Universidade do Minho
Escola de Engenharia
Mestrado Integrado em Engenharia Informática

Unidade Curricular de Aprendizagem e Extração de Conhecimento

Ano Letivo de 2017/2018

Relatório do 2º Trabalho Prático

Bruno Pereira (a75135), Maria Ana de Brito (a73580)

Dezembro, 2017

AEC

Índice

1. Introdução	1
2. Descrição do Conjunto de Dados	2
2.1. Dataset da Qualidade do Vinho	2
2.2. Dataset da Qualidade do Ar	3
3. Classificação	7
3.1. NaïveBayes	7
3.2. J48 e SimpleCART	10
3.2.1 J48	10
3.3. AdaBoost e Bagging	17
3.3.1 AdaBoost	17
3.3.2 Bagging	18
3.4. SimpleCART	19
3.5. Resumo dos resultados de árvores de decisão	20
3.6. Recomendações para a classificação	21
4. Segmentação	22
4.1. Algoritmo k-means	22
4.2. Algoritmo EM	27
4.3. Algoritmo Hierarchical Clustering	28
4.4. Recomendações para a Segmentação	31
5. Regras de Associação	33
5.1. Algoritmo Apriori	33
5.1.1 Discretização com 10 intervalos	33
5.1.2 Discretização com 5 intervalos	37
5.2. Resultados sobre as Regras de Associação	38
5.3. Recomendações sobre as Regras de Associação	39
6. Conclusões	40

Índice de Figuras

Figura 1 - Exemplo de um intervalo mais frequente 5
Figura 2 - Vista global da frequência dos intervalos 5
Figura 3 - Sumário do atributo NMHC(GT) 5
Figura 4 - Exemplo de um classificador de *Bayes* 7
Figura 5 – Resultados com validação cruzada e dados discretizados 8
Figura 6 - Gráfico da curva ROC para a classe 7, com validação cruzada 9
Figura 7 - Erro de x-fold sem pruning 11
Figura 8 - Estatísticas de x-fold sem pruning 11
Figura 9 - Árvore de decisão de x-fold sem pruning 11
Figura 10 - Erro de *pruning* 12
Figura 11 - Estatísticas de *pruning* 12
Figura 12 - Erro de *pruning* e *subtreeRaising* 13
Figura 13 - Estatísticas de *pruning* e *subtreeRaising* 13
Figura 14 - Erro de *pruning* com *subtreeRaising* e *Laplace* 14
Figura 15 - Estatísticas de *pruning* com *subtreeRaising* e *Laplace* 14
Figura 16 - Erro de *training set* sem *pruning* 15
Figura 17 - Estatísticas de *training set* sem *pruning* 15
Figura 18 - Erro de *percentage split* sem *pruning* 16
Figura 19 - Estatísticas de *percentage split* sem *pruning* 16
Figura 20 - Erro de AdaBoost 17
Figura 21 - Estatísticas de AdaBoost 17
Figura 22 - Erro de Bagging 18
Figura 23 - Estatísticas de Bagging 18
Figura 24 - Erro de SimpleCART com *pruning* 19
Figura 25 - Estatísticas de SimpleCART com *pruning* 19
Figura 26 - Erro de SimpleCART sem *pruning* 19
Figura 27 - Estatísticas de SimpleCART sem *pruning* 20
Figura 28 - Exemplo do valor da variância do k-means 23
Figura 29 - Representação gráfica de *clusters* com o número da instância e pH 24
Figura 30 - Representação gráfica de *clusters* com álcool e *cluster* 25
Figura 31 - Resultado do teste k-means com "Classes to clusters evaluation" 26

Figura 32 - Número de instâncias mal segmentadas para o algoritmo EM 28
Figura 33 - Exemplo de um dendrograma com 4 *clusters* 28
Figura 34 - Exemplo de um dendrograma gerado através do algoritmo HC 29
Figura 35 - Resultado gráfico da segmentação utilizando HC 31
Figura 36 - Exemplo de regra redundante 34
Figura 37 - Exemplo de regras com *lift* máximo 35
Figura 38 - Exemplo de regra tendo em conta a convicção 36

Índice de Tabelas

Tabela 1 - Resultados sobre os testes de *NaïveBayes* 8
Tabela 2 - Resultados dos valores de AUC para diferentes classes e modos 9
Tabela 3 - Resultados dos vários testes de árvores de decisão 20
Tabela 4 - Resultados de teste do k-means 22
Tabela 5 - Resultados k-means para diferentes funções de distância 23
Tabela 6 - Resultados com diferentes *link types*, número de *clusters* e funções distância 29
Tabela 7 - Número de regras geradas tendo em conta a confiança (10 *bins*) 33
Tabela 8 - Número de regras geradas tendo em conta o *lift* (10 *bins*) 35
Tabela 9 - Número de regras tendo em conta a convicção (10 *bins*) 36
Tabela 10 - Número de regras geradas tendo em conta a *leverage* (10 *bins*) 36
Tabela 11 - Número de regras tendo em conta a confiança (5 *bins*) 37
Tabela 12 - Número de regras tendo em conta o *lift* (5 *bins*) 37
Tabela 13 - Número de regras tendo em conta a *conviction* (5 *bins*) 38
Tabela 14 - Número de regras geradas tendo em conta a *leverage* (5 *bins*) 38

1. Introdução

O trabalho prático tem por base extrair conhecimento de um conjunto de *datasets*, através da exploração da ferramenta WEKA e das várias funcionalidades e diversos algoritmos que disponibiliza. Foram usados dois tipos diferentes de *datasets* neste projeto: o primeiro lida com dados sobre a concentração de diferentes gases que contribuem para a poluição atmosférica de uma certa cidade da Itália, o segundo contém dados sobre diversos fatores que afetam a qualidade de um vinho.

Desta maneira, numa primeira fase foi realizada uma análise dos conjuntos de dados selecionados, visando uma melhor compreensão do seu contexto e atributos. De seguida, foram identificadas as operações de pré-processamento que teriam de ser realizadas para que os dados fossem sujeitos a treinos e testes de *performance*. Por fim, dentro das três áreas de extração de conhecimento foram escolhidos quais os algoritmos que seriam utilizados. Com os resultados obtidos, foi feita a interpretação dos mesmos e tecidas as conclusões e recomendações necessárias.

Área de Aplicação: Extração de Conhecimento, WEKA, Classificação, Segmentação, Regras de Associação

Palavras-Chave: *Dataset*, dados, algoritmo, percentagem de erro, qualidade do ar, qualidade do vinho

2. Descrição do Conjunto de Dados

2.1. Dataset da Qualidade do Vinho

O *dataset* trabalhado para a segmentação foi um *dataset* referente à qualidade de vinho branco, em que os atributos são características do vinho e a classe representa a classificação do vinho numa escala de [0,10] (apesar do *dataset* só conter valores no intervalo [3,6]). Os atributos tratados podem ser caracterizados da seguinte forma:

- ***fixed acidity*:** ou ácidos não voláteis. Estes ácidos dão o sabor ao vinho;
- ***volatile acidity*:** este atributo refere-se aos gases destilados dos ácidos presentes no vinho, ou seja, mede se o vinho está estragado ou não, possuindo valores de ácidos considerados "estragados", tais como o ácido acético (também conhecido como vinagre);
- ***citric acid*:** um tipo de *fixed acid;*
- ***residual sugar*:** este atributo representa o açúcar que não foi fermentado. Este valor irá depender das uvas;
- ***chlorides:*** este atributo representa os cloretos, que tornam o vinho salgado
- ***free sulfur dioxide*:** dióxido de enxofre existente em equilíbrio com outro ião. Previne o crescimento e oxidação do vinho, podendo a sua concentração afetar o odor e sabor do vinho;
- ***total sulfur dioxide*:** dióxido de enxofre não ligado, ou seja, não tem outro ião.
- ***density*:** densidade da água no vinho;
- ***pH*:** descreve o valor na escala de pH se um vinho é ácido, neutro ou básico. A maioria dos vinhos pertence a um intervalo [3, 4] na escala de pH;
- ***sulfates*:** um aditivo do vinho que contribui para os níveis de gás de dióxido de enxofre, e que atua como antioxidante;
- ***alcohol*:** percentagem de álcool no vinho;

O pré-processamento aplicado ao ficheiro *wine-quality.csv* para a utilização de segmentação seguiu variados passos, mas só atingiu resultados desejáveis na normalização dos dados.

Uma das alterações experimentada foi a discretização de dados. Considerando, por exemplo, o erro obtido através do algoritmo *k-means,* este valor era bastante superior com a discretização dos dados. No entanto, mesmo que este valor fosse inferior, não era confiável, pois a segmentação não lida corretamente com valores discretos.

A segunda alteração proposta consistiu em normalizar os dados. Considerando que os dados apresentam atributos com escalas bastante diferentes, tais como as taxas de dióxido de enxofre, relativamente às anteriores. Assim, surgiu a preocupação de que os algoritmos de

génese de *clusters* iriam colocar mais peso em atributos com maior variância. Assim, para evitar *clusters* elípticos, os dados foram normalizados de duas maneiras diferentes:

- Normalizar todos os atributos;
- Normalizar somente os atributos relativos à taxa de dióxido de enxofre, visto que estes apresentam a maior variância entre valores. A normalização efetuada consistiu em multiplicar todos os valores dos atributos relativos ao dióxido de enxofre por 1000 para o atributo apresentar as mesmas unidades SI (mg para g).

Aquilo que foi verificado na primeira normalização foi que o erro era inferior nos dados originais. No entanto, na segunda normalização, como os atributos já apresentavam as mesmas unidades, o valor do erro é bastante inferior ao valor do erro apresentado nos dados originais. Apesar dos algoritmos de criação de *clusters* no WEKA implementarem uma normalização aos dados que tratam, eles não sabem em que escalas é que os atributos se encontram, logo, é necessário efetuar uma normalização prévia à utilização destes algoritmos.

A terceira alteração foi a mudança da ordem das instâncias. Devido ao facto da segmentação depender da ordem com que as instâncias surgem, achou-se interessante alterar a ordem das instâncias. Esta alteração verificou-se praticamente sem impacto, pois a diferença do erro com valores aleatórios e os iniciais não superava a unidade.

Finalmente, a última alteração passou por transformar o atributo classe *quality* num atributo nominal (previamente numérico), de modo a que possa ser possível utilizar o modo dos *clusters* do WEKA "*Classes to clusters evaluation*", que só aceita valores nominais para a classe.

2.2. Dataset da Qualidade do Ar

O *dataset* atribuído ao grupo, *Air Quality*, lida com contagens de concentração de diferentes gases retirados a partir de vários sensores localizados numa cidade italiana. Estes dados foram recolhidos durante um ano, de hora em hora, logo contamos com 9358 registos que podem conter valores nulos/inexistentes.

Deste modo, ao todo existem 15 atributos diferentes:

- ***Date***: data da recolha dos dados, sob a forma de dd/mm/yyyy. Não contém valores nulos.
- ***Time***: hora da recolha dos dados, sob a forma de hh:mm:ss. Não contém valores nulos.
- **CO(GT)**: Monóxido de carbono. Contém 1592 (17%) valores nulos.
- **PT08.S1(CO)**: Reação do sensor de óxido de estanho, direcionado ao monóxido de carbono. Contém 366 (4%) valores nulos.
- **NMHC(GT)**: Hidrocarbonetos não-metano. Contém 8443 (90%) valores nulos.
- **C6H6(GT)**: Benzeno. Não contém valores nulos.
- **PT08.S2(NMHC)**: Reação do sensor de titânio, direcionado aos hidrocarbonetos não-metano. Contém 366 (4%) valores nulos.

- **NOx(GT)**: Óxidos de azoto. Contém 1639 (18%) valores nulos.
- **PT08.S3(NOx)**: Reação do sensor de tungsténio, direcionado aos óxidos de azoto. Contém 366 (4%) valores nulos.
- **NO2(GT)**: Dióxido de azoto. Contém 1642 (18%) valores nulos.
- **PT08.S4(NO2)**: Reação do sensor de tungsténio, direcionado ao dióxido de azoto. Contém 366 (4%) valores nulos.
- **PT08.S5(O3)**: Reação do sensor de índio, direcionado ao ozono. Contém 366 (4%) valores nulos.
- **T**: Temperatura em ºC. Contém 366 (4%) valores nulos.
- **RH**: Humidade relativa em %. Contém 366 (4%) valores nulos.
- **AH**: Humidade absoluta. Contém 366 (4%) valores nulos.

Este *dataset* foi utilizado para a geração de regras de associação, logo os atributos data e hora não foram considerados relevantes, pois possuem uma menor importância quando comparados com os restantes atributos. Assim, foi realizada uma discretização deste *dataset*, uma vez que existe uma grande variedade de valores. Se não tivéssemos feito esta discretização, seriam geradas um grande número de regras e seriam muito específicas para cada valor existente. Assim, dividindo estes valores em intervalos, obtém-se regras mais abrangentes e um menor número de regras redundantes. Por exemplo, sem discretização (ainda se teve de transformar os atributos numéricos em nominais, uma vez que o algoritmo Apriori o exige – recorreu-se ao filtro de pré-processamento *NumericToNominal*) poderíamos obter uma regra com a seguinte forma:

NOx(GT)=89 ==> NO2(GT)=97

Com a discretização dos valores, obtemos, então, a seguinte regra:

NOx(GT)='(88.5-89.5]' ==> NO2(GT)='(96.5-97.5]'

Como se pode verificar, a segunda regra considera não um valor absoluto, mas sim um intervalo de valores. Tendo em conta a dispersão de valores dos atributos do *dataset*, considerou-se, então, mais útil para retirar conclusões discretizar os valores, considerando-os como intervalos. Assim, para uma regra de associação, não se considera que se o valor de NOx for 89, tem-se que a concentração de NO2 é de 97, mas sim se o valor de NOx for maior do que 88.5, mas menor do que 89.5, então temos a implicação que o valor de NO2 está entre 96.5 e 97.5. Como estamos a lidar com concentrações de gases no ar para aferir a sua qualidade, é mais realista considerarmos intervalos de valores, em vez de apenas um valor único.

Decidiu-se, então, fazer a discretização tendo em conta que se pretendia intervalos sempre com a mesma frequência, se possível. Esta decisão foi tomada, pois tentou-se discretizar mantendo a largura dos intervalos igual, contudo isso levou à existência de intervalos muito frequentes, enquanto que outros eram muito raros, como se pode ver nas duas figuras abaixo. Não se verificava um equilíbrio entre a distribuição dos intervalos, o que não levava, decerto, a bons resultados.

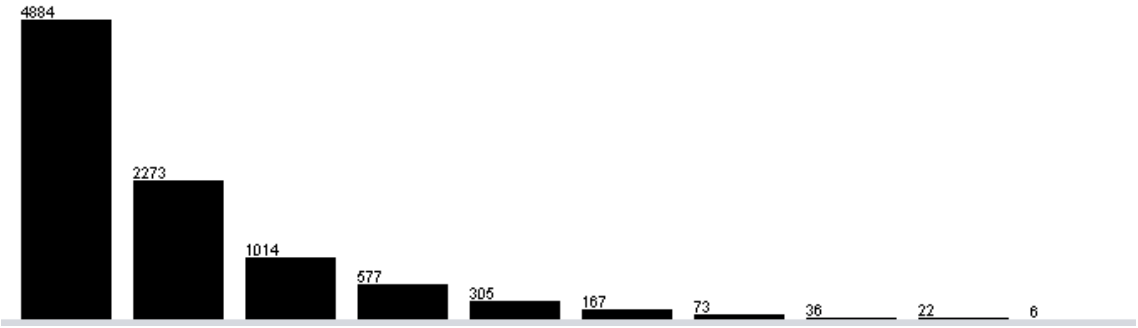

Figura 1 - Exemplo de um intervalo mais frequente

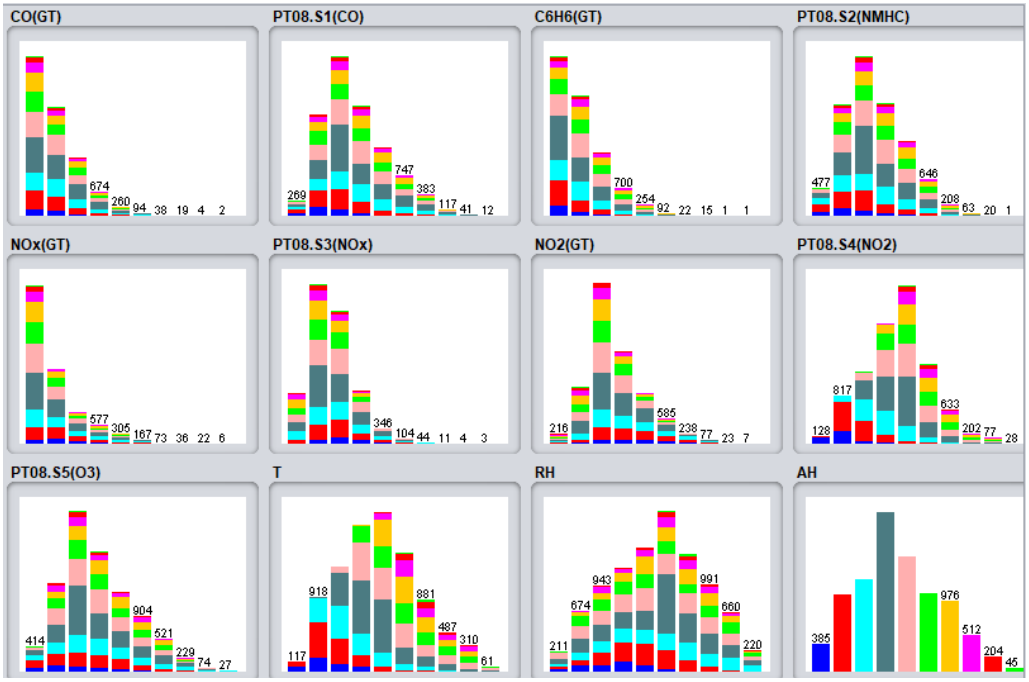

Figura 2 - Vista global da frequência dos intervalos

Analisando, ainda, o *dataset*, conclui-se que a maioria dos atributos tem uma percentagem de valores nulos inferior a 18%, à exceção do atributo NMHC(GT), que, em contraste com os restantes, possui uma percentagem de 90% de valores nulos. Assim, em 9358 registos existem 915 valores que não são nulos.

Name: NMHC(GT)		Type: Numeric
Missing: 8443 (90%)	Distinct: 429	Unique: 208 (2%)

Figura 3 - Sumário do atributo NMHC(GT)

Se nos limitássemos a substituí-los pelo valor mais frequente, obteríamos uma frequência superior a 90% desse valor, o que enviesaria os resultados e as conclusões a que pudéssemos chegar. Assim, como os valores existentes deste atributo não são significativos

para a análise do problema, decidiu-se eliminá-lo. Dos 15 atributos iniciais, estamos, então, a considerar 12 atributos.

3. Classificação

3.1. NaïveBayes

Naïve Bayes consiste na família de algoritmos probabilísticos que utiliza teorias de probabilidades e o Teorema de *Bayes* para prever a classe de um objeto. Visto que são probabilísticos, estes algoritmos calculam a probabilidade de uma classe se referir a um objeto, escolhendo, depois, a classe que possuir um maior valor de probabilidade sobre os atributos. Estas probabilidades são calculadas através do Teorema de *Bayes*, que descreve a probabilidade de um evento, através de conhecimento anterior relacionado com esse mesmo evento. Denote-se que os classificadores de *Naive Bayes* assumem que os atributos têm distribuições independentes e são normalmente representados por grafos, em que a direção da seta indica que cada classe causa certos valores de atributos com determinada probabilidade, como exemplificado na figura seguinte.

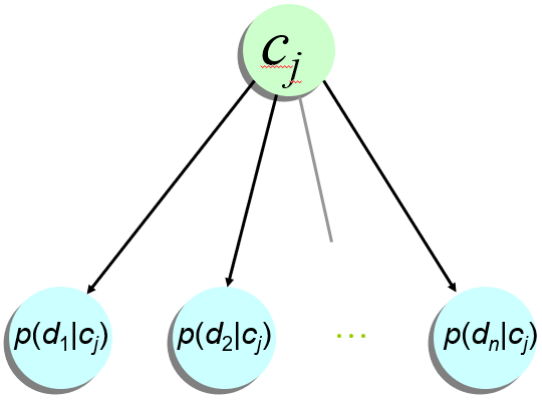

Figura 4 - Exemplo de um classificador de *Bayes*

Assim, possuindo a noção de como este algoritmo funciona, foi necessário efetuar testes. Considerando o algoritmo "*Naive Bayes*" do WEKA, as opções de teste utilizadas foram a validação cruzada (x10) e dividir o *dataset* em 64% para treino e 36% para teste. Não foram consideradas as opções de utilizar o *dataset* inteiro para teste para evitar *overfitting*, nem utilizar um *dataset* de teste, visto que este não existia.

Além do *Naive Bayes*, também foram efetuados testes com o algoritmo *AdaBoost* sobre o *Naive Bayes*, no entanto, não houve melhoria nos resultados, por isso não serão mostrados tais valores. O algoritmo *AdaBoost* é um algoritmo que serve para aumentar a performance de outros algoritmos de aprendizagem. Este algoritmo é adaptável, visto que para criar um novo

modelo de classificação utiliza as instâncias erradamente classificadas em classificações anteriores. Assim, o *AdaBoost* funciona melhor com classificadores fracos, tais como as árvores de decisão, logo não trouxe alterações ao erro.

Outro teste, surgiu com a utilização de *bagging* sobre o *Naïve Bayes*. O algoritmo *bagging* vê as diferenças entre diferentes conjuntos de treino. O problema surge do facto do *Naïve Bayes* ser bom a lidar com variância, logo o *bagging* não afetará o erro.

Finalmente, considerando todas as premissas referidas anteriormente, apresenta-se a tabela com os resultados dos testes sobre *Naïve Bayes* e uma imagem que ilustra como os resultados surgem no WEKA.

Tabela 1 - Resultados sobre os testes de *NaïveBayes*

Discretização	Modo de teste	Classes acertadas (%)	Tempo (s)
Não	Cross (x10)	44	0.06
Não	Teste (66%)	43	0.07
Sim	Cross (x10)	49	0
Sim	Teste (66%)	47	0.01

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        2384               48.6729 %
Incorrectly Classified Instances      2514               51.3271 %
Kappa statistic                          0.2511
Mean absolute error                      0.1631
Root mean squared error                  0.3128
Relative absolute error                 84.4602 %
Root relative squared error            100.689  %
Total Number of Instances             4898

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,000    0,000    0,000      0,000    0,000      0,000    0,677     0,010     3
                 0,227    0,017    0,311      0,227    0,262      0,244    0,807     0,180     4
                 0,634    0,251    0,517      0,634    0,569      0,364    0,757     0,537     5
                 0,436    0,298    0,544      0,436    0,484      0,143    0,614     0,547     6
                 0,520    0,179    0,388      0,520    0,445      0,306    0,766     0,403     7
                 0,034    0,009    0,125      0,034    0,054      0,048    0,796     0,106     8
                 0,000    0,000    0,000      0,000    0,000      0,000    0,505     0,001     9
Weighted Avg.    0,487    0,242    0,482      0,487    0,477      0,237    0,697     0,487

=== Confusion Matrix ===

   a   b   c   d   e   f   g   <-- classified as
   0   2   9   6   3   0   0 |   a = 3
   0  37  71  37  18   0   0 |   b = 4
   0  59 924 401  67   6   0 |   c = 5
   0  18 668 959 538  15   0 |   d = 6
   0   2  97 303 458  20   0 |   e = 7
   0   1  18  58  92   6   0 |   f = 8
   0   0   1   0   3   1   0 |   g = 9
```

Figura 5 – Resultados com validação cruzada e dados discretizados

Analisando a tabela, a primeira conclusão a ser retirada é que obtemos melhores resultados quando os dados foram discretizados. A discretização consistiu em discretizar os

atributos em 10 intervalos com a mesma frequência. O melhor resultado, marcado a azul, verificou-se com os dados discretizados e utilizando validação cruzada (x10), tendo acertado quase 50% das classes treinadas.

Outra forma de avaliar os modelos passa por analisar a área que se forma pela linha ROC, também conhecida como AUC (*Area Under Curve*). Avaliando esta métrica e considerando dois modelos com valores AUC diferentes, o melhor dos dois será o que apresentar o AUC mais elevado. Assim, utilizando a opção "Visualize threshold curve" sobre o *result buffer* obtiveram-se os seguintes resultados, considerando validação cruzada (x10) e 36% do *dataset* para treino sobre os dados discretizados.

Tabela 2 - Resultados dos valores de AUC para diferentes classes e modos

Modo	Classe	AUC	Modo	Classe	AUC
x10	3	0.68	66/34	3	0.60
	4	0.81		4	0.80
	5	0.76		5	0.74
	6	0.61		6	0.59
	7	0.77		7	0.76
	8	0.80		8	0.80
	9	0.51		9	0.51

Analisando a tabela, conseguimos visualizar que os valores de AUC para a validação cruzada são, maioritariamente, superiores aos do teste do *dataset* com 66% dos dados para teste, logo, novamente, a validação cruzada com os dados normalizados gera um modelo de classificação mais confiável do que os restantes.

De seguida, apresenta-se um exemplo de uma destas curvas e respetiva área utilizando as opções descritas anteriormente.

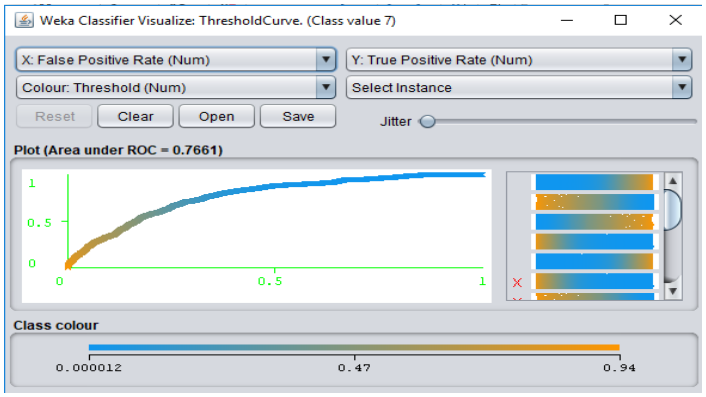

Figura 6 - Gráfico da curva ROC para a classe 7, com validação cruzada

3.2. J48 e SimpleCART

Dentro dos vários algoritmos de classificação disponibilizados pelo WEKA, temos o algoritmo J48 e SimpleCART para a geração de árvores de decisão. Uma árvore de decisão contém várias ramificações onde cada nó intermédio corresponde a testes sobre os valores dos atributos, enquanto que os ramos são os possíveis valores e as folhas as decisões sobre a previsão do valor da classe.

Tendo em conta que o *dataset* de *Wine Quality* não contém valores nulos ou inexistentes, não foi necessário fazer qualquer tratamento para esta situação. No entanto, tendo em conta a grande disparidade do alcance de alguns dos atributos foi feita uma normalização dos atributos "free sulfur dioxide" (dióxido de enxofre livre) e "total sulfur dioxide" (dióxido de enxofre total). Por outro lado, também foi feita uma discretização dos atributos em 10 intervalos de frequência igual.

Em relação ao algoritmo J48, podemos alterar vários parâmetros de modo a analisar a sua *performance*. Deste modo, podemos indicar que não queremos que seja realizado nenhum tratamento de *pruning*, nem usada a correção de Laplace ou *subtreeRaising*. Além disso, ainda podem ser utilizados outros algoritmos que atuam com o J48 de forma a tentar melhorar a percentagem de erro dos resultados, sendo eles o AdaBoost e o Bagging. O primeiro, AdaBoost, cria vários modelos, onde cada um deles tenta corrigir o que o anterior errou. Assim, este algoritmo itera até se atingir um valor de *accuracy* mínimo ou mais nenhum melhoramento possa ser feito. Por sua vez, o algoritmo Bagging (*Bootstrapp Aggregation*) cria amostras do *training set* e cria um modelo para cada uma delas. Os resultados destes modelos são por fim combinados (calculando a sua média ou por votação uniforme). Este algoritmo visa, então, reduzir a variância do modelo.

O algoritmo SimpleCART gera sempre árvores de decisão binárias, logo gera uma árvore que só contém duas ramificações (ou duas subárvores). Neste modelo podemos indicar se queremos ou não fazer a "poda" da árvore.

A análise de *performance* pode ser feita através de várias medidas, tais como a percentagem de erro de previsão, o valor de ROC, o valor de *precision* e de *recall,* entre outros*.*

Primeiramente foram realizados testes sem nenhuma opção de *pruning* e com diferentes opções de teste (*cross-validation*, *training set* e *percentage split*). De seguida, usando apenas o *cross-validation*, foram testados casos em que foi feita a operação de *pruning* (*subTreeRaising*, *unpruned* e *LaPlace*) e ainda uma combinação de modelos (AdaBoost e Bagging).

3.2.1 J48

- ***Cross-validation* 10 *folds***

Usando como opção de teste o *cross-validation* 10 *folds*, em que o *dataset* é dividido em 10 conjuntos e a cada iteração temos 9 *sets* para treino e 1 *set* para teste. A *performance* é a média dos resultados de cada um dos dez *sets*.

➢ Sem *pruning*

Criando a árvore de decisão sem nenhuma opção de *pruning*, obtiveram-se os seguintes resultados:

Correctly Classified Instances	2820	57.5745 %
Incorrectly Classified Instances	2078	42.4255 %
Kappa statistic	0.3668	
Mean absolute error	0.1255	
Root mean squared error	0.311	
Relative absolute error	65.0048 %	
Root relative squared error	100.1159 %	
Total Number of Instances	4898	

Figura 7 - Erro de x-fold sem pruning

```
=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,000    0,003    0,000      0,000    0,000      -0,004   0,492     0,004     3
               0,264    0,028    0,243      0,264    0,253      0,226    0,644     0,118     4
               0,631    0,186    0,589      0,631    0,609      0,436    0,777     0,576     5
               0,626    0,313    0,619      0,626    0,623      0,313    0,704     0,619     6
               0,500    0,091    0,547      0,500    0,522      0,424    0,763     0,450     7
               0,234    0,016    0,350      0,234    0,281      0,265    0,751     0,217     8
               0,000    0,000    0,000      0,000    0,000      -0,000   0,499     0,001     9
Weighted Avg.  0,576    0,214    0,572      0,576    0,573      0,363    0,735     0,542

=== Confusion Matrix ===

    a    b    c    d    e    f    g   <-- classified as
    0    0   12    4    2    2    0 |    a = 3
    1   43   53   52   13    1    0 |    b = 4
    7   67  919  405   51    8    0 |    c = 5
    5   50  479 1377  247   40    0 |    d = 6
    2   14   87  313  440   24    0 |    e = 7
    1    3    9   70   50   41    1 |    f = 8
    0    0    0    2    2    1    0 |    g = 9
```

Figura 8 - Estatísticas de x-fold sem pruning

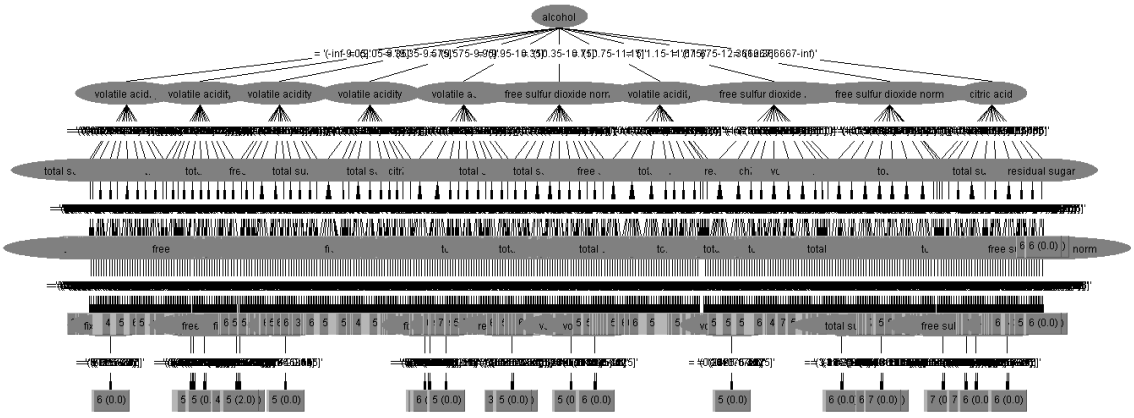

Figura 9 - Árvore de decisão de x-fold sem pruning

Tal como se pode verificar, obteve-se uma percentagem de erro de 42.4255% e um valor de ROC de 0.732. Ainda se pode observar a árvore de decisão gerada através deste algoritmo.

Caso aumentássemos o número de *folds* do *cross-validation* para 20, o erro manter-se-ia, mas o valor de ROC aumentaria para 0.738, o que não é uma mudança muito significativa, por isso vamos continuar com o número de *folds* igual a 10. Caso o número de *folds* fosse 50, o erro reduziria para 41.2005% e o ROC aumentaria para 0.747, o que, novamente, não é um melhoramento considerável, tendo em conta que estamos com cinco vezes mais *folds*. À medida que o número de *folds* aumenta, o número de testes e de treino também aumenta, o que pode levar a um menor erro e a uma melhor previsão do modelo.

➢ Apenas a opção de *pruning*

Neste caso, ligamos apenas a opção de *unpruned*, ou seja, passa de *True* (sem "poda") para *False* (com "poda").

Correctly Classified Instances	2709	55.3083 %
Incorrectly Classified Instances	2189	44.6917 %
Kappa statistic	0.3053	
Mean absolute error	0.1467	
Root mean squared error	0.3032	
Relative absolute error	75.9808 %	
Root relative squared error	97.617 %	
Total Number of Instances	4898	

Figura 10 - Erro de *pruning*

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,000	0,002	0,000	0,000	0,000	-0,003	0,449	0,004	3
	0,110	0,012	0,240	0,110	0,151	0,144	0,698	0,099	4
	0,579	0,169	0,592	0,579	0,585	0,412	0,773	0,577	5
	0,680	0,441	0,557	0,680	0,612	0,239	0,660	0,585	6
	0,370	0,076	0,517	0,370	0,432	0,338	0,765	0,439	7
	0,160	0,010	0,378	0,160	0,225	0,229	0,771	0,206	8
	0,000	0,000	0,000	0,000	0,000	-0,000	0,484	0,001	9
Weighted Avg.	0,553	0,262	0,540	0,553	0,539	0,303	0,716	0,524	

=== Confusion Matrix ===

a	b	c	d	e	f	g	<-- classified as
0	0	10	8	1	1	0	a = 3
1	18	69	69	5	1	0	b = 4
3	28	843	548	28	7	0	c = 5
4	23	435	1494	218	24	0	d = 6
0	6	59	477	326	12	0	e = 7
0	0	9	86	51	28	1	f = 8
0	0	0	2	2	1	0	g = 9

Figura 11 - Estatísticas de *pruning*

Tal como se pode ver, a opção de *pruning* neste caso não melhorou a *performance* do algoritmo, uma vez que o erro sem o *pruning* era de 42.4255% e passou para 44.6917%. Também a capacidade de previsão do algoritmo piorou, uma vez que o valor de ROC, anteriormente de 0.732, passou para 0.716.

➢ *Pruning* e *subtreeRaising*

Neste caso, no processo de *pruning* da árvore de decisão também é realizada a opção de subir as subárvores sempre que necessário.

```
Correctly Classified Instances        2710               55.3287 %
Incorrectly Classified Instances      2188               44.6713 %
Kappa statistic                          0.3066
Mean absolute error                      0.1468
Root mean squared error                  0.3035
Relative absolute error                 76.0143 %
Root relative squared error             97.6897 %
Total Number of Instances             4898
```

Figura 12 - Erro de *pruning* e *subtreeRaising*

```
=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,000    0,001    0,000      0,000    0,000      -0,002   0,455     0,004     3
                 0,110    0,012    0,237      0,110    0,151      0,142    0,697     0,095     4
                 0,581    0,170    0,591      0,581    0,586      0,413    0,773     0,575     5
                 0,676    0,437    0,557      0,676    0,611      0,238    0,658     0,583     6
                 0,377    0,077    0,518      0,377    0,437      0,342    0,766     0,444     7
                 0,160    0,010    0,373      0,160    0,224      0,227    0,772     0,202     8
                 0,000    0,000    0,000      0,000    0,000      -0,000   0,484     0,001     9
Weighted Avg.    0,553    0,261    0,540      0,553    0,540      0,304    0,716     0,523

=== Confusion Matrix ===

    a    b    c    d    e    f    g   <-- classified as
    0    0   10    8    1    1    0 |    a = 3
    0   18   69   69    6    1    0 |    b = 4
    3   28  847  544   28    7    0 |    c = 5
    4   24  438 1485  222   25    0 |    d = 6
    0    6   59  471  332   12    0 |    e = 7
    0    0    9   87   50   28    1 |    f = 8
    0    0    0    2    2    1    0 |    g = 9
```

Figura 13 - Estatísticas de *pruning* e *subtreeRaising*

Em comparação com termos apenas a opção de *pruning* ligada, a opção de *subtreeRaising* revelou-se uma mais valia em relação à percentagem de erro, uma vez que o diminuiu. No entanto, o valor de ROC manteve-se igual ao caso anterior.

➢ *Pruning* com *subtreeRaising* e *Laplace*

Neste caso, para além de fazermos uma “poda” à árvore de decisão, também realizamos a operação de subir subárvores, bem como a correção de Laplace nas suas folhas.

```
Correctly Classified Instances        2710               55.3287 %
Incorrectly Classified Instances      2188               44.6713 %
Kappa statistic                          0.3066
Mean absolute error                      0.1915
Root mean squared error                  0.3011
Relative absolute error                 99.1774 %
Root relative squared error             96.9423 %
Total Number of Instances             4898
```

Figura 14 - Erro de *pruning* com *subtreeRaising* e *Laplace*

```
=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,000    0,001    0,000      0,000    0,000      -0,002   0,491     0,004     3
               0,110    0,012    0,237      0,110    0,151      0,142    0,655     0,089     4
               0,581    0,170    0,591      0,581    0,586      0,413    0,773     0,570     5
               0,676    0,437    0,557      0,676    0,611      0,238    0,624     0,529     6
               0,377    0,077    0,518      0,377    0,437      0,342    0,773     0,437     7
               0,160    0,010    0,373      0,160    0,224      0,227    0,738     0,211     8
               0,000    0,000    0,000      0,000    0,000      -0,000   0,611     0,002     9
Weighted Avg.  0,553    0,261    0,540      0,553    0,540      0,304    0,700     0,496

=== Confusion Matrix ===

    a    b    c    d    e    f    g   <-- classified as
    0    0   10    8    1    1    0 |    a = 3
    0   18   69   69    6    1    0 |    b = 4
    3   28  847  544   28    7    0 |    c = 5
    4   24  438 1485  222   25    0 |    d = 6
    0    6   59  471  332   12    0 |    e = 7
    0    0    9   87   50   28    1 |    f = 8
    0    0    0    2    2    1    0 |    g = 9
```

Figura 15 - Estatísticas de *pruning* com *subtreeRaising* e *Laplace*

Embora a percentagem de erro se tenha mantido igual ao caso anterior, o valor de ROC diminuiu, logo a capacidade de previsão do modelo deteriorou-se em relação ao *pruning* só com a operação de *subtreeRaising.*

- **Usando o próprio *training set* para testes**

Com esta opção de teste, usamos o *dataset* de treino para os testes de avaliação de *performance*. Desta maneira, o algoritmo não é introduzido a novos casos, pois todos os que foram testados, já foram treinados anteriormente.

```
Correctly Classified Instances        4397               89.7713 %
Incorrectly Classified Instances       501               10.2287 %
Kappa statistic                          0.8478
Mean absolute error                      0.0355
Root mean squared error                  0.1332
Relative absolute error                 18.3897 %
Root relative squared error             42.8915 %
Total Number of Instances             4898
```

Figura 16 - Erro de *training set* sem *pruning*

```
=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,700    0,001    0,778      0,700    0,737      0,737    0,999     0,828     3
               0,804    0,009    0,762      0,804    0,782      0,775    0,996     0,886     4
               0,939    0,060    0,870      0,939    0,903      0,861    0,990     0,974     5
               0,916    0,067    0,918      0,916    0,917      0,849    0,987     0,981     6
               0,838    0,015    0,926      0,838    0,879      0,856    0,993     0,964     7
               0,754    0,002    0,923      0,754    0,830      0,829    0,998     0,939     8
               0,400    0,000    1,000      0,400    0,571      0,632    1,000     0,767     9
Weighted Avg.  0,898    0,051    0,899      0,898    0,897      0,850    0,990     0,970

=== Confusion Matrix ===

    a    b    c    d    e    f    g   <-- classified as
   14    0    2    3    1    0    0 |    a = 3
    0  131   18   12    2    0    0 |    b = 4
    3   16 1368   55   12    3    0 |    c = 5
    1   18  137 2013   24    5    0 |    d = 6
    0    7   44   89  737    3    0 |    e = 7
    0    0    4   21   18  132    0 |    f = 8
    0    0    0    1    2    0    2 |    g = 9
```

Figura 17 - Estatísticas de *training set* sem *pruning*

Tal como se pode verificar, o erro neste caso é de 10.2287% e o valor de ROC é de 0.990, que são ambos muito bons, porém temos de ter em consideração que este modelo está demasiado habituado aos casos de testes, logo é normal que o erro seja tão baixo e que a previsão (valor de ROC) seja tão alta. Caso se introduza um novo caso de teste, é muito provável que o modelo erre a previsão, pois é algo novo e as condições de avaliação de *performance* não foram muito realísticas.

- ***Percentage split* de 75%**

Com esta opção de teste, 75% do *dataset* em causa é usado para treino e os restantes 25% são usados para teste, ou seja, estamos a usar ¾ do *dataset* para fins de treino e ¼ para testes.

```
Correctly Classified Instances         664               54.2484 %
Incorrectly Classified Instances       560               45.7516 %
Kappa statistic                          0.3133
Mean absolute error                      0.1359
Root mean squared error                  0.3239
Relative absolute error                 70.399  %
Root relative squared error            104.2716 %
Total Number of Instances             1224
```

Figura 18 - Erro de *percentage split* sem *pruning*

```
=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,000    0,005    0,000      0,000    0,000      -0,005   0,493     0,005     3
                 0,239    0,031    0,234      0,239    0,237      0,206    0,654     0,113     4
                 0,578    0,203    0,559      0,578    0,568      0,372    0,729     0,529     5
                 0,618    0,343    0,591      0,618    0,604      0,274    0,674     0,583     6
                 0,448    0,095    0,497      0,448    0,471      0,368    0,742     0,394     7
                 0,105    0,014    0,190      0,105    0,136      0,121    0,671     0,125     8
                 0,000    0,000    0,000      0,000    0,000      0,000    0,499     0,001     9
Weighted Avg.    0,542    0,233    0,535      0,542    0,538      0,312    0,701     0,498

=== Confusion Matrix ===

   a   b   c   d   e   f   g   <-- classified as
   0   0   2   4   0   0   0 |   a = 3
   0  11  12  18   5   0   0 |   b = 4
   5  17 218 113  18   6   0 |   c = 5
   1  14 131 336  55   7   0 |   d = 6
   0   4  26  83  95   4   0 |   e = 7
   0   1   1  14  18   4   0 |   f = 8
   0   0   0   1   0   0   0 |   g = 9
```

Figura 19 - Estatísticas de *percentage split* sem *pruning*

Tal como podemos verificar, o erro é de 45.7516% e o valor de ROC é de 0.701. Dentro dos testes que foram feitos usando o algoritmo J48 sem *pruning*, este caso foi o que obteve, então, piores valores de erro e de ROC. No entanto, esta opção de teste é mais realística do que usar o próprio *training set* para avaliação, logo a sua percentagem de erro é maior. A opção de *cross-validation* continua a ser a melhor opção de teste, pois confere uma maior oportunidade de testes e treino, que resulta numa melhor previsão de casos.

Caso reduzíssemos a percentagem de *dataset* usado para treino para 66%, aumentamos a percentagem de teste para 34%, ou seja, estamos a usar para treino 2/3 do *dataset* e para teste 1/3 do mesmo. Nestas condições obteríamos um erro de 46.8468% e um valor de ROC de 0.686, que são piores do que o caso anterior, uma vez que estamos a disponibilizar menos casos de treino para o algoritmo.

3.3. AdaBoost e Bagging

Numa tentativa de melhorar ainda mais a *performance* do algoritmo é possível fazer uma combinação de modelos. Deste modo, temos o algoritmo AdaBoost e o Bagging, que visam baixar a percentagem de instâncias incorretamente avaliadas, ou seja, pretendemos atingir uma melhor previsão de classes. Assim, usando o melhor caso de teste da secção anterior: *pruning* com *subtreeRaising*, submetemo-lo a duas novas situações.

3.3.1 AdaBoost

Correctly Classified Instances	3154	64.3936 %
Incorrectly Classified Instances	1744	35.6064 %
Kappa statistic	0.4609	
Mean absolute error	0.1036	
Root mean squared error	0.2965	
Relative absolute error	53.6789 %	
Root relative squared error	95.4469 %	
Total Number of Instances	4898	

Figura 20 - Erro de AdaBoost

```
=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,000    0,000    0,000      0,000    0,000      -0,001   0,537     0,006     3
               0,209    0,011    0,400      0,209    0,274      0,272    0,742     0,253     4
               0,669    0,142    0,667      0,669    0,668      0,527    0,843     0,717     5
               0,710    0,304    0,656      0,710    0,682      0,405    0,774     0,700     6
               0,580    0,085    0,600      0,580    0,590      0,502    0,846     0,642     7
               0,423    0,009    0,632      0,423    0,507      0,503    0,812     0,469     8
               0,000    0,000    0,000      0,000    0,000      -0,000   0,600     0,002     9
Weighted Avg.  0,644    0,194    0,636      0,644    0,638      0,455    0,807     0,668

=== Confusion Matrix ===

    a    b    c    d    e    f    g   <-- classified as
    0    0    9    9    1    1    0 |    a = 3
    0   34   66   55    8    0    0 |    b = 4
    0   30  975  397   51    4    0 |    c = 5
    1   17  359 1561  237   23    0 |    d = 6
    1    3   48  303  510   15    0 |    e = 7
    0    1    4   54   41   74    1 |    f = 8
    0    0    1    2    2    0    0 |    g = 9
```

Figura 21 - Estatísticas de AdaBoost

Deste modo, conseguimos verificar que o erro diminuiu significativamente, pois agora toma o valor de 35.6064% e conseguimos realizar melhores previsões, uma vez que o valor de ROC subiu para 0.807. Este teste foi feito para 10 iterações. Caso aumentássemos esse número para 50, teríamos melhores resultados (erro = 34.9939% e ROC = 0.824). No entanto, como não houve um melhoramento significativo, manteve-se o número de iterações igual a 10.

3.3.2 Bagging

Correctly Classified Instances	3085	62.9849 %
Incorrectly Classified Instances	1813	37.0151 %
Kappa statistic	0.428	
Mean absolute error	0.1408	
Root mean squared error	0.2675	
Relative absolute error	72.941 %	
Root relative squared error	86.1273 %	
Total Number of Instances	4898	

Figura 22 - Erro de Bagging

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,000	0,000	0,000	0,000	0,000	-0,001	0,479	0,005	3
	0,129	0,006	0,438	0,129	0,199	0,224	0,788	0,240	4
	0,647	0,136	0,668	0,647	0,658	0,516	0,848	0,720	5
	0,736	0,365	0,621	0,736	0,674	0,369	0,756	0,716	6
	0,515	0,076	0,597	0,515	0,553	0,465	0,850	0,623	7
	0,291	0,005	0,680	0,291	0,408	0,433	0,843	0,414	8
	0,000	0,000	0,000	0,000	0,000	-0,000	0,483	0,001	9
Weighted Avg.	0,630	0,218	0,624	0,630	0,618	0,426	0,803	0,670	

=== Confusion Matrix ===

a	b	c	d	e	f	g	<-- classified as
0	0	7	12	1	0	0	a = 3
0	21	64	65	13	0	0	b = 4
0	10	943	462	39	3	0	c = 5
1	14	346	1617	208	12	0	d = 6
0	2	47	369	453	9	0	e = 7
0	1	3	77	42	51	1	f = 8
0	0	1	1	3	0	0	g = 9

Figura 23 - Estatísticas de Bagging

Com dez iterações do algoritmo Bagging, usando o classificador J48, obteve-se uma percentagem de erro também menor, com valor de 37.0151% e ROC de 0.803. No entanto, ao contrário do algoritmo de AdaBoost, aumentando o número de iterações para 50, obtém-se uma redução substancial do erro, uma vez que desce para 34.8101% e o valor de ROC sobe para 0.821. Assim, verifica-se uma mudança significativa na percentagem de instâncias incorretamente classificadas, enquanto que a capacidade de previsão não aumenta consideravelmente (sofre apenas um aumento de 0.018).

Em suma, pode-se concluir que a combinação de modelos é uma mais valia para a capacidade de previsão do modelo J48, logo recomenda-se o seu uso para a criação de árvores de decisão.

3.4. SimpleCART

Usando o mesmo *dataset* e ligando a opção de *pruning* obtivemos os seguintes resultados:

```
Correctly Classified Instances        2800               57.1662 %
Incorrectly Classified Instances      2098               42.8338 %
Kappa statistic                          0.354
Mean absolute error                      0.1323
Root mean squared error                  0.3182
Relative absolute error                 68.5458 %
Root relative squared error            102.4203 %
Total Number of Instances             4898
```

Figura 24 - Erro de SimpleCART com *pruning*

```
=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,000    0,002    0,000      0,000    0,000      -0,003   0,500     0,004     3
               0,153    0,018    0,225      0,153    0,182      0,163    0,674     0,112     4
               0,591    0,168    0,598      0,591    0,595      0,424    0,754     0,553     5
               0,658    0,340    0,612      0,658    0,634      0,317    0,689     0,610     6
               0,486    0,106    0,502      0,486    0,494      0,386    0,767     0,431     7
               0,223    0,018    0,320      0,223    0,263      0,244    0,746     0,185     8
               0,000    0,000    0,000      0,000    0,000      0,000    0,494     0,001     9
Weighted Avg.  0,572    0,223    0,562      0,572    0,566      0,352    0,723     0,526

=== Confusion Matrix ===

    a    b    c    d    e    f    g   <-- classified as
    0    1    7    7    4    1    0 |    a = 3
    1   25   77   49   10    1    0 |    b = 4
    5   42  861  463   75   11    0 |    c = 5
    2   31  410 1447  276   32    0 |    d = 6
    1    9   76  328  428   38    0 |    e = 7
    0    3    7   70   56   39    0 |    f = 8
    0    0    1    1    3    0    0 |    g = 9
```

Figura 25 - Estatísticas de SimpleCART com *pruning*

Tendo em conta que a percentagem de erro é de 42.8338% e o ROC é de 0.723, podemos verificar que o algoritmo SimpleCART tem melhores resultados com *pruning* do que o algoritmo anterior J48.

Desligando, então, a opção de *pruning* obtém-se os resultados seguintes:

```
Correctly Classified Instances        2819               57.5541 %
Incorrectly Classified Instances      2079               42.4459 %
Kappa statistic                          0.3647
Mean absolute error                      0.1276
Root mean squared error                  0.3211
Relative absolute error                 66.076  %
Root relative squared error            103.3823 %
Total Number of Instances             4898
```

Figura 26 - Erro de SimpleCART sem *pruning*

```
=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,000    0,004    0,000      0,000    0,000      -0,004   0,510     0,004     3
               0,209    0,025    0,222      0,209    0,215      0,189    0,634     0,105     4
               0,606    0,171    0,601      0,606    0,603      0,434    0,749     0,552     5
               0,652    0,317    0,626      0,652    0,639      0,334    0,696     0,606     6
               0,485    0,102    0,510      0,485    0,497      0,390    0,756     0,413     7
               0,234    0,018    0,328      0,234    0,273      0,255    0,731     0,183     8
               0,000    0,000    0,000      0,000    0,000      -0,000   0,497     0,001     9
Weighted Avg.  0,576    0,213    0,570      0,576    0,572      0,365    0,721     0,520

=== Confusion Matrix ===

    a    b    c    d    e    f    g   <-- classified as
    0    1    8    6    4    1    0 |    a = 3
    1   34   71   44   12    1    0 |    b = 4
   10   58  883  425   71   10    0 |    c = 5
    8   45  412 1434  266   33    0 |    d = 6
    1   12   86  315  427   39    0 |    e = 7
    0    3    9   66   55   41    1 |    f = 8
    0    0    1    1    3    0    0 |    g = 9
```

Figura 27 - Estatísticas de SimpleCART sem *pruning*

Mais uma vez, é o caso sem *pruning* que obtém uma percentagem de erro menor. No entanto, verifica-se que, embora a mudança seja mínima, obtém-se um valor de ROC ligeiramente maior para o caso anterior.

3.5. Resumo dos resultados de árvores de decisão

Tabela 3 - Resultados dos vários testes de árvores de decisão

		Erro (%)	ROC	*Precision*	*Recall*
Sem *pruning*	*x-fold*	42.4255	0.735	0.572	0.576
	Training set	10.2287	0.990	0.899	0.898
	Percentage split	45.7516	0.701	0.535	0.542
Com *pruning*	Só *pruning*	44.6917	0.716	0.540	0.553
	SubtreeRaising	44.6713	0.716	0.540	0.553
	Laplace	44.6713	0.700	0.540	0.553
AdaBoost		35.6084	0.807	0.636	0.644
Bagging		37.0151	0.803	0.624	0.630
SimpleCART	Sem *pruning*	42.4459	0.721	0.570	0.576
	Com *pruning*	42.8338	0.723	0.562	0.572

Atendendo às percentagens de erro de cada um dos casos de teste, temos que o menor erro se verificou quando se usou o próprio *training set* como *set* de teste. Ora, como é óbvio, esta situação não é muito realística, uma vez que o modelo está a ser testado com os mesmos casos com que foi treinado, logo nenhum caso novo foi lhe fornecido para ser avaliado. Deste modo, o próximo melhor resultado registou-se quando se usou uma combinação de modelos através de AdaBoost com o classificador J48. Os seus parâmetros incluíram a opção de *pruning* e a de *subtreeRaising*. O algoritmo AdaBoost realizou 10 iterações. Deste modo, podemos verificar que tanto o valor de ROC (que mede a capacidade preditiva do modelo), como o da precisão (que mede a qualidade de cada previsão individual) e o do *recall* (que mede a proporção de respostas corretas) são os mais elevados quando comparados com os dos restantes casos.

3.6. Recomendações para a classificação

Após a apresentação de resultados há algumas conclusões que se podem tirar acerca da classificação:

- Os algoritmos que trabalham com árvores de decisão tendem a fazer *overfitting* mais frequentemente que outros modelos, logo, quando se trabalha com árvores de decisão é importante testar *pruning* para tentar contrair esta tendência;
- É importante testar diferentes modos de treino dos dados, de modo a entender qual produz melhores resultados, tendo em atenção que ao testar o *dataset* inteiro, os resultados obtidos serão, muito provavelmente, fruto do *overfitting* e não da boa capacidade preditiva do modelo. Com este *dataset*, em regra geral, a validação cruzada apresentou melhores valores;
- Verificar outras métricas de análise do modelo além do erro de avaliação, visto que, utilizando o caso das curvas ROC e a sua área AUC, permitem obter uma ideia da capacidade preditiva do modelo;
- Testar sempre algoritmos de aumento de performance. É possível denotar uma enorme melhoria de performance, através do *bagging* e do *AdaBoost*, para os algoritmos que utilizam árvores de decisão, apesar de isso não acontecer para o *Naive Bayes* pelas razões já referidas anteriormente;

4. Segmentação

4.1. Algoritmo k-means

O algoritmo *k-means* é um algoritmo de segmentação que, recebendo um valor de *k*, define *k* valores aleatórios como centróides dos *k clusters.* De seguida, para cada valor do conjunto de objetos, verifica o *cluster* que lhe está mais próximo (existem várias métricas para calcular a proximidade, tais como a distância euclidiana e a distância Manhattan), e atribui esse valor ao *cluster*. Por fim, calcula qual elemento do *cluster* será o novo centroide. Este processo repete-se até todos os dados estarem associados a um *cluster*.

Assim, temos um problema. Qual o valor de *k* inicial? O algoritmo k-means apresenta, como um dos resultados, o valor da soma dos erros quadrados do *cluster*, ou seja, a variância total dentro dos *clusters.* O objetivo passa então por tentar minimizar este valor, de modo a que a variância seja a menor possível. Como critério de paragem, foi definido que se a variância for inferior a 10 unidades (residual), o melhor valor para o *k* foi encontrado.

Antes de avançar para a determinação do *k* ótimo segundo *k-means*, é necessário falar do contexto em que os seguintes testes vão ocorrer. Para aprendizagem será utilizado o *dataset* completo em que escondemos o atributo *quality*. A razão de esconder o atributo vem do facto de ele ser uma classe, deste modo, ajudaria imensamente à construção dos *clusters* saber onde se situam. No entanto, esta situação não é desejável, pois poderia levar a *overffiting*. Assim, escondemos o atributo *quality* (através da opção do WEKA "Ignore attributes") e avançamos para a determinação do *k* ótimo.

De seguida, apresentam-se os valores da variância começando com *k* =1, estando ilustrado também como o resultado é obtido no WEKA.

Tabela 4 - Resultados de teste do k-means

Nº*clusters*	Variância	Tempo (s)
1	608.84	0
2	442.84	0.02
3	391.04	0.03
4	357.32	0.06
5	335.15	0.08

6	323.84	0.16
7	307.02	0.24
8	294.38	2.48
9	282.32	0.29
10	278.55	0.2

Analisando o critério de paragem definido, temos que o *k* ótimo é k=9 *clusters*, logo, doravante, será este valor o utilizado para os próximos testes em *k-means.*

Após ter sido determinado o número ótimo de *clusters*, o próximo teste a ser efetuado passou por, em vez de se treinar com a totalidade do *dataset*, utilizou-se 66% do *dataset* para treino e 34% do *dataset* para teste e obtemos melhores valores de variância na segunda alternativa, como pode ser visto na seguinte figura:

```
kMeans
======

Number of iterations: 60
Within cluster sum of squared errors: 214.32624095746002
```

Figura 28 - Exemplo do valor da variância do k-means

A melhoria do erro pode surgir do possível *overfitting* causado pelo treino com o *dataset* inteiro, em que o modelo se adapta demasiado bem ao modelo, sendo-lhe difícil avaliar novos casos. Esta melhoria também pode ter surgido do facto da opção “Percentage split” alterar a ordem dos dados de forma aleatória, o que poderia ter um impacto positivo, visto que o *k-means* é sensível à ordem das instâncias no *dataset.*

O próximo parâmetro a ser alterado será o método do cálculo da distância entre duas instâncias. O algoritmo *k-means* suporta dois tipos diferentes de função de distância: a distância Euclidiana e a distância de Manhattan. Nos exemplos anteriores, foi sempre utilizada a distância Euclidiana, logo, considerando a divisão dos dados 66%treino/34%teste, apresenta-se a tabela com o resultado da variância para ambos os casos.

Tabela 5 - Resultados k-means para diferentes funções de distância

Distância Euclidiana	Variância= 221.19	Tempo=0.29s
Distância de Manhattan	Variância= 1991.89	Tempo=0.33s

Analisando a tabela, podemos concluir que a variância entre valores para a distância de Manhattan é muito superior à distância Euclidiana, o que faz com que neste caso, seja melhor utilizar a distância Euclidiana.

De seguida, vamos analisar os resultados das métricas que produziram melhores valores de variância faladas anteriormente. Considerando que os centroides iniciais foram

gerados aleatoriamente para cada *cluster*, de seguida apresentam-se os valores finais para os centróides.

Final cluster centroids:

		Cluster#								
Attribute	Full Data	0	1	2	3	4	5	6	7	8
	(4898.0)	(986.0)	(486.0)	(819.0)	(523.0)	(95.0)	(396.0)	(287.0)	(653.0)	(653.0)
fixed acidity	6.8548	7.11	7.6671	6.5993	6.235	6.7116	6.8659	6.6274	6.5294	7.1211
volatile acidity	0.2782	0.2882	0.2813	0.2934	0.2395	0.3263	0.2715	0.2873	0.3161	0.2282
citric acid	0.3342	0.3856	0.3385	0.2655	0.3084	0.4585	0.3393	0.3311	0.3261	0.3484
residual sugar	6.3914	12.0336	5.1183	7.0053	3.3593	4.3358	7.3735	3.2826	3.9392	4.0002
chlorides	0.0458	0.0498	0.0366	0.0494	0.0411	0.1675	0.0472	0.036	0.0347	0.0425
free sulfur d.	35308.0849	46943.712	27118.3128	33225.8852	31961.7591	39784.2105	43066.9192	29958.1882	30371.3629	31057.4273
total sulfur d.	138360.6574	179368.1542	111061.7284	133569.5971	125055.4493	144536.8421	177750	112069.6864	113563.5528	124990.8116
density	0.994	0.9975	0.9926	0.9952	0.9928	0.9945	0.9959	0.991	0.9905	0.9929
pH	3.1883	3.0854	3.0319	3.2238	3.4191	3.0873	3.2996	3.2036	3.2169	3.1423
sulphates	0.4898	0.4893	0.3825	0.428	0.5023	0.4683	0.6541	0.6959	0.4193	0.5217
alcohol	10.5143	9.2456	11.1503	9.5943	10.7987	9.4105	9.875	12.1791	12.4526	10.7607

A partir destes resultados, podemos ver quais os *clusters* que têm mais instâncias a eles relacionados (neste caso é o *cluster* 0), mas também podemos retirar outras conclusões. Por exemplo, analisando o teor de álcool no vinho, podemos concluir que vinho com mais álcool (na ordem dos 12%) estará provavelmente situado ou no *cluster 5* ou no *cluster 6.*

Caso queiramos efetuar uma análise gráfica, eis um exemplo do seu resultado:

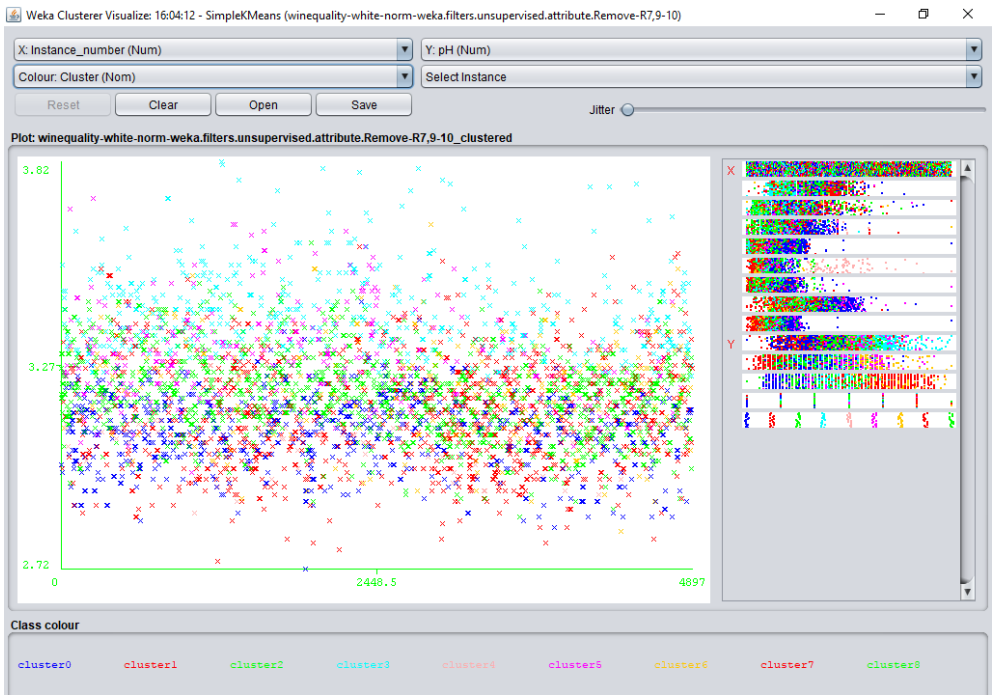

Figura 29 - Representação gráfica de *clusters* com o número da instância e pH

Neste exemplo, estão representadas as instâncias (eixo X) por pH (eixo Y). Como a cor representada é a correspondente ao seu *cluster*, podemos concluir, por exemplo, que no cluster 3 estão representadas as instâncias com maior valor de pH (neste caso superior a 3.27).

No entanto, podemos mudar as unidades dos eixos X e Y se desejarmos. No exemplo seguinte, estão representados os valores do álcool (eixo X) e os *clusters* a que pertencem (eixo Y). Para este caso, podemos concluir que o *cluster 2* não apresenta valores de álcool superiores a 11,1. Assim, podemos ver a distribuição dos *clusters* para cada atributo.

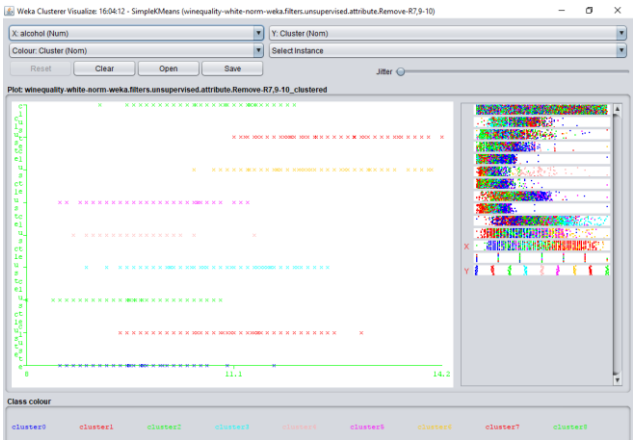

Figura 30 - Representação gráfica de *clusters* com álcool e *cluster*

Finalmente, o último teste com o algoritmo *k-means* implicou o uso do método de aprendizagem "*classes to clusters evaluation"* que analisa o problema de *clusters* como se se tratasse de um problema de classificação. Assim sendo, obtemos os seguintes valores para este teste (teste demorou 0.31 segundos):

```
Class attribute: quality
Classes to Clusters:

   0   1   2   3   4   5   6   7   8  <-- assigned to cluster
   6   2   2   2   1   3   0   3   1 | 3
  27  27  46  13   4  18   5   9  14 | 4
 468 123 415  84  48 130  15  28 146 | 5
 388 233 324 255  40 196 117 291 354 | 6
  82  84  27 147   2  45 120 250 123 | 7
  15  17   5  22   0   4  29  69  14 | 8
   0   0   0   0   0   0   1   3   1 | 9

Cluster 0 <-- 5
Cluster 1 <-- No class
Cluster 2 <-- 4
Cluster 3 <-- No class
Cluster 4 <-- No class
Cluster 5 <-- 3
Cluster 6 <-- 8
Cluster 7 <-- 7
Cluster 8 <-- 6

Incorrectly clustered instances :       3748.0   76.521  %
```

Figura 31 - Resultado do teste k-means com "Classes to clusters evaluation"

O primeiro dado que salta à vista é o facto do modelo ter avaliado incorretamente 76% das instâncias. Isto pode levar a crer que o modelo foi mal construído, no entanto, neste caso, temos que seria muito difícil encontrar um modelo que representasse corretamente este *dataset*, pois o valor da classe é subjetivo, logo não segue nenhuma ciência, variando mediante a pessoa que avaliou o vinho. Assim, vinhos com atributos muito distantes conseguem ter a mesma avaliação, sendo difícil representar o *dataset* através da segmentação.

Analisado os restantes dados, podemos ver como estão distribuídos os diferentes valores de classe para cada *cluster* diferente, sendo que o nível de qualidade 5 é onde se encontram a maior parte dos dados, possuindo o *cluster 0* 468 instâncias da classe *quality*=5. Em baixo da matriz, temos o valor de classe que se associa a cada *cluster*, sendo que o *k-means* associa o *cluster* que gera menor erro de classificação. Apesar de estar representado que alguns *clusters* não apresentam classe, isto não é verdade e é uma peculiaridade do WEKA que não associa um valor de classe a um *cluster* caso já esteja associado a outro *cluster.* Assim, mesmo não estando explicitamente representado, podemos analisar a matriz que indica que, por exemplo, para o *cluster 1* estaria associado o valor de classe 6.

4.2. Algoritmo EM

O algoritmo EM (de *Expectation Maximization*) é um algoritmo implementado pelo WEKA que utiliza métodos estatísticos para produzir *clusters* probabilísticos. Assim, a cada instância está associada uma probabilidade de pertencer a determinado *cluster*. Caso as instâncias sejam atributos nominais, o EM apresenta a probabilidade de cada valor. Caso as instâncias sejam numéricas, o EM apresenta a média e o desvio padrão.

Este algoritmo apresenta uma medida de comparação entre modelos denominada "log likelihood". Contrariamente, ao algoritmo *k-means* o objetivo passar por maximizar este valor.

Contrariamente ao algoritmo *k-means*, o EM não necessita saber qual o valor de *clusters* a considerar *a priori.* Como parâmetro do algoritmo, se considerarmos k=-1, o WEKA saberá que o nosso desejo é encontrar o valor ideal de *clusters.* Assim, de futuro, quando forem efetuados testes o valor ideal de *clusters* será calculado aquando a execução do teste.

O primeiro dos dois testes feitos consistiu em utilizar 66% do *dataset* para treino e 34% para teste. Este teste terminou a execução em 29 minutos (1714 segundos) e determinou como número ótimo de *clusters* k=19. Como os resultados gerados para 19 *clusters* não caberiam numa única página, só serão ilustrados os valores para os três primeiros *clusters* e os três primeiros atributos.

	Cluster		
Attribute	0	1	2
	(0.06)	(0.04)	(0.09)
fixed acidity			
mean	6.207	7.5618	6.9272
std. dev.	0.6693	1.415	0.6526
volatile acidity			
mean	0.3189	0.4019	0.2746
std. dev.	0.1201	0.1775	0.093
citric acid			
mean	0.2486	0.3648	0.2949
std. dev.	0.1547	0.2272	0.0563
residual sugar			
mean	1.2806	8.9784	11.1417
std. dev.	0.2609	4.6267	2.4312

Como referido previamente, estes *clusters* são probabilísticos. Assim, representado em baixo do número de *cluster*, temos a probabilidade desse *cluster* apresentar esses valores, ou seja, o *cluster 0* apresenta 6% de probabilidade do atributo *fixed acidity* possuir uma média de valor 6.207. A soma das probabilidades de todos os *clusters* será igual a 100%. Além disto, tal como foi efetuado em estudos anteriores, podemos analisar que tipos de valores apresenta cada *cluster*. Finalmente, como medida de comparação de modelos, observamos que o "log likelihood" é igual a -16.39559.

O segundo teste a este algoritmo consistiu em analisar o problema como se tratasse de um problema de classificação. Desde o início ao fim da execução deste algoritmo, passaram-se 29 minutos (1727 segundos). O valor de "log likelihood" é superior ao teste prévio, logo este modelo é considerado melhor.

```
Incorrectly clustered instances :       4184.0    85.4226 %
```

Figura 32 - Número de instâncias mal segmentadas para o algoritmo EM

Considerando o valor apresentado na imagem anterior, podemos ver que, relativamente ao algoritmo *k-means*, o número de instâncias incorretamente segmentadas é bastante superior, porém isto pode apresentar uma razão bastante fácil de explicar. Como o atributo classe apresenta valores entre [3,9] e estamos a trabalhar com 19 *clusters*, a maior parte dos *clusters* estariam associados a valores de classe que não existem.

4.3. Algoritmo Hierarchical Clustering

Os métodos de segmentação hierárquicos classificam-se por criar uma decomposição hierárquica dos objetos segundo um determinado critério. Estes algoritmos têm capacidade de produzir estruturas capazes de avaliar/discriminar exemplos de um *dataset*, chamados de dendrogramas. A similaridade entre dois objetos num dendrograma é representada pela altura do nó interno mais baixo que ambos os objetos partilham. De seguida, apresenta-se um exemplo de um dendrograma. Analisando a seguinte imagem, podemos identificar 4 *clusters* diferentes. Geralmente, árvores muito separadas sugerem *clusters* diferentes.

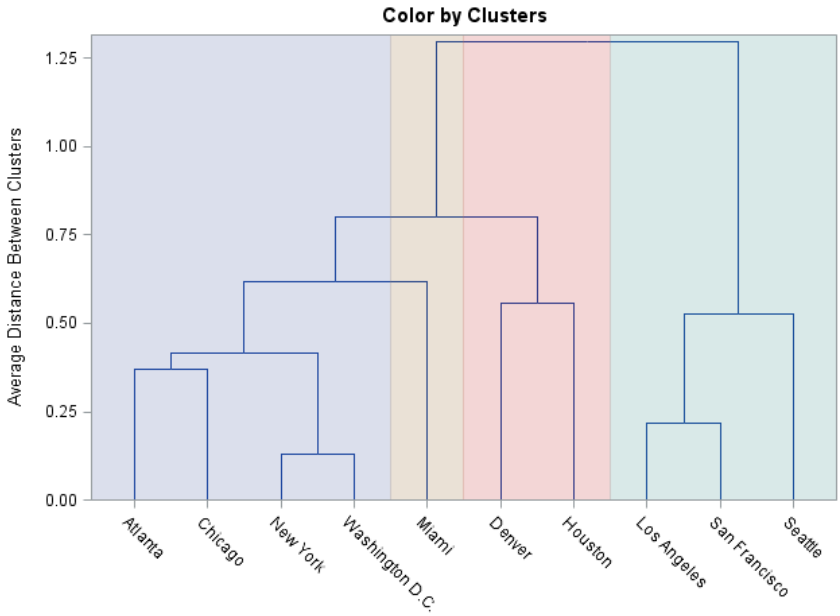

Figura 33 - Exemplo de um dendrograma com 4 *clusters*

Apesar dos dendrogramas serem uma boa forma de identificar diferentes *clusters*, é inconcebível considerar analisar todos os dendrogramas possíveis para um *dataset*, daí serem utilizados métodos que definem a distância entre um objeto e um *cluster* ou entre dois *clusters*. Estes métodos serão discutidos mais à frente aquando a discussão dos resultados.

Outra desvantagem dos dendrogramas, surge quando analisamos dendrogramas de *datasets* com grande volume de dados. No caso que estamos a trabalhar, temos um *dataset* com um grande volume de dados, ou seja, quando tentamos analisar o dendrograma, deparamo-nos com uma árvore com inúmeros ramos, muito difícil de examinar, como se pode verificar na figura seguinte.

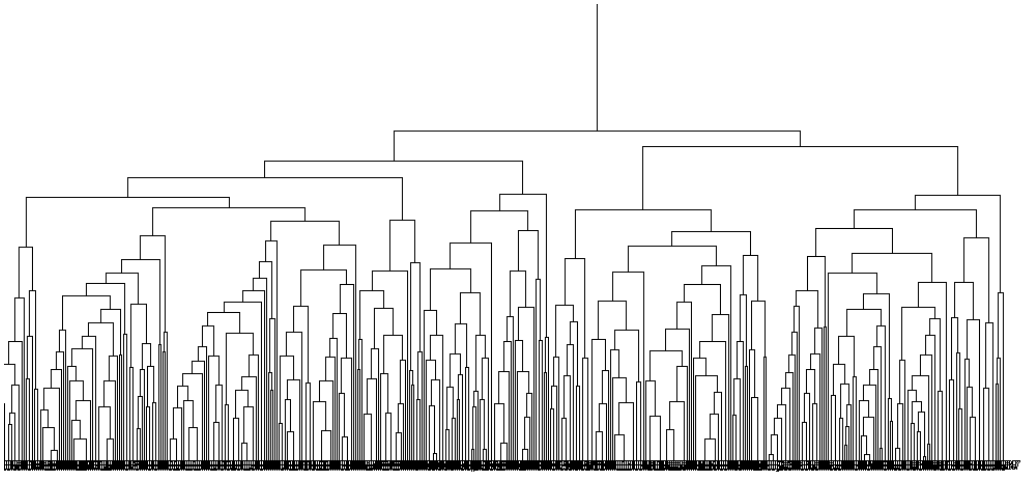

Figura 34 - Exemplo de um dendrograma gerado através do algoritmo HC

Assim, para se efetuar uma análise mais objetiva dos dados, o único modo de utilização dos dados para aprendizagem utilizado foi o "*classes to clusters evaluation*", visto que apresenta um valor de erros (número de classes incorretamente segmentadas).

Prosseguindo para os resultados, de seguida apresenta-se uma tabela que apresenta os resultados obtidos utilizando o algoritmo do WEKA "*Hierarchical Clustering*", com diferentes parâmetros.

Tabela 6 - Resultados com diferentes *link types*, número de *clusters* e funções distância

Nº*clusters*	*Link type*	Função distância	Erro (%)	Tempo (s)
6	MEAN	*EuclidianDistance*	69	72
6	MEAN	*ChebyshevDistance*	62	74
6	WARD	*EuclidianDistance*	69	127
6	WARD	*ChebyshevDistance*	72	122
9	MEAN	*EuclidianDistance*	72	70
9	MEAN	*ChebyshevDistance*	69	67
9	WARD	*EuclidianDistance*	76	121
9	WARD	*ChebyshevDistance*	75	134

A escolha para o valor dos *clusters* baseou-se em dois fatores. A escolha de k=9, surgiu dos testes realizados com o algoritmo *k-means*, que apresentava o *k* ótimo como 9 e obtinha melhores resultados que o algoritmo EM, logo uma porção dos testes efetuados teve como número de *clusters 9.* A escolha de k=6, surgiu do facto do *dataset* original possuir 6 valores identificáveis da classe *quality*, logo, se dividirmos o *dataset* no número de *clusters* correspondente ao número de classes, é mais provável obtermos um erro de classificação menor, o que analisando a tabela, se verifica.

Apesar da tabela só apresentar valores para dois *link types* e funções de distância distintas, nos testes efetuados, também foram avaliados todos os restantes *link types* e funções de distância que o WEKA disponibilizava, no entanto, como os resultados obtidos não representavam corretamente o problema (os resultados apresentavam geralmente o *dataset* dividido num único *cluster)*, não existiu a necessidade de apresentar esses resultados na tabela.

Analisando a tabela, conseguimos concluir que o melhor resultado surgiu quando o número de *clusters* era 6, o *link type MEAN* e a distância era calculada através da distância de Chebyshev. O tipo de ligação MEAN calcula a distância média de um *cluster* fundido (ou seja, média do grupo de segmentação de aglomeração), enquanto que o tipo de ligação WARD encontra a distância da alteração causada com a fusão do cluster. Como o WARD necessita de efetuar mais cálculos, é notório que seja de execução mais lenta.

Também podemos verificar que o tipo de ligação MEAN apresenta melhores valores para o erro, isto porque com o *clustering* hierárquico a soma dos quadrados (através deste cálculo será determinado o centro do *cluster* a cada iteração*)* começa em zero e aumenta à medida que juntamos *clusters*. O método de *Ward* tenta manter este crescimento o mais pequeno possível, o que pode funcionar caso a soma dos quadrados seja pequena, no entanto, como neste *dataset* os *clusters* apresentam variâncias grandes, ou seja, será muito difícil obter *clusters* que representem círculos perfeitos, este método terá resultados inferiores ao MEAN, que se limita a calcular a distância média.

Assim, podemos analisar visualmente os novos *clusters* formados e retirar conclusões tal como fizemos para o algoritmo *k-means.*

Figura 35 - Resultado gráfico da segmentação utilizando HC

4.4. Recomendações para a Segmentação

Após a apresentação de resultados há algumas conclusões que se podem tirar:

- Caso o fator tempo seja importante, o algoritmo a seguir será o k-means que apresenta a resposta em segundos, se bem que o *Hierarchical Clustering* demora entre 1 e 2 minutos e apresenta resultados mais fiáveis. O EM, considerando o *dataset* utilizado, nunca seria escolhida, visto que cada teste superou sempre os 28 minutos em tempos de execução;
- O algoritmo apresenta bons resultados quando a forma dos clusters é esférica, o que leva a concluir que se os dados não apresentarem esta distribuição, o algoritmo de *Hierarchical Clustering* seria uma melhor escolha. O algoritmo EM apresentou erros na ordem dos 95%, logo é claramente o pior dos três;
- Caso o objetivo seja a confiabilidade, a melhor escolha será o *Hierarchical Clustering*, visto que contrariamente ao k-means não apresenta resultados diferentes com execuções diferentes (k-means inicia-se com valores aleatórios de centroides, o que pode levar a diferentes resultados em diferentes execuções);
- O algoritmo k-means é um algoritmo forte, mas só se se souber o número de clusters a serem criados, enquanto que o o EM determina o valor ótimo e o *Hierarchical Clustering* não necessita de saber, visto que se pode parar no nível desejado;
- É muito importante analisar os dados antes de os processar, de modo a melhorar o máximo o comportamento do algoritmo. Considerando o dataset utilizado, se a normalização não tivesse sido efetuada, estaríamos a trabalhar com atributos em escalas diferentes, o que tornaria os clusters mais elípticos e menos esféricos;

Finalmente, é importante descobrir as limitações do *dataset* a analisar e investigar sobre quais as ferramentas, algoritmos e métodos que melhor se adaptam a cada *dataset*, pois, cada *dataset* é único e é impossível definir uma estratégia que indique quais os passos a seguir para obter o modelo ideal.

5. Regras de Associação

5.1. Algoritmo Apriori

Na geração de regras de associação, foi usado o algoritmo Apriori, que se divide em duas fases: encontrar itemsets frequentes e gerar regras a partir deles. A primeira parte só considera os itens que tenham um valor superior ao mínimo especificado para o suporte. A segunda parte considera apenas as regras cujo nível de confiança (caso seja esta a medida de interesse usada) seja também superior a um certo valor mínimo. Não foi usado o algoritmo FPGrowth, também disponível no WEKA, uma vez que este só lida com valores binários, o que não é apropriado para o problema em questão.

5.1.1 Discretização com 10 intervalos

Em relação ao algoritmo Apriori, considerou-se três situações distintas em que iriamos variar o valor de suporte mínimo. Assim, o suporte seria de 0.1, 0.05 e 0.01. Para estes três valores, temos que o valor mínimo de confiança que uma regra tem de possuir é de 0.85, 0.9 e 0.95.

Tabela 7 - Número de regras geradas tendo em conta a confiança (10 *bins*)

Suporte	0.1			0.05			0.01		
Confiança	0.75	0.85	0.95	0.75	0.85	0.95	0.75	0.85	0.95
Número de regras geradas	5	4	4	77	38	33	24442	20931	14376

Deste modo, como podemos verificar analisando a tabela acima, à medida que o valor de suporte mínimo decresce, o número de regras geradas é maior, uma vez que o número de *itemsets* frequentes aumenta, logo temos mais itens candidatos a fazerem parte de regras de associação. O contrário acontece quando para cada valor de suporte tornamo-nos mais exigentes em relação à geração de regras, isto é, o valor mínimo de confiança aumenta. Pois com um valor maior de confiança regista-se, em geral, um menor número de regras geradas.

Exportando os resultados do teste com o suporte mínimo = 0.1 e a confiança = 0.95, podemos facilmente analisar as regras obtidas e identificar possíveis casos em que algumas das regras geradas sejam redundantes. Uma regra redundante é identificada quando verificamos que existem duas regras semelhantes, em que uma é mais específica do que a outra, e ambas apresentam o mesmo valor de suporte e o valor de confiança não aumenta ou não aumenta o suficiente (tendo por base um limite mínimo de *improvement*).

Assim, são identificadas as seguintes regras:

```
PT08.S1(CO)='(1406.5-inf)' C6H6(GT)='(20.15-inf)' 629 ==> CO(GT)='(3.95-inf)' 481    <conf:(0.76)> lift:(8.05) lev:(0.05) [421] conv:(3.82)
PT08.S1(CO)='(1406.5-inf)' C6H6(GT)='(20.15-inf)' PT08.S2(NMHC)='(1299.5-inf)' 629 ==> CO(GT)='(3.95-inf)' 481    <conf:(0.76)> lift:(8.05) lev:(0.05)
```

Figura 36 - Exemplo de regra redundante

(1) PT08.S1(CO)='(1406.5-inf)' C6H6(GT)='(20.15-inf)' 629 ==> CO(GT)='(3.95-inf)' 481, com confiança de 0.76

(2) PT08.S1(CO)='(1406.5-inf)' C6H6(GT)='(20.15-inf)' PT08.S2(NMHC)='(1299.5-inf)' 629 ==> CO(GT)='(3.95-inf)' 481, com confiança de 0.76

Como podemos verificar temos duas regras em que uma é mais geral (1) e a outra é a mais específica (2), uma vez que contém mais um elemento no antecedente, mantendo o consequente igual. O WEKA indica-nos a frequência do antecedente e do consequente das regras. Assim, como podemos ver, ambas as regras possuem a mesma frequência. Além disso, as duas regras têm também o mesmo valor de confiança (0.96), logo a regra mais específica (2) não adiciona nada de novo ao problema em causa. Deste modo, esta regra é redundante e pode ser descartada, mantendo apenas a regra mais geral (1).

O conjunto total de regras de associação obtidas com o suporte = 0.1 e confiança = 0.95 foi:

1. CO(GT)='(0.95-1.05]' NOx(GT)='(88.5-89.5]' 1249 ==> NO2(GT)='(96.5-97.5]' 1245 <conf:(1)>

2. CO(GT)='(0.95-1.05]' NO2(GT)='(96.5-97.5]' 1258 ==> NOx(GT)='(88.5-89.5]' 1245 <conf:(0.99)>

3. NOx(GT)='(88.5-89.5]' 1680 ==> NO2(GT)='(96.5-97.5]' 1639 <conf:(0.98)>

4. NO2(GT)='(96.5-97.5]' 1720 ==> NOx(GT)='(88.5-89.5]' 1639 <conf:(0.95)>

Como se pode verificar, todas têm uma confiança igual ou superior ao limite definido. Estas são as regras geradas para o caso mais exigente usando como métrica de interesse a confiança (ou *confidence*).

Outra medida de interesse que o WEKA disponibiliza é o *lift*, ou seja, indica a importância de uma regra e pode tomar valores entre 0 e infinito:

- Um valor de *lift* inferior a 1 indica que o antecedente e o consequente aparecem juntos menos vezes do que o esperado, isto é, a ocorrência do antecedente tem um efeito negativo na ocorrência do consequente;

- Um valor de *lift* perto de 1 indica que o antecedente não tem qualquer efeito na ocorrência do consequente;

- Um valor de *lift* superior a 1 indica que a ocorrência do antecedente tem um efeito positivo no consequente.

Assim, pretendemos filtrar as regras geradas de acordo com um valor mínimo de *lift* maior do que 1, pois quanto maior for o seu valor, maior é a probabilidade da ocorrência do antecedente e do consequente juntos não ser apenas uma coincidência, mas sim de existir uma relação entre os dois.

Deste modo, continuamos com as três situações de suporte mínimo: 0.1, 0.05 e 0.01. Para cada uma delas, temos valores de *lift* igual a 3, 4 e 5.

Tabela 8 - Número de regras geradas tendo em conta o *lift* (10 *bins*)

Suporte	0.1			0.05			0.01		
Lift	3	4	5	3	4	5	3	4	5
Número de regras geradas	12	6	6	238	232	226	51418	49802	48212

À semelhança da confiança, quando temos um valor maior para o suporte mínimo menos regras são geradas quando em comparação com o número de regras geradas quando o suporte mínimo é mais baixo. Também verificamos que quando o valor de *lift* aumenta, o número de regras é menor, pois ficamos mais exigentes em relação às regras que queremos que sejam geradas.

Mesmo quando temos um valor de *lift* mínimo superior a 6, não significa que a confiança da regra seja também muito elevada. Por exemplo, quando o suporte é 0.1 e o *lift* é máximo (5.51) temos duas regras distintas com diferentes valores de confiança:

```
1. NOx(GT)='(88.5-89.5]' 1680 ==> CO(GT)='(0.95-1.05]' NO2(GT)='(96.5-97.5]' 1245    conf:(0.74) < lift:(5.51)> lev:(0.11) [1019] conv:(3.34)
2. CO(GT)='(0.95-1.05]' NO2(GT)='(96.5-97.5]' 1258 ==> NOx(GT)='(88.5-89.5]' 1245    conf:(0.99) < lift:(5.51)> lev:(0.11) [1019] conv:(73.72)
```

Figura 37 - Exemplo de regras com *lift* máximo

(1) NOx(GT)='(88.5-89.5]' 1680 ==> CO(GT)='(0.95-1.05]' NO2(GT)='(96.5-97.5]' 1245, com confiança de 0.74

(2) CO(GT)='(0.95-1.05]' NO2(GT)='(96.5-97.5]' 1258 ==> NOx(GT)='(88.5-89.5]' 1245, com confiança de 0.99

Assim, tal como podemos verificar temos as duas regras com o valor de *lift* máximo (5.51), porém não implica que o valor de confiança seja também ele máximo, pois temos uma regra (1) com valor de confiança 0.74 e outra com confiança de 0.99.

Por outro lado, também temos como medida de interesse a convicção (*conviction*), que avalia uma regra de associação como uma verdadeira implicação. Assim, esta medida, que

leva em conta tanto o suporte do antecedente como do consequente, toma o valor 1 se existir uma independência entre os dois. No entanto, caso a ocorrência do antecedente implique a ocorrência do consequente, a convicção toma um valor a tender para +infinito.

Deste modo, para os valores mínimos de suporte 0.1, 0.05 e 0.01, temos como limite mínimo de *conviction* 1.5, 2.5 e 3.5.

Tabela 9 - Número de regras tendo em conta a convicção (10 *bins*)

Suporte	0.1			0.05			0.01		
Conviction	1.5	2.5	3.5	1.5	2.5	3.5	1.5	2.5	3.5
Número de regras geradas	12	9	4	238	132	76	39002	27955	24718

Tal como podemos verificar através dos dados contidos na tabela, à medida que o suporte diminui, o número de regras geradas aumenta. Contudo, sempre que o valor de *conviction* aumenta (queremos regras mais significativas), o número de regras mineradas diminui.

Assim, para o caso em que temos um suporte mínimo de 1.5 e um valor para a convicção mínima de 1.5, obtemos 238 regras de associação, sendo que a seguinte é uma das regras geradas:

```
CO(GT)='(-inf-0.75]' 1057 ==> PT08.S3(NOx)='(1146.5-inf)' 470    conf:(0.44) lift:(4.44) lev:(0.04) [364] < conv:(1.62)>
```

Figura 38 - Exemplo de regra tendo em conta a convicção

CO(GT)='(-inf-0.75]' 1057 ==> PT08.S3(NOx)='(1146.5-inf)' 470, com *conviction* de 1.62

A regra acima tem uma *conviction* de 1.62, ou seja, este valor indica que a probabilidade de ocorrência da concentração do gás CO ser inferior ou igual a 0.75 acontecer sem que a ocorrência do da concentração do gás NOx ser maior ou igual a 1146.5 é 1.62 vezes menor do que o esperado.

Por fim, temos outra medida de interesse chamada *leverage*, que, por sua vez, varia entre -0.25 e 0.25 e mede o número de casos extra obtidos em relação ao esperado.

Deste modo, para os três valores de suporte mínimo 0.1, 0.05 e 0.01, temos o valor mínimo de *leverage* como sendo 0.06, 0.1 e 0.14.

Tabela 10 - Número de regras geradas tendo em conta a *leverage* (10 *bins*)

Suporte	0.1			0.05			0.01		
Leverage	0.06	0.1	0.14	0.06	0.1	0.14	0.06	0.1	0.14
Número de regras geradas	12	6	2	36	6	2	36	6	2

Assim, tal como se pode analisar, para os dois primeiros valores de *leverage* temos que o número de regras geradas aumenta quando o *leverage* mínimo é de 0.06. No entanto, não se verifica nenhuma mudança para os outros valores desta medida de interesse. O mesmo acontece quando se compara os resultados para o suporte de 0.05 e 0.01 – não ocorre um aumento de regras geradas.

5.1.2 Discretização com 5 intervalos

Dividindo os atributos em cinco intervalos com uma frequência semelhante, sempre que possível, testamos três casos de suporte: 0.1, 0.05 e 0.01. Para estes, temos, então, também três situações diferentes em que a confiança toma os valores 0.75, 0.85 e 0.95.

Tabela 11 - Número de regras tendo em conta a confiança (5 *bins*)

Suporte	0.1			0.05			0.01		
Confiança	0.75	0.85	0.95	0.75	0.85	0.95	0.75	0.85	0.95
Número de regras geradas	393	195	57	3904	2167	587	464396	317162	168265

Como podemos verificar, quando o valor de suporte mínimo aumenta e a confiança também aumenta, o número de regras geradas diminui. Quando o suporte é de 0.1, temos um número significativamente menor do que quando o suporte é de 0.01, pois, neste último caso, estamos a considerar mais itens candidatos para gerar regras.

Para as restantes medidas de interesse, já não consideramos o suporte de 0.01, pois havia um número muito elevado de itens a serem considerados e regras a serem geradas, logo o tempo de execução do programa não era apropriado para testar as diversas situações. Desta maneira, consideramos os seguintes valores: 0.1, 0.075 e 0.05, ou seja, estamos a limitar o número de itens frequentes a serem considerados, pois o valor de suporte mínimo é maior. Para estes três casos, temos os valores de *lift* 3, 4 e 5.

Tabela 12 - Número de regras tendo em conta o *lift* (5 *bins*)

Suporte	0.1			0.075			0.05		
Lift	3	4	5	3	4	5	3	4	5
Número de regras geradas	864	622	160	3782	3144	1384	12484	10918	6208

Neste caso, com um suporte mínimo mais permissivo (0.05), temos um número maior de regras a serem geradas em comparação com um suporte mais rigoroso. O mesmo acontece quando temos um valor de *lift* menor, onde há menos regras geradas.

Para a medida de interesse convicção, temos que para os suportes 0.1, 0.075 e 0.05 consideramos os valores de *conviction* 1.5, 2.5 e 3.5.

Tabela 13 - Número de regras tendo em conta a *conviction* (5 *bins*)

Suporte	0.1			0.075			0.05		
Conviction	1.5	2.5	3.5	1.5	2.5	3.5	1.5	2.5	3.5
Número de regras geradas	919	567	363	3789	2079	1427	11076	5780	3792

As conclusões da medida de interesse *conviction* são as mesmas que as da medida anterior, *lift*.

Tabela 14 - Número de regras geradas tendo em conta a *leverage* (5 *bins*)

Suporte	0.1			0.05			0.01		
Leverage	0.06	0.1	0.14	0.06	0.1	0.14	0.06	0.1	0.14
Número de regras geradas	914	154	6	3700	154	6	3700	154	6

Por outro lado, quando o valor de *leverage* é 0.1 e 0.14 (um valor mais alto), temos o mesmo número de regras a serem geradas, independentemente do valor de suporte escolhido. No entanto, quando temos o valor de suporte mais elevado (0.1) existe um menor número de regras geradas quando comparadas com os outros valores de suporte.

5.2. Resultados sobre as Regras de Associação

Nas regras de associação foram considerados dois casos em que os dados estavam organizados de uma maneira distinta: divididos em 5 intervalos e divididos em 10 intervalos. Esta divisão foi realizada com o intuito de testar quantas regras eram geradas com cada uma das discretizações e como seriam as suas regras.

Ora, chegou-se à conclusão que uma divisão dos valores dos atributos em 5 intervalos leva a que sejam considerados intervalos demasiado extensos, o que pode permitir uma grande variabilidade de concentrações de gases num só intervalo. Assim, uma regra gerada teria em conta um intervalo cuja gama demasiado extensa levaria a que a sua precisão não fosse muito boa. Como o número de intervalos é menor, é normal que estes apareçam mais frequentemente, logo o seu suporte possui valores mais elevados. Deste modo, mesmo para um suporte mais alto (0.1) são considerados muitos *itens* frequentes, o que leva, claramente, a um maior número de regras a serem geradas. Tal como se pode verificar no caso mais exigente da confiança (suporte=0.1 e confiança=0.95) são geradas 57 regras. Em contraste, quando os valores dos atributos são divididos em 10 intervalos, apenas são geradas 4 regras para os mesmos valores de suporte e confiança. Como estes intervalos são menos extensos,

englobam um menor número de valores de concentração de gases, logo as regras são mais específicas e corretas.

Desta maneira, chegou-se à conclusão que o número mais acertado para se fazer a discretização era de 10 intervalos, uma vez que as mesmas conclusões retiradas para a confiança se aplicam às restantes medidas de interesse.

5.3. Recomendações sobre as Regras de Associação

Escolhendo como medida de interesse a confiança, temos que o caso mais exigente é o de suporte igual a 0.1 e valor mínimo de confiança de 0.95. Assim, como já foi mostrado obtemos as seguintes regras:

1. CO(GT)='(0.95-1.05]' NOx(GT)='(88.5-89.5]' 1249 ==> NO2(GT)='(96.5-97.5]' 1245 <conf:(1)>
2. CO(GT)='(0.95-1.05]' NO2(GT)='(96.5-97.5]' 1258 ==> NOx(GT)='(88.5-89.5]' 1245 <conf:(0.99)>
3. NOx(GT)='(88.5-89.5]' 1680 ==> NO2(GT)='(96.5-97.5]' 1639 <conf:(0.98)>
4. NO2(GT)='(96.5-97.5]' 1720 ==> NOx(GT)='(88.5-89.5]' 1639 <conf:(0.95)>

Como se pode verificar as duas últimas regras implicam a mesma coisa, logo temos a certeza que a presença de um dos dois gases NOx e NO2 naquelas quantidades, implicam necessariamente a presença do outro na quantidade acima indicada. Deste modo, temos uma implicação nos dois sentidos, o que pode ser útil para os analistas, pois sabem que na presença de um gás, temos também o outro. Deste modo, podem tomar as precauções necessárias.

Como elemento comum nas duas primeiras regras temos o gás CO (monóxido de carbono), que na presença do gás NOx numa certa quantidade pode implica a presença do gás NO2 e vice-versa. Deste modo, basta aos investigadores detetarem a presença de dois destes três gases para saberem que o terceiro gás também está presente na concentração indicada pelas regras.

Em relação às medidas de interesse existem quatro que podemos selecionar de modo a organizar as regras de associação geradas. O conjunto mais frequente costuma ser suporte + confiança ou *lift*. Normalmente, a confiança é uma medida de interesse boa, porém pode ser enganosa na geração de regras, uma vez que não leva em conta a independência dos atributos. Aí podemos escolher o *lift* como a nova medida. No entanto, mesmo assim existem a *leverage* e a *conviction* que tentam compensar as falhas das outras duas medidas anteriormente mencionadas. Deste modo, a análise das regras de associação deve ser feita tendo em conta todas estas medidas e a sua escolha deve ter em conta o contexto da área de negócio.

6. Conclusões

Neste trabalho apresentamos uma visão diferentes sobre três tipos de aprendizagem diferente: a classificação, a segmentação e as regras de associação, analisando diferentes algoritmos de aprendizagem e modos de treino dos dados. Todos estes parâmetros foram avaliados entre dois *datasets* diferentes.

Antes de avançar para o processamento dos dados, qualquer que fosse o tipo de aprendizagem, foi necessário avaliar bem os *datasets*, analisando de forma detalhada os atributos e concluindo se seria possível introduzir um pré-processamento dos dados (tal com a discretização e a normalização dos dados), de modo a serem melhor aceites pelos diferentes tipos de aprendizagem.

Os três tipos de aprendizagem apresentam características diferentes, mas o mesmo objetivo final: caracterizar o *dataset* através de um modelo. Relativamente ao *dataset* referente à qualidade do vinho, a classificação obteve melhores resultados do que a segmentação, ou seja, conseguiu criar um modelo que melhor caracteriza os dados, visto que os valores de erro eram inferiores. Como este *dataset* apresentava variância enormes de dados e a segmentação caracteriza-se por tentar agrupar dados semelhantes nos mesmos conjuntos, este *dataset* apresentou piores resultados que a classificação, que não apresenta esse problema e cujo objetivo principal passa por, após construído o modelo, ser capaz de prever novos casos corretamente. Relativamente às regras de associação, conseguimos descobrir relações entre os determinados atributos do *dataset* relativo à qualidade do ar numa cidade italiana e, também conseguimos descobrir, por exemplo, quais dessas regras eram as mais confiáveis e quais surgiam em maior frequência

Em suma, como referido previamente, não há uma fórmula definida que indique qual o melhor tipo de aprendizagem, sendo necessário testar os vários tipos, de modo a obter resultados passíveis de serem comparados entre si.