Universidade de São Paulo
SCC - 230 Inteligência Artificial
Profª Solange Oliveira Rezende

Mineração de Dados e Aprendizado de Máquina na base Wine Quality

8598861 - Bernardo Simões Lage G. Duarte
8122585 - Eder Rosati Ribeiro
8936993 - Gabriel Luiz Ferraz Souto
8936648 - Giovani Ortolani Barbosa
8531887 - Giovanni Robira
8937271 - Rafael Bueno da Silva

São Carlos - SP
Novembro de 2017

1) Introdução

No ramo de Mineração de Dados e Aprendizado de Máquina pode-se extrair diversas informações relevantes de grandes bases de dados, além disso é possível o aprendizado de padrões e comportamentos de objetos dessas bases de dados. A partir desse conhecimento a máquina consegue prever resultados sem que haja interferência humana no processo.

O objetivo do trabalho é, usando abordagens de Aprendizado de Máquina, prever a qualidade de um vinho através de informações referentes à sua composição, fazendo uma escolha de metodologia de aprendizado supervisionado que seja mais adequada para o problema em questão. Utilizaremos a ferramenta *Weka* bem como as bibliotecas *scikit-learn*, *pandas* e *matplotlib* com a linguagem *Python* como auxílio nos processos de Mineração/Aprendizado de Máquina e de pré-processamento.

Existem diversas possibilidades de aplicação real do conhecimento obtido, seja em: aplicativos para pessoas leigas que recebem como entrada algumas características do vinho e conseguem realizar a predição de sua qualidade; automatização do processo de controle de qualidade em vinícolas entre outros. A mesma ideia pode ser estendida para outros tipos de alimentos e bebidas.

2) Mudanças em relação às entregas anteriores

Nas entregas passadas foi decidido utilizar as 2 bases de dados, uma referente à vinhos brancos e outra referente à vinhos tintos. Por fim, apenas a base de vinhos brancos foi utilizada, pois é a que apresenta maior número de instâncias e também para facilitar a sumarização dos resultados.

Em relação à fase de pré-processamento ocorreram as seguintes mudanças.

Primeiramente, todos os atributos exceto a classe (qualidade) foram normalizados a fim de que escalas diferentes não interfiram em algoritmos que são sensíveis à diferentes escalas. Em seguida, as classes foram discretizadas em 3 grupos, que não apresentaram resultados satisfatórios e, por esse motivo, foram discretizadas em apenas 2 grupos. Um maior enfoque ao motivo de tal discretização será dado na próxima seção.

3) Pré-processamento

Nota: as etapas de discretização que serão referidas abaixo foram realizadas pelo *script* preproc.py que se encontra no diretório do projeto.

A base possui rótulos (qualidade do vinho) variando de 0 a 10, porém alguns desses valores não aparecem na base e há um grande desbalanceamento entre as classes como é

mostrado na Figura 1. O menor e maior valor de qualidade são, respetivamente, 3 e 9. As classes mais frequentes são a 5, 6 e 7.

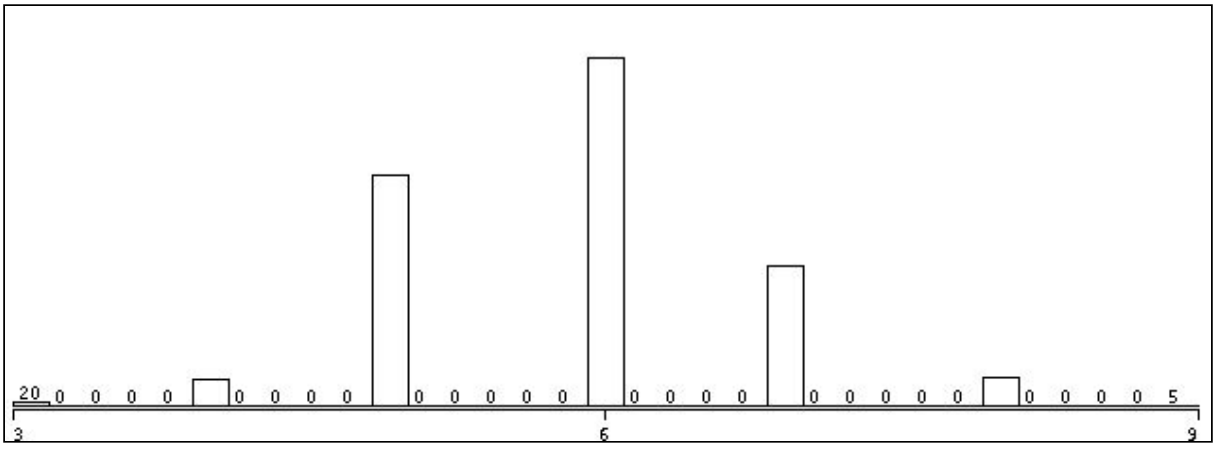

Figura 1 - histograma da distribuição das classes na base de vinhos brancos

Para contornar o problema referido no parágrafo anterior a qualidade foi discretizada da seguinte maneira: intervalo [0-4] = *ruim*, intervalo [5-7] = *médio* e intervalo [8-10] = *bom*. A Figura 2 apresenta a quantidade de instâncias em cada classe.

Label	Count
medio	4535
bom	180
ruim	183

Figura 2 - número de instâncias em cada classe após a discretização em 3 grupos

Mesmo após a discretização referida ainda assim as classes continuam altamente desbalanceadas e métodos de *resampling* tanto para igualar as classes minoritárias com a majoritária quanto para o inverso não apresentaram resultados satisfatórios, uma vez que ocorreu o *underfitting* e o *overfitting*, respetivamente.

A fim de exemplificar o *overfitting*, foi feito o *resampling* como indicado na Figura 3.

Label	Count
medio	1632
bom	1632
ruim	1632

Figura 3 - número de instâncias em cada classe após a discretização em 3 grupos com resampling

Através de teste rápidos com algoritmo *kNN* com 5 vizinhos apenas para verificar se o *resampling* é eficaz a acurácia obtida foi de 96.97% e a respectiva matriz de confusão é:

```
   a    b    c   <-- classified as
1484   78   70 |    a = medio
   0 1632    0 |    b = bom
   0    0 1632 |    c = ruim
```

Podemos perceber que como houve uma grande replicação de um pequeno grupo de 180 instâncias no caso das classes *bom* e *ruim* ocorreu o *overfitting*, pois o algoritmo conseguiu acertar 100% dos casos nessas classes.

Por fim, a solução encontrada foi discretizar os rótulos em apenas 2 classes, sendo elas *bom* e *ruim*. A discretização foi realizada da seguinte maneira: intervalo [0-6] = *ruim*, intervalo [7-10] = *bom*. A Figura 4 apresenta a quantidade de instâncias em cada classe.

Label	Count
ruim	3838
bom	1060

Figura 4 - número de instâncias em cada classe após a discretização em 2 grupos

Pode-se perceber que as classes estão melhor balanceadas após a discretização em 2 grupos e também apresentaram resultados em relação à *overfitting* e *underfitting*, mesmo tendo acurácias menores. Tais resultados serão apresentados na seção subsequente.

4) Mineração de Dados e Aprendizado de Máquina

Anteriormente à etapa de Aprendizado, calculamos a matriz de correlação da base de dados para verificar quais pares de atributos estavam mais correlacionados com outros. A matriz de correlação é apresentada na Figura 5 por meio de um *heat map*. O *heat mat* foi gerado por um *script* nomeado corr.py que se encontra no diretório do projeto.

É notável a forte correlação positiva entre o par de atributos (*density, residual sugar*) bem como uma forte correlação negativa entre o par (*alcohol, density*). Tal fato poderia ser utilizado para remover um atributo dentro de cada um desse pares, pois trazem informação redundante. Porém, optamos por deixá-los na base.

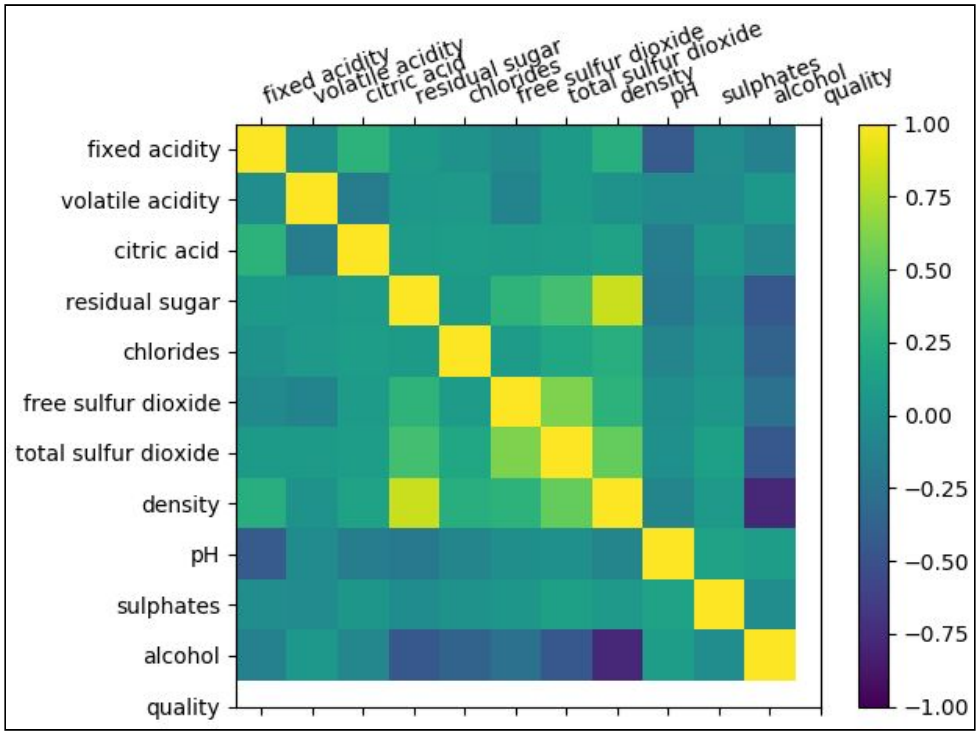

Figura 5 - matriz de correlação dos atributos da base (goo.gl/23k7AV)

Durante a etapa de Aprendizado de Máquina, utilizamos diferentes algoritmos de classificação com intuito de verificar com possui melhor desempenho frente aos dados utilizados, alguns deles são do tipo caixa-preta como os que utilizam redes neurais (SVM e MLP). Eles são: *k-Nearest-Neighbors (kNN)*, *Multi-Layer Perceptron (MLP), Support Vector Machines (SVM)*, Árvore de Decisão, *Naive Bayes* e Regra de Associação.

Em todos os testes foi utilizada a validação cruzada com 10 *folds* para computar as medidas de eficácia dos algoritmos.

Com o intuito de verificar se o balanceamento (Figura 4) considerado no fim da seção de pré-processamento possuía desempenho satisfatório, aplicamos o *kNN* com 5 vizinhos nestes dados e também aplicamos a mesma configuração após realizar o balanceamento das classes (Figura 6) para possuírem o mesmo número de instâncias (algumas instâncias da classe minoritária foram replicadas e algumas instâncias da classe majoritária foram removidas aleatoriamente). Os resultados são apresentados a seguir.

- Utilizando o balanceamento apresentado na Figura 4

Acurácia: 87.34%
Matriz de confusão:

```
   a    b   <-- classified as
3561  277 |    a = ruim
 343  717 |    b = bom
```

TP Rate	FP Rate	Class
0,928	0,324	ruim
0,676	0,072	bom
0,873	0,269	

- Utilizando o balanceamento que iguala o número de instâncias em cada classe

Label	Count
ruim	2448
bom	2448

Figura 6 - número de instâncias em cada classe após o balanceamento

Acurácia: 91.79%
Matriz de confusão:

```
   a    b   <-- classified as
2106  343 |    a = ruim
  59 2390 |    b = bom
```

TP Rate	FP Rate	Class
0,860	0,024	ruim
0,976	0,140	bom
0,918	0,082	

Portanto, percebemos que a segunda abordagem de balanceamento aparenta ser melhor e por esse motivo escolhemos para ser aplicada em todos os testes a seguir, pois além de possuir melhor acurácia também apresenta uma alta taxa de verdadeiros positivos para cada uma das classes. A primeira abordagem possui uma acurácia razoável, porém deixa a desejar na taxa de verdadeiros positivos para a classe *bom*.

4.1) kNN

Abaixo são apresentados os resultados utilizando k-vizinhos - 3, 5 e 7. A medida de distância utilizada foi o inverso da distância, pois evita empates e apresentou uma melhor acurácia em testes rápidos (tais testes não foram incluídos no relatório pois acreditamos não serem de grande relevância).

- k = 3

Acurácia: 92.46%
Matriz de confusão:

```
   a    b   <-- classified as
2134  314 |    a = ruim
  55 2393 |    b = bom
```

	TP Rate	FP Rate	Class
	0,872	0,022	ruim
	0,978	0,128	bom
Weighted Avg.	0,925	0,075	

- k = 5

Acurácia: 92.12%
Matriz de confusão:

```
   a    b   <-- classified as
2115  333 |    a = ruim
  53 2395 |    b = bom
```

	TP Rate	FP Rate	Class
	0,864	0,022	ruim
	0,978	0,136	bom
Weighted Avg.	0,921	0,079	

- k = 7

Acurácia: 92.03%
Matriz de confusão:

```
   a    b   <-- classified as
2104  344 |    a = ruim
  46 2402 |    b = bom
```

	TP Rate	FP Rate	Class
	0,859	0,019	ruim
	0,981	0,141	bom
Weighted Avg.	0,920	0,080	

Podemos concluir que o melhor resultado obtido foi através da utilização de k = 3.

4.2) Naive Bayes

Acurácia: 70.18%
Matriz de confusão:

```
   a    b   <-- classified as
1467  981 |    a = ruim
 479 1969 |    b = bom
```

	TP Rate	FP Rate	Class
	0,599	0,196	ruim
	0,804	0,401	bom
Weighted Avg.	0,702	0,298	

O resultado da aplicação do algoritmo *Naive Bayes* gerou um classificador com desempenho não satisfatório. Acreditamos que o mesmo pode ser melhorado através de uma melhor escolha dos parâmetros de entrada, porém como não temos uma base teórica sólida em relação ao método, decidimos manter o padrão de parâmetros do *Weka*.

4.3) MLP

Através da configuração padrão do *Weka* e com 10 neurônios na camada oculta, obtivemos o resultado abaixo.

Acurácia: 80.15%
Matriz de confusão:

```
   a    b   <-- classified as
1778  670 |    a = ruim
 302 2146 |    b = bom
```

	TP Rate	FP Rate	Class
	0,726	0,123	ruim
	0,877	0,274	bom
Weighted Avg.	0,801	0,199	

O método *Multi-Layer Perceptron* apresenta desempenho satisfatório, porém não ótimo, para a base de dados. Acreditamos que assim como o método *Naive Bayes*, o desempenho poderá ser melhorado através de uma configuração diferente de parâmetros de entrada.

4.4) SVM

Através da configuração padrão do *Weka*, obtivemos o resultado abaixo.

Acurácia: 73.51%
Matriz de confusão:

```
   a    b   <-- classified as
1670  778 |    a = ruim
 519 1929 |    b = bom
```

	TP Rate	FP Rate	Class
	0,682	0,212	ruim
	0,788	0,318	bom
Weighted Avg.	0,735	0,265	

A utilização de *SVM* não apresenta resultados satisfatório, assim como dito anteriormente, acreditamos que o desempenho poderá ser melhorado através de uma configuração diferente de parâmetros de entrada.

4.5) Árvore de Decisão (algoritmo J48)

Através da configuração padrão do *Weka*, obtivemos o resultado abaixo.

Acurácia: 91.05%
Matriz de confusão:

```
   a    b   <-- classified as
2156  292 |    a = ruim
 146 2302 |    b = bom
```

	TP Rate	FP Rate	Class
	0,881	0,060	ruim
	0,940	0,119	bom
Weighted Avg.	0,911	0,089	

A árvore gerada possui 285 nós-folha e também apresenta um desempenho bom em relação à classificação. A esquematização da árvore do classificador não será colocada no relatório por justificativa de espaço.

4.6) Regra de Associação (algoritmo JRip)

Através da configuração padrão do *Weka*, obtivemos o resultado abaixo.

Acurácia: 86.85%
Matriz de confusão:

```
   a    b   <-- classified as                    TP Rate  FP Rate  Class
2043  405 |    a = ruim                          0,835    0,098    ruim
 239 2209 |    b = bom                           0,902    0,165    bom
                                  Weighted Avg.  0,868    0,132
```

Um exemplo de regra gerada é:

```
(residual sugar >= 0.025806) and (alcohol >= 0.774194)
 and (density <= 0.114273) => quality=bom (129.0/12.0)
```

Ao todo foram geradas 60 regras de associação. Ficamos surpresos por obter uma acurácia alta utilizando este tipo de algoritmo para um problema cujos atributos possuem apenas valores numéricos. Também acreditamos que o resultado pode ser melhorado com a variação dos parâmetros de entrada do *Weka* bem como a aplicação de um pré-processamento que favoreça o algoritmo de decisão.

5) Problemas e soluções

Alguns empecilhos encontrados pelo grupo já foram discutidos nas seções 2 e 3, referindo à tarefa de normalização dos resultados a fim de não afetar alguns algoritmos sensíveis à escala e também ao balanceamento das classes.

Durante a fase de Mineração/Aprendizado notamos que os algoritmos possuem diversos parâmetros de entrada, e em alguns casos nem sempre tínhamos embasamento teórico para saber quais parâmetros e quais valores seriam adequados, pois não aprendemos em aula. Com o intuito de contornar o problema, decidimos deixar os parâmetros padrão que são providos pelo *Weka*.

6) Experimentos e resultados

A Tabela 1 contém a sumarização dos resultados obtidos na seção 4 ordem decrescente de acurácia.

Tabela 1 - método e acurácia do método na classificação da base

Método	Acurácia (%)
kNN - 3 vizinhos	92.46
kNN - 5 vizinhos	92.12
kNN - 7 vizinhos	92.03
Árvore de Decisão (*J48*)	91.05
Regra de Associação (*JRip*)	86.85
MLP	80.15
Naive Bayes	73.51
SVM	70.18

Portanto, o melhor método classificador é o *kNN* considerando 3 vizinhos (porém qualquer variação no número de vizinhos ainda gera um bom classificador), seguido de perto pela Árvore de Decisão. A acurácia dos métodos baseados em redes neurais ficou abaixo da expectativa esperada, pois são métodos que conseguem tratar de problemas não-linearmente separáveis que é o caso da base de dados utilizada.

Ao fazermos uma análise geral, todos os algoritmos apresentaram acurácia acima de 50%, ou seja, melhor que um classificador aleatório. Acreditamos que todos eles poderiam ser melhorados de alguma forma, seja por diferentes técnicas de pré-processamento ou de alteração dos parâmetros dos algoritmos de classificação.

7) Conclusão

Logo, o melhor algoritmo que poderia ser utilizado em um aplicativo de classificação de vinhos é o *kNN*. Por ser um algoritmo que suporta aprendizado incremental e relativamente rápido na tarefa de classificação, ele poderia ser implementado em um servidor que seria responsável por processar as consultas dos usuários. Deve-se atentar para sua necessidade de espaço de armazenamento, pois não realiza um forte processamento na etapa de treino. Caso a visualização do resultado seja necessária, o algoritmo de Árvores de Decisão poderá ser aplicando, pois é próximo em acurácia do *kNN*.

Através de realização do Projeto o grupo pode perceber que extrair padrões e ensinar à máquina a descobrir informação em uma base de dados grande não é uma tarefa trivial. Existe uma gama de tarefas a serem feitas de modo a deixar a base pronta para a tarefa de Mineração/Aprendizado, seja realizando o pré-processamento junto às diversas possibilidades

de filtragem dos dados, seja experimentando os diversos algoritmos e seus diversos parâmetros para que as saídas sejam as melhores possíveis.