Teste Axur – Classificação de SMS

Rodrigo Biasuz

A presente documentação detalha as etapas criadas para a construção da solução de um classificador binário de SMS, que avalia o texto da mensagem como “bloqueado” ou “ok”.

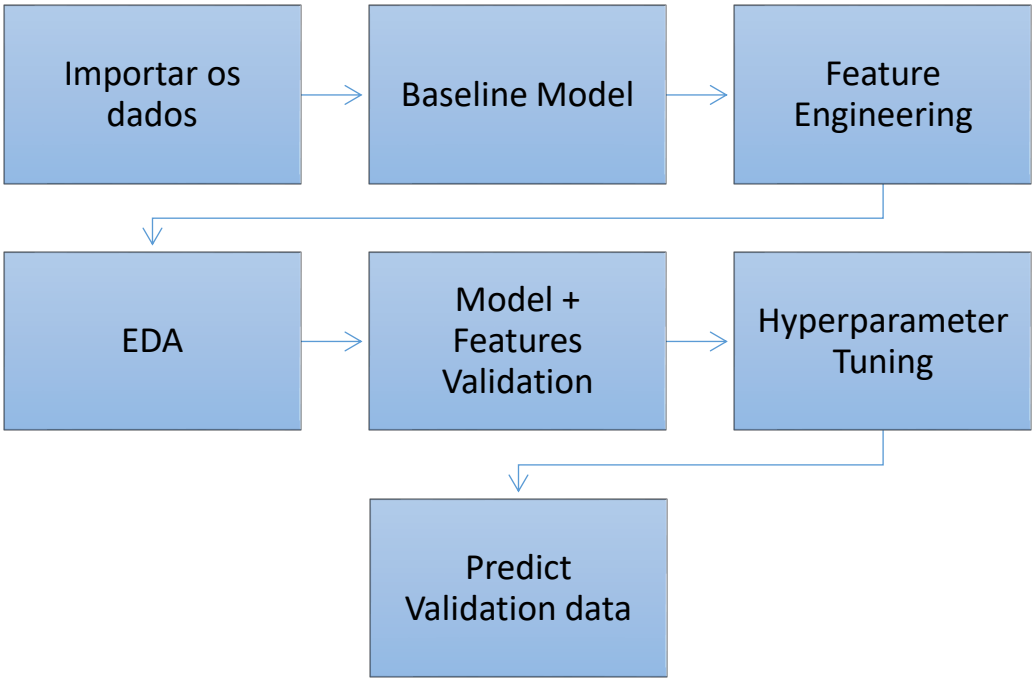
Sumário

Descrição dos arquivos

- Requirements.txt: Arquivo com as bibliotecas necessárias pare executar o código criado
- SMS_Classification.html: Arquivo com o código em html para facilitar a visualização
- SMS_Classification.ipynb: Arquivo com o código original
- Submission.csv: Arquivo com os resultados

Metodologia Aplicada

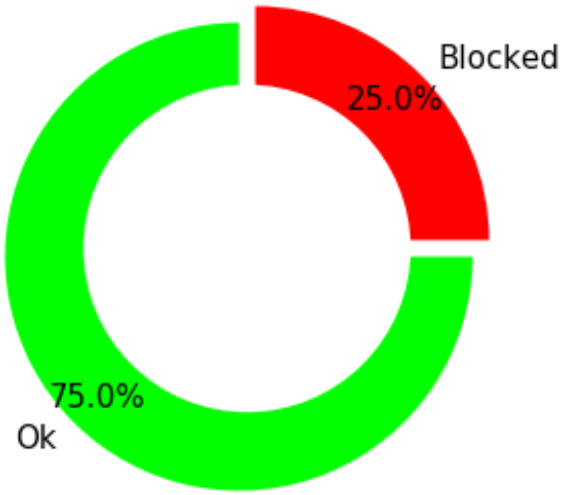
Para solucionar o desafio, desenhou-se a seguinte estratégia:



Importar os dados

Nesta etapa apenas carregamos os Datasets, visualizamos algumas ocorrências específicas, procuramos por Missing Values e verificamos como os dados estão distribuídos.

Obs.: Importante verificar qual encoding se aplica melhor para a linguagem em avaliação.



Distribuição dos dados de teste

Baseline Model

Primeiramente, criou-se um modelo base para comparação dos resultados utilizando novas Features e outros modelos.

Nesta etapa já aproveitei para remover as “StopWords” da língua portuguesa e verificar qual dos tipos de vetorização de SMS’s se condicionava melhor aos dados apresentados:

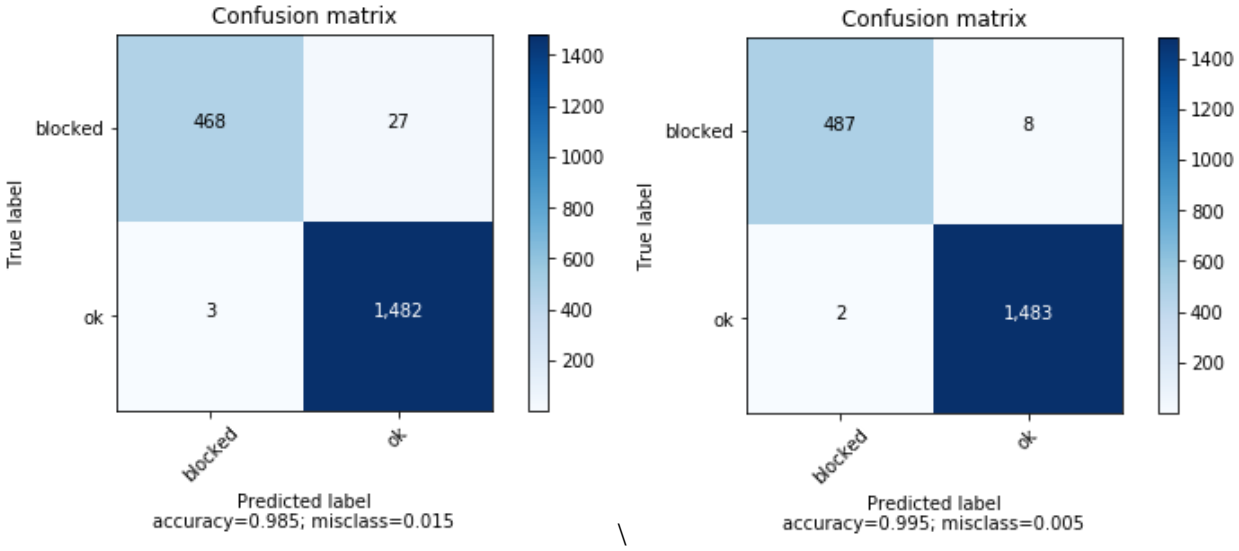
- CountVectorizer ou
- TfidfVectorizer

O primeiro apenas conta a frequência de cada token no texto e usa isso como peso, enquanto o outro, chamado de “term frequency-inverse document frequency” também leva em conta a presença do Token ao longo de todo o Corpus (todos os conjuntos de textos).

Os resultados indicaram pouca diferença na utilização direta de cada um dos métodos, aproximadamente **1%** de diferença na acurácia do modelo, dessa forma foi optado por CountVectorizer, pois seu tempo de execução é menor (além de ter uma redução no tempo de treino dos modelos, pois cria apenas aprox. 6000 parâmetros contra aprox.. 8500 do TfidfVectorizer).

Obs.: para evitar problemas de overfitting, os dados do treino fornecidos foram separados novamente em treino e teste (33% para teste e o restante para treino) de forma aleatória, além disso, levou-se em consideração os dados desbalanceados estratificando a separação por categoria, com intuito de manter a proporção entre “bloqueados” e “oks”

Abaixo as matrizes de confusão de cada um:



TfidfVectorizer

CountVectorizer

Feature engineering

Criou-se então algumas ‘colunas’ extras utilizando as informações presentes no texto, que foram:

- Número de Characters do SMS;
- Número de Palavras do SMS;
- Número de Hashtags (#) presentes no SMS;
- Variável binária para a presença de alguma URL no SMS;
- Variável binária para a presença de algum número no SMS;
- Variável binária para a presença da palavra ‘senha’ no SMS;

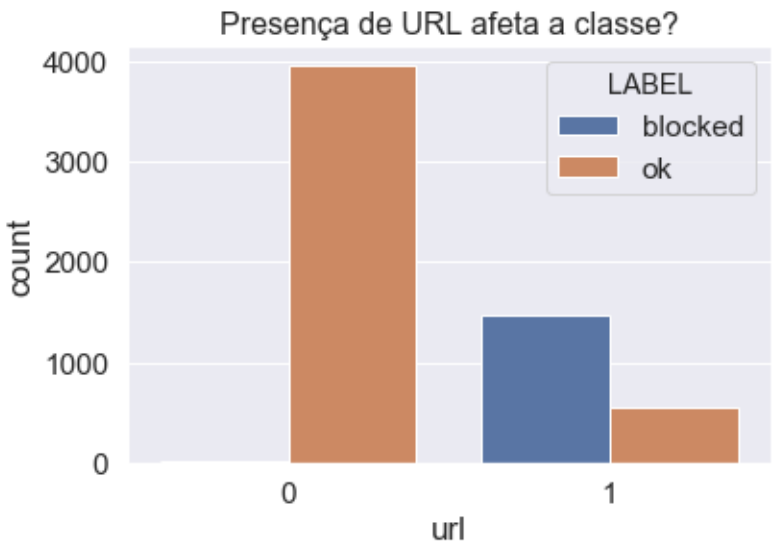
EDA - Exploratory Data Analysis

Procurou-se verificar algumas estatísticas intrínsecas no dataset com o objetivo de detectar padrões, utilizando o próprio conteúdo de cada mensagem e as Features criadas.

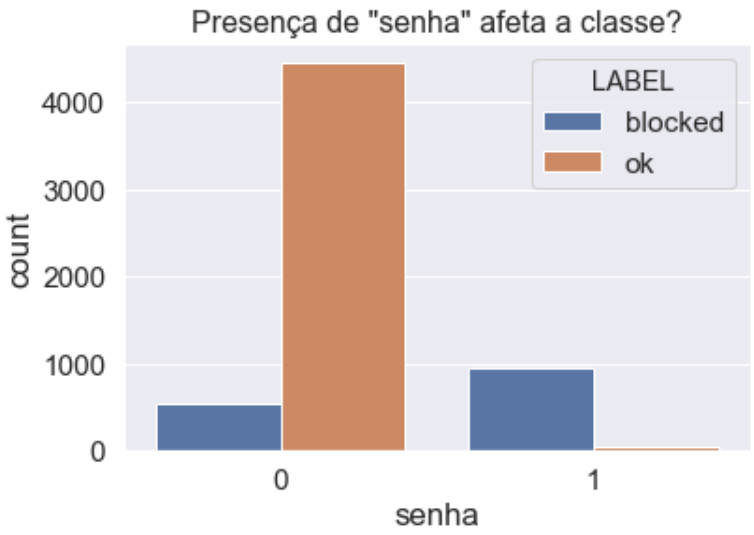
Principais pontos a destacar:

Observa-se que **98.7%** dos SMS's bloqueados apresentam uma URL,

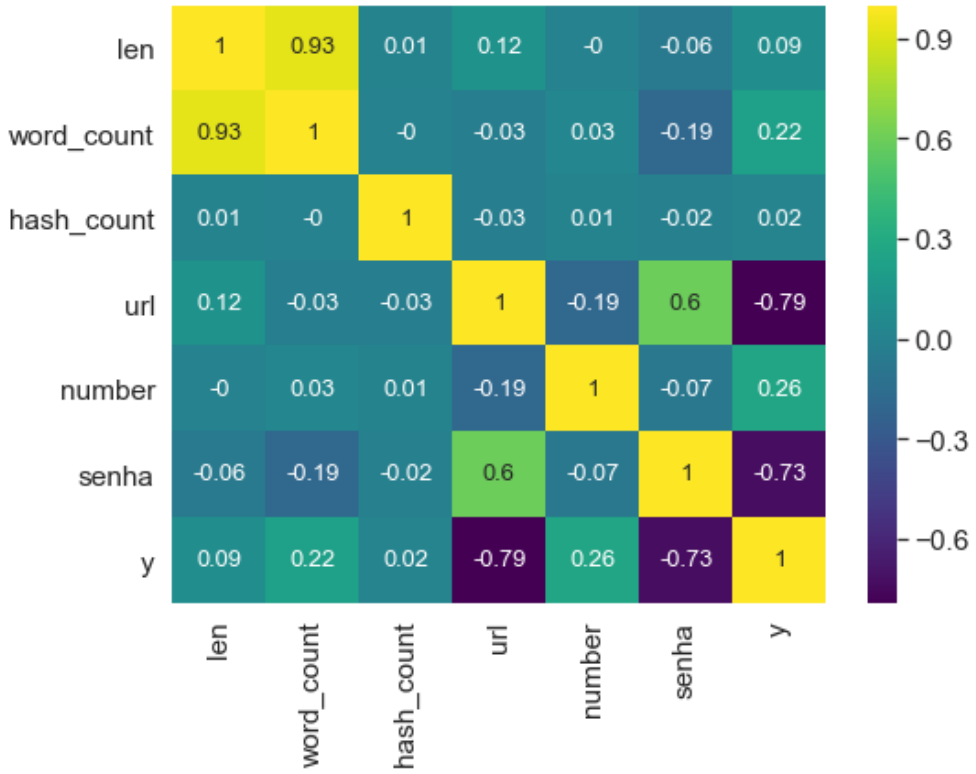
Apenas **12,22%** dos SMS's ok também apresentam



Observa-se que **63.3%** dos SMS's bloqueados apresentam a palavra "**senha**", contra menos de **1%** dos SMS's ok

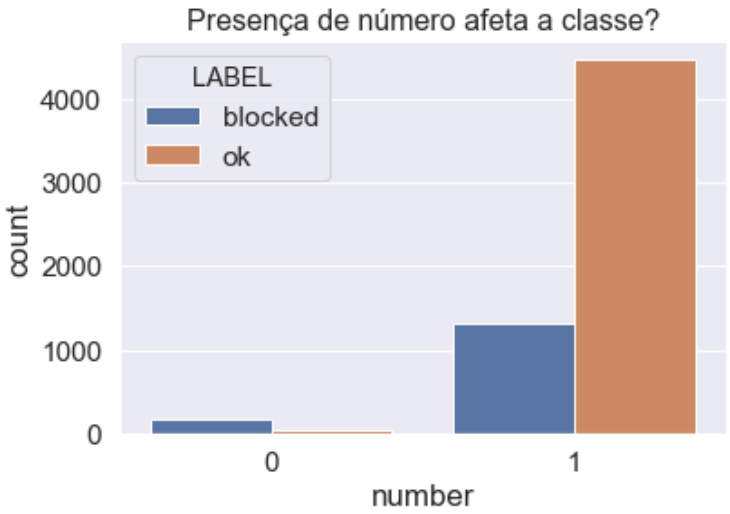


Quando observamos o mapa de correlação entre a variável de saída e as Features criadas, os resultados que se destacam são justamente a presença de URL e da palavra “Senha” (considerando o coeficiente de Pearson), com 79% e 73%, respectivamente:

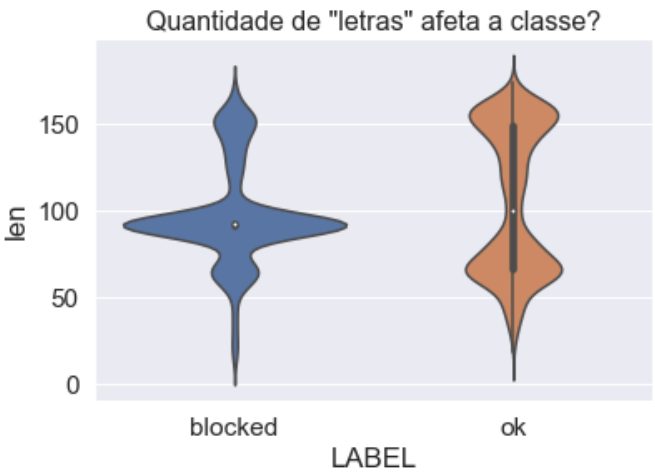


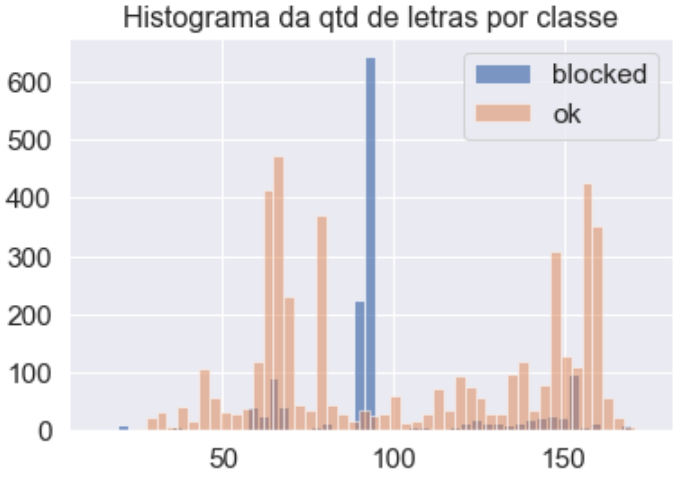
Heatmap das correlações de Pearson

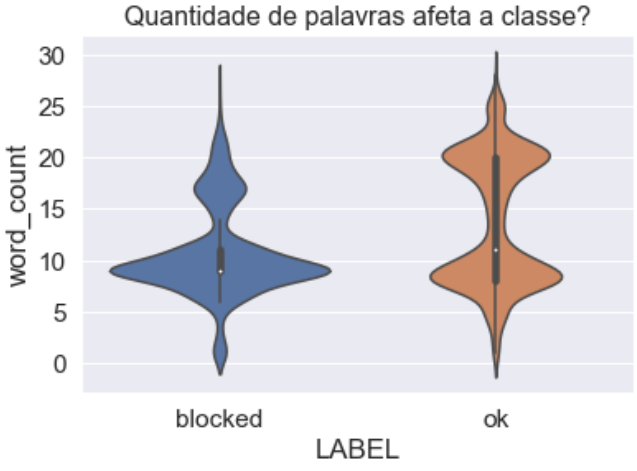
A Feature de presença de número não apresentou uma relação tão direta, talvez nesse caso fosse melhor investigar o que esse número representa (usando mascaras de nro de telefone, conta bancaria, etc...)

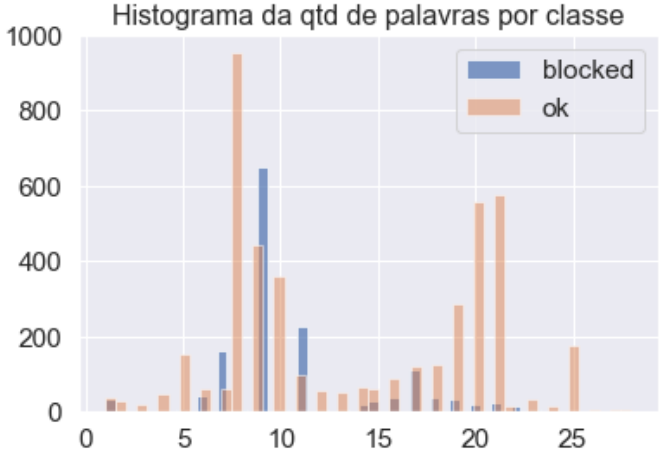


As Features de quantidade de Characteres e de palavras também não aparentam uma relação direta, mas podemos notar que as distribuições na categoria de "ok" tendem a ter um padrão mais distribuído entre mensagens curtas e mensagens compridas, enquanto as "bloqueadas" ficam em uma faixa mais intermediária.

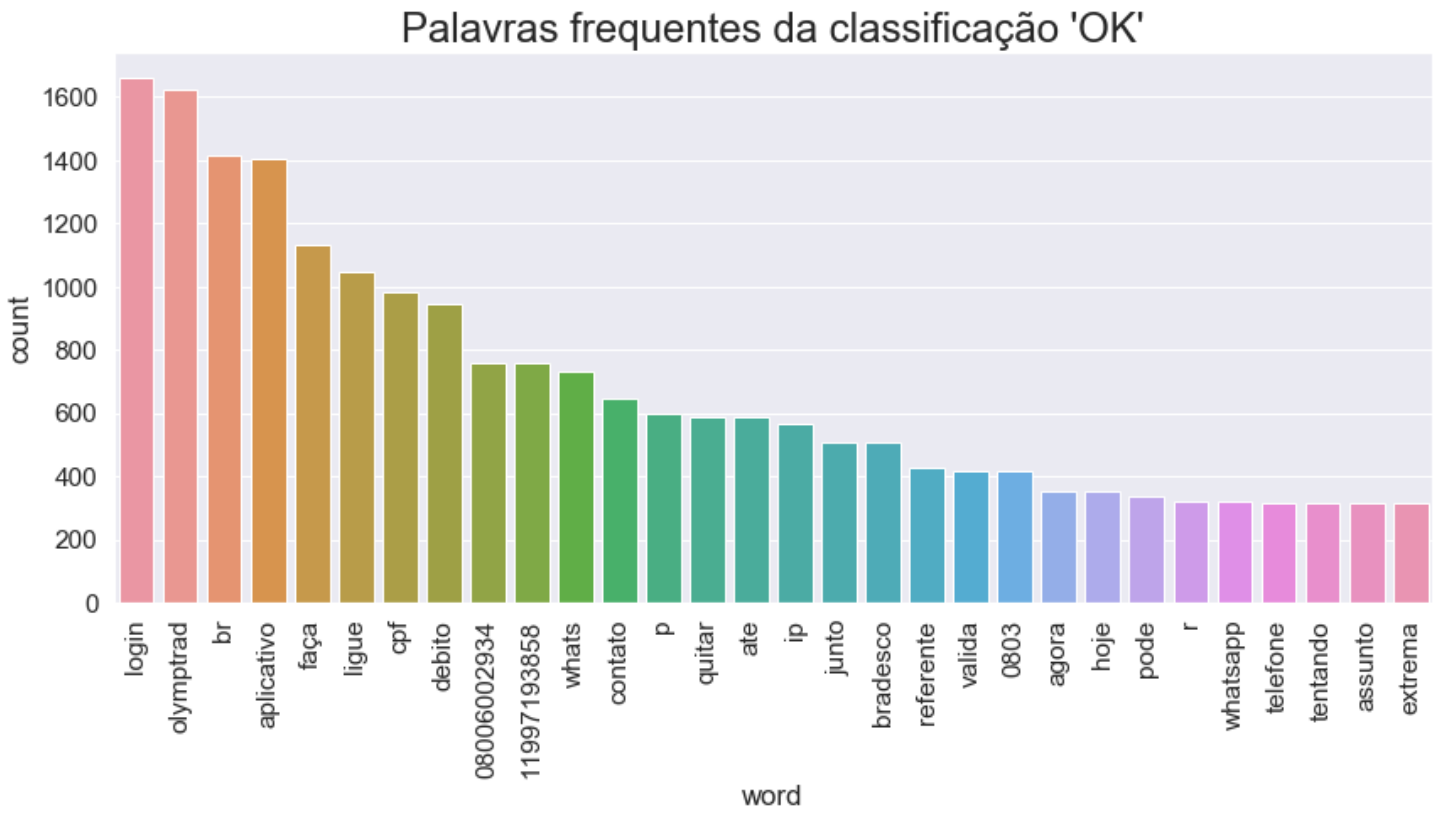


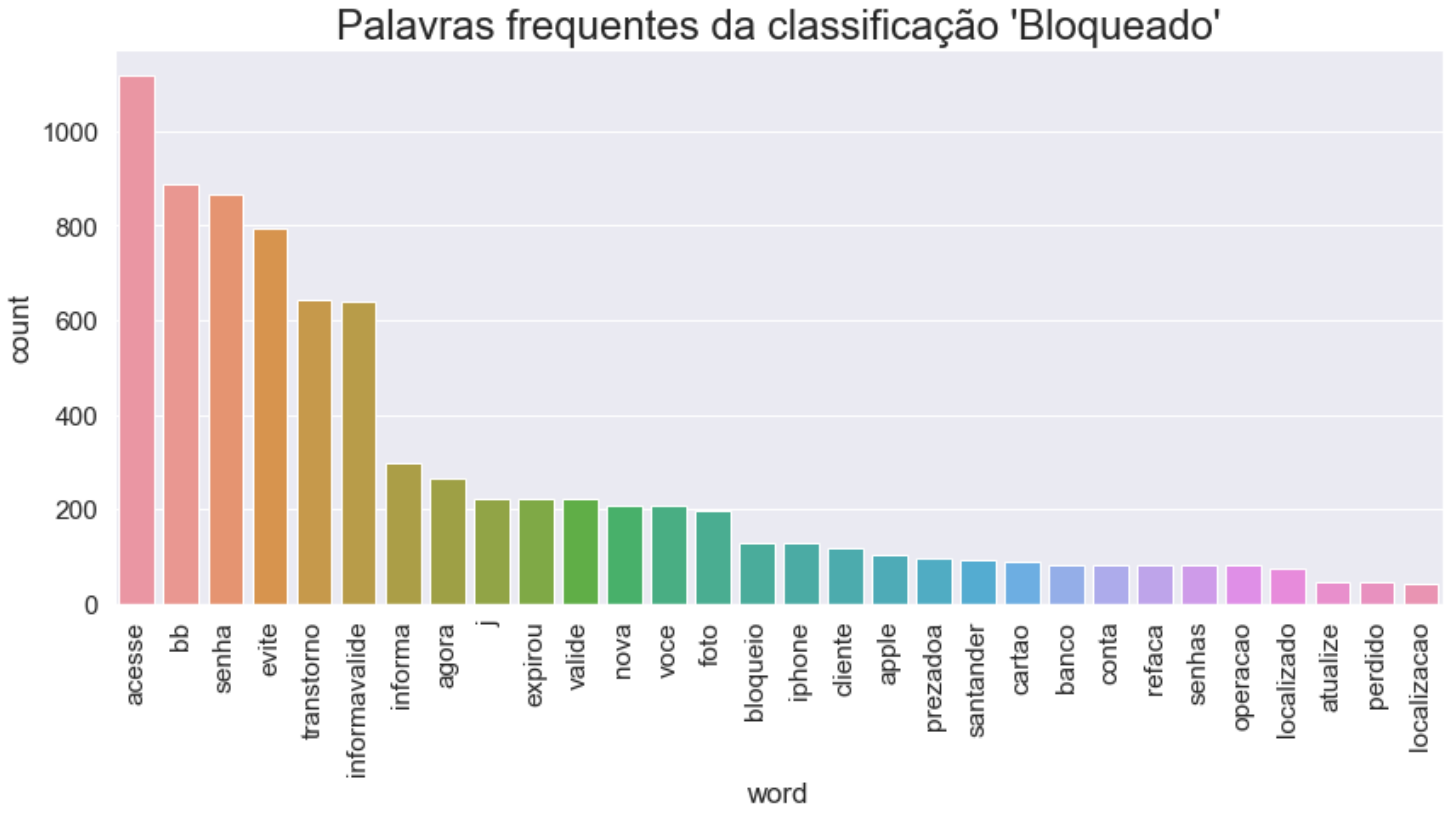






Foi levantado também as palavras mais frequentes nos SMS’s em cada categoria:





Percebe-se um padrão nos bloqueados em termos de palavras que tentam influenciar a pessoa a agir rápido e sem muita criticidade, como:

- *Acesse;*
- *Senha;*
- *Evite transtorno;*
- *Agora;*
- *Expirou;*
- *Validade...*

Enquanto nos e-mails OK, a distribuição é mais uniforme entre as palavras e encontramos termos que levam a pessoa a buscar por mais informações, como:

- *Ligue;*
- *Whatsap;*
- *Whats;*
- *Contato…*

Uma sugestão para continuação desse trabalho é criar Features que levem em consideração essas palavras mais frequentes em cada uma das categorias, mascaras de email (@gmail, @outlook, @hotmail, @yahoo, etc...), máscaras de números de contas bancárias, números de telefone, etc...

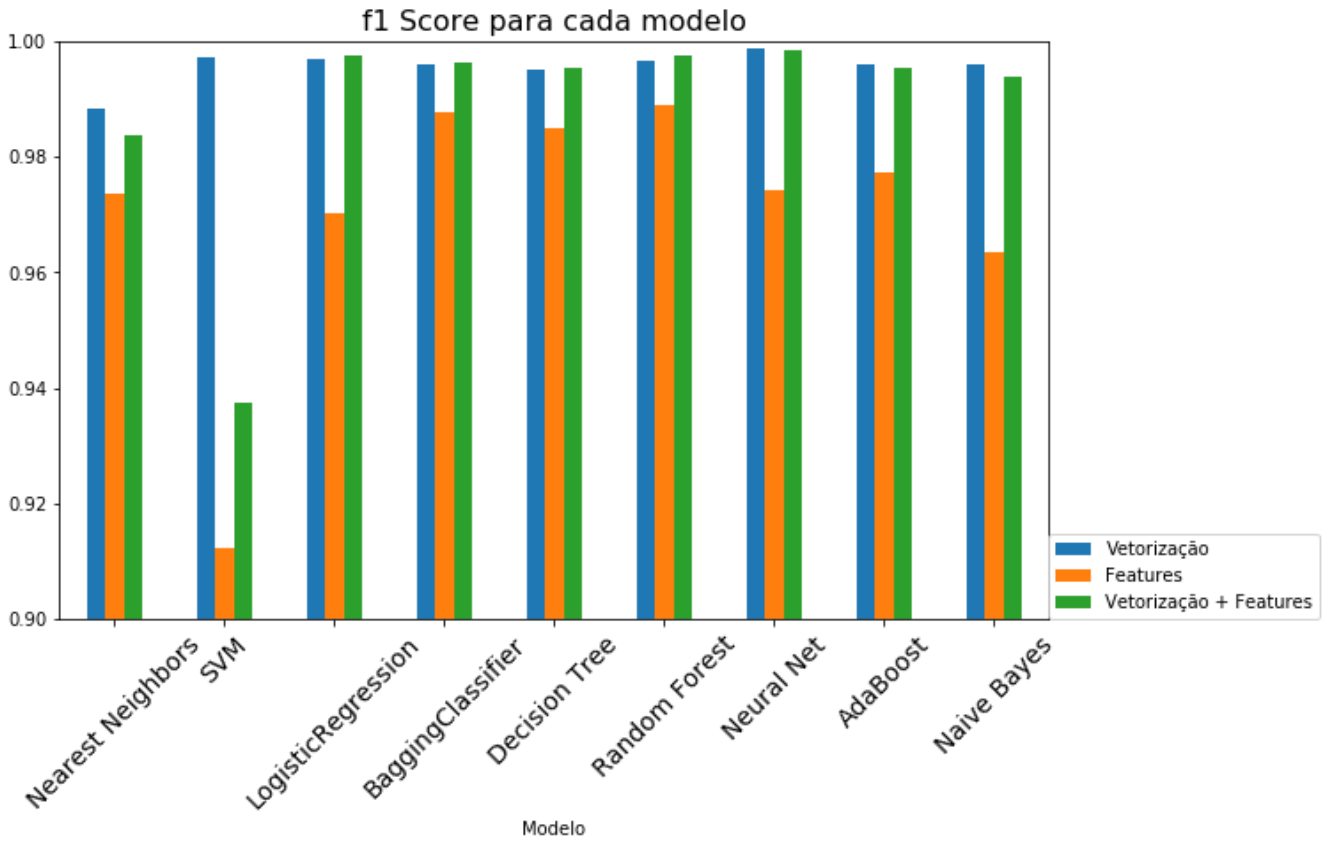
Model + Features Validation

Realizaram-se testes utilizando diferentes combinações de Features com os seguintes modelos de classificação:

(Os resultados podem ser visualizados na tabela e no gráfico abaixo)

- Nearest Neighbors
- SVM
- LogisticRegression
- BaggingClassifier
- Decision Tree
- Random Forest
- Neural Net
- AdaBoost
- Naive Bayes

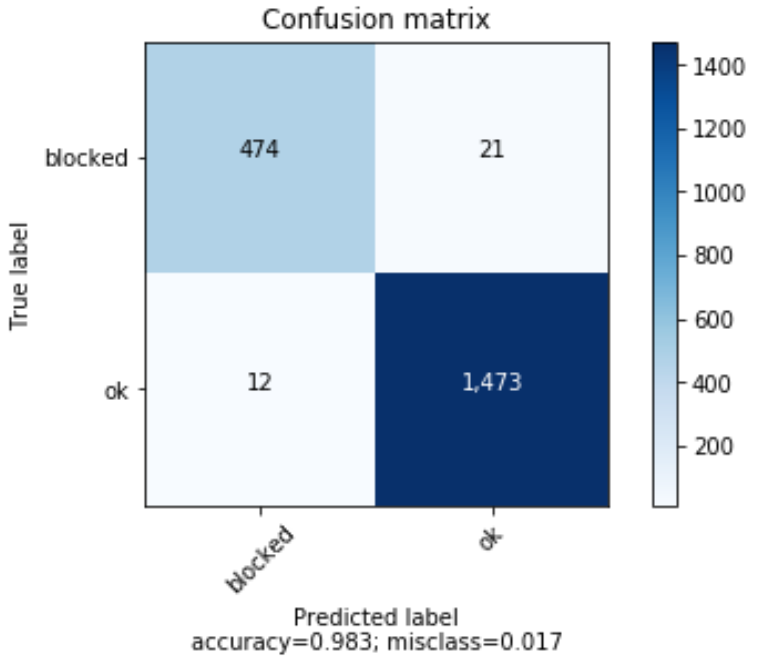
	Modelo	Vetorização	Features	Vetorização + Features
0	Nearest Neighbors	0.988164	0.973492	0.983784
1	SVM	0.997312	0.912357	0.937401
2	LogisticRegression	0.996977	0.970139	0.997647
3	BaggingClassifier	0.995965	0.987563	0.996303
4	Decision Tree	0.994955	0.984915	0.995293
5	Random Forest	0.996638	0.988922	0.997647
6	Neural Net	0.998654	0.974240	0.998318
7	AdaBoost	0.995968	0.977373	0.995296
8	Naive Bayes	0.995951	0.963499	0.993907



Foi simulado o comportamento isolado das Features, sem os dados dos SMS, para a classificação, utilizando a seguinte estrutura de entrada:

	len	word_count	url	number	senha
0	142	24	0	1	0
1	52	2	1	1	0
2	47	2	1	1	0
3	109	14	1	0	0
4	47	1	1	1	0

O resultado dessa simulação pode ser visualizado pela Matriz de Confusão abaixo:



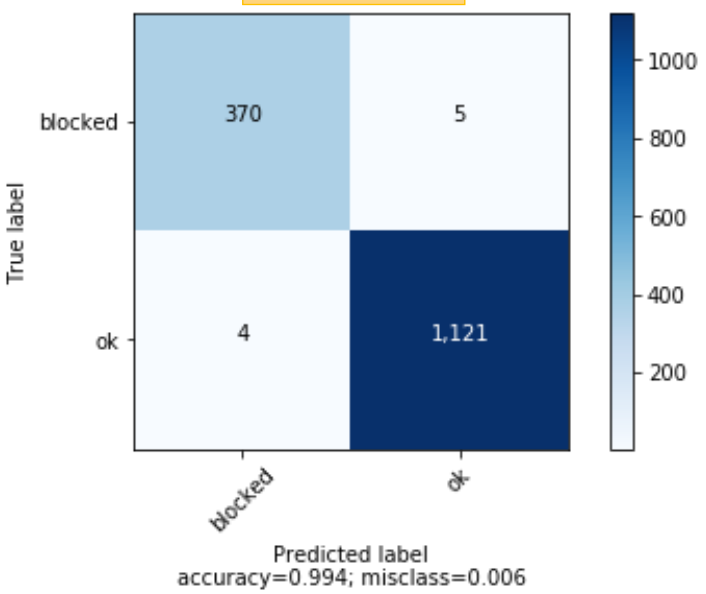
É interessante destacar que apenas as Features extraídas já conseguiram no conjunto de teste uma acurácia de 98% (inferior aos modelos completos, porém significativamente boa, considerando que o tempo de treinamento e de execução é menos da metade).

Caso o desafio exija um tempo baixo de processamento, otimizar e criar modelos com base apenas nas Features criadas pode ser uma boa estratégia.

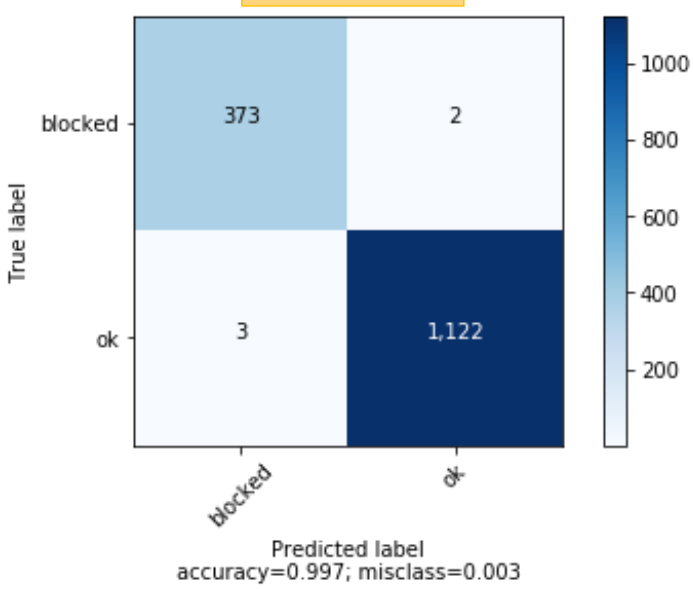
Hyperparameter Tuning

Dos modelos com o melhor desempenho, realizou-se uma otimização nos parâmetros utilizando GridSearchCV

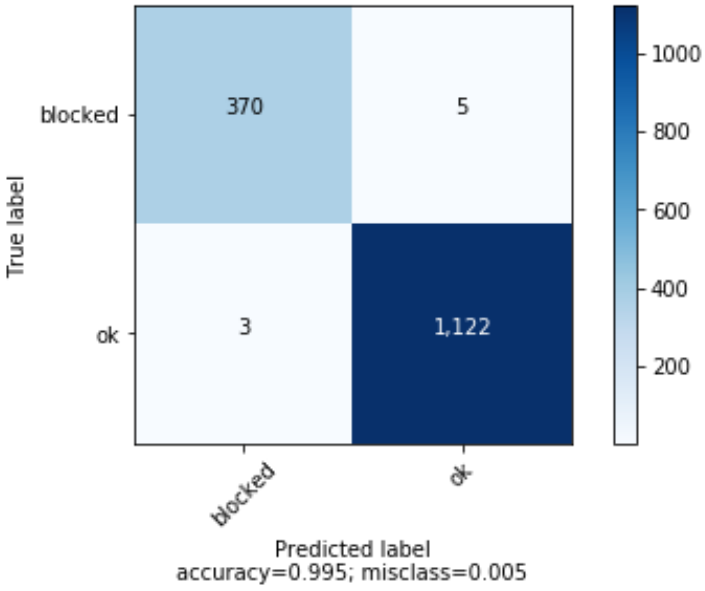
Random Forest



Neural Net



Naive Bayes



Predict Validation data

Na etapa final, aplicou-se os três modelos finais construídos, para posteriormente verificar se em alguma linha eles deram retornos incompatíveis, buscando detectar possíveis erros.

As seguintes linhas retornaram diferentes previsões para os modelos:

```
In [85]: # Verificando possíveis erros (previsões diferentes pra cada modelo)
         validation.query('predict_mean < 1 and predict_mean > 0')
```

Out[85]:

	SMS	predict1	predict2	predict3	predict_mean
5	Brastemp, Sem Duvida a melhor experiencia de a...	0	0	1	0.333333
59	Prezado(a) Cliente, os exames da O.S. 009-646...	0	0	1	0.333333
60	Brastemp, Sem Duvida a melhor experiencia de a...	0	0	1	0.333333
64	Brastemp, Sem Duvida a melhor experiencia de a...	0	0	1	0.333333
65	Disponivel: Seu boleto esta vencido e seu aces...	1	0	1	0.666667
66	Brastemp, Sem Duvida a melhor experiencia de a...	0	0	1	0.333333
67	Let's video chat and text on imo! Get the free...	0	0	1	0.333333
73	Brastemp, Sem Duvida a melhor experiencia de a...	0	0	1	0.333333
79	Alerta FREE: Prezado(a), Houve 3 tentativas de...	0	0	1	0.333333
84	Prezado cliente , por motivos de seguranca , a...	0	0	1	0.333333
94	050003EB0202|ardian30hors.xyz/Unibank/S.A/BR/c...	0	1	0	0.333333
101	This service does not exist. You wrote: eio po...	0	0	1	0.333333
104	Boa noite, voce tem 30,00 em credito cadastre-...	0	0	1	0.333333
160	Attempt to login from your Apple ID. We recomm...	1	0	1	0.666667
187	http://seguranca_caixa.segurobr.top/	0	0	1	0.333333
188	http://seguroGG.top/seguranca_caixa	0	0	1	0.333333
190	www.segurobr.top/seguranca_caixa	0	1	1	0.666667

A partir disso, a abordagem mais comum é optar pela classificação que dois modelos haviam dado (arredondar o valor para o inteiro mais próximo), porém optei por classificar esse grupo como “bloqued” considerando que o SMS pode conter algum vírus ou outro tipo de malefício para o usuário e, no caso da dúvida, seria melhor bloquear do que classificar como OK.

Métricas Escolhidas

Ao longo da solução, as métricas utilizadas para comparar os modelos foram:

- Acurácia
- Precisão
- Recall
- F1-score

Para comparação final dos modelos, optou-se pelo f1-score, por ser uma combinação da precisão com o recall, e, na posição de cliente desse possível produto, eu não desejo nem que os bloqueados cheguem para mim, nem que os "oks" sejam bloqueados, então um intermediário entre ambos pode ser a melhor opção.

(A métrica de acurácia pura não é recomendada para esse tipo de modelo, pois o desbalanceamento de classe faz com que a acurácia seja alta até mesmo de um modelo que simplesmente diz que todos os SMS's estão ok.)

Para melhor visualizar os resultados, também se utilizou da Matriz de Confusão.

Sugestões para melhorar o modelo

Algumas sugestões que podem vir a melhorar o modelo, mas não foram abordadas na solução:

Ensemble Learning

Combinar as decisões de múltiplos métodos para melhorar o desempenho geral (minimizar overfitting e a variância em geral). Podemos aplicar isso de diferentes formas, como utilizar a saída de um modelo como parâmetro de entrada para outro modelo.

Url scraping

Além de verificar se o SMS contém ou não uma URL, realizar o web scraping dessa URL (procurar por conteúdo específicos dentro dos HTML's) também pode gerar Features importantes para a análise. Existem ferramentas específicas em Python para ajudar nessa tarefa, como a biblioteca BeautifulSoup.

Parameter_Space + Cross_Validation

Outra forma de melhorar o desempenho do classificador é testar mais modelos utilizando Cross_Validation e um espaço de parâmetros maior (na solução, foram testados apenas 3 tamanhos relativamente pequenos de rede neural, por exemplo, esse espaço poderia ser muito maior)

Deep Learning e Processamento pela GPU

Uma rede profunda, de diversas camadas, pode apresentar um resultado superior também, porém leva mais tempo para ser treinada, dessa forma, combinar o uso de uma rede profunda (como uma arquitetura criada usando TensorFlow) com o processamento por GPU agiliza o processo e torna esse tipo de abordagem possível.

Algumas libs para executar isso em Python: Pytorch, Keras, CUDA (GPUs da NVIDEA)

Stemming / Lemmatização

Reduzir as palavras ao radical pode ajudar muito nas análises, por exemplo, as palavras 'gato', 'gata', 'gatos', 'gatas', 'gatinho', 'gatinha', 'gatinhos', 'gatinhas' todas, em sua essência, podem representar a mesma ideia, dessa forma, transformar todas em 'gat' traz inúmeras vantagens como redução do tamanho das matrizes de palavras, abstração de significado, etc...

Python também tem bibliotecas para ajudar nessa tarefa, como NLTK, Gensim e spaCy