Análise e implementação de algoritmos de aprendizado sobre a base de dados Mushroom Database.

Claudio Carvalho de Oliveira
Centro de Informática
Universidade Federal de Pernambuco
Recife, Brasil
cco2@cin.ufpe.br

Guilherme Gouveia Figueiredo Lima
Centro de Informática
Universidade Federal de Pernambuco
Recife, Brasil
ggfl@cin.ufpe.br

Marina Oliveira Barros
Centro de Informática
Universidade Federal de Pernambuco
Recife, Brasil
mob@cin.ufpe.br

***Abstract*—Visando colocar em prática os conhecimentos adquiridos durante as aulas de Sistemas Inteligentes, ministradas pelo professor Cleber Zanchettin, aplicamos sobre a base de dados Mushroom Database (disponível no UCI Machine Learning Repository) quatro algoritmos de aprendizado supervisionado e três não-supervisionado. Posteriormente, implementamos nossa própria versão do classificador mais bem avaliado em cada categoria, realizando ainda uma análise sobre os resultados.**

Keywords—classficador, aprendizagem de máquina, aprendizagem supervisionada, aprendizagem não-supervisionada, projeto de disciplina

I. Introdução

Nosso projeto consiste basicamente em três etapas básicas, divididas entre a escolha de uma base de dados, o teste destas utilizando diversos algoritmos de aprendizagem e, por fim, a implementação de alguns deles.

Para a obtenção de uma base de dados, escolhemos dentre as disponíveis no UCI Machine Learning Repository, tendo como parâmetros para escolha um mínimo de 1000 instâncias e 6 atributos. Desse modo, acabamos por optar pela Mushroom Database, um *Data Set* contendo informações sobre cogumelos de vinte e três espécies dentre as famílias Agaricus e Lepiota, divididas em vinte e dois atributos e uma classificação entre comestível ou venenoso. Os dados originais para a formação dessa base vêm de [1].

Quanto à aplicação dos algoritmos de aprendizado, optamos pelo uso do WEKA, um software neozelandês que agrega uma vasta coleção de ferramentas para aprendizado de máquina e mineração dados, contando ainda com uma interface intuitiva para nossas análises.

Foram aplicados os seguintes algoritmos de aprendizagem supervisionada: Árvores de Decisão, K-Nearest Neighbor (KNN), Suppor Vector Machine (SVM) e Rede Neural (usando back-propagation). Já para os algoritmos de aprendizagem não-supervisionada, utilizamos: K-Means, Mistura de Gaussianas e Expectation-Maximization.

Por fim, além de realizar uma breve análise sobre os resultados obtidos, implementamos o classificador mais bem-sucedido tanto na categoria dos supervisionados quanto na dos não-supervisionados.

II. Nossa Base de Dados

A base de dados escolhida (Mushroom Database) contém exemplares de cogumelos de vinte e três espécies diferentes dentre as famílias Agaricus e Lepiota, classificando-os como comestíveis, venenosos ou de comestibilidade desconhecida. Essa última classificação foi agregada à categoria de venenosos pelo autor do *Data Set* para garantir uma maior segurança em seu uso.

Os atributos disponibilizados para que possamos realizar nossa classificação, todos nominais, são: *Cap Shape*, *Cap Surface*, *Cap Color*, *Bruises*, *Odor*, *Gill Attachment*, *Gill Spacing*, *Gill Size*, *Gill Color*, *Stalk Shape*, *Stalk Root*, *Stalk Surface Above Ring*, *Stalk Surface Below Ring*, *Stalk Color Above Ring*, *Stalk Color Below Ring*, *Veil Type*, *Veil Color*, *Ring Number*, *Ring Type*, *Spore Print Color*, *Population* e *Habitat.*

Quanto à proporção de exemplos de cada classificação em nossa base de dados, temos 3916 instâncias de venenosos (48,2%) e 4208 instâncias de comestíveis (51,8%). Fica claro então que o conjunto é bem balanceado quanto às classes, facilitando assim nosso trabalho em análises posteriores.

III. Pré-Processamento

Como o autor de nossa base de dados já a disponibilizou bem balanceada e organizada, o trabalho de pré-processamento necessário resumiu-se apenas à conversão dos arquivos originais para o formato suportado por nossa plataforma de análise.

Os dados originais encontravam-se em um formato de texto simples, com uma instância em cada linha e seus atributos ordenados e separados por vírgula. Um arquivo de apoio também foi utilizado para nos indicar à que remetia cada atributo na ordem apresentada, além do significado de cada um de seus valores nominais, representados apenas por um caracter para diminuir o tamanho do arquivo original.

Nosso primeiro passo então foi a conversão do arquivo de texto para um formato CSV. Essa etapa não demandou muito esforço pois, pela própria estrutura do arquivo original, bastou a adição de uma linha com os nomes das colunas para que ele apresentasse-se agora no formato desejado. Por fim, utilizando a ferramenta ARRF Viewer, disponibilizado também pelo WEKA, convertemos do formato CSV para o ARFF, tendo então nosso conjunto de dados no formato ideal suportado pelas ferramentas de análise que seriam utilizadas.

Pensando em facilitar o entendimento quanto aos valores dos atributos nominais, tentamos utilizar esta mesma ferramenta para substituir os caracteres por seus devidos valores utilizando a função *Replace Values With* disponível para cada coluna de atributos. Esta função, porém, mostrou comportamento inesperado e não nos forneceu os devidos resultados, nos levando a crer que a mesma não encontra-se em funcionamento estável. Dessa maneira, optamos por não realizar essa substituição de valores, visto que, com o auxílio do arquivo de nomes, isso não deveria afetar nosso entendimento quanto às análises.

IV. Utilizando os Algoritmos

A. Métricas para Análise

Para realizar a avaliação e comparação entre os algoritmos utilizados, assim como a subsequente escolha do melhor em cada tipo, optamos por analisar a taxa de acerto obtida, assim como a taxa de erro. Como critério de desempate, foi escolhido o tempo de execução.

Visando uma comparação sob as mesmas condições, testamos todos os algoritmos supervisionados utilizando *Cross-Validation 10-Fold* e os não-supervisionados utilizando a técnica de avaliação com relação ao rótulo de saída.

B. Algoritmos Supervisionados

1) Árvores de Decisão

Na aplicação desse algoritmo, utilizamos a árvore J48 disponibilizada também pelo WEKA. Considerando a grande variedade de atributos nominais encontrados em nosso conjunto de dados, já era esperado que esse caminho nos apresentasse bons resultados. Ainda assim, fomos surpreendidos pela acurácia encontrada.

Correctly Classified Rows	8124	100%
Incorrectly Classified Rows	0	0%
Time Taken to Build Model	0.02 *Seconds*	

Fig. 1. Resultados da análise sobre a Árvore de Decisão J48.

Podemos observar que o algoritmo basicamente conseguiu realizar uma classificação perfeita das instâncias presentes em nossa base de dados, além de fazê-la em tempo significativamente baixo.

2) K-Nearest Neighbor (KNN)

Utilizando o IBk, disponível por meio do WEKA, aplicamos o algoritmo KNN e obtivemos resultados igualmente impressionantes aos em comparação ao algoritmo anterior.

Correctly Classified Rows	8124	100%
Incorrectly Classified Rows	0	0%
Time Taken to Build Model	0 *Seconds*	

Fig. 2. Resultados da análise sobre o algoritmo KNN (IBk).

Assim como as Árvores de Decisão, o algoritmo KNN também conseguiu acertar com precisão ótima todas as instâncias de nosso *Data Set*. Apesar de seu tempo para construir o modelo ter sido menor que o do algoritmo anterior (mesmo que infimamente), o melhor desempenho ainda pertence às Árvores de Decisão por apresentarem um tempo de classificação menor, sendo praticamente instantâneo enquanto o KNN, por mais que seja muito rápido, ainda apresenta um ligeiro *delay* visível ao olho humano.

3) Support Vector Machine (SVM)

O próximo algoritmo a testarmos foi o SMO, aplicação do WEKA para o SVM. Os resultados, como já começamos a esperar dos algoritmos de aprendizado supervisionado a esse ponto, não decepcionaram nem um pouco.

Correctly Classified Rows	8124	100%
Incorrectly Classified Rows	0	0%
Time Taken to Build Model	1.56 *Seconds*	

Fig. 3. Resultados da análise sobre o algoritmo SVM (SMO).

Apesar da mesma taxa de acerto perfeita igual aos algoritmos anteriores, o SVM apresentou um tempo significativamente maior tanto para construir o modelo quanto para realizar a classificação, demorando alguns segundos para esta segunda tarefa.

4) Rede Neural Usando Backpropagation (Multilayer Perceptron)

Para finalizar nossos testes sobre os algoritmos de aprendizagem supervisionada, utilizamos uma rede neural com uso de backpropagation, por meio do Multilayer Perceptron disponibilizado também no WEKA. A primeira observação que notamos nessa execução é que, diferentemente dos algoritmos que utilizamos até então, este apresentou um tempo muito maior tampo para construir o modelo e, principalmente, para realizar a classificação.

Correctly Classified Rows	8124	100%
Incorrectly Classified Rows	0	0%
Time Taken to Build Model	*638.17 Seconds*	

Fig. 4. Resultados da análise sobre o algoritmo de Rede Neural usando backpropagation (Multilayer Perceptron).

Novamente, o resultado de classificação perfeita das instâncias de nosso *Data Set* foi obtido. Dessa vez, porém, o tempo necessário para executar a construção do modelo esteve na casa dos minutos (comparado com as frações de segundo dos algoritmos passados). Ainda assim, o maior problema esteve na fase de classificação das instâncias, que chegou a demorar horas para finalizar, mesmo sendo executada em um computador com poder de processamento relativamente alto. Com isso, podemos ter muita certeza ao afirmar que esse definitivamente não é o modelo ideal a ser aplicado para nossa base de dados.

5) Resultados

Sabendo então que todos os algoritmos apresentados conseguiram obter o mesmo resultado ótimos sobre nossos critérios de avaliação, foi necessário recorrer ao critério de desempate e selecionar o melhor baseado no tempo de execução de cada um deles.

É importante ressaltar antes de tudo, claro, que com resultados como esses era natural que houvesse uma desconfiança da equipe quanto à possibilidade de overfitting em nossas análises. Pesquisando um pouco, porém, nos arquivos auxiliares do *Data Set*, podemos ver que o próprio autor do conjunto deixa clara a presença de alguns atributos que, por si só, já são capazes de classificar bem mais de 90% dos casos. Considerando que existem múltiplos atributos com tais características, não é improvável que possa se obter uma precisão de 100% ao aplicar as técnicas de aprendizagem. Tendo essa questão resolvida, voltemos à seleção do melhor algoritmo.

A primeira a ser eliminada nesse processo sem dúvidas foi a Rede Neural, que apresentou uma quantia de tempo muito significativamente maior que os outros candidatos para acabar gerando o mesmo resultado, mesmo que ótimo.

O próximo a ser excluído da lista de possibilidades foi o SVM que, apesar de também ser bastante rápido e ter uso viável, ainda apresentou um tempo maior que seus outros dois concorrentes.

Dessa maneira, acabamos ficando com dois candidatos de execução praticamente instantânea, com um vencendo no tempo para construção do modelo e o outro no tempo de classificação sobre o modelo construído. Após executar diversas vezes ambos algoritmos prestando medindo seus tempos de execução, chegamos à conclusão que as Árvores de Decisão conseguem ser ligeiramente mais rápidas que o KNN, sendo às vezes tão velozes que impossibilitavam uma medida manual realizada pelos membros da equipe.

Concluímos então que o classificador mais bem-avaliado nessa categoria para nosso conjunto de dados é o algoritmo de Árvores de Decisão.

C. Algoritmos Não-Supervisionados

1) K-Means (SimpleKMeans)

Nossa primeira análise de um algoritmo de aprendizado não-supervisionado foi feita aplicando o K-Means, representado pelo SimpleKMeans presente no WEKA. O resultado de nossa execução pode ser representado pela Fig. 5

0	***1***	***Assigned to Cluster***
2093	1823	p
1234	2974	e
Cluster 0 ← p		
Cluster 1 ← e		
Incorrectly Clustered Instances	*3057*	37.6292%
Time Taken to Build Model (Full Training Data)		0.07 seconds

Fig. 5. Resultados da análise sobre K-Means usando SimpleKMeans.

Como esperado vindo de um algoritmo não-supervisionado, obtivemos aqui um resultado consideravelmente menor que nos nossos testes supervisionados. Ainda assim, é considerado um resultado bastante positivo, dividindo bem nossas instâncias entre os dois clusters gerados.

2) Expectation Maximization (EM)

Continuando nossas análises, partimos então para o algoritmo de Expectation Maximization, representado como EM na plataforma WEKA. Foram selecionados então treza clusters por cross-validation e, após a realização de vinte e cinco iterações, obtivemos alguns resultados.

Cluster #	***Clustered Instances***	***Clustered Instances %***	***Classification***
Cluster 0	576	7%	No Class
Cluster 1	1728	21%	e
Cluster 2	1728	21%	P
Cluster 3	88	1%	No Class
Cluster 4	1296	16%	No Class
Cluster 5	960	12%	No Class
Cluster 6	144	2%	No Class
Cluster 7	36	0%	No Class
Cluster 8	608	7%	No Class
Cluster 9	192	2%	No Class
Cluster 12	768	9%	No Class
Incorrectly Clustered Instances	*4668*		57.4594%
Time Taken to Build Model (Full Training Data)			571.09 seconds

Fig. 6. Resultados da análise sobre Expectation Maximization usando EM.

Esse algoritmo, além de ser bastante lento tanto na parte de construir o modelo quanto, principalmente, na parte da clusterização, resultou em uma quantia muito elevada de instâncias clusterizadas incorretamente. Certamente, dentre nossas opções, não é o mais adequado para nosso tipo de conjunto de dados utilizado.

3) Mistura de Gaussianas

A técnica de Mistura de Gaussianas (ou *Gaussian Mixture* em inglês) foi o último algoritmo de aprendizado não-supervisionado que tentamos utilizar para realizar nossa análise. É importante ressaltar justamente o termo 'tentar', pois não fomos capazes de realizar realmente uma classificação utilizando-o.

Inicialmente buscamos alguma implementação desse algoritmo dentre os *clusterers* presentes no WEKA, mas não conseguimos encontrar nada que realizasse essa função. Confirmamos então esta suspeita com os monitores da disciplina, que recomendaram o uso de outros meios caso realmente desejássemos concluir esta análise.

Nosso próximo passo então foi optar pelas ferramentas oferecidas por meio do *scikit-learn*, uma biblioteca de *Python* focada em aprendizado de máquina e mineração de dados. Aparentemente nesta existe uma função que possivelmente nos seria útil para aplicar em nosso *data set* mas, após muito tentar usar a ferramenta sem sucesso, optamos por abdicar dessa análise por recomendação de um dos monitores da disciplina.

Pelas pesquisas realizadas, porém, era de se esperar que esse algoritmo obtivesse um desempenho similar ou até mesmo inferior ao do Expectation Maximization. Essa hipótese, claro, apenas pode ser confirmada mediante a uma verdadeira análise aplicando-o na prática.

4) Resultados

Conseguindo apenas utilizar dois dos três classificadores desejados para algoritmos de aprendizado não-supervisionado, nossa escolha para o melhor ficou bastante limitada e, até mesmo, um tanto quanto fácil. Vale ressaltar que acreditamos que a falta de testes sobre a técnica de Mistura de Gaussianas não fez uma grande diferença sobre o resultado final pois não era de se esperar um bom desempenho dela baseado na teoria de sua implementação.

Dessa maneira, a escolha clara fora o algoritmo K-Means como o mais bem-avaliado dessa categoria, visto que o Expectation Maximization obtivera uma taxa de erro significativamente maior além de gastar muito mais tempo de execução.

V. Implementando os Melhores Algoritmos

A. K-Means

Começamos nossas implementações pelo algoritmo de aprendizado não-supervisionado com melhor performance dentre os testados, K-Means. Para isso, decidimos usar a ferramenta Jupyter Notebook para programar na linguagem Python 3.6.5 e testar nossos resultados em tempo real. A escolha foi feita pela familiaridade dos membros da equipe com esse ambiente de trabalho, assim como algumas de suas bibliotecas, mais especificamente NumPy e Pandas.

Nossa primeira problemática quanto a esse algoritmo foi que, após pesquisas iniciais, ficou bem claro que o mesmo é mais indicado para dados numéricos e contínuos, tendo mais dificuldades ao lidar com dados discretos, especialmente quando estes são não-ordinais. Tais problemas vês pela necessidade de plotar cada instância em um espaço n-dimensional e calcular a distância euclidiana entre eles, algo não tão diretamente aplicável quando estamos tratando com dados nominais, que representam todos os atributos de nosso conjunto.

Para podermos realizar estas operações, foi definida inicialmente que a função para o cálculo de distância entre duas categorias distintas de um atributo seria definida como 0 caso estes fossem iguais ou definida como 1 caso contrário. Como consequência, o cálculo de distância euclidiana entre dois pontos representados cada por uma instância de nossos dados seria totalizada como a quantia de atributos com valores diferentes que estes têm entre si.

Outro conceito necessário para este modelo que não pode ser diretamente aplicado para nosso conjunto de dados nominais era o cálculo de média entre vários pontos para gerar novos centros de clusters. Para realizar essa operação, optamos pelo cálculo da moda entre cada atributo das instâncias, podendo assim gerar uma espécie de instância média para um determinado conjunto de pontos quaisquer.

Tendo resolvido essas problemáticas nas funções, pudemos então começar a implementar de fato o algoritmo em questão. Primeiramente, para realizarmos o aprendizado não-supervisionado, retiramos a coluna de classe de nosso *data set* para que essa informação não influenciasse nossa decisão. A única informação extraída inicialmente desta coluna é a quantia de classes existentes, que será usada como quantia de clusters a serem gerados naquela execução, apenas 2 em nosso caso.

Criamos uma função para gerar instâncias aleatórias válidas dentre as categorias de cada atributo em nossos dados e aplicamos-na para gerar os centros de cluster iniciais. Feito isso, atribuímos cada uma das instâncias de nossa base de dados a um dos clusters optando pela cluster cujo centro tem menor distância em relação a esta e, uma vez que todas já houvessem sido atribuídas, calculamos novos centros para os cada cluster baseado na média dos valores das instâncias que atualmente a ele pertencem em cada dimensão.

O processo descrito foi repetido *ad infinitum* até que o valor dos pontos em cada cluster não mudasse mais entre repetições. Nesse momento, sabemos que os clusters já estão estáveis e está pronto o resultado de nossa clusterização.

Para checar nosso desempenho, atribuímos agora um valor de classe para cada cluster baseado nas instâncias ali presentes e, por fim, checamos a proporção de instâncias que foram corretamente classificadas pelos clusters.

Em algumas de nossas melhores execuções, conseguimos resultados de classificação bastante satisfatórios, superiores até aos nossos testes na plataforma WEKA.

Correctly Classified Rows	6499	85.73%
Incorrectly Classified Rows	*1625*	14126%

Fig. 7. Melhores resultados obtidos com nossa implementação do algoritmo K-Means.

Nem todas as nossas execuções, porém, obtiveram resultados tão elevados como estes. Ainda assim foram resultados significativamente satisfatórios, aproximando-se mais do que conseguimos com os testes no WEKA.

Correctly Clustered Instances	4692	57.76%
Incorrectly Clustered Instances	*3432*	42.23%

Fig. 8. Outro exemplo de resultados obtidos com nossa implementação do algoritmo K-Means.

Em geral o desempenho da classificação, apesar de variar bastante, sempre manteve-se entre 56% e 87%, tendo uma média aproximada de 65%. O tempo de execução total, englobando tanto a etapa de clusterização quanto os testes de classificação dá-se em uma média de aproximadamente 9 segundos.

B. Árvores de Decisão

Para nossa segunda implementação algorítmica utilizamos as mesmas ferramentas, linguagem e bibliotecas citadas anteriormente, dessa vez para criar um classificador supervisionado por meio de uma árvore de decisão. Após algumas pesquisas, encontramos vários algoritmos diferentes para a criação de árvores e optamos pelo C4.5, um algoritmo ideal para o tipo de dados nominais presentes em nosso *data set*. Tal algoritmo trata-se de uma evolução sobre o algoritmo ID3, que adequaria-se igualmente bem a nossos dados, mas é mais suscetível a *overfitting*.

Com o intuito de fazer um teste simples, dividimos nosso conjunto de dados em uma proporção de 70-30 para obter um subconjunto de treinamento e outro de testes.

Nosso primeiro passo para a implementação do C4.5 foi a definição de uma função para o cálculo da entropia da informação de um dado conjunto de valores categóricos, ou seja, o grau médio de incerteza ali presente. Naturalmente, é possível deduzir que atributos de maior entropia tem também maior potencial de obtenção de informação em sua observação.

Outra função foi então definida para calcular o ganho de informação agregado a um atributo em relação à sua classificação. Para realizar esse cálculo, utilizamos a entropia do atributo juntamente com a classificação das instâncias.

Dividindo então o ganho de informação de um determinado atributo por uma determinada taxa de probabilidade de seus valores em relação à suas classificações, podemos determinar a razão de ganho (ou *gain ratio*) desse atributo. Foi criada então uma função para calcular esta razão, que será a métrica utilizada para determinar o atributo de particionamento de cada nó.

Tendo agora todas as funções auxiliares prontas, partimos finalmente para a criação de nossa árvore de decisão. Esta árvore inicia-se apenas com um nó e, ao receber o conjunto de treinamento, calcula o atributo de maior razão de ganho de informação e define-o como seu atributo de particionamento. Em seguida, para cada valor possível a ser atribuído ao atributo de particionamento, é criado um nó filho que será construído assim como o original, porém a partir de um subconjunto do conjunto de treinamento, este contendo apenas as instâncias cujo valor do atributo de particionamento de seu pai igualam ao referente àquele filho.

A árvore constrói-se recursivamente, formando folhas (nós sem filhos) quando efetivamente não houver uma maior razão de ganho de informação daquele nó, o que implicitamente quer dizer que todas as instâncias ali representadas apresentam a mesma classificação. Dessa maneira, atribuímos essa classificação à essa folha. Uma vez com toda a árvore calculada, com cada um de seus ramos terminando apenas em folhas, temos nosso classificador completo.

Para realizar a classificação de uma nova instância, basta colocá-la na raiz da árvore e repassá-la para a subárvore correta baseada no atributo de particionamento daquele nó. Ao final da execução, a instância encontrará-se em uma folha e receberá a classificação da mesma.

Com o intuito de checar a performance de nossa implementação, passamos então cada instância de nosso conjunto de testes pelo classificador e comparamos sua classificação predita com sua classe real. Baseando-nos em nossas análises pelo WEKA, sabíamos que era possível a obtenção de resultados ótimos, o que de fato ocorreu, provando o desempenho de nosso C4.5.

Correctly Classified Rows	8124	100%
Incorrectly Classified Rows	*0*	0%

Fig. 9. Performance ótima obtida por nossa implementação de árvores de decisão por meio do algoritmo C4.5.

Assim como em nossa análise pelo WEKA, alguns atributos mostraram-se altamente eficientes para a classificação, destacando-se dentre estes *odor* e *spore-print-color*. O tempo global de execução de nossa implementação desse algoritmo, englobando tanto a etapa de construção da árvore de decisão quanto os testes de classificação dá-se em uma média de aproximadamente 4 segundos.

VI. Questões Investigativas

A. Qual classificador/configuração apresenta melhores resultados em termos de acurácia e de tempo?

Com base nos testes realizados na fase da análises de nosso estudo podemos concluir que, ao menos para nosso conjunto de testes, o melhor classificador baseado nessas duas métricas é a árvore de decisão. Assim como os outros classificadores supervisionados testados, ela apresenta uma taxa de acerto perfeita, além de custar menos tempo ao combinar a fase de construção do modelo com a fase de classificação.

B. Avalie os parâmetros: taxa de aprendizado, número de épocas de treinamento, funções de ativação de neurônio, parâmetros de configuração, etc. Caso julgue necessário outros atributos podem ser inseridos na análise. Analise e discuta os resultados obtidos.

Os parâmetros citados tratam-se de hiper-parâmetros de redes neurais que devem ser ajustados antes do treinamento da rede para obter-se bons resultados.

A taxa de aprendizado refere-se, basicamente, à predisposição que a rede tem de abandonar seus conhecimentos anteriores para substituí-los por novos. Esse hiper-parâmetro deve ser tratado com cuidado pois, se muito alto, pode fazer com que a rede esqueça-se muito do que já aprendeu até então, gerando resultados de menor qualidade. Quando muito baixo, porém, esse hiper-parâmetro pode fazer com que a rede tenha um custo de tempo consideravelmente alto para gerar novos conhecimentos. Em geral, busca-se valores para esse parâmetro para que a rede tenha bons resultadas mas não demore demasiado tempo em treinamento.

O número de épocas de treinamento refere-se à quantia de passagens completas que a rede fará pelo conjunto de dados até seu término. Logicamente, a precisão da rede tende a aumentar mais e mais a cada época, mas a partir de um certo momento isso pode começar a gerar *overfitting*, enganando-nos com esse desempenho. Dessa maneira, definir um valor máximo de passagens para fazermos um *early-stopping* pode ser uma boa ideia para evitá-lo.

A função de ativação, em suma, é quem ditará a saída devolvida por um neurônio dada a entrada que o mesmo recebe. Esta saída será utilizada como entrada pelo próximo neurônio na rede. Existem diversos tipos de implementação diferentes para funções de ativação e sua escolha deve ser meticulosamente pensada antes de iniciar-se a aprendizagem da rede.

Em geral, os hiper-parâmetros mencionados anteriormente, assim como um conjunto de muitos outros, juntos, fazem parte dos parâmetros de configuração de uma rede neural. Seus valores, dependendo de como ajustados, podem ser a diferença crucial dentre redes tanto em questão de eficiência quanto de precisão.

C. *Quais as características do modelo que influenciam seu desempenho sobre a base de dados?*

Os desempenhos obtidos sobre nossa base foram fortemente influenciados, para todo tipo de modelos, pela característica de nosso *data set* de ter apenas atributos nominais.

Esse caso é ideal para as árvores de decisão, por exemplo, pois elas dependem de categorização dentre os valores possíveis dos atributos, precisando realizar inclusive o agrupamento caso receba valores numéricos contínuos. Outra grande vantagem para esse modelo é a presença de alguns atributos que são muito mais significativos, permitindo um filtro inicial robusto quando aplicado o aprendizado supervisionado.

Alguns outros tipos de algoritmos, como o K-Means e o Expectation-Maximization, por exemplo, assim como a maioria dos outros não-supervisionados, têm uma certa dificuldade ao lidar com essas características. Os dados nominais realmente não facilitam o trabalho desses algoritmos, assim como a presença de atributos mais relevantes que outros.

D. *Como determinar a melhor quantidade de agrupamentos?*

A quantia de agrupamentos (*clusters*) à serem criados em algoritmos de clusterização (em aprendizado não-supervisionado) deve ser definida, primeiramente, baseada no conhecimento ou não da quantia de classificações possíveis.

Caso saibamos previamente a quantia de classificações que devemos gerar, podemos atribuir esse valor para definir nossa quantidade de agrupamentos. Caso contrário, uma boa maneira de descobrir essa quantia é utilizar o *Elbow Method*.

Esse método, basicamente, olha para a variância gerada ao subir a quantia atual de agrupamentos. A partir do momento em que a variância não está mais aumentando significativamente, podemos inferir que a adição de ainda mais *clusters* não está nos agregando muitas novas informações e, consequentemente, podemos parar por aí.

E. *Como melhorar o desempenho do classificador?*

Para aumentar o desempenho de nossos classificadores, sempre há a possibilidade de modificar os parâmetros e hiper-parâmetros utilizados em sua inicialização. Na maioria dos casos, não existe um "valor mágico" que sempre servirá, cada modelo e conjunto de dados demandará valores diferentes que se adequem às suas necessidades específicas.

O importante é sempre que o analista não espere que o modelo/algoritmo realize todo o trabalho para ele e retorne resultados perfeitos apenas ao receber um *data set* arbitrário. As configurações devem ser muito bem pensadas e manualmente ajustadas para obter um bom funcionamento.

F. *Quais técnicas podem ser aplicadas para o pré-processamento ou pós-processamento dos dados?*

Para o pré-processamento dos dados, é importante o uso de técnicas para que estes adequem-se inteiramente ao tipo de entrada esperada pelo algoritmo que venha a ser utilizado. Um caso simples pode ser quando existe um atributo com dados contínuos em uma árvore de decisão, tornando importante a categorização dos elementos desse atributo.

Já para o pós-processamento, devemos analisar bem as respostas obtidas quando tratamos de nossa classificação. Em caso de clusterização, por exemplo, é importante analisar os *clusters* para agrupá-los em relação às classes predominantes ali presentes. É importante também sempre fazer um estudo sobre a taxa de acerto de um determinado modelo para buscar *under* ou *overfitting*.

G. *Os classificadores podem ser combinados? Como? Quais as implicações?*

Sim, é possível combinar variados modelos e isso pode nos gerar bons resultados. Existem diversas técnicas para estes fins, variando desde médias simples dos conjuntos de resultados até sistemas de votação simples e ponderada entre os modelos. No caso da ponderada, podemos definir os pesos de cada voto baseado na precisão de cada modelo.

Em geral, uma combinação bem feita pode ser capaz de aumentar a precisão global da análise e, principalmente, diminuir consideravelmente a possibilidade de *overfitting*.

H. Qual o desempenho dos classificadores escolhidos como os melhores sobre outra base de dados do UCI?

Utilizando a plataforma WEKA realizamos novas análises, dessa vez para a base de dados Abalone, também disponível no UCI. Esse *data set* foi escolhido principalmente por sua diferença em relação ao nosso quanto ao tipo dos atributos. Enquanto trabalhamos apenas com atributos nominais durante nossas análises, esse novo conjunto possui apenas dados numéricos contínuos além, claro, de sua classificação.

Correctly Classified Rows	2206	52.813%
Incorrectly Classified Rows	1971	47.187%
Time Taken to Build Model	0.38 *Seconds*	

Fig. 10. Resultados da análise do conjunto Abalone sobre a Árvore de Decisão J48.

0	***1***	***Assigned to Cluster***
463	1064	M
332	975	F
1142	200	I
Cluster 0 ← I		
Cluster 1 ← M		
Incorrectly Clustered Instances	*1971*	47.187%
Time Taken to Build Model (Full Training Data)		0.16 seconds

Fig. 11. Resultados da análise sobre K-Means usando SimpleKMeans.

Como podemos observar, os classificadores que melhor funcionaram em nossa base de dados, obtiveram resultados consideravelmente inferiores quando aplicadas à base de dados Abalone.

Boa parte dessa perda em desempenho certamente pode ser conectada às diferenças entre os dados dos dois *data sets*. Como já dito anteriormente, o fato de um ser formado por dados nominais enquanto o outro é composto por dados numéricos contínuos, o que muda completamente o tipo de paradigmas que devem ser aplicados durante as análises feitas pelos algoritmos.

Temos então essa comparação entre os dois conjuntos como uma prova de que não existe um "melhor algoritmo" ou "melhor modelo" que se aplica a todos os casos, mostrando a importância da escolha certa pelo analista para obter um bom desempenho na classificação.

VII. Conclusão

O trabalho realizado permitiu à equipe aprofundar os conhecimentos obtidos em sala de aula sobre determinados tópicos dentro da área de aprendizado de máquina por meio de análises e implementações algorítmicas que necessitam de um bom entendimento de seus funcionamentos para ocorrerem.

Realizando esse estudo prático, sentimos-nos agora mais dominantes sobre estes tópicos especialmente nos aspectos de desempenho e execução, algo que dificilmente seria atingido caso nos limitasse-mos meramente ao estudo teórico.

Por fim, a equipe concorda que os conhecimentos adquiridos serão de grande importância para futuros trabalhos nesse campo, especialmente tratando-se da possível necessidade de escolha de algoritmos para usos práticos.

Referências

[1] The Audubon Society Field Guide to North American Mushrooms (1981). G. H. Lincoff (Pres.), New York: Alfred A. Knopf