ACH2118

Introdução ao Processamento de Língua Natural

Relatório de Implementação do Classificador de Faixa Etária para Respostas na Plataforma eSIC

Aluno

Felipe Mateos Castro de Souza - 11796909

Nilton Tadashi Enta - 12730911

Docente

Ivandré Paraboni

Escola de Artes, Ciências e Humanidades - EACH

18 de dezembro de 2023

Sumário

1 Introdução

Este relatório apresenta os resultados da implementação de um classificador de faixa etária para respostas publicadas na plataforma eSIC. O objetivo central do projeto foi desenvolver um modelo capaz de categorizar as respostas em quatro grupos de faixa etária, identificados como 'a1', 'a2', 'a3' e 'a4'. O trabalho foi conduzido em duas partes distintas: a primeira envolveu o desenvolvimento do classificador e a apresentação dos resultados médios de acurácia utilizando validação cruzada de 10 partições sobre o conjunto de dados de treinamento. A segunda parte consistiu na geração de previsões para o conjunto de teste.

Ao longo deste processo, realizamos uma análise exploratória dos dados, aplicamos técnicas de pré-processamento, vetorização de texto e implementamos modelos de aprendizado de máquina clássicos e neurais. O relatório detalha cada etapa do desenvolvimento, desde a importação e análise dos dados até a avaliação final dos modelos no conjunto de teste.

Os algoritmos testados incluíram Naive Bayes, Support Vector Machine, Random Forest e LSTM, sendo que a otimização dos modelos foi realizada através do optuna para SVM e Random Forest. As métricas de avaliação, como acurácia média na validação cruzada e acurácia no conjunto de teste, foram registradas para cada algoritmo, proporcionando uma visão abrangente do desempenho de cada modelo.

Os resultados obtidos neste relatório são essenciais para a escolha do modelo mais adequado para a tarefa de classificação de respostas na plataforma eSIC, contribuindo para a eficácia na análise de clareza dessas respostas.

2 Análise Exploratória

Realizamos uma análise preliminar do conjunto de dados utilizando as bibliotecas pandas e numpy.

Verificamos as informações gerais da base como presença de valores nulos, tipo de cada feature e quantidade de linhas. O dataframe possui 8.2 mil linhas e nenhuma vazia, todas do tipo 'object'.

Características do dataset:

Classe	Faiza etária
a1	1200
a2	2000
a3	3000
a4	2000

E analisamos também as palavras que se repetiam em cada classe. Mapeamos as dez maiores classes e em comum ambas tinham como as três palavras mais presente em seus textos: 'informação', 'por' e 'para'.

3 Pré-processamento

3.1 Tratamento dos dados

No âmbito do tratamento de dados textuais, a análise precisa e eficiente requer uma série de técnicas avançadas. Abaixo descrevemos o processo que utilizamos, abordando desde a tokenização até a codificação de rótulos.

a. Tokenização: A primeira etapa envolve a tokenização, um processo fundamental que divide o texto em unidades semânticas chamadas tokens. Cada palavra ou expressão é isolada, facilitando análises subsequentes e fornecendo uma visão granular do conteúdo textual. Optamos por utilizar a biblioteca NLTK para esta tarefa.

b. Lemmatização e remoção de pontuações: A lematização entra em cena para simplificar a análise ao reduzir palavras flexionadas às suas formas base. Essa técnica garante que diferentes formas de uma palavra sejam tratadas como uma única entidade, facilitando a compreensão e reduzindo a complexidade do conjunto de dados. Já para esta tarefa, optamos por usar a biblioteca SpaCy, onde fizemos uso do modelo 'pt_core_news_sm', vale ressaltar que juntamente desse processo, fizemos a remoção de pontuações, usando o tagger da biblioteca SpaCy.

c. POS Tagging (Etiquetagem de Partes da Fala): A etiquetagem de partes da fala adiciona camadas de informações valiosas, atribuindo etiquetas a cada palavra de acordo com sua função gramatical. Essa abordagem aprimora a compreensão contextual, permitindo uma análise mais profunda das relações sintáticas e semânticas entre as palavras.

d. Label Encoding (Codificação de Rótulos): A etapa final do processo consiste na codificação de rótulos, transformando informações textuais em representações numéricas. Essa técnica é crucial para a utilização eficiente de algoritmos de aprendizado de máquina, permitindo a aplicação de modelos preditivos a dados previamente tratados.

Para sermos mais eficientes, optamos por armazenar o tratamentos feitos em arquivos do tipo CSV, sendo o mais relevante deles o "enc_tok_nopunct_lemm.csv".

3.2 Separação do Conjunto de Dados

Com a finalidade de aumentar nossa confiança nos resultados das métricas que seriam obtidas após o treinamento dos modelos, optamos por dividir o conjunto de dados em duas partes, um conjunto de treino, correspondente a 80% dos dados, e um conjunto de teste, correspondente a 20% dos dados.

Dessa forma, podemos realizar uma média de uma validação cruzada no co conjunto de treino e depois comparar seus resultados com a métrica de acurácia no conjunto de teste, para aferir se houve *overfitting*.

3.3 Técnicas de Vetorização de Texto

Uma abordagem eficaz para essa tarefa é a utilização do método TF-IDF (Term Frequency-Inverse Document Frequency), e para implementar essa técnica de maneira robusta e eficiente, a biblioteca Scikit-Learn oferece ferramentas poderosas.

O Scikit-Learn fornece uma implementação flexível e fácil de usar do TF-IDF, permitindo que os usuários ajustem parâmetros conforme necessário e integrem facilmente essa etapa em seus pipelines de PLN e aprendizado de máquina. Desse modo optamos por utilizá-la em nosso projeto, com o hiperparâmetro 'max_features=5000' na coluna 'lemma' dos conjuntos de treino e teste.

Optamos por armazenar as matrizes resultantes em arquivos NPZ e NPY, para os atributos e rótulos respesctivamente, nos poupando, então, de ter que gerá-los novamente tod vez que reiniciássemos o kernel do Jupyter.

3.4 Técnicas de Word Embeddings

Uma abordagem inovadora para essa tarefa é o uso do modelo BERTimbau, uma variação do BERT (Bidirectional Encoder Representations from Transformers), presente na renomada biblioteca transformers. Optamos por utilizá-lo na seguinte configuração: return_tensors="pt", truncation=True, padding=True, max_length=512, add_special_tokens = True. Pois foi a que conferiu maior resultado dentre as abordagens usando embeddings,

4 Treinamento de modelos

4.1 Teste de Modelos de classificadores

Sob o data frame disponibilizado foram aplicados diversos classificadores, não neurais e neurais. Dentre os não neurais foram testados o Naive Bayes, Random Forest e SVM. Para os modelos neurais foram testados a MLP e LSTM.

Testamos os modelo inicialmente com a separação 80/20 de treino e teste, e em seguida com o cross-validation de 10 folds.

As redes neurais demandam um tempo maior de processamento e apresentaram uma acurácia similar ou próxima das não neurais, porém com uma variação maior entre resultados. Sendo assim, escolhemos seguir com os algoritmos não neurais.

4.2 Modelagem do Algoritmo

Realizamos a otimização utilizando grid search do sklearn nos modelos de SVM e RF, escolhidos por ter a acurária maior dentre os modelos.

5 Avaliação dos Modelos

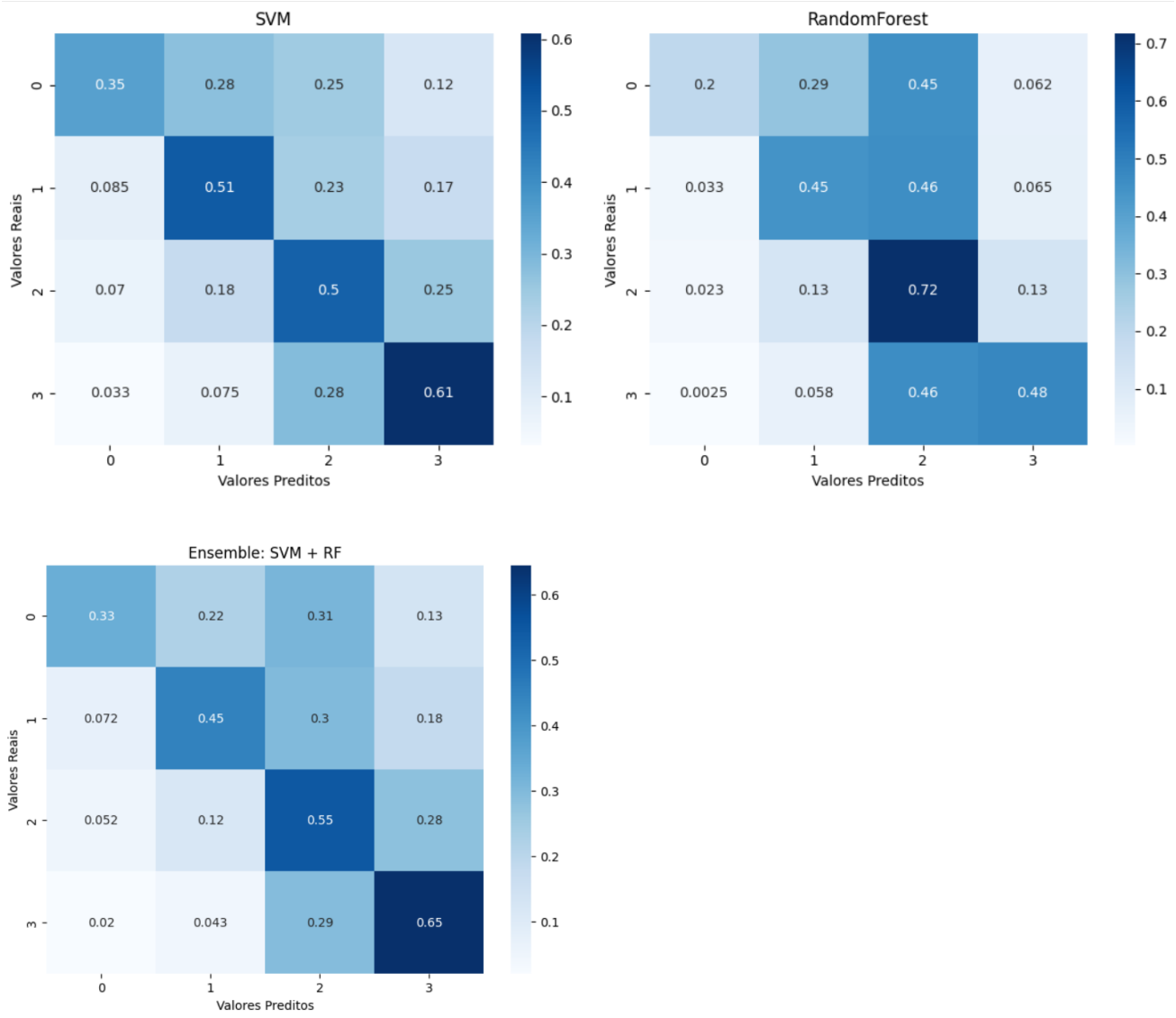
Após termos realizado a otimização de busca de hiperparâmetros usando o optuna nos algoritmos mais promissores, chegamos à conclusão que o algoritmo de SVM (Support Vector Machine) se saíria melhor que os demais (Naive Bayes e LSTM).

Desse modo, nosso *ensemble* final ficou com as seguintes características:

Vetorização: TFIDF dos textos lemmatizados e sem pontuação, com 'max_features'=5000

Algoritmos combinados: SVM com (C=1.0, kernel='linear', class_weight='balanced', probability=True), RandomForest com (n estimators=167, max depth=176, class weight=class_weights)

Abaixo, apresentamos duas matrizes de confusão dos resultados apra cada classe do conjunto de dados. Mais adiante temos a matriz de confusão do *ensemble* que consiste na união dois melhores classificadores (SVM e Random Forest) usando um Voting Classifier do sklearn.



5.1 Conjunto de treinamento

Dentro do conjunto de treinamento, que corresponde a 80% dos do conjunto de dados, fizemos uma validação cruzada de 10 *folds*. Após a obtenção da média dos valores dessa métrica, constatamos que o modelo obteve uma acurácia média de 51.28%.

5.2 Conjunto de teste

Modelos testados:

Modelo	Emb	Parâmetro	acc
NB	tfidf	Default	45.30%
SVM	tfidf	class_weight='balanced', probability=True	48.90%
RandForest	tfidf	167'nestimators, 176'maxdepth'	51.76%
DNN	tfidf	512'relu', 0.8'dout', 126'relu', 4'stmax'	49.02%
LSTM	tfidf	64'lstm', 0.4'dout', 32'relu', 4'stmax'	36.59%
LSTM+C	tfidf	f32'Conv1d', 500'mpool', 64'lstm', 0.4'dout', 64'relu', 4'stmax'	36.59%
SVM + RF	tfidf	-	51.28%
RandForest	w2vec	Default	37.86%
DNN	BERT	128'relu', 64'relu', 0.6'dout', 4'stmax'	failed
hugface	hugface	formality classifier	0.3684

6 Análise do Resultado

Não obtivemos sucesso ao utilizar as bibliotecas do python Eli5, Lime e Shap. Dada a estrutura de dados em matriz esparsa e o kernel não linear, não foi possível atender as exigência dos parâmetros das funções das bibliotecas de interpretação disponíveis. Contudo, tentamos alterar o kernel para linear mas sem sucesso, uma vez que um dos parâmetros da SVM='probability=True' fazia com que as funções e métodos ainda não pudessem interpretar nossa base de dados.

Em alternativa a isso realizamos uma análise das principais features e utilizamos o método 'sample' para buscar aleatoriamente alguns textos. Notamos que havia um certo nível de informalidade maior nas classes 'a1' e 'a2' mas no geral nada que diferenciasse-as do restante.

7 Reprodução dos resultados

Basta seguir as instruções presentes no arquivo main.ipynb