2.0
12.86	1.35	2.32	18.0	122.0	1.51	1.25	0.21	0.94	4.1	0.76	1.29	630.0	1.0	1.0
12.7	3.55	2.36	21.5	106.0	1.7	1.2	0.17	0.84	5.0	0.78	1.29	600.0	1.0	1.0
12.51	1.24	2.25	17.5	85.0	2.0	0.58	0.6	1.25	5.45	0.75	1.51	650.0	1.0	1.0
12.6	2.46	2.2	18.5	94.0	1.62	0.66	0.63	0.94	7.1	0.73	1.58	695.0	1.0	1.0
12.25	4.72	2.54	21.0	89.0	1.38	0.47	0.53	0.8	3.85	0.75	1.27	720.0	1.0	1.0
13.49	3.59	2.19	19.5	88.0	1.62	0.48	0.58	0.88	5.7	0.81	1.82	580.0	1.0	1.0
12.84	2.96	2.61	24.0	101.0	2.32	0.6	0.53	0.81	4.92	0.89	2.15	590.0	1.0	1.0
12.93	2.81	2.7	21.0	96.0	1.54	0.5	0.53	0.75	4.6	0.77	2.31	600.0	1.0	1.0
12.87	4.61	2.48	21.5	86.0	1.7	0.65	0.47	0.86	7.65	0.54	1.86	625.0	1.0	1.0
13.32	3.24	2.38	21.5	92.0	1.93	0.76	0.45	1.25	8.42	0.55	1.62	650.0	1.0	1.0
13.23	3.3	2.28	18.5	98.0	1.8	0.83	0.61	1.87	10.52	0.56	1.51	675.0	1.0	1.0
12.58	1.29	2.1	20.0	103.0	1.48	0.58	0.53	1.4	7.6	0.58	1.55	640.0	1.0	1.0
13.17	5.19	2.32	22.0	93.0	1.74	0.63	0.61	1.55	7.9	0.6	1.48	725.0	1.0	1.0
14.34	1.68	2.7	25.0	98.0	2.8	1.31	0.53	2.7	13.0	0.57	1.96	660.0	1.0	1.0
13.48	1.67	2.64	22.5	89.0	2.6	1.1	0.52	2.29	11.75	0.57	1.78	620.0	1.0	1.0
13.69	3.26	2.54	20.0	107.0	1.83	0.56	0.5	0.8	5.88	0.96	1.82	680.0	1.0	1.0
12.96	3.45	2.35	18.5	106.0	1.39	0.7	0.4	0.94	5.28	0.68	1.75	675.0	1.0	1.0
13.78	2.76	2.3	22.0	90.0	1.35	0.68	0.41	1.03	9.58	0.7	1.68	615.0	1.0	1.0
13.45	3.7	2.6	23.0	111.0	1.7	0.92	0.43	1.46	10.68	0.85	1.56	695.0	1.0	1.0
12.82	3.37	2.3	19.5	88.0	1.48	0.66	0.4	0.97	10.26	0.72	1.75	685.0	1.0	1.0
13.4	4.6	2.86	25.0	112.0	1.98	0.96	0.27	1.11	8.5	0.67	1.92	630.0	1.0	1.0
14.16	2.51	2.48	20.0	91.0	1.68	0.7	0.44	1.24	9.7	0.62	1.71	660.0	1.0	1.0
13.71	5.65	2.45	20.5	95.0	1.68	0.61	0.52	1.06	7.7	0.64	1.74	740.0	1.0	1.0
12.72	1.81	2.2	18.8	86.0	2.2	2.53	0.26	1.77	3.9	1.16	3.14	714.0	2.0	1.0
12.08	1.13	2.51	24.0	78.0	2.0	1.58	0.4	1.4	2.2	1.31	2.72	630.0	2.0	1.0
11.65	1.67	2.62	26.0	88.0	1.92	1.61	0.4	1.34	2.6	1.36	3.21	562.0	2.0	1.0
11.64	2.06	2.46	21.6	84.0	1.95	1.69	0.48	1.35	2.8	1.0	2.75	680.0	2.0	1.0
12.08	1.33	2.3	23.6	70.0	2.2	1.59	0.42	1.38	1.74	1.07	3.21	625.0	2.0	1.0
12.37	1.07	2.1	18.5	88.0	3.52	3.75	0.24	1.95	4.5	1.04	2.77	660.0	2.0	1.0
12.08	2.08	1.7	17.5	97.0	2.23	2.17	0.26	1.4	3.3	1.27	2.96	710.0	2.0	1.0
12.51	1.73	1.98	20.5	85.0	2.2	1.92	0.32	1.48	2.94	1.04	3.57	672.0	2.0	1.0
11.61	1.35	2.7	20.0	94.0	2.74	2.92	0.29	2.49	2.65	0.96	3.26	680.0	2.0	1.0
11.76	2.68	2.92	20.0	103.0	1.75	2.03	0.6	1.05	3.8	1.23	2.5	607.0	2.0	1.0
12.0	3.43	2.0	19.0	87.0	2.0	1.64	0.37	1.87	1.28	0.93	3.05	564.0	2.0	1.0
11.45	2.4	2.42	20.0	96.0	2.9	2.79	0.32	1.83	3.25	0.8	3.39	625.0	2.0	1.0
12.04	4.3	2.38	22.0	80.0	2.1	1.75	0.42	1.35	2.6	0.79	2.57	580.0	2.0	1.0
13.24	2.59	2.87	21.0	118.0	2.8	2.69	0.39	1.82	4.32	1.04	2.93	735.0	3.0	1.0
13.24	3.98	2.29	17.5	103.0	2.64	2.63	0.32	1.66	4.36	0.82	3.0	680.0	3.0	1.0
13.24	3.98	2.29	17.5	103.0	2.64	2.63	0.32	1.66	4.36	0.82	3.0	680.0	3.0	1.0
12.81	2.31	2.4	24.0	98.0	1.15	1.09	0.27	0.83	5.7	0.66	1.36	560.0	1.0	1.0
13.36	2.56	2.35	20.0	89.0	1.4	0.5	0.37	0.64	5.6	0.7	2.47	780.0	1.0	1.0
13.62	4.95	2.35	20.0	92.0	2.0	0.8	0.47	1.02	4.4	0.91	2.05	550.0	1.0	1.0
13.08	3.9	2.36	21.5	113.0	1.41	1.39	0.34	1.14	9.4	0.57	1.33	550.0	1.0	1.0
12.85	3.27	2.58	22.0	106.0	1.65	0.6	0.6	0.96	5.58	0.87	2.11	570.0	1.0	1.0
13.58	2.58	2.69	24.5	105.0	1.55	0.84	0.39	1.54	8.66	0.74	1.8	750.0	1.0	1.0
13.4	3.91	2.48	23.0	102.0	1.8	0.75	0.43	1.41	7.3	0.7	1.56	750.0	1.0	1.0
14.13	4.1	2.74	24.5	96.0	2.05	0.76	0.56	1.35	9.2	0.61	1.6	560.0	1.0	1.0
11.81	2.12	2.74	21.5	134.0	1.6	0.99	0.14	1.56	2.5	0.95	2.26	625.0	2.0	1.0

12.6	1.34	1.9	18.5	88.0	1.45	1.36	0.29	1.35	2.45	1.04	2.77	562.0	2.0	1.0
11.46	3.74	1.82	19.5	107.0	3.18	2.58	0.24	3.58	2.9	0.75	2.81	562.0	2.0	1.0
14.06	1.63	2.28	16.0	126.0	3.0	3.17	0.24	2.1	5.65	1.09	3.71	780.0	3.0	1.0
12.93	3.8	2.65	18.6	102.0	2.41	2.41	0.25	1.98	4.5	1.03	3.52	770.0	3.0	1.0
14.22	3.99	2.51	13.2	128.0	3.0	3.04	0.2	2.08	5.1	0.89	3.53	760.0	3.0	1.0
14.22	3.99	2.51	13.2	128.0	3.0	3.04	0.2	2.08	5.1	0.89	3.53	760.0	3.0	1.0
12.88	2.99	2.4	20.0	104.0	1.3	1.22	0.24	0.83	5.4	0.74	1.42	530.0	1.0	1.0
12.25	3.88	2.2	18.5	112.0	1.38	0.78	0.29	1.14	8.21	0.65	2.0	855.0	1.0	3.0
13.16	3.57	2.15	21.0	102.0	1.5	0.55	0.43	1.3	4.0	0.6	1.68	830.0	1.0	3.0
12.45	3.03	2.64	27.0	97.0	1.9	0.58	0.63	1.14	7.5	0.67	1.73	880.0	1.0	3.0
13.27	4.28	2.26	20.0	120.0	1.59	0.69	0.43	1.35	10.2	0.59	1.56	835.0	1.0	3.0
13.17	2.59	2.37	20.0	120.0	1.65	0.68	0.53	1.46	9.3	0.6	1.62	840.0	1.0	3.0
12.47	1.52	2.2	19.0	162.0	2.5	2.27	0.32	3.28	2.6	1.16	2.63	937.0	2.0	3.0
14.23	1.71	2.43	15.6	127.0	2.8	3.06	0.28	2.29	5.64	1.04	3.92	1065.0	3.0	3.0
13.2	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.4	1050.0	3.0	3.0
13.16	2.36	2.67	18.6	101.0	2.8	3.24	0.3	2.81	5.68	1.03	3.17	1185.0	3.0	3.0
14.37	1.95	2.5	16.8	113.0	3.85	3.49	0.24	2.18	7.8	0.86	3.45	1480.0	3.0	3.0
14.2	1.76	2.45	15.2	112.0	3.27	3.39	0.34	1.97	6.75	1.05	2.85	1450.0	3.0	3.0
14.39	1.87	2.45	14.6	96.0	2.5	2.52	0.3	1.98	5.25	1.02	3.58	1290.0	3.0	3.0
14.06	2.15	2.61	17.6	121.0	2.6	2.51	0.31	1.25	5.05	1.06	3.58	1295.0	3.0	3.0
14.83	1.64	2.17	14.0	97.0	2.8	2.98	0.29	1.98	5.2	1.08	2.85	1045.0	3.0	3.0
13.86	1.35	2.27	16.0	98.0	2.98	3.15	0.22	1.85	7.22	1.01	3.55	1045.0	3.0	3.0
14.1	2.16	2.3	18.0	105.0	2.95	3.32	0.22	2.38	5.75	1.25	3.17	1510.0	3.0	3.0
14.12	1.48	2.32	16.8	95.0	2.2	2.43	0.26	1.57	5.0	1.17	2.82	1280.0	3.0	3.0
13.75	1.73	2.41	16.0	89.0	2.6	2.76	0.29	1.81	5.6	1.15	2.9	1320.0	3.0	3.0
14.75	1.73	2.39	11.4	91.0	3.1	3.69	0.43	2.81	5.4	1.25	2.73	1150.0	3.0	3.0
14.38	1.87	2.38	12.0	102.0	3.3	3.64	0.29	2.96	7.5	1.2	3.0	1547.0	3.0	3.0
13.63	1.81	2.7	17.2	112.0	2.85	2.91	0.3	1.46	7.3	1.28	2.88	1310.0	3.0	3.0
14.3	1.92	2.72	20.0	120.0	2.8	3.14	0.33	1.97	6.2	1.07	2.65	1280.0	3.0	3.0
13.83	1.57	2.62	20.0	115.0	2.95	3.4	0.4	1.72	6.6	1.13	2.57	1130.0	3.0	3.0
14.19	1.59	2.48	16.5	108.0	3.3	3.93	0.32	1.86	8.7	1.23	2.82	1680.0	3.0	3.0
13.64	3.1	2.56	15.2	116.0	2.7	3.03	0.17	1.66	5.1	0.96	3.36	845.0	3.0	3.0
13.71	1.86	2.36	16.6	101.0	2.61	2.88	0.27	1.69	3.8	1.11	4.0	1035.0	3.0	3.0
12.85	1.6	2.52	17.8	95.0	2.48	2.37	0.26	1.46	3.93	1.09	3.63	1015.0	3.0	3.0
13.5	1.81	2.61	20.0	96.0	2.53	2.61	0.28	1.66	3.52	1.12	3.82	845.0	3.0	3.0
13.05	2.05	3.22	25.0	124.0	2.63	2.68	0.47	1.92	3.58	1.13	3.2	830.0	3.0	3.0
13.39	1.77	2.62	16.1	93.0	2.85	2.94	0.34	1.45	4.8	0.92	3.22	1195.0	3.0	3.0
13.3	1.72	2.14	17.0	94.0	2.4	2.19	0.27	1.35	3.95	1.02	2.77	1285.0	3.0	3.0
13.87	1.9	2.8	19.4	107.0	2.95	2.97	0.37	1.76	4.5	1.25	3.4	915.0	3.0	3.0
14.02	1.68	2.21	16.0	96.0	2.65	2.33	0.26	1.98	4.7	1.04	3.59	1035.0	3.0	3.0
13.73	1.5	2.7	22.5	101.0	3.0	3.25	0.29	2.38	5.7	1.19	2.71	1285.0	3.0	3.0
13.58	1.66	2.36	19.1	106.0	2.86	3.19	0.22	1.95	6.9	1.09	2.88	1515.0	3.0	3.0
13.68	1.83	2.36	17.2	104.0	2.42	2.69	0.42	1.97	3.84	1.23	2.87	990.0	3.0	3.0
13.76	1.53	2.7	19.5	132.0	2.95	2.74	0.5	1.35	5.4	1.25	3.0	1235.0	3.0	3.0
13.51	1.8	2.65	19.0	110.0	2.35	2.53	0.29	1.54	4.2	1.1	2.87	1095.0	3.0	3.0
13.48	1.81	2.41	20.5	100.0	2.7	2.98	0.26	1.86	5.1	1.04	3.47	920.0	3.0	3.0
13.28	1.64	2.84	15.5	110.0	2.6	2.68	0.34	1.36	4.6	1.09	2.78	880.0	3.0	3.0
13.05	1.65	2.55	18.0	98.0	2.45	2.43	0.29	1.44	4.25	1.12	2.51	1105.0	3.0	3.0
13.07	1.5	2.1	15.5	98.0	2.4	2.64	0.28	1.37	3.7	1.18	2.69	1020.0	3.0	3.0

13.56	1.71	2.31	16.2	117.0	3.15	3.29	0.34	2.34	6.13	0.95	3.38	795.0	3.0	3.0
13.41	3.84	2.12	18.8	90.0	2.45	2.68	0.27	1.48	4.28	0.91	3.0	1035.0	3.0	3.0
13.88	1.89	2.59	15.0	101.0	3.25	3.56	0.17	1.7	5.43	0.88	3.56	1095.0	3.0	3.0
13.05	1.77	2.1	17.0	107.0	3.0	3.0	0.28	2.03	5.04	0.88	3.35	885.0	3.0	3.0
14.21	4.04	2.44	18.9	111.0	2.85	2.65	0.3	1.25	5.24	0.87	3.33	1080.0	3.0	3.0
14.38	3.59	2.28	16.0	102.0	3.25	3.17	0.27	2.19	4.9	1.04	3.44	1065.0	3.0	3.0
13.9	1.68	2.12	16.0	101.0	3.1	3.39	0.21	2.14	6.1	0.91	3.33	985.0	3.0	3.0
14.1	2.02	2.4	18.8	103.0	2.75	2.92	0.32	2.38	6.2	1.07	2.75	1060.0	3.0	3.0
13.94	1.73	2.27	17.4	108.0	2.88	3.54	0.32	2.08	8.9	1.12	3.1	1260.0	3.0	3.0
13.05	1.73	2.04	12.4	92.0	2.72	3.27	0.17	2.91	7.2	1.12	2.91	1150.0	3.0	3.0
13.83	1.65	2.6	17.2	94.0	2.45	2.99	0.22	2.29	5.6	1.24	3.37	1265.0	3.0	3.0
13.82	1.75	2.42	14.0	111.0	3.88	3.74	0.32	1.87	7.05	1.01	3.26	1190.0	3.0	3.0
13.77	1.9	2.68	17.1	115.0	3.0	2.79	0.39	1.68	6.3	1.13	2.93	1375.0	3.0	3.0
13.74	1.67	2.25	16.4	118.0	2.6	2.9	0.21	1.62	5.85	0.92	3.2	1060.0	3.0	3.0
13.56	1.73	2.46	20.5	116.0	2.96	2.78	0.2	2.45	6.25	0.98	3.03	1120.0	3.0	3.0
14.22	1.7	2.3	16.3	118.0	3.2	3.0	0.26	2.03	6.38	0.94	3.31	970.0	3.0	3.0
13.29	1.97	2.68	16.8	102.0	3.0	3.23	0.31	1.66	6.0	1.07	2.84	1270.0	3.0	3.0
13.72	1.43	2.5	16.7	108.0	3.4	3.67	0.19	2.04	6.8	0.89	2.87	1285.0	3.0	3.0
13.56	1.71	2.31	16.2	117.0	3.15	3.29	0.34	2.34	6.13	0.95	3.38	795.0	3.0	3.0
13.41	3.84	2.12	18.8	90.0	2.45	2.68	0.27	1.48	4.28	0.91	3.0	1035.0	3.0	3.0
13.88	1.89	2.59	15.0	101.0	3.25	3.56	0.17	1.7	5.43	0.88	3.56	1095.0	3.0	3.0
13.05	1.77	2.1	17.0	107.0	3.0	3.0	0.28	2.03	5.04	0.88	3.35	885.0	3.0	3.0
14.21	4.04	2.44	18.9	111.0	2.85	2.65	0.3	1.25	5.24	0.87	3.33	1080.0	3.0	3.0
14.38	3.59	2.28	16.0	102.0	3.25	3.17	0.27	2.19	4.9	1.04	3.44	1065.0	3.0	3.0
13.9	1.68	2.12	16.0	101.0	3.1	3.39	0.21	2.14	6.1	0.91	3.33	985.0	3.0	3.0
14.1	2.02	2.4	18.8	103.0	2.75	2.92	0.32	2.38	6.2	1.07	2.75	1060.0	3.0	3.0
13.94	1.73	2.27	17.4	108.0	2.88	3.54	0.32	2.08	8.9	1.12	3.1	1260.0	3.0	3.0
13.05	1.73	2.04	12.4	92.0	2.72	3.27	0.17	2.91	7.2	1.12	2.91	1150.0	3.0	3.0
13.83	1.65	2.6	17.2	94.0	2.45	2.99	0.22	2.29	5.6	1.24	3.37	1265.0	3.0	3.0
13.82	1.75	2.42	14.0	111.0	3.88	3.74	0.32	1.87	7.05	1.01	3.26	1190.0	3.0	3.0
13.77	1.9	2.68	17.1	115.0	3.0	2.79	0.39	1.68	6.3	1.13	2.93	1375.0	3.0	3.0
13.74	1.67	2.25	16.4	118.0	2.6	2.9	0.21	1.62	5.85	0.92	3.2	1060.0	3.0	3.0
13.56	1.73	2.46	20.5	116.0	2.96	2.78	0.2	2.45	6.25	0.98	3.03	1120.0	3.0	3.0
14.22	1.7	2.3	16.3	118.0	3.2	3.0	0.26	2.03	6.38	0.94	3.31	970.0	3.0	3.0
13.29	1.97	2.68	16.8	102.0	3.0	3.23	0.31	1.66	6.0	1.07	2.84	1270.0	3.0	3.0
13.72	1.43	2.5	16.7	108.0	3.4	3.67	0.19	2.04	6.8	0.89	2.87	1285.0	3.0	3.0

3 Análise das Execuções

O algoritmo foi executado 3 vezes, em um computador Core 2 duo, 3.2 GHz, 3GB de memória principal.
A tabela abaixo mostras os resultados:

-	Tamanho	Execução 1	Execução 2	Execução 3	Weka
Grupo 1	48	48	57	56	48
Grupo 2	51	60	59	60	51
Grupo 3	79	70	62	62	79
Tempo	-	0.9	0.5	0.4	0.03

Tabela 2: Resultados da Análise RS

4 Conclusão

Analisando os resultados percebe-se que o método *K-means* desenvolvido tem um desempenho mediano comparado ao Weka, porém o metodo se mostrou adequado para resolver problemas de classificação, resolvendo rapidamente o problema com poucas iterações.

5 Algoritmo

```
package br.cloud.dataming.kmeansjava;

/**
 *
 * @author Adriele Alvarenga
 * @author Claudson Oliveira
 */
public class KmeansJava {

    static final int NUMERO_INSTANCIAS = 178;
    static final int K = 3;
    static final int LOOP_MAX = 100;



    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        Leitor leitor = new Leitor("Wine.txt");
        Wine[] instancias = new Wine[NUMERO_INSTANCIAS];
        String linha = null;
        int i = 0;
        try{
            i =0;
            while ((linha = leitor.getLinha()) != null) {
                instancias[i++] = WineFactory.criarWine(linha);
            }
        }catch(Exception e){
            System.out.println("erro!");
        }

        // pegando k centros aleatórios
        i = 0;
        Wine[] centros = new Wine[K];
        while(i < K){
            int indice = (int) (Math.random() * NUMERO_INSTANCIAS - 1) ;
            centros[i++] = instancias[indice];
        }
        GrupoManager grupos = new GrupoManager(K, centros);
        i = 0;
```

```
        while(i < LOOP_MAX) {
            int j = 0;
            while(j < NUMERO_INSTANCIAS) {
                Wine instanciaAtual = instancias[j];
                int idgrup = grupos.seleciona(instanciaAtual);
                grupos.addInstancia(instanciaAtual,idgrup);
                j++;
            }

            for(j=0; j < grupos.k; j++) {
                Grupo grupo = grupos.grupos[j];
                System.out.println("grupo: ("+i+","+j+")"+grupo+", centro:"+grupo.centro);
                Wine pseudo = grupos.geraPseudoInstanciaMedia(grupo);
                centros[j] = grupos.instanciaMaisProxima(pseudo, grupo);

            }
            System.out.println("\n");
            for(j=0; j<grupos.k; j++){
                grupos.grupos[j].setCentro(centros[j]);
            }
            i++;

            if(!grupos.temMudancas()){
               break;
            }
            grupos.zerarMudancas();
        }
        System.out.println("Iterou por "+i+" vezes");
        grupos.estatisticas(instancias);
        for(int j=0; j<grupos.k; j++){
            Grupo grupo = grupos.grupos[j];
            System.out.println("grupo ordenado("+j+"): "+grupo+", centro:"+grupo.centro);
        }
    }

}
```

```
package br.cloud.dataming.kmeansjava;

/**
 * Arquivo que define uma instancia Wine, uma
 * linha no arquivo base
 * @author Claudson Oliveira
 */
public class Wine
{
    public double alcohol;
    public double malic_acid;
    public double ash;
    public double alcalinity_ash;
    public double magnesium;
    public double total_phenols;
    public double flavanoids;
    public double nonflavanoid_phenols;
    public double proanthocyanins;
```

```
public double color_intensity;
public double hue;
public double diluted_wines;
public double proline;

public int real_class;
public int found_class;

public double distance_euclidian(Wine center)
{
    double distance = 0;

    distance = Math.pow(center.alcalinity_ash - this.alcalinity_ash,2);
    distance += Math.pow(center.alcohol - this.alcohol,2);
    distance += Math.pow(center.ash - this.ash,2);
    distance += Math.pow(center.color_intensity - this.color_intensity,2);
    distance += Math.pow(center.diluted_wines - this.diluted_wines,2);
    distance += Math.pow(center.flavanoids - this.flavanoids,2);
    distance += Math.pow(center.hue - this.hue,2);
    distance += Math.pow(center.magnesium - this.magnesium,2);
    distance += Math.pow(center.malic_acid - this.malic_acid,2);
    distance += Math.pow(center.nonflavanoid_phenols - this.nonflavanoid_phenols,2);
    distance += Math.pow(center.proline - this.proline,2);
    distance += Math.pow(center.total_phenols - this.total_phenols,2);
    distance += Math.pow(center.proanthocyanins - this.proanthocyanins,2);

    return Math.sqrt(distance);
}

public void set(int i,double valor)
{

    switch(i){
        case 0:
            this.alcalinity_ash = valor;
            break;

        case 1:
            this.alcohol = valor;
            break;

        case 2:
            this.ash = valor;
            break;

        case 3:
            this.color_intensity = valor;
            break;

        case 4:
            this.diluted_wines = valor;
            break;

        case 5:
            this.flavanoids = valor;
            break;
```

```
        case 6:
            this.hue = valor;
            break;

        case 7:
            this.magnesium = valor;
            break;

        case 8:
            this.malic_acid = valor;
            break;

        case 9:
            this.nonflavanoid_phenols = valor;
            break;

        case 10:
            this.proline = valor;
            break;

        case 11:
            this.total_phenols = valor;
            break;

        case 12:
            this.proanthocyanins = valor;
            break;

        case 13:
            this.real_class = (int)valor;
            break;

    }
}

public double get(int i)
{

    switch(i){
        case 0:
            return this.alcalinity_ash;

        case 1:
            return this.alcohol;

        case 2:
            return this.ash;

        case 3:
            return this.color_intensity;

        case 4:
            return this.diluted_wines;

        case 5:
```

```
                return this.flavanoids;

            case 6:
                return this.hue;

            case 7:
                return this.magnesium;

            case 8:
                return this.malic_acid;

            case 9:
                return this.nonflavanoid_phenols;

            case 10:
                return this.proline;

            case 11:
                return this.total_phenols;

            case 12:
                return this.proanthocyanins;

            case 13:
                return this.real_class;

            case 14:
                return this.found_class;


        }
        return 0;
    }

    @Override
    public String toString() {
        String texto = "[";
        for(int i=0; i<= 14; i++){
            texto += this.get(i)+", ";
        }
        texto += "]";
        return texto;
    }

}
```

```
package br.cloud.dataming.kmeansjava;

import java.util.ArrayList;

/**
 * Representa um grupo de wines
 * @author Adriele Alvarenga
 */
```

```
public class Grupo
{
    protected Wine centro;
    protected ArrayList<Wine> instancias;

    public Grupo() {
        this.instancias = new ArrayList<Wine>();
    }


    public void add(Wine instancia){
        this.instancias.add(instancia);
    }

    public boolean contains(Wine instancia){
        return this.instancias.contains(instancia);
    }

    public void remove(Wine instancia){
        this.instancias.remove(instancia);
    }

    @Override
    public String toString() {
        String texto = " Grupo ("+this.instancias.size()+") ";
        return texto;
    }

    public Wine getCentro() {
        return centro;
    }

    public void setCentro(Wine centro) {
        this.centro = centro;
    }

    public int size(){
        return this.instancias.size();
    }

    public Wine get(int i){
        return this.instancias.get(i);
    }

}
```

```
package br.cloud.dataming.kmeansjava;

import java.util.ListIterator;

/**
 * Gerenciador de Grupos, encapsula particularidades
 * que nao cabiam ao grupo e nem a Wine
 * @author Claudson Oliveira
```

```
 */
public class GrupoManager {

    protected Grupo[] grupos;
    protected int k;
    protected int mudancas = 0;


    public GrupoManager(int k, Wine[] centros){
        this.grupos = new Grupo[k];
        for(int i = 0; i < k; i++){
            this.grupos[i] = new Grupo();
            this.grupos[i].setCentro(centros[i]);
        }
        this.k = k;

    }

    public void addInstancia(Wine instancia, int id){
        if(this.k <= id){
            throw new IllegalArgumentException("Grupo inexistente");
        }
        instancia.found_class = id + 1;

        if(!this.grupos[id].contains(instancia)){
            this.grupos[id].add(instancia);

            for(int i=0; i < this.k; i++){
                if(i != id && this.grupos[i].contains(instancia)){
                    this.grupos[i].remove(instancia);
                    break;
                }
            }
            this.mudancas++;
        }

    }

    public void rmInstancia(Wine instancia){
        for(int i=0; i < this.k; i++){
            if(this.grupos[i].contains(instancia)){
                this.grupos[i].remove(instancia);
                break;
            }
        }
    }

    public int seleciona(Wine wine)
    {
        int i = 0, centro = 0;
        double distancia = 0,menorDistancia = Double.MAX_VALUE;
        while(i < this.k){
            distancia = wine.distance_euclidian(this.grupos[i].centro);
            if(distancia < menorDistancia){
                menorDistancia = distancia;
                centro = i;
```

```
        }
        i++;
    }

    return centro;
}

public Wine instanciaMaisProxima(Wine pivo, Grupo grupo)
{
    Wine instancia = null, maisProxima = null;
    int i = 0;
    double distancia = 0,menorDistancia = Double.MAX_VALUE;
    ListIterator<Wine> list = grupo.instancias.listIterator();

    while(list.hasNext()){
        instancia = list.next();
        distancia = pivo.distance_euclidian(instancia);
        if(distancia < menorDistancia){
            menorDistancia = distancia;
            maisProxima = instancia;
        }
        i++;
    }

    return maisProxima;
}


@Override
public String toString() {
    String texto = "";
    for (int i = 0; i < this.grupos.length; i++) {
        texto += this.grupos[i];
    }

    return texto;
}

public boolean temMudancas(){
    return (this.mudancas > 0);
}

public void zerarMudancas(){
    this.mudancas = 0;
}

public Wine geraPseudoInstanciaMedia(Grupo grupo) {
    Wine instancia = new Wine();
    int tamanhoGrupo = grupo.size();
    for(int i= 0; i< tamanhoGrupo; i++) {
        Wine atual = grupo.get(i);
        for(int a=0; a< 13; a++){
           double valor = atual.get(a);
           instancia.set(a, instancia.get(a) + valor);
        }
    }
```

```
    for(int i= 0; i< 13; i++) {
        instancia.set(i, instancia.get(i)/grupo.size());
    }
    return instancia;
}

public void estatisticas(Wine[] instancias){
    this.ordenaGrupos();
    int i=0;
    int[] contadores_classes = new int[this.k];
    while(i < this.k) {
        contadores_classes[i] =0;
        i++;
    }
    i=0;
    while(i < KmeansJava.NUMERO_INSTANCIAS) {
        Wine atual = instancias[i];
        contadores_classes[atual.real_class - 1]++;
        i++;
    }

    i=0;
    // imprimo o numero de instancias que o grupo X esperava
    while(i < this.k) {
        System.out.println("Grupo "+(i+1)+" esperava "+contadores_classes[i++]);
    }
    i=0;
    while(i < this.k) {
        contadores_classes[i] =0;
        i++;
    }
    System.out.println("");
    i = 0;
    int numAcertos = 0;
    while(i < this.k) {
        int j = 0;
        while(j < this.grupos[i].size()) {
            Wine instancia = this.grupos[i].get(j);
            System.out.println(instancia);
            if(instancia.real_class == instancia.found_class){
               // System.out.println(instancia.real_class+","+instancia.found_class);
                numAcertos++;
            }
            j++;
        }
        i++;
    }

    float percentagem = (100*numAcertos)/KmeansJava.NUMERO_INSTANCIAS;
    System.out.println("Acerto :"+(percentagem)+" % , Falha: "+(100 - percentagem)+" %");
}

protected void ordenaGrupos(){
    for(int i=0; i<this.k;i++ ){
        Grupo fixo = this.grupos[i];
```

```
            for(int j=0; j<this.k; j++){
                Grupo atual = this.grupos[j];
                if(atual.size() > fixo.size()){
                    Grupo aux = atual;
                    atual = fixo;
                    fixo = aux;

                    this.grupos[i] = fixo;
                    this.grupos[j] = atual;
                }
            }
        }
    }
}
```

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package br.cloud.dataming.kmeansjava;

/**
 *
 * @author cloud
 */
public class WineFactory
{
    public static Wine criarWine(String linha)
    {
        Wine wine = new Wine();
        String[] atributos = linha.split("\t");
        for(int i = 0; i < atributos.length; i++){
            double valor = Double.parseDouble(atributos[i]);
            wine.set(i,valor);
        }

        return wine;
    }


}
```

```
package br.cloud.dataming.kmeansjava;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

/**
 *
 * @author Claudson Oliveira
 * @author Adriele Alvarenga
 */
```

```
public class Leitor
{
    protected FileReader reader;
    protected BufferedReader buffer;

    public Leitor(String baseDeDados) {
        try{
            this.reader = new FileReader(baseDeDados);
            this.buffer = new BufferedReader(reader);
        }catch(Exception e){

        }
    }

    public String getLinha() throws IOException
    {
        String linha = null;
        linha = this.buffer.readLine();
        return linha;
    }

    public void fechar()throws IOException
    {
        this.buffer.close();
        this.reader.close();
    }

}
```