Nanodegree Engenheiro de Machine Learning

Projeto final

Yuri Martins Campolongo
03 de abril de 2018

Sumário

I. Definição

Visão geral do projeto

Uma das grandes utilizações de machine Learning nos últimos tempos, e que se mostra cada vez mais promissora é a interpretação de linguagem natural. Essa técnica se tornou muito útil e procurada nos últimos tempos devido ao grande aumento do uso de chatbots, que são robôs de atendimento que interpretam textos digitados pelo cliente, de maneira natural, para realizar a busca de possíveis soluções para aquele problema.

De acordo com grandes influenciadores do mercado de tecnologia como Mark Zuckerberg (Facebook) o autoatendimento trará mais eficácia das interações entre usuários e empresas, diminuindo o tempo de resposta e auxiliando clientes na resolução de problemas, essa definição foi dada por ele em 2016 durante um evento para desenvolvedores.

A Criação de um algoritmo para a análise de linguagem natural surgiu pois atualmente trabalho em uma companhia que cria sistemas de atendimento para empresas, e após análise da área comercial, verifiquei que as dúvidas dos possíveis novos clientes que entram em contato via chat, podem ser classificadas e posteriormente ensinadas para um algoritmo de aprendizagem, que pode também utilizar essas classificações para exibir a melhor resposta, sem a necessidade de uma pessoa para responder dúvidas frequentes e recorrentes.

Descrição do problema

O objetivo é criar um algoritmo de processamento de linguagem natural capaz de classificar corretamente entradas de dúvidas de clientes em um chat, esse algoritmo poderá se tornar útil para a empresa na qual eu trabalho pois trabalhamos diariamente com atendimentos e com grandes volumes, com o algoritmo funcionando e com uma taxa alta de acuracidade será possível fornecer esse aprendizado para outros sistemas que poderão responder automaticamente para os clientes, sem a necessidade de uma atendente humana respondendo questões repetitivas que consomem tempo, liberando essa atendente para atendimentos mais personalizados e específicos.

Para atingir o objetivo citado acima, foi obtido um dataset com dúvidas que possíveis futuros clientes da empresa têm, e que entram em contato utilizando o site da empresa. Cada dúvida dessa foi classificada manualmente como um tipo de dúvida relacionada a um dos produtos que a empresa fornece, com esse dataset, e os passos mostrados no tópico **'pré-processamento de dados'** poderemos avaliar a criação de um algoritmo de linguagem natural que atenda a necessidade.

Métricas

Para medir e avaliar a qualidade do algoritmo foi utilizada a métrica F1-Score, segundo a biblioteca sklearn (scikit-learn, 2018) essa métrica utiliza a média ponderada da precisão e da revocação, variando entre 1 e 0, sendo 1 a melhor nota possível e 0 a pior.

A fórmula para o cálculo do F1-Score é:

$$F1 = 2 \cdot \frac{(precisão * revocação)}{(precisão + revocação)}$$

Para facilitar a visualização de possíveis erros no algoritmo foi também criada uma matriz de confusão, essa tabela mostra a performance da seguinte forma:

- Cada linha da tabela representa a classe que foi prevista pelo algoritmo, chamaremos de X;
- Cada coluna representa a classe real, chamaremos de Y;
- As células mostram quantas instâncias de X que foram previstas corretamente em relação a Y.

Com essa tabela é simples verificar a performance do algoritmo, pois apenas os valores na diagonal da tabela indicam as previsões corretas, enquanto todas as outras células indicam os erros (Wikipedia, 2018).

É interessante usar as duas formas, pois o F1 Score dá apenas um valor que indica a performance do seu algoritmo, porém não é possível visualizar em qual ponto a performance está sendo reduzida, com a matriz de confusão é possível verificar isso, e visualizar o porquê uma classe está com problemas de previsão e corrigir esse ponto em específico, seja por meio de parametrização do algoritmo ou até mesmo revisão do dataset.

II. Análise

Exploração dos dados

O dataset utilizado para a criação do algoritmo contém 2 colunas de dados separados por vírgula (CSV).

- A primeira coluna representa os textos que foram digitados por possíveis clientes com dúvidas sobre os produtos fornecidos pela empresa na qual eu trabalho.
- A segunda coluna representa o tipo de dúvida sobre qual tipo de produto aquele texto informado pelo cliente representa.

Foram obtidos 172 registros, separados em 11 categorias diferentes com a seguinte distribuição:

- Categoria CHAT – 20 registros
- Categoria CHATBOT – 23 registros
- Categoria COST – 20 registros
- Categoria DISCADOR – 20 registros
- Categoria ME_ATENDE – 17 registros
- Categoria MULTI_CHANNEL – 5 registros
- Categoria PRODUTO – 17 registros
- Categoria SMS – 14 registros
- Categoria TRIAL – 7 registros
- Categoria URA – 18 registros

- Categoria VOS_PCI – 11 registros

Os dados foram previamente tratados para remover informações de nome dos clientes, nome das empresas e outras informações sobre números de documentos e informações pessoais.

Visualização exploratória

Os gráficos abaixo mostram a distribuição dos dados no dataset

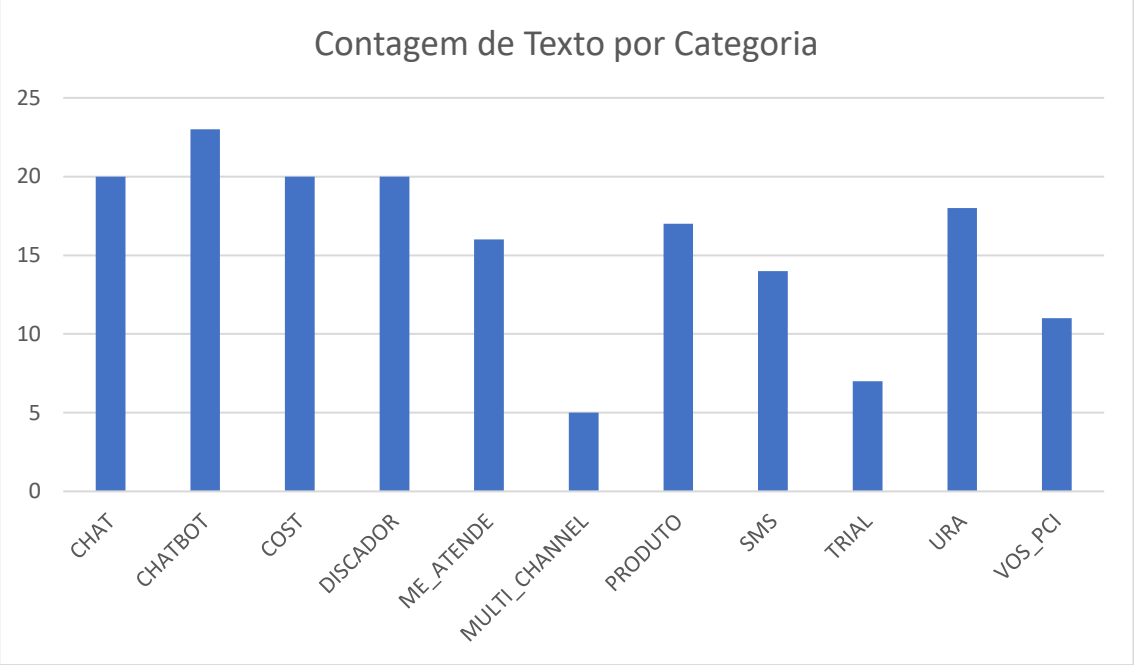

Figura 1 - Contagem de texto por categoria

Observamos que as categorias MULTI_CHANNEL e TRIAL possuem poucos dados, isso pode tornar necessário aplicar a técnica de oversampling afim de aumentar a quantidade de entradas para essas categorias, visando aumentar a acuracidade do algoritmo.

Algoritmos e técnicas

Existem diversos algoritmos que poderiam ser utilizados para realizar a classificação proposta, como: SVM, Redes neurais, etc. Porém neste projeto, irei aplicar algoritmos baseados no teorema de Bayes, devido a sua simplicidade e ampla utilização em algoritmos de classificação para processamento de linguagem natural.

Irei utilizar o algoritmo baseado no teorema de Bayes, o Multinomial Naive Bayes, abaixo uma breve descrição desse algoritmo

- O Multinomial Naive Bayes, como o próprio nome diz, utiliza para cada feature uma distribuição multinominal. E de acordo com o Sckiti-Learn, mesmo que essas distribuições multinominais tem como requerimentos features com valores inteiros, valores fracionados como os que o TF-IDF retorna, também funcionam.

Benchmark

Os dados serão comparados com as respostas de um modelo de regressão logística, para comparar se a utilização de algoritmos bayesianos realmente são a melhor forma de tratar esse problema. A métrica para avaliação de acuracidade será utilizando o f1 score: isso nos dará uma porcentagem de acuracidade que indica se o algoritmo está conseguindo prever corretamente as corretas respostas para as dúvidas dos clientes.

III. Metodologia

Pré-processamento de dados

Os dados de entrada passarão pelo seguinte processo para prepará-los para o algoritmo:

- A coluna com a dúvida do cliente passará por um pré-processamento de texto, com o objetivo de remover palavras que não são importantes para o contexto, ou seja, palavras sem significância semântica, essas palavras estarão disponíveis em um arquivo .csv que o algoritmo utilizará para removê-las dos textos em questão.
- Conforme analisado no tópico 'Visualização exploratória', temos algumas categorias com poucos registros, para solucionar esse problema, será aplicada a técnica SMOTE (Synthetic Minority Over-sampling Technique).
  Essa técnica funciona da seguinte forma: Para a classe com poucas entradas, serão criados exemplos sintéticos, para isso, os dados da classe serão utilizados juntamente com a técnica *k-neighbors.* Os passos para isso são: Obter a diferença entre o vetor de entrada e os seus 'vizinhos' mais próximos; Multiplicar a diferença por um número aleatório entre 0 e 1, adicionar isso no o vetor de entradas;
  Esses passos fazem com que sejam gerados novos pontos entre as diferentes classes, o que torna a classe com poucos dados um pouco mais generalizada. (Nitesh, Kevin, Lawrence , & W. Philip, 2002)
- As palavras restantes passarão por um processo chamado de TF-IDF que é uma sigla em inglês para term-frequency-inverse document frequency, ou seja, frequência do termo–inverso da frequência nos documentos, o intuito dessa técnica é indicar a importância de cada palavra contida em um documento, analisando todos os documentos disponíveis. Com isso, poderemos definir um conjunto de treinamento que pode ser aplicado para os algoritmos bayesianos que serão utilizados. (Scikit-Learn, 2018)
- Treinar um classificador bayesiano multinomial com os vetores de entrada.

Implementação

O processo de implementação passou pelas seguintes etapas:

- Carregamento dos datasets necessários;

- Pré-processamento de dados;
- Aplicação do Oversampling;
- Treinamento do classificador bayesiano;

O detalhamento de cada processo e a parte relacionada a esse processo no Jupyter notebook do projeto estão descritas abaixo:

O **Carregamento dos datasets necessários** está no tópico 1.1 do Jupyter notebook do projeto e o fluxo dessa parte é:

1. Importa a biblioteca *panda*, que é utilizada para análise de dados;
2. Carrega o arquivo dataset.csv em memória, esse arquivo é o responsável por guardar as entradas principais do algoritmo, e está mais detalhado no tópico **Análise de dados;**
3. Carrega o arquivo exclude.csv em memória, esse arquivo é o responsável por guardar as palavras sem importância semântica para esse processo em específico, e que devem ser removidas previamente afim de reduzir a quantidade de dados que serão trabalhados pelo algoritmo;

A execução do trecho 1.1 do Notebook nos mostra a quantidade de registros contidos no arquivo dataset.csv, que na execução foi de 172 linhas.

O **pré-processamento de dados** está no tópico 1.2 do Jupyter notebook do projeto e o fluxo dessa parte é:

1. Importa a biblioteca *re* que é utilizada para a execução de expressões regulares;
2. Um laço nos registros do dataset carregado no passo anterior é executado e, para cada registro:
    a. O conteúdo é alterado apenas para letras minúsculas;
    b. Remove a pontuação e caracteres especiais do texto usando expressão regular;
    c. Remove as palavras sem importância semântica que estão contidas no arquivo exclude.cvs (carregado em memória no passo anterior)
3. São atribuídas duas variáveis, X e y, respectivamente sendo responsáveis por guardar os dados de texto de entrada e categoria do texto.

A **aplicação de oversampling** está no tópico 1.3 do Jupyter notebook do projeto, e o fluxo dessa parte é:

1. Carrega a biblioteca SMOTE, que é a responsável por aplicar o oversampling nos dados;
2. Carrega a biblioteca TfIdfVectorizes do Scikit Learn, responsável por aplicar a técnica TF-IDF no dataset;
3. Realiza o processo de TF-IDF no dataset, transformando os valores da variável X (atribuída no passo anterior), calculando a frequência e importância de cada termo;

4. Aplica a técnica SMOTE, ampliando a quantidade de registros que temos no dataset. O gráfico abaixo mostra a quantidade de registros que foi inserida por categoria:

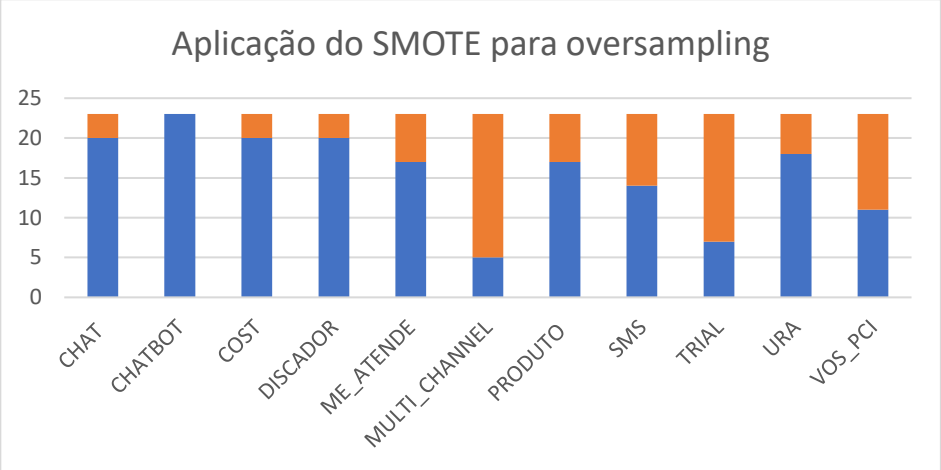

Figura 2 - Aplicação do SMOTE e resultados

A barra azul no gráfico acima mostra os registros que tínhamos anteriormente enquanto a barra laranja mostra o quanto de cada categoria que foi adicionada com a técnica. Podemos observar que o algoritmo SMOTE equalizou todas as categorias colocando cada uma com 23 registros, isso ocorreu pois a categoria original que tinha o maior número de dados era a 'CHATBOT' com 23 registros ao total. As categorias que mais ganharam entradas sintéticas foram: MULTI_CHANNEL, TRIAL e VOS_PCI;

O **treinamento do classificador bayesiano** está no tópico 2.1 do Jupyter notebook do projeto, e o fluxo dessa parte é:

1. Carrega a biblioteca *train_test_split* do scikit-learn;
2. Carrega a biblioteca *MultinomialNB* também do scikit-learn;
3. Realiza o split do dataset alterado no passo anterior pelo SMOTE em 4 variáveis: X_train, X_test, y_train, y_test, sendo a X_train a que guarda os dados de treinamento relacionados aos textos de entrada e y_train a que guardaas categorias desses textos, as variáveis X_test e y_test seguem a mesma ideia. Foi separado 25% dos 253 registros para testes;
4. É criado um classificador do tipo Multinomial Naive Bayes
5. O classificador é treinado com os dados de X_train e y_train;

Refinamento

O processo de refinamento para o treinamento do classificador bayesiano foi feito aplicando o parâmetro alpha, esse parâmetro indica se o algoritmo irá utilizar a técnica *Laplace Smoothing* caso esse valor seja próximo de 1, ou próximo de 0 caso não seja utilizado *smoothing*, isso significa que, caso a entrada e a classe nunca ocorram juntas no TF-IDF, a probabilidade estimada será 0, o parâmetro em questão ajusta esse

valor, inserindo uma correção nessa probabilidade fazendo com que ela nunca seja zerada, isso é importante pois probabilidades zeradas podem remover dados para outras probabilidades que venham a ser calculadas. (Wikipedia the free encyclopedia, 2018)

Ao aplicar o parâmetro *alpha* foram comparados os valores decimais entre 0 e 1 e o algoritmo foi treinado e o f1 score calculado, os resultados foram:

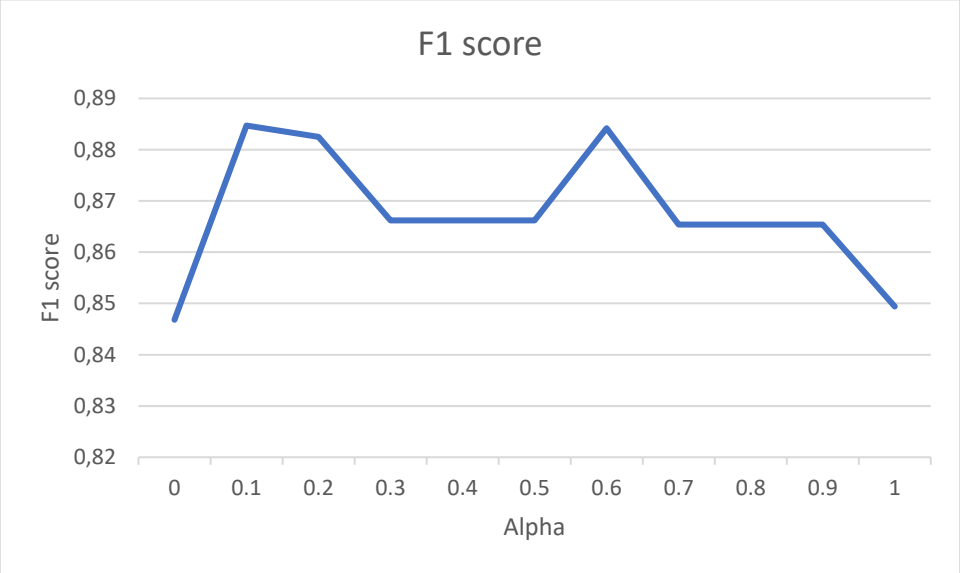

Figura 3 - F1 score alteração de alpha

De acordo com o gráfico acima, um valor menor de alpha se mostrou ter uma leve melhora em relação a valores mais altos, o valor padrão era 1, e esse se mostrou ser um dos piores valores possíveis, por isso, o algoritmo será treinado com o alpha sendo 0.1.

No pré-processamento de dados, foi removido o passo de remover palavras sem importância semântica e 10 execuções foram comparadas com antes e depois, lembrando que o parâmetro alpha do Multinominal Naive Bayes já estava em 0.1 durante todas essas execuções

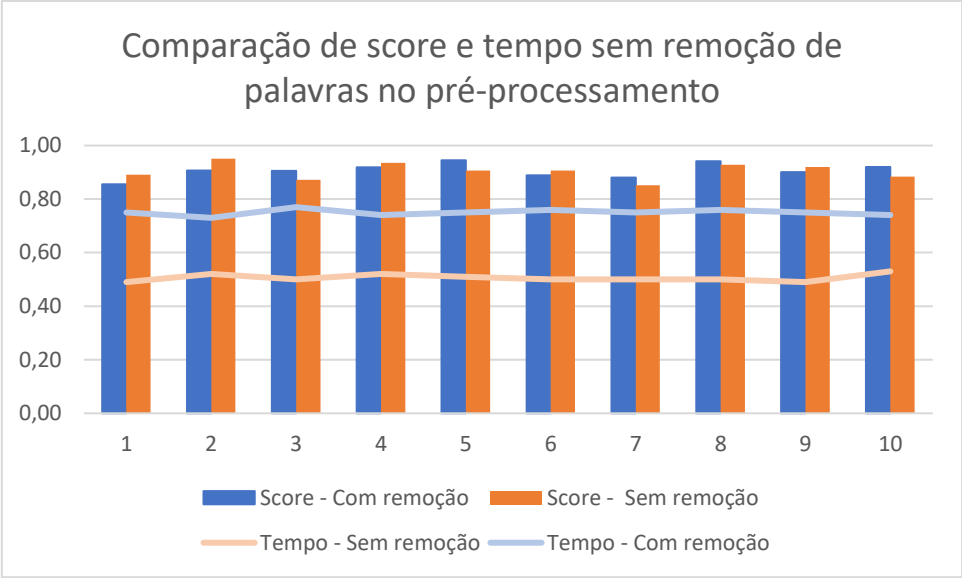

Figura 4 - Comparação de score e tempo sem remoção de palavras no pré-processamento

O gráfico acima mostra que o score do algoritmo não variou muito com a remoção de palavras, e no geral, remover palavras obtém um score mais baixo e gasta mais tempo de processamento, mesmo que esse tempo seja baixo (20ms a mais normalmente), é um processamento desnecessário e que pode ser evitado. Portanto, no projeto final, a remoção de palavras e o carregamento do dataset 'exclude.csv' será descartada.

IV. Resultados

Modelo de avaliação e validação

Durante o desenvolvimento foi utilizado como modelo para comparação o treinamento do mesmo dataset em um algoritmo de regressão logística.

Após o processo de refinamento, a arquitetura final disponível no notebook se mostrou a que possui o maior score, e o modelo se mostrou robusto pois a alteração de parâmetros e pré-processamento de dados não causou grandes perturbações no score final, variando de 0.80 até 0.95 no F1 score, o que são scores bem altos.

De acordo com a matriz de confusão, a categoria que se mostrou mais problemática durante todas as execuções foi a MULTI_CHANNEL, isso se deve ao fato de que no dataset original haviam poucas entradas dessa categoria, 5, e mesmo após a aplicação do algoritmo SMOTE, que aumentou esse número para 23, o número de erros continuou alto, conforme imagem da matriz de confusão abaixo.

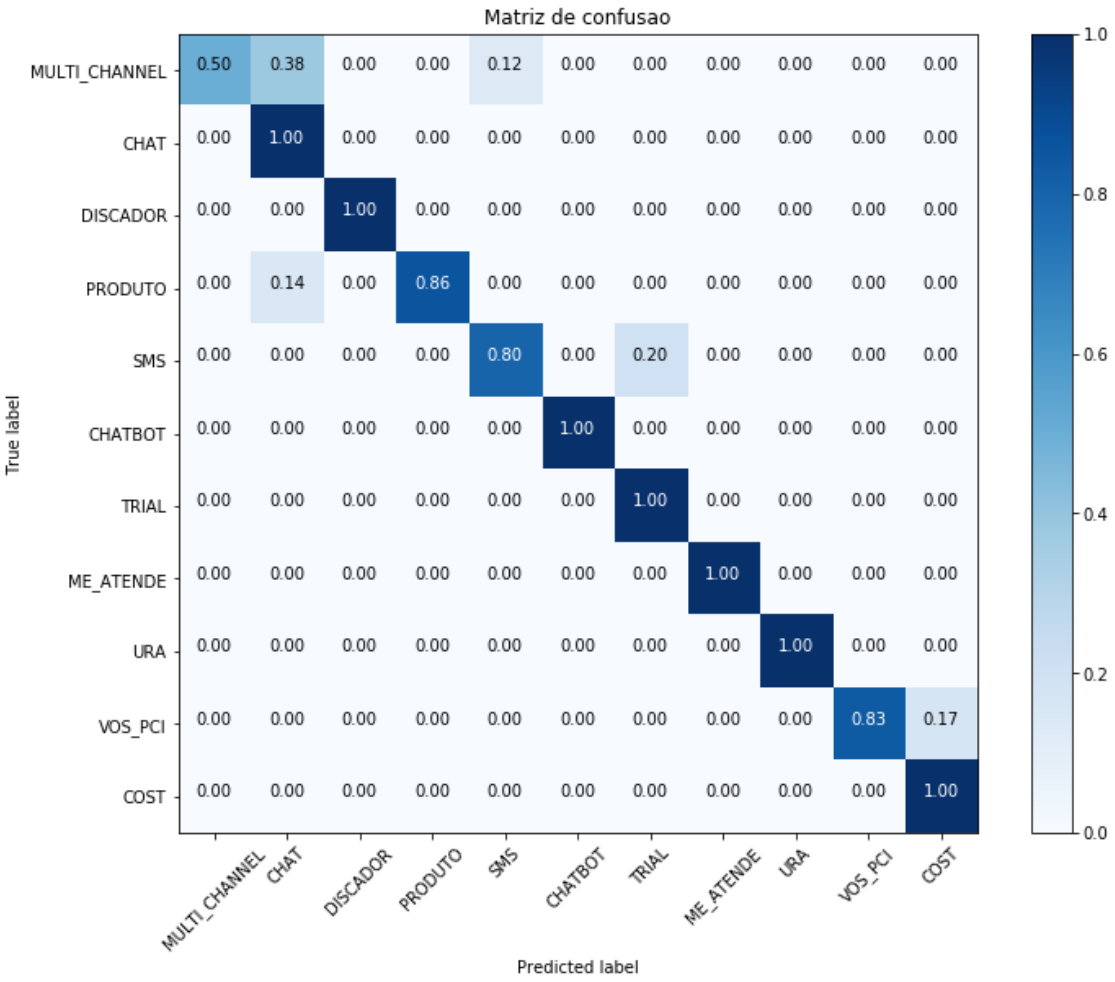

Figura 5 - Matriz de confusão

Conforme a imagem acima, vemos que apenas metade dos testes da categoria MULTI_CHANNEL foram bem sucedidos, houve muito erro de confusão dessa categoria em relação a categoria CHAT.

Justificativa

Inicialmente o modelo final utilizando o algoritmo Multinomial Naive Bayes estava com uma performance inferior ao modelo de benchmark em cerca de 2%, o que não justificaria a utilização do modelo proposto, porém após o processo de refinamento e análise dos parâmetros do algoritmo e pré-processamento de dados, esse valor aumentou e a performance do modelo proposto superou a do modelo de benchmark com uma variação de 1% até 8% superior, sendo que durante todas as execuções, o modelo proposto após refinamento nunca obteve um F1 score inferior ao do modelo de benchmark.

Os dois modelos se mostraram bastante robustos, sendo que mesmo após manipulações no pré-processamento e diversas execuções com variações de tamanho de dataset o F1 score nunca teve variações drásticas em nenhum caso.

No geral a solução apresentada se mostrou útil para a resolução do problema com uma alta margem de confiança, o que indica que essa solução pode ser utilizada para o problema final.

V. Conclusão

Forma livre de visualização

Algumas entradas manuais foram enviadas para previsão pelo classificador final, o resultado da matriz de confusão foi:

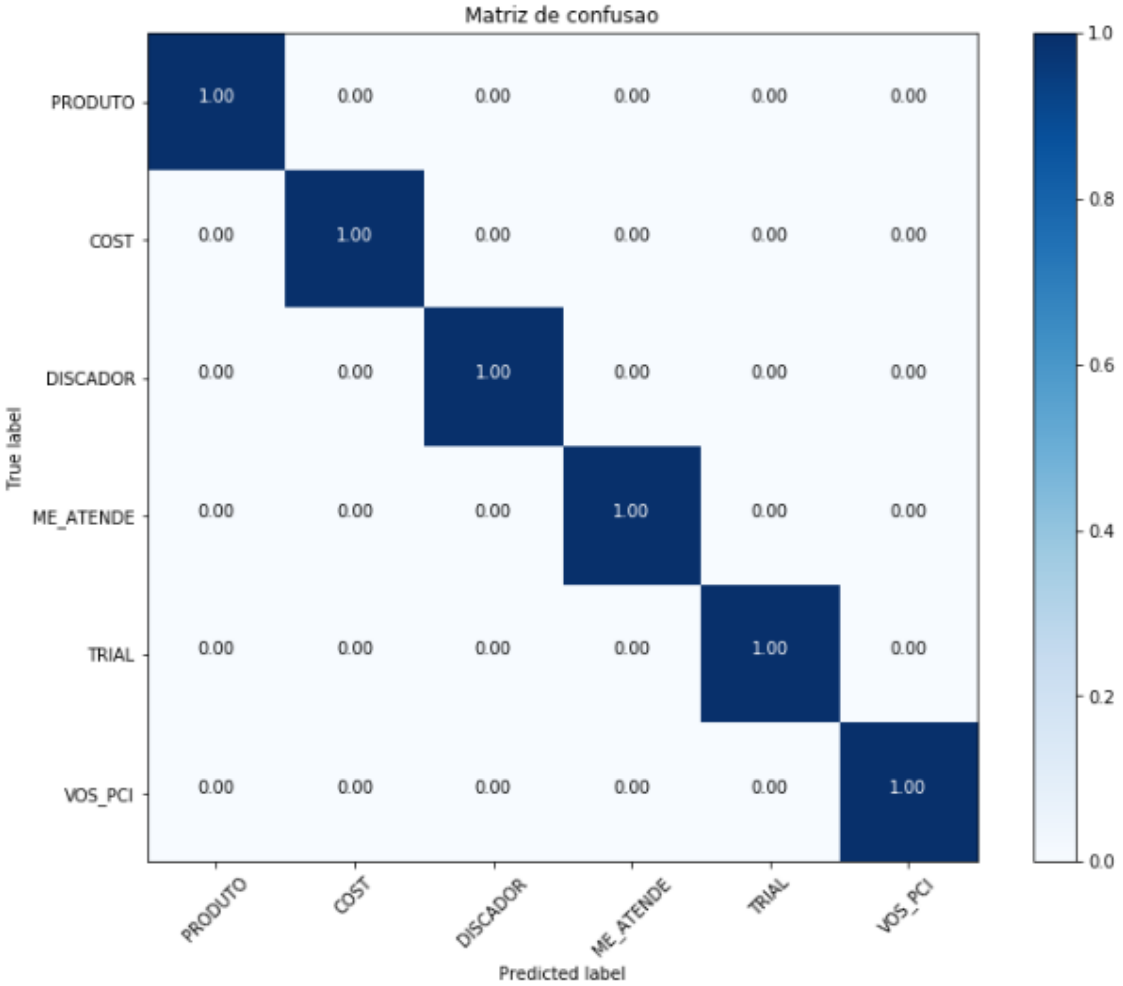

Figura 6 - Matriz de confusão dados manuais de teste

Como podemos observar, todas as entradas de teste manuais foram identificados e previstas com sucesso pelo classificador, durante alguns testes, foi possível identificar que algumas palavras que não tem tanta importância semântica causaram alguns erros durante outras execuções, como por exemplo a palavra ‘saber’ que estava causando confusão no classificador em relação a ‘DISCADOR’, pois nos dados de treinamento, essa palavra possui um valor alto de score para essa categoria.

Reflexão

Desde a idealização em fornecer uma maneira de lidar com processamento de linguagem natural com os dados de possíveis clientes e as dúvidas que eles tinham sobre os produtos da empresa na qual trabalho, me interessei pelo Machine Learning e fui atrás para aprender por conta dessa necessidade que surgiu, e durante o desenvolvimento desse projeto

pude ver que a comunidade em volta disso é muito grande e o compartilhamento de informações é crucial para que possamos cada vez mais evoluir rapidamente e fornecer cada vez mais sistemas que auxiliem em resoluções cada vez mais complexas.

Os passos que segui para a resolução desse problema foram:

1. Identificação da necessidade de solucionar o problema;
2. Obtenção de dados de entrada para treinamento do algoritmo;
3. Quais seriam os tratamentos necessários para o dataset inicial, de modo a obter o máximo de score possível;
4. Treinar o classificador com os dados tratados e avaliar as métricas de desempenho;
5. Refinar o classificador para torna-lo melhor para a solução final do problema;

Nos passos acima, particularmente gostei muito do 3 e 4, mesmo que tenha sido os mais difíceis, pois pude aprender algoritmos de oversampling e como modificações pequenas em parâmetros de algoritmos fazem diferença no resultado final, isso me mostrou que a avaliação de métricas e realização de testes é muito importante, Machine Learning não é só obter dados e enviar para treinamento, é entende-los a fundo, analisar as métricas e ir atrás de quais as necessidades para melhorar cada vez mais os resultados.

Melhorias

Acredito que a melhoria necessária para o projeto seria torna-lo mais geral para diversas categorias diferentes, não apenas para as categorias propostas, e torna-lo útil para diversas implementações diferentes de processamento de linguagem natural, com processos automáticos para identificação de parâmetros para refinamento de acordo com o dataset inserido e avaliação de métricas, para indicar ao usuário quais os pontos necessários de atenção no dataset.

Outra melhoria possível seria alguma forma de não deixar o TF-IDF dar muita importância para palavras que semanticamente não tem tanta importância assim, por mais que essas palavras ocorram poucas vezes no dataset, isso melhoraria em muito a previsão para novas entradas do algoritmo.

Bibliografia

Huang, O. (09 de 04 de 2018). *Medium*. Fonte: https://medium.com/@Synced/applying-multinomial-naive-bayes-to-nlp-problems-a-practical-explanation-4f5271768ebf

jlund3. (2018, 04 09). *Stack Exchange*. Retrieved from https://stats.stackexchange.com/questions/33185/difference-between-naive-bayes-multinomial-naive-bayes

Nitesh, V. C., Kevin, W. B., Lawrence , O. H., & W. Philip, K. (2002, 04 09). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 321 --- 357. Retrieved from https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume16/chawla02a-html/node6.html#SECTION00042000000000000000

Rai, A. (2018, 04 09). *Medium*. Retrieved from https://medium.com/@theflyingmantis/text-classification-in-nlp-naive-bayes-a606bf419f8c

scikit-learn. (2018, 04 04). *Scikit-learn*. Retrieved from http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

Scikit-Learn. (2018, 04 09). *sklearn.feature_extraction.text.TfidfVectorizer*. Retrieved from Scikit-learn: http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

Wikipedia. (2018, 04 04). *Wikipedia*. Retrieved from https://en.wikipedia.org/wiki/Confusion_matrix

Wikipedia the free encyclopedia. (2018, Abril 26). *Naive Bayes classifier*. Retrieved from Wikipedia Naive Bayes classifier: https://en.wikipedia.org/wiki/Naive_Bayes_classifier