Projeto Prático 2
“There and back again”

Adlla Katarine Aragão Cruz Passos
Universidade Estadual de
Feira de Santana
UEFS
Email: adllakatarine@hotmail.com

Daniel Alves Costa
Universidade Estadual de
Feira de Santana
UEFS
Email: dancostafsa@hotmail.com

Ramon de Cerqueira Silva
Universidade Estadual de
Feira de Santana
UEFS
Email: ramondecerqueirasilva@gmail.com

***Resumo*—O estudo de Mineração de Dados vem crescendo exponencialmente no mundo da ciência da computação. Assim, propicia à população elementos capazes de interpretar e usar, a grande massa de dados gerada diariamente. Saber qual produto recomendar para uma pessoa acessando uma loja online, e descobrir padrões para determinada doença, são exemplos do seu uso. Muitas técnicas foram surgindo durante a evolução dessa ciência. Alguns dos modelos que estão sendo bastante utilizados atualmente são os de Ensemble Learning. Técnicas como o Bagging, Boosting, AdaBoost vem ganhando bastante visibilidade no meio da ciência de dados.**

O mundo dos super heróis está crescendo na cultura popular, tanto entre jovens, quanto entre adultos. Atualmente esse mundo apresenta diversos heróis que foram surgindo através do tempo. Cada um com suas próprias características e super poderes. Visando isso, o objetivo deste trabalho é, a partir de duas bases de dados, prever determinadas características e poderes dos heróis presentes nelas. E para isso, foram utilizadas técnicas de classificação de dados e de Ensemble Learning para resultar em uma previsão satisfatória.

I. Introdução

O desenvolvimento deste trabalho foi proposto pelo professor da disciplina *EXA864 - Mineração de Dados de 2019.1*, Rodrigo Calumby. A intenção é fazer com que seus alunos apliquem as técnicas adquiridas em suas aulas, para a criação de uma avaliação experimental. E assim, foi solicitado métodos de classificação de dados e modelos de *Ensemble Learning* para a resolução do problema.

O objetivo é predizer, a partir de algoritmos de aprendizagem de máquina, determinadas características e super poderes dos super heróis disponibilizados. A equipe decidiu optar por mais um objetivo, chega a uma acurácia de pelo menos 70% em todos os atributos a ser previstos, sejam por classificadores, ou Ensemble Learning. Para isso foi fornecida duas bases de dados, contendo nome dos personagens, características e seus poderes. Com isso, os grupos receberam o trabalho de preparar os dados para a análise, realizando um pré-processamento, utilizar técnicas de classificação de dados e modelos de Ensemble Learning, para a predição das características determinadas pelo professor e também pela equipe. Além do uso de métodos para avaliação dos classificadores, sendo capaz de visualizar a efetividade dos algoritmos aplicados.

II. Algoritmos e Ferramentas Utilizadas

A. Tecnologias

No trabalho realizado, optou-se por usar a linguagem de programação *Python* em sua versão 3.6, juntamente com uma IDE para sua utilização, sendo essa o *Spyder* na versão 3.3.4, facilitando cálculos estatísticos e visualização dos dados requeridos. E para isso, foi criado um ambiente pela ferramenta *Anaconda Navigator*, apenas para a utilização do pacote *fancyimpute* utilizado no pré-processamento dos dados[4] [6].

B. Ferramentas

Com a necessidade de técnicas de manipulação de base dados, foi utilizado métodos da biblioteca *Pandas*, que fornece ferramentas de análise de dados e estruturas de dados de alta performance. Ela cria *DataFrames* com arquivos *csv*, por exemplo, como foi o caso do projeto. E a partir disso, foi possível trabalhar facilmente com os dados e fazer o uso de outras bibliotecas do *Python*[1].

Para a maior parte do desenvolvimento do projeto, foi utilizada uma biblioteca de aprendizagem de máquina, *Sklearn*[3]. Ela contém vários algoritmos de classificação, regressão e agrupamento. Foram usados algoritmos para a classificação: Árvore de Decisão, *Naive Bayes*, *Random Forest*; algoritmos de Ensemble: *VotingClassifier*, *Bagging* e *Addaboost*; avaliação dos modelos de predição. Para o pré-processamento, usou-se o pacote fancyimpute, destinando o algoritmo *KNN*, a predições de atributos ausentes.

A biblioteca *Numpy* permite trabalhar com arranjos, vetores e matrizes de N dimensões de forma eficiente. Sendo utilizada durante o pré-processamento dos dados, para o tratamento de valores NaN[2].

Para a avaliação da qualidade dos modelos de predição, foram usados os seguintes métodos: matriz de confusão, recall e precision com validação cruzada, acurácia, e a curva ROC; sendo todos eles da biblioteca Sklearn e que serão explicados melhor mais adiante.

III. Análise Proposta e Metodologia

Neste projeto, cada atributo que será previsto tem seu pré-processamento, pois usar o mesmo tratamento para atributos diferentes pode influenciar diretamente no desempenho da

predição. Portanto,houve uma divisão em sete arquivos, cada um contendo uma classe a ser prevista.

A. Base de Dados

As bases de dados utilizadas neste projeto foram disponibilizadas pelo professor da disciplina, estando elas em formato *csv*, com os seguintes nomes: "herois" e "superpoderes". No *dataset* "herois" tem-se nome e características de heróis e vilões. Visualizando o *dataset*, é possível perceber algumas particularidades. Há 734 nomes de heróis, atributo nominal, com mais nove características, sendo que em alguns casos, é possível notar características com valores ausentes ou nulos. O segundo *dataset* apresenta o nome de personagens, sendo notável a presença de 667 nomes, além de 167 poderes representados por atributos binários em que pode-se saber se o personagem possui ou não um determinado super poder.

B. Pré-processamento

Em muitos casos, pode-se dizer que o pré-processamento de dados é a fase mais importante de um projeto de aprendizagem de máquina. É nesse estágio que se realiza a "limpeza" dos dados. É o responsável por verificar, na base de dados, os valores nulos, vagos, negativos, repetidos, inconsistentes. Por fim, o pré-processamento trata cada um deles.

1) Remoção de Variáveis: Inicialmente, como um dos requisitos do projeto, foi feita a integração das duas bases de dados, "herois" e "superpoderes". Após isso, observou-se a existência de heróis em "superpoderes" ausentes na base de dados "herois"; por fim foi decidida a exclusão desses heróis em superpoderes, pois um tratamento nas características existentes na base de dados de "herois" tornaria esses heróis muito artificiais. Outra observação foi a presença de heróis duplicados, nome e atributos, e para isso, foram tomadas duas decisões: uma parte desses heróis foram removidos da base de dados, pois, além do nome, seus atributos eram bem parecidos entre si; enquanto os restantes tiveram seus nomes modificados.

2) Agrupamento: Os atributos numéricos, *Height* e *Weight*, continham alguns valores ausentes e foram tratados utilizando a média geral de cada um. Com o objetivo de reduzir o número de valores desses atributos foi realizada uma discretização desses dados, transformando-os de atributos numéricos contínuos em discretos. No projeto, os valores de *Height* levam, 0 e 1, respectivamente baixo, menor ou igual a média (0), e alto (1), maior que a média. Valores de *Weight* foram divididos em 0, menor que a média, 1, a média, 2, maior que a média, representando respectivamente, leve, médio e pesado.

No atributo *Publisher*, a classe "Marvel Comics" ocorre em mais de 50% dos heróis, assim houve a decisão de tornar o atributo binário, onde todos os valores diferentes de "Marvel Comics" fossem substituídos por "Outher".

3) Preenchimento de Valores Ausentes: Essa etapa se demonstrou difícil devido ao fato de que os valores ausentes no *dataset* não estavam simplesmente ausentes, mas constavam apresentando valor '-', além dos *np.nan*. Utilizando disso, os valores '-' foram trocados por *np.nan*, modificação feita para atender ao algoritmo do *KNN*, utilizado pelo pacote *fancyimpute*. No *KNN*, os k vizinhos são escolhidos com base em alguma medida de distância e sua média é usada como estimativa de imputação. O método requer a seleção do número de vizinhos mais próximos e uma métrica de distância[13].

Pode-se prever atributos discretos (valor mais frequente entre os k vizinhos mais próximos) e atributos contínuos (média entre os k vizinhos mais próximos). Caso o dado seja discreto, usa-se a distância *Hamming*, onde observa-se todos os atributos categóricos e, para cada um, some 1 se o valor não for o mesmo entre dois pontos. Com dado contínuo, as métricas de distância comumente usadas são *Euclidean*, *Manhattan* e *Cosine*. Ficou decidido usar o método *KNN* para predição dos atributos no projeto usando os 5 ou 3, depende da classe, vizinhos mais próximos, e a distancia de Hamming, pois a base de dados trabalha apenas com atributos discretos.

C. Classificadores

A classificação é uma tarefa de mineração de dados responsável por dizer se um determinado objeto pertece a uma classe, de acordo com suas características. Para isso acontecer, primeiramente deve-se pegar a base de dados, depois de pré-processada, e escolher um algoritmo de classificação. Este algoritmo irá realizar seu treinamento a partir de uma parte da base de dados, que foi dividida previamente. Em seguida, ela irá realizar o seu teste com a parte restante dos dados. Após a fase de teste do classificador, é importante que haja uma avaliação do classificador para determinar a eficácia de sua previsão. Para este projeto, foram escolhidos os seguintes classificadores:

1) Árvore de Decisão: Árvore de decisão é um método de aprendizado de máquina supervisionado não-paramétrico. Ele funciona da seguinte forma: cada nó de decisão contém um conjunto de informações para algum atributo e cada ramo descendente corresponde a um possível valor deste atributo. O conjunto de ramos são distintos, cada folha está associada a uma classe e, cada percurso da árvore, da raiz à folha, corresponde uma regra de classificação. Para a criação da Árvore, foi usada a classe *DecisionTreeClassifier*, da biblioteca *Sklearn*, bem como hiperparâmetros. Os hiperparâmetros são usados para encontrar os melhores parâmetros para a árvore e neles foram definidos: um intervalo de profundidade da árvore, sendo **[3,20]**; um intervalo de quantidade de nós, sendo **[1,5]**; além do critério para impureza, sendo o de **Gini**, ou **Entropy**. Esses hiperparâmetros foram usados no *GridSearchCV*, que funciona com uma validação cruzada testando todos os parâmetros, sendo decidido no projeto, **K=3**, a quantidade de validações cruzadas, buscando a melhor parametrização para a árvore e assim evitando *Overfitting*.

2) Naive Bayes: O *Naive Bayes* é um classificador baseado no Teorema de Bayes. A principal característica do algoritmo é que ele desconsidera qualquer correlação entre os atributos da base de dados, ou seja, um atributo não interfere no valor de qualquer outro atributo. O algoritmo funciona convertendo o conjunto de dados em uma tabela de frequência e, a partir

disso, é feito um cálculo de probabilidade em cada atributo individualmente e também na classe previsora, depois é feita a probabilidade entre cada atributo com a classe.

Dentre os modelos de *Naive Bayes* disponíveis, o Bernoulli foi o escolhido. Sendo adequado para dados discretos, este algoritmo foi optado por ser projetado para aplicações em dados binários. Ele possui quatro parâmetros que podem ser definidos para a criação do classificador, porém nenhum precisou ser modificado a fim de que pudesse melhorar a predição.

3) Random Forest: O *Random Forest* é um algoritmo de aprendizagem supervisionada e faz uso de um conjunto de árvores de decisão. Como o próprio nome expressa, ele cria uma floresta aleatória, porém não tão aleatória assim. Inicialmente, ele separa um conjunto aleatório de atributos para criar uma árvore de decisão, o processo da criação da árvore de decisão é a mesma que explicada anteriormente e faz isso quantas vezes for definida. Após isso, o *Random Forest* executa uma votação, entre todas as suas árvores de decisões, a fim de decidir o valor da classe prevista. Para a criação do *Random Forest*, usou-se a classe *RandonForestClassifier*, do *Sklearn*. Dentre os seus parâmetros disponíveis, três foram modificados: quantidade de árvores a serem geradas, sendo escolhido **50** por ter produzido os melhores resultados; critério de ganho de informação, sendo **entropy**; e o *randomstate* que, quando **igual a 0**, guarda o estado atual do treinamento e impede que ocorra uma nova instância do modelo classificatório.

D. Ensemble Learning

Foram utilizados modelos de Ensemble para se ter uma melhor predição dos atributos. Com a finalidade de minimizar os erros, foi proposto o uso desses modelos. Um ensemble é um conjunto de classificadores cujas decisões individuais são combinadas de alguma forma para classificar um novo caso. [8]. A proposta principal dos ensembles é combinar classificadores e aproveitar as contribuições de cada um deles para conseguir melhores resultados de classificação em detrimento da complexidade computacional e também maior custo computacional[7]. Após utilizar os métodos de votação para determinar um preditor mais adequado para se utilizar, em que aplicou-se as técnicas de ensemble. Foram escolhidos os seguintes modelos de ensemble:

1) Voting-Classifier: O classificador *Voting* é um método de aprendizagem de máquina que combina classificadores distintos a fim de obter uma melhor predição. Ele funciona fazendo uma votação, onde cada classificador recebe um peso e quanto maior este peso, maior confiabilidade o classificador ele terá. Por fim, a resposta que tiver a maior quantidade de votos, é a considerada correta. Aqui está um exemplo para maior entendimento: se temos N estados e cada um deles pode assumir o valor de ativo ou inativo e, cada classificador poderá responder a probabilidade de um estado estar ativo ou inativo, o *Voting Classifier* escolherá qual um desses estados, a partir da votação dos outros classificadores.

Na criação do *Voting*, usou-se a classe *VotingClassifier*, do *Sklearn*. Nele, foram usados os três classificadores discutidos anteriormente: *Árvore de Decisão*, *Random Forest* e *Naive Bayes* e receberam, respectivamente, os pesos 1.1, 2 e 1, com o propósito do classificador alcançar um resultado final melhor. Finalmente, seu resultado final foi aplicado nos algoritmos de AdaBoost e Bagging, sendo também o seu propósito neste projeto.

2) Bagging: Publicado pela primeira vez por *Breiman* em 1996[9], esse método utiliza de um mesmo algoritmo para realizar as predições, e após isso combina seus resultados. São gerados conjuntos sucessivos e independentes de amostras, denominadas *bootstrap*, a partir do conjunto de treinamento com n exemplos. Assim, cada *bootstrap* é gerado escolhendo de forma aleatória n exemplos, com substituição, de forma que todos apresentem a mesma quantidade de exemplos de T[10]. Os classificadores são gerados de forma paralela e o classificador final é uma média realizada pelos classificadores gerados.

No projeto ele foi aplicado utilizando a classe *Bagging-Classifier* da biblioteca *Sklearn*. Utilizando como parâmetros o classificador do método de votação. A quantidade máxima de amostras para treinar cada previsor igual a **0.5**, e o número de previsores no ensemble de **5**, com todos seus outros parâmetros no seu valor padrão.

3) AdaBoost: Publicado em 1995 por *Freund* e Schapire[11], ele funciona de forma repetitiva por uma série de rodadas. Utilizando de pesos para melhorar sua classificação ele repete o processo de classificação do algoritmo escolhido por uma certa quantidade de vezes. Esses pesos variam a cada nova rodada, diminuindo daqueles atributos que ele acertou a previsão e aumentando dos que errou até obter uma classificação aceitável. Seu funcionamento finaliza obtendo uma combinação ponderada das diversas saídas de iterações, dando uma importância maior para as saídas que obtiveram um maior número de acertos[12].

No projeto foi utilizado a classe *AdaBoostClassifier*, do *Sklearn*. Tendo como parâmetros o classificador gerado pelo *VotingClassifier*, a quantidade de previsores, no qual o processo de *boosting* é finalizado, igual a **5**, e tendo todos seus outros parâmetros como padrão da biblioteca.

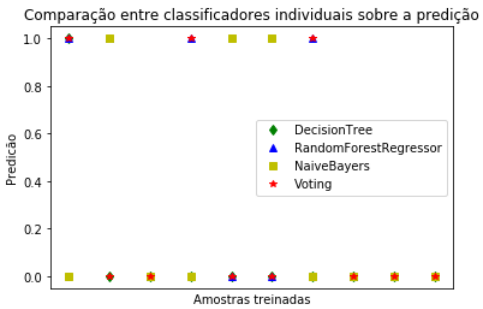

Figura 1. Predição do atributo Accelerated Healing, nota-se a influência nos pesos na 4 predição.

E. Validação

Após criação e treinamento de um classificador, é necessário começar os testes e avaliar sua performance em diferentes métricas de desempenho. Para tal, foram escolhidas algumas métricas para avaliação dos classificadores, enquanto que os métodos de *Ensemble Learning* foram avaliados apenas com sua acurácia.

Seguem abaixo resumos das definições de métricas de avaliação de desempenho usados neste projeto:

1) Matriz de confusão: Tabela que mostra as frequências de classificação de cada classe do modelo classificando-os como verdadeiro positivo, verdadeiro negativo, falso positivo e falso negativo;

2) Validação Cruzada: Particiona os conjuntos de dados em subconjuntos para o treinamento, repetindo o processo K vezes para avaliar a capacidade de generalização de um modelo. No projeto foi definido usar **K=3** ou **K=5**;

3) Acurácia: Taxa total de classificação correta. Ela é determinada pela razão entre a quantidade de verdadeiros positivos e a quantidade de entradas;

$$ACC = \frac{VP + VN}{VP + FP + FN + VN}$$

Figura 2. Função da Acurácia

4) Recall: Calcula a capacidade de predição do modelo;

5) Precision: Calcula quantas classificações de classe o modelo fez corretamente;

6) Curva ROC: Demonstra o desempenho de um classificador com dados binários e possui dois parâmetros: verdadeiro positivo e falso positivo.

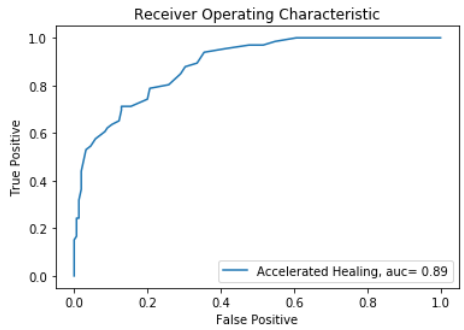

Figura 3. Curva ROC do atributo Accelerated Healing, plotagem pela biblioteca MatPlotLib

7) F1: A pontuação F1 pode ser interpretada como uma média ponderada da *precision* e *recall*.

$$F1 = \frac{2 * (PRE * REC)}{(PRE + REC)}$$

Figura 4. Função da F1

IV. Resultados e Discussão

Para criar um classificador ou modelo de classificação, é preciso coletar os dados, limpar, tratar, modificar, treinar, testar e por fim, validar. E, caso o modelo de classificação não contenha a qualidade ideal, repete-se o processo até que a qualidade esteja de acordo com o desejado.

E por isso é necessário dividir a base de dados em treinamento e teste. Na prática, é comum utilizar-se cerca de 70% dos dados para treinamento e 30% para teste, contudo, isso depende muito de cada projeto. Assim sendo, é importante garantir que a base de dados não seja inteiramente testada, pois isso pode causar *overfitting*, que acontece quando o algoritmo se adapta muito aos dados de treinamento. Uma solução para este problema, é usar a Validação Cruzada que, de uma forma dinâmica, faz com que todos os dados passem pelo treinamento e teste. Com isso, foi decidido para o projeto, após vários testes com diferentes divisões entre treinamento/teste, que a melhor configuração para estes dados seria a distribuição em 70% para treinamento e 30% para teste. Para isso, foi usada a função *train_test_split* da biblioteca *Sklearn*.

Abaixo, é possível visualizar tabelas que representam o score de um *cross_val_score*, parametrizando 5 *cross_validation* e escore pela acurácia, em cada classificador.

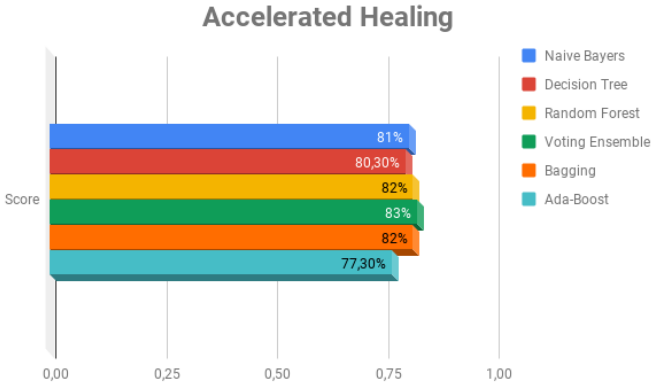

Figura 5. Tabela geral por score, Accelerated Healing

Ao analisar a atuação dos classificadores em cada atributo a partir das tabelas, verificamos que o atributo *Accelerated Healing* atingiu o objetivo proposto pela equipe, atingir ao menos 70% de acurácia. Em todos os modelos de classificação o atributo obteve valores maiores que 80% de acurácia, dando destaque para o *Voting Ensemble* que se beneficia da boa acurácia de outros modelos.

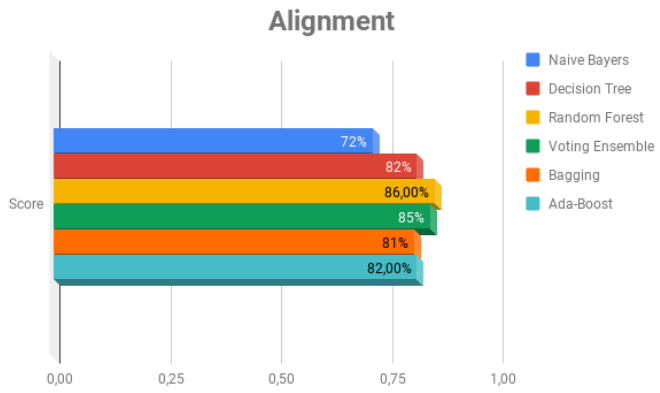

Figura 6. Tabela geral por score, Alignment

O atributo *Alignment* obteve sucesso no objetivo da equipe, todos atingiram ao menos 70% de acurácia. Para isso houve uma modificação em cada classe do atributo. Na primeira modificação, eliminou-se a classe *'bad'* transformando-o em atributo vago, representado por '-', assim o atributo foi binariado, trabalhando apenas com as classes *'good'* e *'neutral'*, além da transformação de atributo nominal para numérico. Com os restante, houve uma previsão por *KNN*. Assim, conclui-se que essa decisão no pré-processamento ajudou na melhora da acurácia do atributo, onde antes chegava a 69%, agora com o uso de métodos de *Ensemble*, chega a uma acurácia de 85%. Uum destaque para o modelo *Forest Random* que chegou a acurácia de 81%.

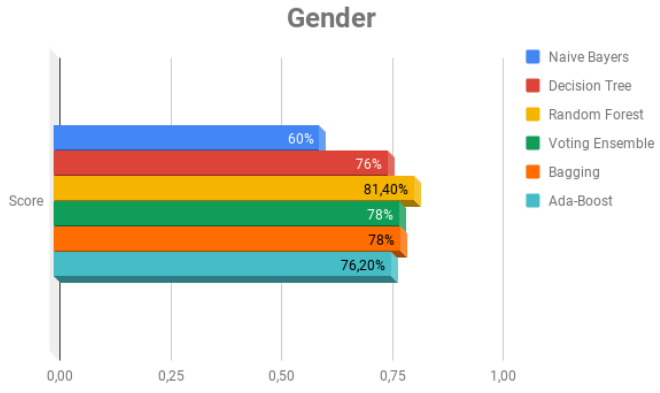

Figura 7. Tabela geral por score, Gender

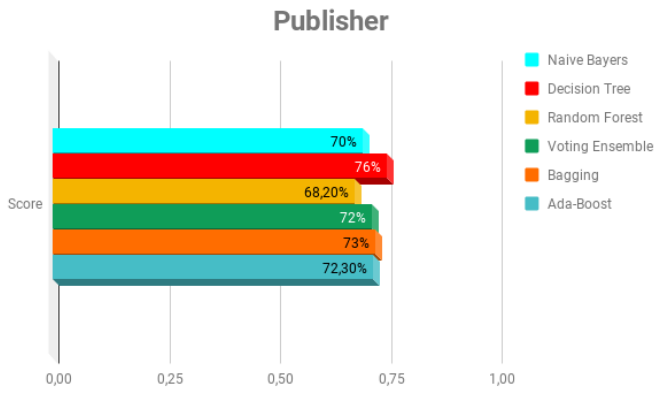

Figura 8. Tabela geral por score, Publisher

Os atributos *Gender* e *Publisher* tiveram acurácia abaixo de 70% em dois modelos. O *Gender* obteve 60% de acurácia no modelo *Naive Bayes*, o que difere do resultado dos outros modelos que chegaram a 81%, no caso do *Random Forest*. Já o *Publisher*, teve baixa no *Random Forest* atigindo 68% de acurácia.

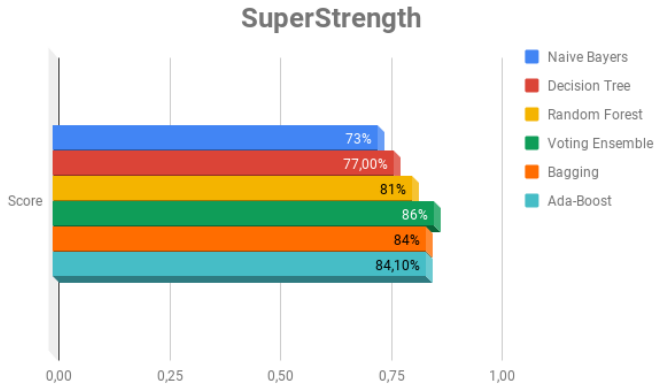

Figura 9. Tabela geral por score, SuperStrength

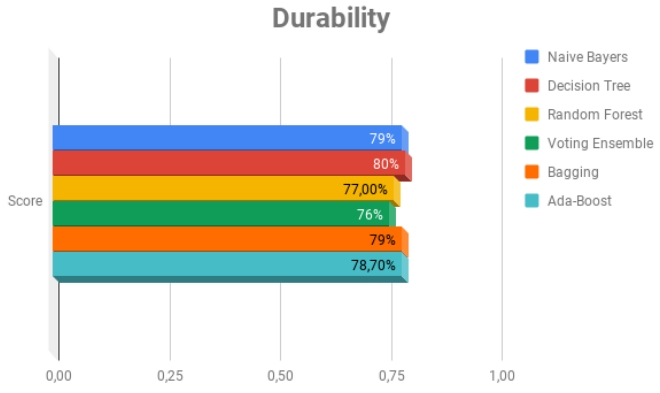

Figura 10. Tabela geral por score, Durability

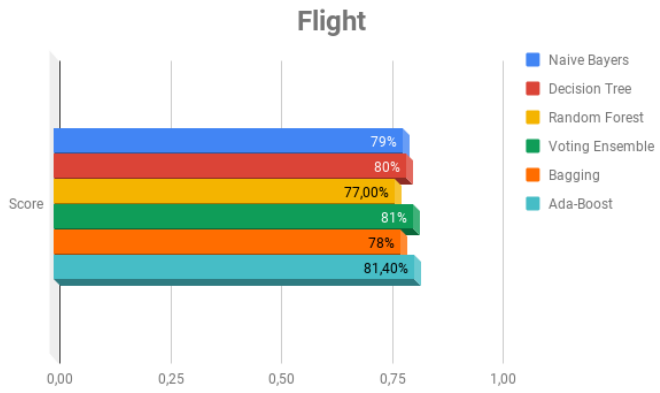

Figura 11. Tabela geral por score, Flight

Possivelmente por tratarem, antes do pré-processamento, de atributos *multiclass* ou por serem atributos nominais, *Gender* e *Publisher* obtiveram um baixo desempenho, comparado as outras classes, mesmo após o pré-processamento onde foram transformandos em atributos binários, no caso do *Publisher*.

Os atributos restantes, *Durability , SuperStrength e Flight* que representam superpoderes, obtiveram taxas satisfatórias em relação a todos os modelos. Atinge o objetivo de 70% de acurácia, e sobressaem-se nos métodos de Ensemble, tanto o modelo Bagging, chegando a 84% para SuperStrength, quanto ao AdaBoost chegando a 81% para o atributo *Flight*.

Como é possível observar nas figuras acima, em relação aos classificadores, o Random Forest aparenta ser o modelo que obteve os melhores resultados em termos gerais, porém sua média de score em todas as classes chegou a 78,94, enquanto que a Árvore de Decisão conseguiu uma média de 78,75%. Já Naive Bayes teve uma média de 73,42%, a mais baixa dentre os três. Assim, o Random Forest conseguiu ser melhor por muito pouco, e por isso ele teve um peso maior no método *VotingClassifier*; seus scores oscilaram um pouco, o maior chegando a 86% e o mais baixo a 68,2%. A Árvore de Decisão e Naive Bayes conseguiram oscilar menos em seus resultados.

V. Conclusões

De modo geral, o requisito de aplicação de múltiplos algoritmos de classificação e a execução de modelos de *Ensemble Learning* foi devidamente atendido. Os passos executados, o pré-processamento, o pós-processamento, os testes e análises das melhores configurações, auxiliaram a melhorar os resultados obtidos.

Ao analisar a Figura 12, a aplicação de Ensemble Learning em específico, é possível visualizar que a utilização destas técnicas aumentam a capacidade preditiva do projeto sobre a classe. Portanto, ao realizar também uma comparação entre os classificadores(Árvore de Decisão, Naive Bayes e Random Forest) e os modelos de Ensemble empregados(Voting-Classifier, Bagging e Adaboost), nota-se diferença entre os resultados obtidos. Os modelos de Ensemble melhoraram significativamente as predições dos classificadores, aumentando assim a confiança na predição para possíveis aplicações práticas.

ENSEMBLE LEARNING - KNN			
Ensemble	Voting Ensemble	Bagging	Ada-Boost
Score	86%	84%	84,10%

Evaluation - KNN preprocessing - SuperStrength			
Classifier	Naive Bayers	Decision Tree	Random Forest
Accurancy	73%	77,00%	81%
Score	79%	83,20%	85%
Recall	80,60%	87%	87%
Precision	81,20%	87,10%	87%
AUC	88%	89%	92%
F1	79%	84,60%	86%

Figura 12. Tabela de Avaliação Geral do atributo SuperStrength

Registrou-se o melhor poder de discriminação no classificador Random Forest (AUC 92%), e o melhor recall nos classificadores *Random Forest* e *Decision Tree* (REC 87%), confirmando, com outras avaliações, a eficácia dos classificadores neste atributo, bem como a eficiência do *Random Forest*.

Referências

[1] Documentação Pandas. Disponível em: https://pandas.pydata.org. Acesso em Julho de 2019.

[2] Documentação Numpy. Disponível em: https://numpy.org. Acesso em Julho de 2019.

[3] Documentação scikit-learn. Disponível em: https://scikit-learn.org/stable/. Acesso em Julho de 2019.

[4] Jones Granatyr. Machine Learning e Data Science com Python de A a Z. Disponível em: hhttps://www.udemy.com/machine-learning-e-data-science-com-python-y/. Acesso em Agosto de 2019.

[5] Nicolas L. Gentille. Aprendizado de máquina e caracterização de aterosclerose subclínica: um estudo de caso. Disponível em: http://cassiopea.ipt.br/teses/2017_EC_Nicolas-Gentille.pdf. Acesso em Julho de 2019.

[6] Documentação Python. Disponível em: https://www.python.org/doc/. Acesso em Julho de 2019.

[7] Babenko, D., e H. Marmanis. Algorithms of the Intelligence Web. Manning Publications, 2009.

[8] Bernardini, F. C. Combinação de classificação simbólicos para melhorar o poder preditivo e descritivo de ensembles. Dissertação de Mestrado, São Carlos: Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2002.

[9] Breiman, L. "Bagging Predictors." Machine Learning 24, 1996: 123 - 140.

[10] Pugliesi, J. B., R. A. Sinoara, e S. O. Rezende. Combinação de Regressores Homogêneos e Heterogêneos: Precisão e Compreensibilidade. Relatório Técnico, São Carlos: Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, 2003.

[11] Freund, Y., e R. E. Schapire. "A short introduction to Boosting." Journal of Japanese Society for Artificial Intelligence, Vol. 14, Nº5., 1999: 771 - 780.

[12] CHAVES, Bruno Butilhão. Estudo do algoritmo AdaBoost de aprendizagem de máquina aplicado a sensores e sistemas embarcados. Dissertação de Mestrado, Escola Politécnica, Universidade de São Paulo, 2011.

[13] Georgios Drakos. Handling Missing Values in Machine Learning: Part 2. Disponível em: https://towardsdatascience.com/handling-missing-values-in-machine-learning-part-2-222154b4b58e. Acesso em Agosto de 2019.