ACH2016 Inteligência Artificial
Exercício Programa 2 (EP2)
Classificação

Alunos:
Denise Keiko Ferreira Adati - 10430962
Fernando Karchiloff Gouveia de Amorim - 10387644
Gabriela Brindo Domingues - 9788030

São Paulo
2019

Introdução

Como tarefa aos alunos da matéria ACH2016 - Inteligência Artificial, como exercício programa 2 (EP2), era necessário utilizar um dos *datasets* propostos pela professora para desenvolver classificadores estudados durantes as aulas e fazer diversas comparações entre eles de acordo com as exigências do enunciado, utilizando diferentes parâmetros para cada um dos classificadores e observar como se comportam de acordo com as modificações feitas. Os resultados dos modelos devem ser comparados e analisados para determinar qual o classificador com melhor robustez para o *dataset* escolhido.

Weka 3.8.3

O software utilizado para executar as tarefas que foram propostas pela professora no exercício programa foi o *Weka 3.8.3*. Desenvolvido na Universidade de Waikato, em Hamilton - Nova Zelândia, o *Waikato Environment for Knowledge Analysis (Weka)*, foi criado para desempenhar análises estatísticas, produção de inteligências artificiais e aprendizado de máquina, ele foi escrito na linguagem de programação *Java*, seu código é aberto ao público para quem gostaria de melhorar o programa.

O computador utilizado para a execução dos algoritmos no software foi um notebook DELL com processador i5 com 8 núcleos, 8GB de memória RAM, 1TB de memória de armazenamento de disco rígido com o sistema operacional Windows 10 e uma JVM versão 1.8_201.

Conjunto de treinamento e teste

Os algoritmos e testes feitos a seguir utilizam um *dataset* que indica se um email é *spam* ou não baseado em 57 atributos. O número total de instâncias é de 4.601, o Conjunto de Treinamento foi constituído por 70% dos dados, o que equivale a 3220 instâncias e o Conjunto de Teste foi constituído por 30% dos dados, o que equivale a 1381 instâncias. Essa quantidade se deve a função *Supervised Resample* do software *Weka*, localizada na parte de *Filters* em *Preprocess* dos dados. Essa função seleciona aleatoriamente e proporcionalmente de acordo com as classes, uma certa porcentagem passada por parâmetro, determinada por nós, dos dados e separa em um novo subconjunto, ele busca as instâncias considerando a porcentagem passada. Seu cálculo para obter a porcentagem desejada ignora o valor decimal (70% de 4601 = 3220,7) resultando na separação baseada nesse número. Para aplicar a mesma, é acessado através do menu *Explorer* do *Weka*, abre o arquivo *spambase.data* para acessar o *dataset* inteiro de *spam* com *Open File*, então selecione em *Filter*, na parte *Supervised*, a função *Supervised Resample*, clique na linha da função para abrir uma caixa de diálogo que permite alteração de configurações disponíveis para a mesma, insira a quantidade de dados que você gostaria na porcentagem, e então dê *Apply* para aplicar no *dataset* o filtro, podendo assim salvar posteriormente o *dataset* em *.arff*.

Para a execução dos algoritmos, salvamos diversos *datasets* de treinamento e teste para cada parte, utilizando ruído, padronização, discretização e etc. Garantindo que sejam de acesso mais fácil.

Execução dos Algoritmos

Cada classificador foi treinado com um conjunto de treinamento, escolhido na aba "*Preprocess" do Weka,* e utilizado no conjunto de testes, escolhido em "*Supplied test set*" na aba "*Classify*". Sendo que os conjuntos eram padronizados ou não e discretizados ou não, de acordo com o enunciado e as necessidades na implementação do algoritmo.

Padronização

O enunciado do exercício programa exigia diversas vezes que seja feita uma padronização para comparar o resultado dos classificadores, rodando sem a padronização e com a padronização. A padronização é o procedimento de tornar a média e desvio-padrão de um ou mais atributos, excluindo a classe, respectivamente 0 e 1, reduzindo o espaço entre as instâncias dos atributos e tentando garantir uma melhor distribuição, muitas vezes é uma distribuição Normal(0,1). Para ser possível fazer isso no *Weka*, utilizamos a função *Unsupervised Standardize*, ela faz a padronização dos atributos passados como parâmetros, ela se encontra na parte de *Preprocess* de dados do *Weka*, em *Filters* a serem aplicados a *datasets*. Não são necessários outros parâmetros além dos atributos que se querem padronizar daquele *dataset*.

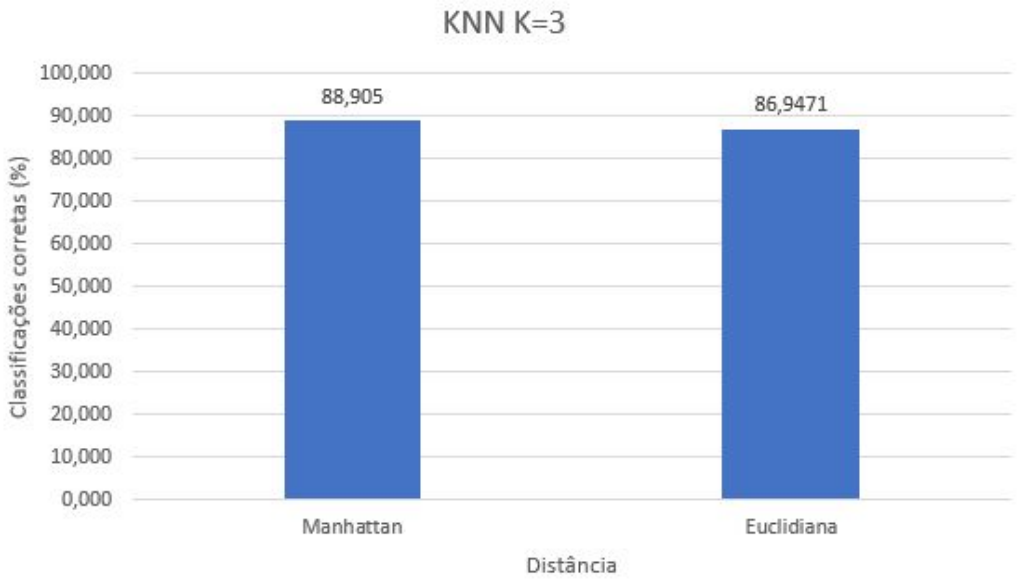
Algo importante a ser notado é que, para utilizar um algoritmo com um conjunto de treinamento padronizado, o seu conjunto de teste também **deve estar padronizado** para que não ocorram problemas na classificação das instâncias.

KNN

Executamos a implementação do algoritmo *K-Nearest Neighbors* do software *Weka* denominado *lazy IBk*.

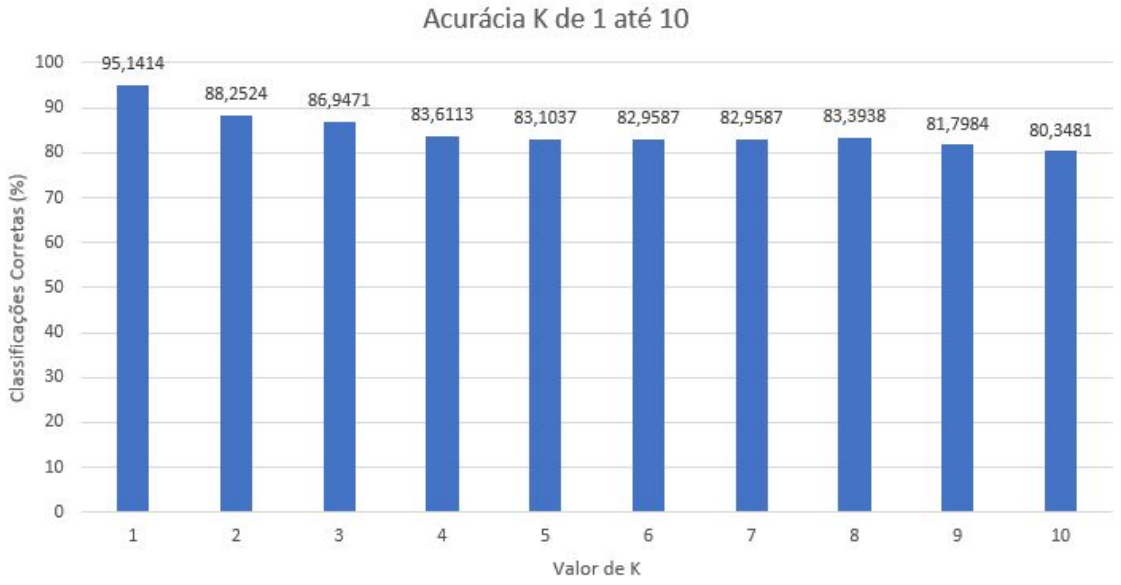
Primeiramente executamos o algoritmo para os valores de k = 3 sem padronizar os dados para as o cálculo das distâncias, sendo elas Euclidiana e Manhattan. A opção *dontNormalize* é alterada para True, para que os dados não sejam automaticamente normalizados.

Obtivemos os seguintes resultados com relação a sua acurácia:



Percebemos que a distância de Manhattan obteve uma acurácia ligeiramente maior com relação à distância Euclidiana. Isso se deve a distância de Manhattan ser menos sensível a outliers em comparação a distância Euclidiana

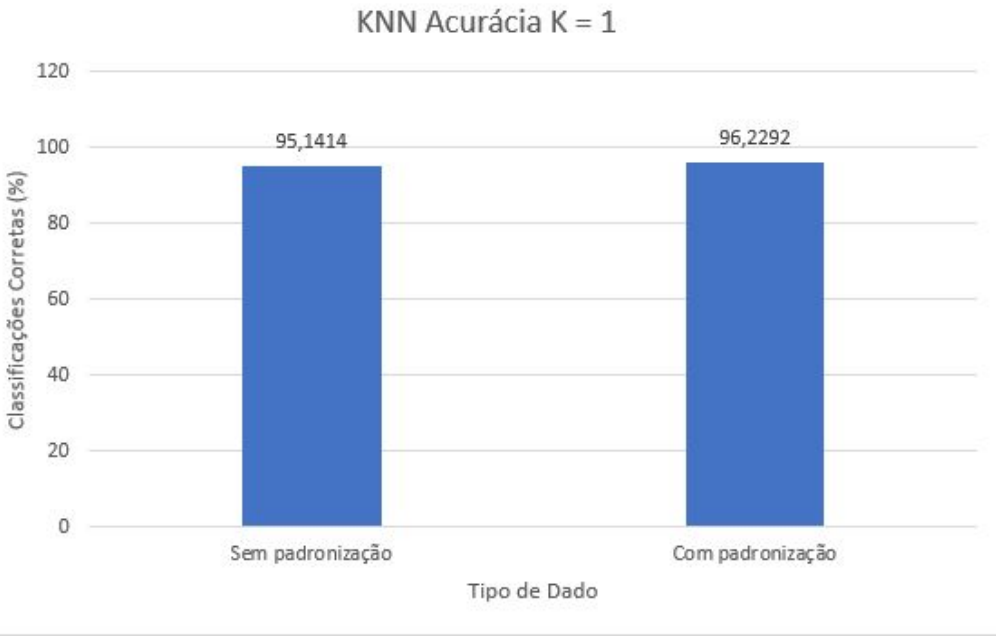
Logo após, testamos o Algoritmo para valores de K entre 1 e 10, ainda sem padronizar e utilizando a distância euclidiana. O gráfico Acurácia versus Valor de K é mostrado abaixo:



Percebemos que a acurácia diminui conforme o valor de K aumenta, os resultados variam por cerca de 95% e 80%. Ela chega a ficar estável entre os valores de K de 4 a 8, flutuando por volta de 83% e 82 %, mas uns números a frente ela volta a cair.

A melhor acurácia encontrada foi para o valor de K = 1, chegando a classificar 95,1414% das instâncias corretamente. Um K de valor tão pequeno pode trazer uma boa acurácia, porém torna o algoritmo muito sensível a outliers e pode indicar overfitting.

Por fim, testamos o mesmo algoritmo com o valor de K para a melhor acurácia encontrada anteriormente(K = 1). Desta vez, os dados foram padronizados e obtivemos o resultado abaixo:



Podemos notar que, para este caso os dados padronizados resultaram em uma melhor acurácia. Isso se deve a padronização minimizar possíveis *outliers* dos dados, e o algoritmo KNN é muito sensível a esse tipo dado, principalmente com um valor de K tão pequeno.

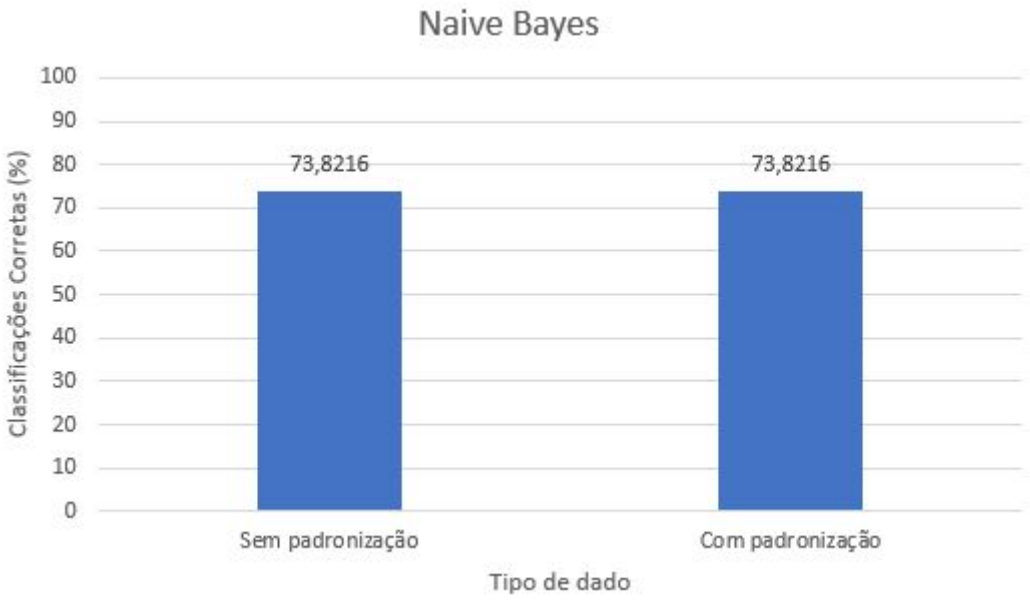
Naive Bayes

Executamos a implementação do algoritmo Naive Bayes do software *Weka* denominado *NaiveBayes*. Esse algoritmo utiliza probabilidade e dados discretos para a sua classificação, e considera os atributos como independentes.

Então, primeiramente, os atributos de valores contínuos foram discretizados para ser possível o cálculo das probabilidades e executar o algoritmo, o exercício programa exigia que para cada atributo, fosse pego o intervalo definido pelo valor mínimo e máximo do mesmo e o dividisse em 6 intervalos de tamanho aproximadamente iguais para que a discretização fosse feita. Para executar tal procedimento, foi utilizado a função *Unsupervised Discretize* que se encontravam nos Filtros do *Weka* para fazer a separação, na função foram escolhidos 6 *Bins* (que seriam a quantidade de intervalos) e o programa utilizaria os valores de máximo e mínimo para colocar as instâncias dentro de cada um destes intervalos que correspondem ao seu respectivo valor contínuo. Com isso, o conjunto de treinamento e teste estão prontos para utilização após a aplicação desta filtragem.

Após a discretização, no *Weka,* o classificador foi treinado com um conjunto de treinamento sem padronização e depois com padronização.

Com isso, obtivemos os seguinte resultados com relação à acurácia:



Percebemos que a acurácia é igual a 73.8216% com os dados sem padronização ou com, o que mostra que padronizar os dados não interferiu na qualidade de classificação do algoritmo Naive Bayes. Isso se deve ao fato de que o Naive Bayes considera que os atributos são independentes, e a padronização não interfere na dependência dos atributos.

Além disso, Naive Bayes é um classificador probabilístico, e, com a padronização, os valores de média e variância se alteram, mas as probabilidades continuam iguais.

Árvore de Decisão

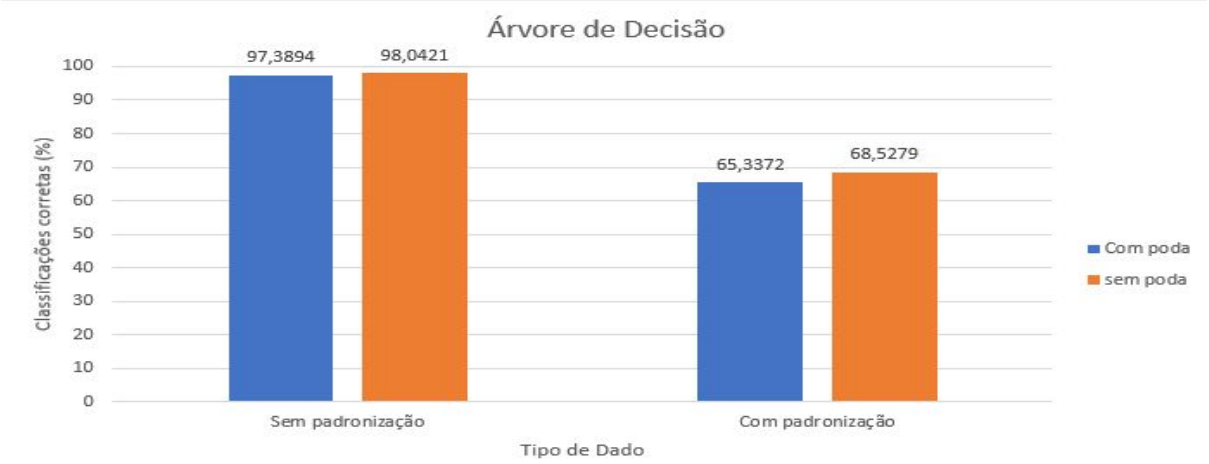
Executamos a implementação do algoritmo de Árvores de Decisão no software *Weka*, ele se encontra na parte de "*Classify*" selecionando na pasta de *trees* usando o algoritmo *J48* das Árvores de Decisão.

Para esta parte do trabalho, o enunciado ressalta a necessidade de fazermos uma execução do algoritmo com os dados não padronizados e outra execução com os dados padronizados da forma descrita na seção "Padronização" deste documento.

O algoritmo *J48*, versão melhorada do algoritmo *C4.5*, é um dos melhores para a montagem de árvores de decisão, ele aplica a técnica de dividir para conquistar, repartindo o problema em subproblemas para ser possível calcular os ganhos de informação dos ramos e montar a melhor árvore que pode encontrar, fazendo essa conquista recursivamente.

Foi executado o algoritmos em dois *datasets* distintos do mesmo conjunto de treinamento, um sem padronização e outro com padronização, com a variação de corte de ramos (com poda) e sem o corte (sem poda) para ambos esses *datasets*.

Com isso, obtivemos os seguinte resultados com relação à acurácia:

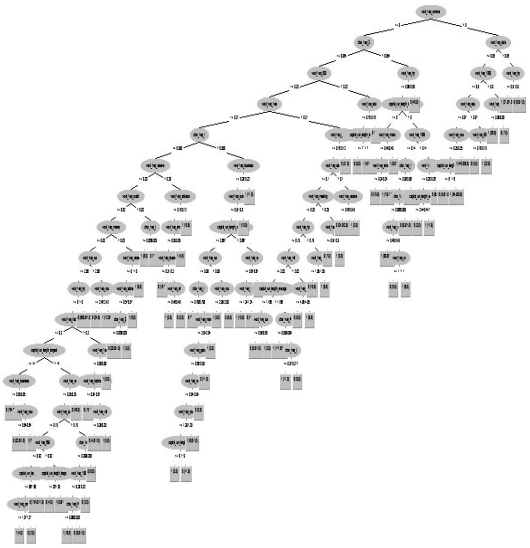


Em primeira análise, observamos que a padronização influencia muito o algoritmo na criação da árvore de decisão e em seus resultados para classificar as instâncias do conjunto de teste, com uma diferença de cerca de aproximadamente 30% entre o não padronizado e o padronizado, deixando mais claro de que a padronização piora o resultado do algoritmo. Na segunda análise, observamos as diferenças entre as podas, a execução do algoritmo sem poda se mostra mais eficiente do que com poda por uma pequena porcentagem, podendo variar, caso se utilize um outro conjunto de treinamento.

O algoritmo *J48* gerou todas as árvores de decisão com a raiz sendo o atributo '*word_freq_remove*', com ramificações para '*char_freq_$*' e '*word_freq_data*' ou seja, eles são os mais importantes para fazer as primeiras separações de um e-mail que é um *spam* e do que não é. Observando as imagens das árvores, nota-se pouca diferença entre o conjunto de treinamento não padronizado e o padronizado, são quase iguais. Porém existe diferença entre a sem poda e com poda, a sem poda é muito maior e mais complexa do que a com poda, que é mais enxuta, ambas pendem para o lado esquerdo, devido a quantidade de atributos presentes no *dataset*.

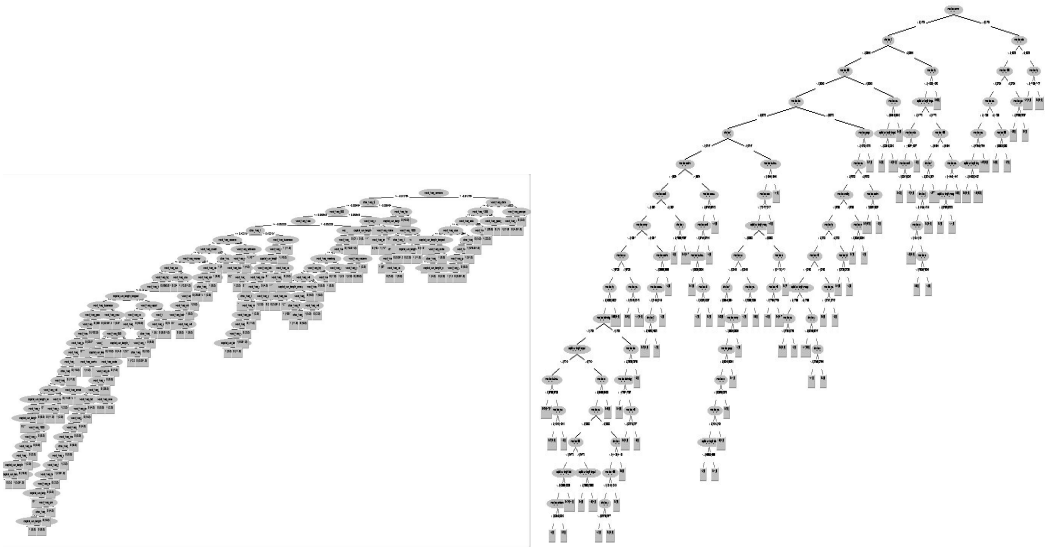
Sem poda, não padronizado

Com poda, não padronizado



Sem poda, padronizado

Com poda, padronizado

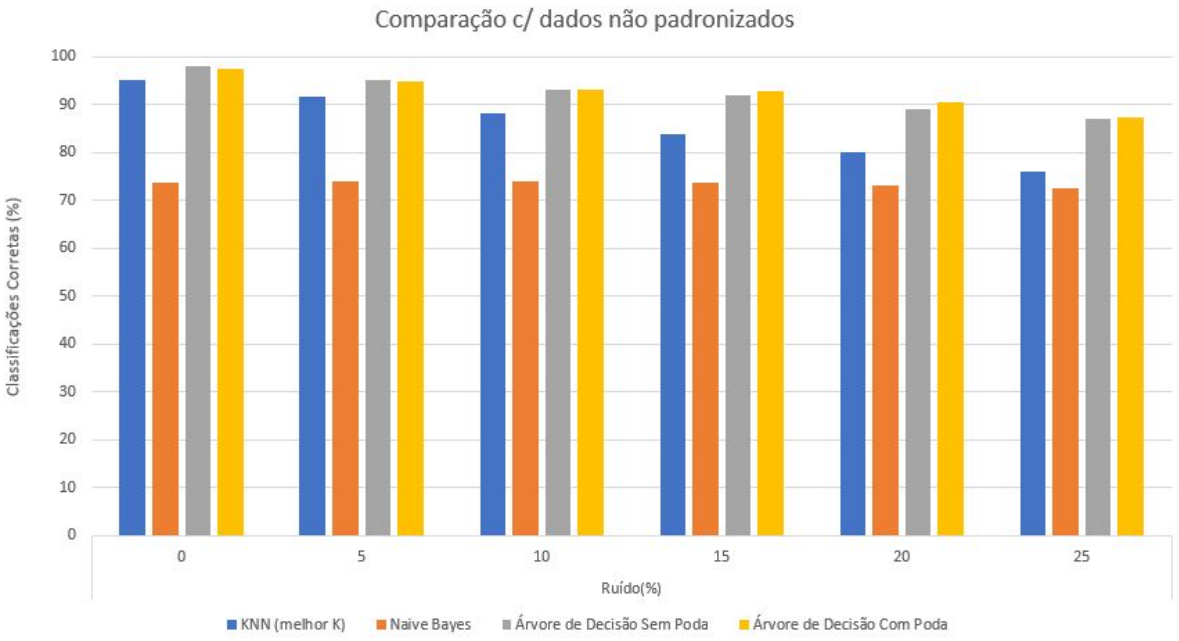


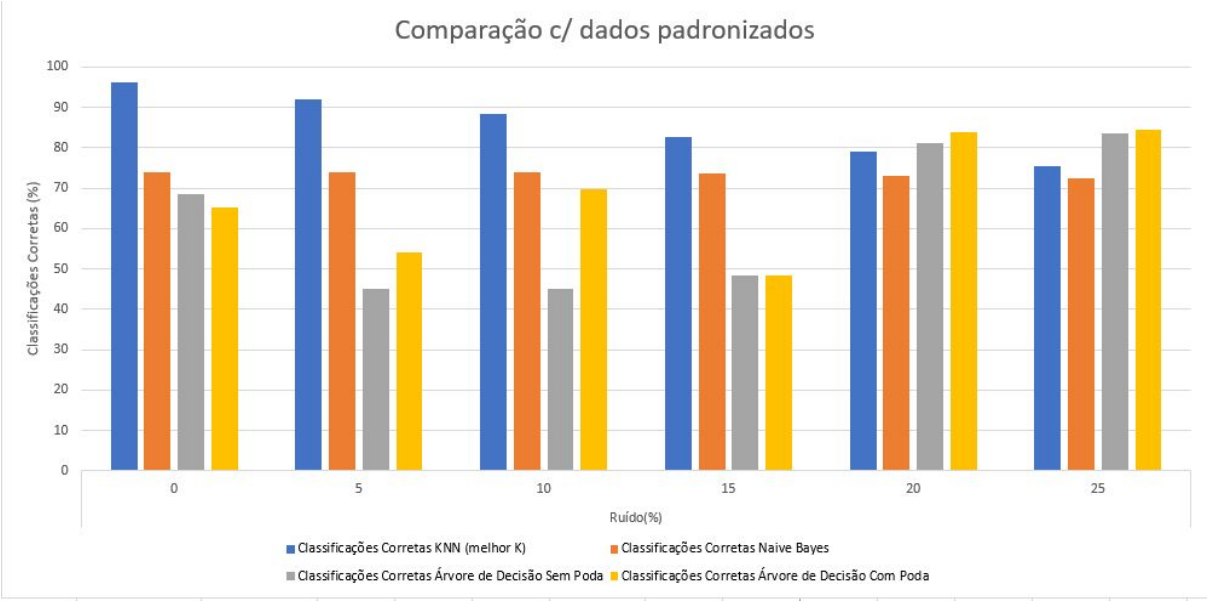
Robustez dos Classificadores

Feitas as execuções dos algoritmos apresentados nas outras seções, o enunciado faz a requisição de que os algoritmos sejam executados novamente, porém com ruídos no *dataset* de conjunto de treinamento, lembrando que ele deve seguir a padronização, caso seja um *dataset* padronizado, o de teste também deve ser.

A intenção de aplicar o ruído é verificar quão preciso o classificador consegue ser mesmo com interferências em seus dados de treinamento, e destacar qual seria o mais robusto entre os classificadores que foram utilizados. Foi construída uma tabela com todos os resultados para os classificadores com a porcentagem de ruído deles, sem padronização e com padronização, além de gráficos representando esses valores.

	Sem padronização				Com padronização			
Ruído (%)			Árv. de Decisão				Árv. de Decisão	
	KNN (melhor K)	Naive Bayes	Sem Poda	Com Poda	KNN (melhor K)	Naive Bayes	Sem Poda	Com Poda
0	95,1414%	73,8216%	98,0421%	97,3894%	96,2292%	73,8216%	68,5279%	65,3372%
5	91,7331%	73,9666%	95,2139%	94,9239%	92,0957%	73,9666%	45,1051%	54,0247%
10	88,1073%	73,8941%	93,1835%	93,2560%	88,4699%	73,8941%	45,1051%	69,8332%
15	83,8289%	73,6041%	91,9507%	92,9659%	82,8136%	73,6041%	48,4409%	48,4409%
20	80,2030%	73,1690%	89,1226%	90,5004%	78,9703%	73,1690%	81,2908%	83,7563%
25	75,9971%	72,5888%	86,9471%	87,3822%	75,3445%	72,5888%	83,4663%	84,4090%





Pelos resultados apresentados na tabela e nos gráficos, nota-se que o classificador KNN, com ou sem padronização e Árvore de Decisão sem padronização decaem de acordo com o percentual de ruído apresentado, quanto maior for o ruído, menor a precisão de classificação do classificador, o que sugere uma relação linear entre o ruído e a taxa de acerto do classificador. No entanto, temos outro classificador que se manteve estável durante todas as etapas de ruído, o Naive Bayes, independente se ele estivesse padronizado ou não, seu comportamento segue a probabilidade entre as relações dos atributos, sendo menos suscetível aos níveis de ruído inseridos. Ao analisarmos a parte padronizada da Árvore de Decisão, encontramos uma anomalia com relação a mesma, quando feita com a padronização, sua taxa de acerto reduz muito em comparação com a não padronizada, levando em conta só o conjunto de treinamento sem ruídos, ao adicionarmos ruído, notamos que a taxa de acerto começa a reduzir ainda mais drasticamente, mas ao chegar

em 15%, ela volta a aumentar sutilmente, em 20% e 25% ela dobra sua taxa de acerto com esse ruído maior, sendo maior do que a taxa de acerto do padronizado sem ruído, isso pode demonstrar que a padronização dos dados afeta o funcionamento da AD completamente, por se tratar de variáveis contínuas, e a árvore de decisão não ser tão eficiente com variáveis contínuas, e ao padronizá-las, ela perde a capacidade de distinguí-las.

Observamos assim, que, quando analisamos a robustez, o KNN, em média, apresenta uma boa taxa de acurácia, mesmo com alta taxa de ruído. Já, o Naive Bayes, apresentou alta estabilidade em relação ao crescimento da taxa de ruído, com uma variação por volta de 1% da acurácia com relação às diferentes taxas de ruído.

Conclusão

Com isso, pudemos concluir que, existem diversos tipos de algoritmos classificadores, cada um com suas propriedades, vantagens e desvantagens. Apesar de só analisarmos uma pequena porção dos algoritmos disponíveis para essas tarefas, entendemos o quão complexa pode ser sua escolha para solucionar determinado problema.

No algoritmo KNN, percebemos que outliers podem interferir muito na sua acurácia e uma boa normalização ajuda muito nesse quesito. Além disso, a escolha de um K ideal, nem muito pequeno e nem muito grande, pode ser fundamental para a eficiência dele.

Já com o algoritmo Naive Bayes, chegamos a conclusão que por conta do algoritmo ser probabilístico (a padronização altera somente a média e a variância, e não as probabilidades) e considerar os atributos como independentes (a padronização não interfere na dependência dos atributos), a acurácia se manteve a mesma sem ou com padronização.

Em relação às Árvores de Decisão, conseguimos observar que, diferentemente do Naive Bayes, a padronização influencia (negativamente) no algoritmo, tanto na criação das árvores, quanto em seus resultados. E, comparando sem poda e com, o algoritmo sem poda se mostrou mais eficiente por uma pequena porcentagem apenas.

E, então, adicionando ruídos em todos os algoritmos citados até agora, tanto o classificador KNN, com ou sem padronização, quanto a Árvore de Decisão sem padronização decaem de acordo com o percentual de ruído apresentado, sugerindo uma relação linear entre o ruído e a taxa de acerto do classificador. Enquanto isso, Naive Bayes se manteve estável durante todas as etapas de ruído, independente de padronização, já que é menos suscetível aos ruídos inseridos. E, por fim, analisando a Árvore de Decisão com padronização, sua taxa de acerto reduz muito, e depois aumenta sutilmente para enfim ultrapassar a taxa de acerto de uma com padronização sem ruído. Isso se deve ao fato da Árvore de Decisão não ser tão eficiente com variáveis contínuas, e quando padronizadas, a AD perde a capacidade de distinguí-las.

Referências

ABERNETHY, Michael. **Nearest Neighbor and server-side library**. 2010. Disponível em: https://www.ibm.com/developerworks/library/os-weka3/index.html (acessado em 30 de maio de 2019)

BROWNLEE, Jason. **How To Use Classification Machine Learning Algorithms in Weka.** 2016. Disponível em: https://machinelearningmastery.com/use-classification-machine-learning-algorithms-weka/ (acessado em 15 de junho de 2019)

DIFFERENCES BETWEEN THE L1-NORM AND THE L2-NORM (LEAST ABSOLUTE DEVIATIONS AND LEAST SQUARES). 2013. Disponível em: http://www.chioka.in/differences-between-the-l1-norm-and-the-l2-norm-least-absolute-deviations-and-least-squares/ (acessado em 17 de junho de 2019).

LIBRELOTTO, Solange Rubert; MOZZAQUATRO, Patricia Mariotto. Análise dos Algoritmos de Mineração J48 e Apriori Aplicados na Detecção de Indicadores da Qualidade de Vida e Saúde. **Revista Interdisciplinar de Ensino, Pesquisa e Extensão,** Rio Grande do Sul, vol.1, nº1, 2013. Disponível em: http://www.revistaeletronica.unicruz.edu.br/index.php/eletronica/article/viewFile/26-37/pdf
(acessado em 16 de junho de 2019)

OVERFITTING. Disponível em: https://www.datarobot.com/wiki/overfitting/ (acessado em 17 de junho de 2019)

OVERFITTING IN MACHINE LEARNING: WHAT IT IS AND HOW TO PREVENT IT. 2017. Disponível em: https://elitedatascience.com/overfitting-in-machine-learning (acessado em 17 de junho de 2019)

SRIVASTAVA, TAVISH. **Introduction to k-Nearest Neighbors: A powerful Machine Learning Algorithm(with implementation in Python)**. [*S. l.*], 26 mar. 2018. Disponível em: https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/. Acesso em: 14 jun. 2019.

WekaMOOC. **Data Mining with Weka (2.2 Training and testing).** 2013. Disponível em: https://www.youtube.com/watch?v=FMiCOx95lAc (acessado em 12 de junho de 2019)

ZUEGE. Tiago. **Aplicação de Técnicas de Mineração de Dados para Detecção de Perdas Comerciais na Distribuição de Energia Elétrica.** 2018. Disponível em: https://www.univates.br/bdu/bitstream/10737/2240/1/2018TiagoZuege.pdf (acessad em 17 de junho de 2019)