Explorando algoritmos: analisador de sentimentos em avaliações de companhias aéreas dos EUA

Rodrigo Bifulco

***Resumo*— Este trabalho apresenta um estudo que envolve a implementação e análise de três algoritmos distintos - LSTM bidirecional, árvore de decisão e Naive Bayes Multinomial - para a criação de um analisador de sentimentos a partir de uma base de dados contendo *tweets* de avaliações sobre experiências de passageiros de companhias aéreas dos EUA no ano de 2015. O algoritmo será capaz de classificar uma nova avaliação em positiva, negativa ou neutra. O objetivo é explorar e discutir as vantagens e desvantagens de cada algoritmo para essa tarefa específica.**

# Descrição do Problema

Com este trabalho espera-se obter uma compreensão mais clara sobre a eficácia e o desempenho dos algoritmos que permitirá uma escolha mais informada e embasada para a criação do analisador de sentimentos na base de dados específica. Isso também pode fornecer insights valiosos para aprimorar a compreensão das opiniões dos usuários e tomar decisões informadas com base nesses sentimentos expressos.

A base de dados contém 14452 *tweets* únicos de passageiros das companhias aéreas dos EUA e foi simplificada para conter apenas as duas variáveis principais para o estudo: *text* que representa o *tweet* escrito pelo passageiro para a companhia aérea e *category* que classifica o *tweet* em *positive* (2298 casos), *negative* (9087 casos) ou *neutral* (3067 casos).

# Métodos e Implementação

PRÉ-PROCESSAMENTO DOS DADOS

Após remover linhas duplicadas e verificar a existência de dados nulos os seguintes tratamentos foram aplicados aos textos escritos.

1. Conversão de todas as letras para minúsculo;
2. Remoção de hiperlinks e *retweets*;
3. Remoção de marcações de *tweets*;
4. Remoção dos demais caracteres especiais;
5. Tokenização dos textos: conversão do texto em lista de palavras onde cada palavra na lista é chamada de *token*.
6. Remoção de *stopwords* – palavras que aparecem com alta frequência nos textos, porém não trazem informações significativas. Exemplo: *that*, *this*; Devido à relevância, algumas *stopwords* da lista default de *stopwords* da biblioteca NLTK foram mantidas Exemplo: as palavras *down*, *few* e *don’t* podem ser úteis para identificar *tweets* com sentimentos negativos;
7. Aplicação do *stemming* que reduz as palavras ao seu radical ajudando a reduzir o tamanho do vocabulário criado;

Abaixo segue um exemplo de um texto que foi pré-processado utilizando os passos acima.

Texto original

@VirginAmerica current bug on website shows ‘select departure city’ when selecting destination city http://t.co/SLLYIBE2vQ

Texto pré-processado

['current', 'bug', 'websit', 'show', 'select', 'departur', 'citi', 'select', 'destin', 'citi']

Para que o algoritmo consiga trabalhar com palavras é necessário convertê-las em vetores numéricos, também chamados de *word embeddings*. Como destacado por Mikolov em [1], "*word embeddings* representam palavras em um espaço vetorial onde palavras semelhantes são mapeadas para pontos próximos", permitindo que modelos de PLN entendam o significado e a similaridade entre palavras com base em sua distribuição nos dados. Para gerar os *word embeddings* de cada *tweet* foram utilizadas as seguintes técnicas (ambas nativas da biblioteca Keras).

*Tokenizer*: converte os *tweets* em vetores de inteiros onde cada palavra será substituída pelo valor correspondente a sua posição no vocabulário de palavras - este por sua vez ordena as palavras que aparecem com mais frequência em ordem decrescente. Para este trabalho foi utilizado um vocabulário com as 5000 palavras mais frequentes em todo corpus (todas as palavras existentes em todos os textos).

*Padding*: preenche o vetor com zeros para que a dimensão de todos os vetores seja a mesma - no caso, 50. O parâmetro *padding* foi ajustado para *post*, ou seja, os zeros foram preenchidos ao final do vetor para se chegar a um vetor com a dimensão desejada.

Ao final do pré-processamento da variável *text* cada entrada foi representada como um vetor numérico de dimensão 50 com variabilidade de 5000 palavras distintas.

O tratamento utilizado na variável *category* foi a criação de variáveis *dummy*, ou seja, foram inseridas 3 colunas ao conjunto de dados original representando cada uma das classes do problema. Os valores dessas colunas são binários e têm valor 1 para a coluna da classe correspondente à classe da linha observada e 0 para as demais colunas.

A próxima etapa foi separar os dados em conjunto de treino, validação e teste. A separação adequada dos dados em conjuntos de treinamento, validação e teste desempenha um papel crucial no desenvolvimento de modelos de aprendizado de máquina. Como mencionado por Brownlee em [2] essa prática é fundamental para avaliar e ajustar o desempenho do modelo de forma justa. O conjunto de treinamento é usado para ajustar os parâmetros do modelo, permitindo que ele aprenda com os padrões dos dados. O conjunto de validação é empregado para ajustar os hiperparâmetros do modelo e evitar o super ajuste (*overfitting*) aos dados de treinamento bem como vazamento dos dados (*data leakage*). Por fim, o conjunto de teste é utilizado para avaliar o desempenho final do modelo em dados não vistos anteriormente. A correta separação dos dados em treino, validação e teste é essencial para garantir que o modelo seja confiável e generalizável. Para este caso foi utilizada uma separação em treino (64% dos dados originais), validação (16% dos dados originais) e teste (20% dos dados originais).

MODELAGEM

LSTM Bidirecional

O modelo LSTM (*Long Short-Term Memory*) bidirecional é amplamente utilizado em processamento de linguagem natural devido à sua capacidade de capturar informações contextuais em ambos os sentidos de uma sequência de texto. Essa arquitetura, como mencionado por Hochreiter and Schmidhuber em [3] combina células LSTM que processam a sequência na ordem direta e inversa, permitindo que o modelo capture relações entre palavras anteriores e posteriores.

Foi desenvolvido um modelo de rede neural sequencial utilizando a biblioteca Keras. Inicialmente, foi adicionada uma camada de *Embedding*, responsável por mapear os índices das palavras em vetores densos de números reais de dimensão 50. Em seguida, uma camada de convolução 1D foi inserida para aplicar 32 filtros convolucionais de tamanho 3x3 e extrair características relevantes dos dados de entrada. Após a camada de convolução, foi utilizada uma camada de *max-pooling1D* para reduzir a dimensionalidade dos recursos extraídos, preservando as informações mais importantes. Uma camada de LSTM bidirecional com 32 blocos foi então adicionada ao modelo para capturar informações contextuais de longo alcance, processando as sequências de entrada em ambas as direções.

Com o objetivo de evitar *overfitting* e maximizar a capacidade de generalização do algoritmo , uma camada de *dropout* foi incluída para desativar aleatoriamente alguns neurônios e conexões sinápticas durante o treinamento. Por fim, uma camada densa com ativação *softmax* foi adicionada para produzir uma distribuição de probabilidade sobre as 3 classes do problema. O modelo foi compilado utilizando a função de perda *categorical\_crossentropy*, adequada para problemas de classificação multiclasse. O otimizador escolhido foi o *Stochastic Gradient Descent* (SGD) com momentum para tornar a trajetória do gradiente mais suave e acelerar a convergência do algoritmo. A métrica de avaliação selecionada foi a acurácia.

Por fim foi aplicada validação cruzada com 3 *folds* com busca em grade variando a quantidade de épocas de treinamento e porcentagem de *dropout* com intuito de tornar os resultados de acurácia mais robustos e encontrar a melhor combinação desses hiperparâmetros. Para avaliar o modelo utilizando o conjunto de dados de teste foi treinada uma nova rede neural com os melhores hiperparâmetros encontrados na etapa de treinamento.

Árvore de decisão

Uma árvore de decisão é um modelo de aprendizado de máquina que utiliza uma estrutura em forma de árvore para representar decisões baseadas em dados. Ela é amplamente utilizada devido à sua capacidade de interpretar e classificar dados complexos. Segundo Mitchell [4], uma árvore de decisão é "uma estrutura de fluxo de controle hierárquica, composta por nós de decisão e nós de ação, onde cada nó de decisão contém um teste em um atributo dos dados e cada nó de ação representa uma decisão ou resultado final". Essa estrutura em árvore permite a representação visual dos processos de tomada de decisão e facilita a interpretação dos resultados obtidos.

Foi construído um modelo de classificação baseado em árvore de decisão aplicando validação cruzada com 5 *folds* e busca em grade variando os seguintes hiperparâmetros da árvore: profundidade máxima da árvore (*max\_depth*), quantidade mínima de amostras para criar uma ramificação nos nós de decisão (*min\_sample\_split*), quantidade mínima de amostras para que um nó seja considerado um nó folha (*min\_sample\_leaf*) e peso que cada uma das classes terá na classificação final (*class\_weight*) para lidar com desbalanceamento de classes.

Naive Bayes Multinomial

O Naive Bayes Multinomial é um algoritmo de classificação probabilístico amplamente utilizado em aprendizado de máquina. Especificamente projetado para lidar com dados discretos e contar a frequência das ocorrências, esse algoritmo é frequentemente aplicado em problemas de classificação de texto e processamento de linguagem natural. Segundo McCallum e Nigam [5], o Naive Bayes Multinomial assume a independência condicional entre os atributos, dada a classe, e utiliza a distribuição multinomial para modelar a probabilidade de ocorrência dos atributos em cada classe. Essa abordagem simplifica o cálculo das probabilidades e torna o Naive Bayes Multinomial um classificador eficiente em conjuntos de dados com alta dimensionalidade e presença de termos repetidos.

Para este modelo de classificação também foi utilizada validação cruzada com 5 *folds* e busca em grade variando os seguintes hiperparâmetros: correção de Laplace (*alpha*) responsável por evitar a probabilidade zero de ocorrência de uma palavra em um texto e (*class\_prior*) que representa o peso que cada classe terá na classificação final para lidar com desbalanceamento de classes.

# Resultados

Os melhores hiperparâmetros encontrados para cada algoritmo foram:

Para a LSTM Bidirecional: {*'dropout'*: 0.4, *'epochs'*: 40}

Para a árvore de decisão: {*'class\_weight'*: *None*, '*max\_depth*': 10, *'min\_samples\_leaf'*: 1, '*min\_samples\_split*': 2}

Para o Naive Bayes Multinomial: {*'alpha'*: 0.1, '*class\_prior*': [0.5, 1.2, 0.8]}

Um ponto a ser destacado é que a árvore de decisão obteve o melhor valor de acurácia utilizando *None* no hiperparâmetro *class\_weight*, ou seja, o desbalanceamento de classes não foi um problema para o algoritmo. Já para o Naive Bayes Multinomial foi necessário balancear os pesos de cada classe.

Os resultados de acurácia utilizando o conjunto de dados de teste, quantidade de modelos executados e tempo de processamento total para os três algoritmos estão apresentados na tabela abaixo.

OBS: Para executar o classificador criado com o Naive Bayes Multinomial foi necessário usar a variável *text* com seus valores originais e não como variáveis *dummy*.



Tabela 1 – Resultados obtidos pelos três algoritmos propostos

Nota-se que o algoritmo LSTM Bidirecional obteve o melhor valor de acurácia e maior custo computacional. Mesmo variando vários hiperparâmetros da árvore de decisão ela não foi capaz de gerar uma boa acurácia.

O Naive Bayes Multinomial lida bem com dados esparsos. Como melhoria no pré-processamento dos dados de entrada desse algoritmo foi feita a conversão das sequências numéricas resultantes da função *Tokenizer* em matrizes de frequência de atributos, onde cada valor representa a contagem de ocorrência de um atributo específico para cada instância de texto. Anteriormente a variável *text* foi utilizada com vetores de dimensão 50 onde cada posição continha o índice correspondente a palavra no vocabulário dada sua frequência. Após a conversão efetuada os vetores passaram a ter dimensão 5000, onde as colunas representam todas as palavras do vocabulário (*max\_words*) e os valores continuaram representando a frequência das palavras em todo vocabulário.

Com intuito de melhorar o resultado da árvore de decisão foi utilizado a variável *category* com seus valores originais e não mais como variáveis *dummy*. Após esses ajustes a árvore de decisão e o Naive Bayes Multinomial foram executados novamente e os resultados são apresentados na tabela abaixo.



Tabela 2 - Resultados após efetuar os ajustes na árvore de decisão e Naive Bayes Multinomial

Nota-se que a árvore de decisão teve um aumento de 4,1% na acurácia e o Naive Bayes Multinomial um aumento de 28,8%, o que indica que ele conseguiu lidar bem com matrizes de entrada esparsas e frequência de palavras. O tempo de processamento de ambos os algoritmos aumentou.

Abaixo seguem os novos melhores hiperparâmetros encontrados para os dois algoritmos ajustados.

Para a árvore de decisão: {'*class\_weight'*: *None*, '*max\_depth*': 5, '*min\_samples\_leaf*': 2, '*min\_samples\_split*': 2}

Para o Naive Bayes Multinomial: {*'alpha'*: 0.5, '*class\_prior*': *None*

Nota-se que agora ambos os algoritmos estão com hiperparâmetro *class\_weight* *None* portanto não foi necessário ponderar as classes. Também é possível notar que a árvore que gerou o melhor resultado foi menos profunda (antes o *max\_depth* foi de 10).

# Discussão

Após executar os 3 algoritmos propostos e efetuar alguns ajustes conclui-se que a LSTM Bidirecional obteve o melhor resultado dado que ela analisa os textos de entrada tanto na ordem direta quanto na ordem reversa capturando informações contextuais tanto do passado quanto do futuro de cada palavra na sequência e possui um mecanismo onde é possível armazenar memórias de longo prazo que facilitam a interpretação do texto e captura de semântica. Em contrapartida este foi o algoritmo com custo computacional mais elevado para executar o menor número de modelos e ajustar seus hiperparâmetros e arquitetura é uma tarefa bastante desafiadora.

A árvore de decisão obteve um bom custo computacional dado que executou 500 modelos em um curto período de tempo, porém foi o algoritmo que obteve o menor valor de acurácia mesmo efetuando alguns ajustes e também necessita do ajuste de muitos hiperparâmetros para evitar o crescimento exagerado da árvore que leva ao *overfitting*.

O Naive Bayes Multinomial obteve um bom resultado de acurácia e baixo custo computacional além de ser o mais simples de ser implementado, com poucos hiperparâmetros para serem ajustados. O ponto negativo dele é que considera que todas as palavras são independentes entre si, o que não é verdadeiro e pode atrapalhar a criação da semântica das palavras.

Portanto, o melhor algoritmo em termos de acurácia foi a LSTM Bidirecional e o Naive Bayes Multinomial foi um bom concorrente dado que obteve um valor de acurácia próximo ao valor obtido pela LSTM Bidirecional e custo computacional e complexidade extremamente baixos.

##### Referências

##### [1] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. ArXiv, abs/1301.3781)

##### [2] Brownlee, J. (2017). Why and How to Divide Your Data into Train, Validation, and Test Sets. Machine Learning Mastery. Recuperado https://machinelearningmastery.com/difference-test-validation-datasets/.

##### [3] Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural computation, 9(8), 1735-1780.

##### [4] Mitchell, T. M. (1997). Machine Learning. McGraw-Hill Education.

##### [5] McCallum, A., & Nigam, K. (1998). A comparison of event models for naive Bayes text classification. In AAAI-98 Workshop on "Learning for Text Categorization" (Vol. 752, No. 1, pp. 41-48)