Introdução

Dados usados: data/2018 já pré-processados, recolhidos durante as eleições presidenciais de 2018.
Experiência: TF-IDF, uni-bi-trigram com random oversampling.

Misinformation = 0 → Real; Misinformation = 1 → Fake; Misinformation = -1 → Não foi rotulada.

Existem mais dados fakes (10152) e menos reais (8799), mas são quase a mesma quantidade. Depois de remover os duplicados fica-se com 2041 fakes e 2547 reais.

- texts é a coluna com as mensagens, e o filtro midia = 0, serve para selecionar apenas as que não são midiáticas, como mensagen do whatsapp - contém as mensagens que serão usadas como entrada no modelo de deteção de desinformação.
- y é a coluna das labels filtrada também com midia = 0 e indica se o texto é ou não desinformação (1) ou não (0).

O número de mensagens/textos que vai ser usado nos modelos é 5284 (len(texts) = len(y)).
A coluna “types” tem o número de palavras únicas de cada mensagem (mede a variedade do vocabulário).

- types/token ratio representa a proporção de palavras únicas em relação ao número total de palavras do texto.
- char/word ratio representa o número de caracteres que cada palavra tem.
- average chars by word representa a média de caracteres por palavra.

basic statistics: misinformation

	words	types	types/token ratio	characters	char/word ratio	average chars by word	shares
count	2041.000000	2041.000000	2041.000000	2041.000000	2041.000000	2041.000000	2041.000000
mean	113.870652	78.719255	0.845943	719.406663	6.590809	5.529126	4.974032
std	173.284886	99.627016	0.134153	1072.223053	2.121217	1.697369	7.027175
min	10.000000	10.000000	0.364807	48.000000	4.000000	3.076923	2.000000
25%	21.000000	20.000000	0.740741	137.000000	5.709677	4.702381	2.000000
50%	39.000000	34.000000	0.880000	246.000000	6.116809	5.100000	3.000000
75%	132.000000	100.000000	0.954545	870.000000	6.620690	5.595745	5.000000
max	2203.000000	998.000000	1.000000	13007.000000	61.925000	18.878947	91.000000

basic statistics: non-misinformation

	words	types	types/token ratio	characters	char/word ratio	average chars by word	shares
count	2547.000000	2547.000000	2547.000000	2547.000000	2547.000000	2547.000000	2547.000000
mean	60.044366	44.574401	0.899166	408.918728	7.231410	6.174667	3.454653
std	136.732108	71.514831	0.113192	894.444405	3.071629	2.993315	4.138486
min	10.000000	10.000000	0.208163	38.000000	2.560284	1.524823	2.000000
25%	15.000000	14.000000	0.846154	103.000000	5.651559	4.639957	2.000000
50%	25.000000	23.000000	0.928571	183.000000	6.193182	5.181818	2.000000
75%	54.000000	46.000000	1.000000	353.500000	7.468627	6.400000	3.000000
max	2664.000000	1179.000000	1.000000	15683.000000	46.615385	45.692308	86.000000

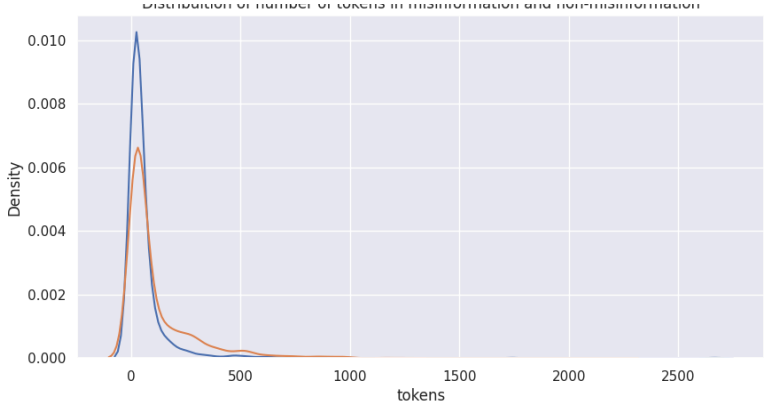
Média:

	misinformation	non-misinformation
words	113.870652	60.044366
types	78.719255	44.574401
types/token ratio	0.845943	0.899166
characters	719.406663	408.918728
char/word ratio	6.590809	7.231410
average chars by word	5.529126	6.174667
shares	4.974032	3.454653

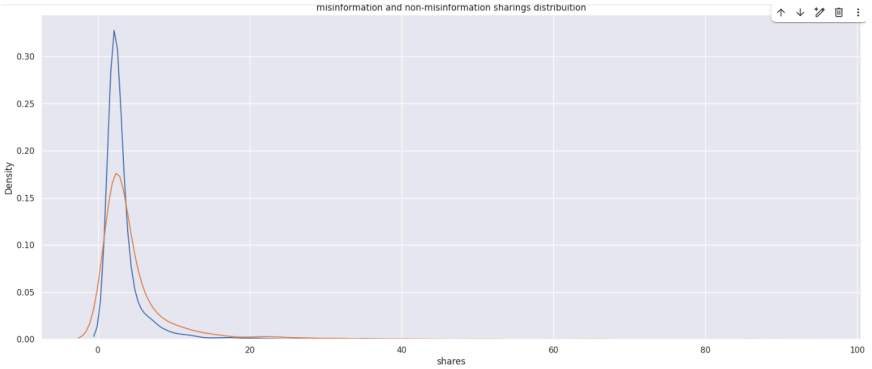
Desvio Padrão:

	misinformation	non-misinformation
words	173.284886	136.732108
types	99.627016	71.514831
types/token ratio	0.134153	0.113192
characters	1072.223053	894.444405
char/word ratio	2.121217	3.071629
average chars by word	1.697369	2.993315
shares	7.027175	4.138486

Displot:



1 - representa número de tokens/densidade, ou seja, o tamanho dos textos por quanto comum é cada tamanho dentro de cada classe. Como as curvas têm um pico muito à esquerda, em torno de poucas dezenas de tokens, indica que a maioria dos textos é curto. Textos mais curto tendem a ser mais curtos, daí o pico azul ser maior que o pico laranja. Os textos das fake news podem ser mais elaborados.



2 - Fez-se a mesma coisa para o número de partilhas e concluiu-se que textos reais são menos partilhados e os fakes têm mais partilhas, mas ambos não sendo muito partilhados, uma vez que estão perto de zero.

Pré-processamento

Verifica se já existe um arquivo pré-processado (processed_texts-) e se existir carrega-o, caso contrário ele executa o pré-processamento manualmente.
Eles no pre-processamento manual fazem remoção de elementos que não ajudam o modelo:

- Pontuação, números, símbolos, emojis, caracteres especiais, links, espaços extras e quebras de linhas.

Fazem a normalização em que se colocam todas as palavras em minúsculas, pode renovar acentos e remove stopwords (reduz o ruído e faz o modelo focar nas palavras mais importantes).
Faz a tokenização em que divide o texto em tokens pelo que cada linha vira uma lista de palavras.
Faz a lematização em que reduz as palavras à sua forma bas (correu,correndo → correr).

Train test split

Divide em conjunto em 80% treino e 20% teste e garante que a proporção entre classes (fake vs real) seja mantida nos dois conjuntos e o random_state faz com que use sempre os mesmos dados, obtendo sempre os mesmos resultados mesmo em caso de executar de novo.

Vetorização

Para transformar os textos em vetores numéricos, para poderem ser usadas no modelo. Como estamos a usar na experiência TF-IDF e uni-bi-trigram com random oversampling, vai realizar essa vetorização.
Aprende o vocabulário com base os textos de treino, converte este conjunto de treino numa matriz numérica esparsa (grandes e cheias de zeros) e converte o conjunto de teste usando o mesmo vocabulário e cria uma versão vetorizada de todos os textos.
Na matriz numérica cada linha é um texto e cada coluna é uma palavra ou n-grama. O número de vocabulário máximo é 500 palavras mais relevantes.
TF-IDF:
O vocabulário é composto por termos únicos, duplos e triplos de palavras. Aprende o peso para cada termo em cada texto e cria matrizes em que cada linha é um texto, cada coluna é um termo (palavra, bigrama, trigrama) e cada valor é um peso TF-IDF desse texto.

O conjunto de treino é composto por 4227 textos (80% do dataset) e 338064 colunas geradas pelo vetorizar em que cada uma representa aquilo que foi dito antes sobre a matriz.

Balanceamento de Dados

Usa-se o random oversampler para aumenta a quantidade da classe minoritária copiando exemplos dela até equilibrar as duas. Permite ao modelo treinar com a mesma quantidade de exemplos de cada classe, neste caso passam a ser 6111 exemplos de treino.

Resultados do treino dos modelos

	vocab	model	fpr	pre	rec	Observação principal
0	**338064**	logistic regression	nan	0.690105	Melhor equilíbrio geral; modelo estável e robusto
1	**338064**	bernoulli naive-bayes	nan	0.606429	Muito fraco — incompatível com TF-IDF
2	**338064**	multinomial naive-bayes	nan	0.642460	Bom desempenho base; simples, mas perde contexto
3	**338064**	linear svm	nan	0.646234	Consistente, muito próximo da regressão logística
4	**338064**	**sgd**	nan	0.657912	Equivalente à SVM normal; ligeiro ganho em velocidade
5	**338064**	knn	nan	**0.695082**	Desastroso — KNN não lida bem com TF-IDF esparso
6	**338064**	random forest	nan	0.638947	Melhor recall (detecta mais fakes), mas menor precisão

A vetorização feita funciona melhor com modelos lineares, uma vez que ela cria um espaço de alta dimensionalidade, em que poucos valores são não nulos. Os modelos lineares tratam cada palavra (feature) como uma variável independente, aprendem o peso para cada termo (coeficiente) e calculam um score que é uma soma ponderada, combinando perfeitamente com o que faz o TF-IDF, em que cada valor já é uma medida linear (proporcional à importância de cada termo).

O treino dos modelos demorou 20.50 min.

LIME Implementation

Explicador com:

- classes fake e real definida
- fixa o random_state=42
- diz para tratar o texto como tfidf
- usa palavras inteiras como features e não caracteres individuais

- controla o quanto as amostras perturbadas influenciam na explicação (atribui-se um valor de 1-5, dei 2)
- escolhe as palavras com maior impacto.

A função preditora é necessário porque o LIME necessita disso para passar versões alteradas do texto e ver como a previsão se altera.
Escolhe-se um texto do conjunto de teste para realizar e representa o local em que o LIME vai ser aplicado e explicar.
Exp serve para gerar a explicação local, onde o LIME cria versões perturbadas do texto, usa a função para ver como o modelo reage à mudança, ajusta o modelo linear simples que imita o comportamento do logreg só para aquele texto e calcula o peso de cada palavra para cada classe.
Ainda se calcula a local fidelity e o peso de cada palavra.
Testa-se também para diferentes seeds, para ver se variam os pesos das palavras.
Faz-se isso para cada modelo.

De seguida faz-se uma implementação global, para ver quais as palavras são mais importantes para as previsões dos modelos.