Deteção de *E-Mail* Fraudulento Através de Aprendizagem Automática

Cristiano Santos
Departamento de Informática
Universidade da Beira Interior
Manteigas, Portugal
cristiano.miguel.santos@ubi.pt

Sara Martins
Departamento de Informática
Universidade da Beira Interior
Mação, Portugal
sara.maria.martins@ubi.pt

***Resumo*—A Aprendizagem Automática é um mundo em constante desenvolvimento que permite a utilização de algoritmos e agentes para a resolução de problemas que exijam a execução de determinadas regras e procedimentos demasiado complexos para serem realizadas em tempo útil por humanos. A deteção de *e-mail* fraudulento é, numa era de utilização maliciosa dos recursos informáticos e tecnológicos, cada vez mais importante. Assim, neste documento, encontra-se uma hipótese de resolução para este problema — a utilização do algoritmo de aprendizagem *perceptron*, cuja precisão na deteção de fraude em *e-mail* pode atingir o valor aproximado de `95%`.**

Index Terms*—Aprendizagem Automática, Aprendizagem Supervisionada, *E-Mail*, *Spam

ACRÓNIMOS

AA Aprendizagem Automática
CCE Com Carateres Especiais
IDF *Inverse Document Frequency*
ML *Machine Learning*
SCE Sem Carateres Especiais
SGD *Stochastic Gradient Descent*
SVM *Support Vector Machine*
TFIDF *Term Frequency - Inverse Document Frequency*

I. INTRODUÇÃO

Este documento funciona como relatório do projeto final da unidade curricular de Aprendizagem Automática (AA). Este trabalho visa aplicar e consolidar os conhecimentos adquiridos nas aulas teóricas e práticas de AA, através de um programa que implemente algoritmos de *Machine Learning* (ML)[1] para detetar *e-mail* fraudulento.

O problema apresentado, de deteção de *e-mail* fraudulento, foi apresentado pelo estudante Cristiano Santos, um dos membros deste grupo de trabalho, no início do semestre. Em conjunto com a estudante Sara Martins, espera-se que consigam obter resultados satisfatórios para a sua resolução.

A. Objetivos

O principal objetivo deste projeto prende-se com a implementação de algoritmos de ML para a deteção de *e-mail* fraudulento.

[1]A nomenclatura em inglês é utilizada de modo a não haver confusão entre o nome da unidade curricular e o nome do campo na área da Inteligência Artificial, que partilham o mesmo nome em português.

A realização deste trabalho pretende culminar também no cumprimento dos seguintes objetivos:

- Compreender como os diversos algoritmos de ML podem ser aplicados sobre os dados em estudo;
- Implementar algoritmos de ML pelo uso de bibliotecas predefinidas na linguagem `Python`;
- Manipular as características (parâmetros) de cada algoritmo de modo a obter os melhores resultados.

B. Organização do Documento

Este relatório está estruturado da seguinte forma:

- Na secção I descrevem-se os pontos base do trabalho e os objetivos propostos;
- Na secção II detalham-se pormenores de implementação, tais como algoritmos e bibliotecas, e os pormenores do código de maior relevo;
- Na secção III apresentam-se os problemas encontrados e os resultados obtidos;
- Na secção IV descrevem-se as principais conclusões e o trabalho futuro relativo ao projeto.

II. IMPLEMENTAÇÃO

A implementação do programa é feita com recurso a um *dataset* já existente, que é apresentado e discutido na secção II-A. O código é escrito na linguagem `Python`, através de cinco algoritmos que são explicitados em maior detalhe na secção II-B. Os detalhes considerados de maior relevo no código são mostrados na secção II-C.

A. Dataset

Embora tenha sido proposto o desafio de criar um *dataset* de raiz, na língua portuguesa, o desconhecimento dos pormenores legais sobre a legislatura em vigor concernente à propriedade e proteção de dados foi um fator desmotivador desta abordagem. Assim, o grupo decidiu utilizar um *dataset* já recolhido e de domínio público de modo a não ter problemas a nível legal.

Como supramencionado, o *dataset* escolhido está disponível *online*. Os seus dados, na sua maioria categóricos (ou seja, texto), estão escritos na língua inglesa. Contém `5171` entradas, das quais `3672` são *e-mails* legítimos e `1499` são *e-mails* fraudulentos [1]. Estas entradas estão distribuídas por quatro colunas, a saber:

- A primeira coluna, `id`, identifica o número da entrada;
- A segunda coluna, `label`, identifica se um *e-mail* é legítimo (`ham`) ou fraudulento (`spam`);
- A terceira coluna, `text`, contém os dados de cada *e-mail*, com título e corpo do texto;
- A quarta coluna, `label_num`, tem uma função idêntica à coluna `label`, mas numericamente: se um *e-mail* é legítimo é identificado com o valor `0`, se um *e-mail* é fraudulento é identificado com o valor `1`.

Este *dataset* é uma fração de um conjunto de seis *datasets* com o mesmo propósito, em que todos os *datasets* têm uma quantidade similar de dados. No entanto, de forma a manter a simplicidade do projeto — e para garantir que os resultados obtidos o são de forma limpa — escolheu-se apenas utilizar o primeiro.

B. Algoritmos

1) K-Vizinhos Mais Próximos: O algoritmo K-Vizinhos mais próximos é um método não paramétrico usado para problemas de classificação e regressão. Neste algoritmo, todos os dados de treino são guardados e quando é necessário fazer uma predição, vão ser selecionados os K-valores de treino mais próximos como dados de *input*. Se for usado para um problema de classificação, o *output* corresponde a uma classe constituinte do *dataset* onde o objeto vai ser classificado através do "voto da maioria"dos seus vizinhos. Por sua vez, se for usado para um problema de regressão, o *output* é o valor próprio do objeto, sendo este obtido calculando a média dos valores dos K-vizinhos mais próximos [2].

Os passos básicos para usar este algoritmo são os seguintes:

- Escolher o valor de K e a distância métrica;
- Encontrar os K-vizinhos mais próximos da amostra que se quer classificar;
- Atribuir uma *label* à classe através do "voto da maioria"ou computar o valor médio para o interesse pretendido.

2) Naïve Bayes Multinomial: Este algoritmo é um classificador probabilístico baseado, como o próprio nome indica, no teorema de Bayes assumindo-se a independência entre *features*. É muito usado para problemas de classificação de texto [3].

Para aplicar este algoritmo é necessário seguir os seguintes passos:

- Recolher e pré-processar os dados;
- Treinar o classificador. Consiste em estimar a probabilidade de cada palavra pertencer a uma dada classe, usando o conjunto de treino;
- Classificar o conjunto de teste. Para cada documento no conjunto de teste, o classificador computa a probabilidade do documento pertencer a uma dada classe, baseando-se nas probabilidades de cada palavra do documento individualmente. O documento é então atribuído à classe onde revelou maior probabilidade.

3) Stochastic Gradient Descent: Em [4], encontra-se uma definição muito simples sobre aquilo que é o algoritmo *Stochastic Gradient Descent*, que é aplicado em código como mostrado no excerto 4 para a implementação dos algoritmos de *perceptron*, regressão logística e *Support Vector Machine*s, que são apresentados nas secções II-B3a a II-B3c. Aqui, explicitam-se os pontos mais importantes.

A descida do gradiente pretende encontrar o valor `x` para o qual o `y` é mínimo, ou seja, encontrar o ponto mais baixo no algoritmo de descida do gradiente sobre o qual a função objetivo opera.

O algoritmo *Stochastic Gradient Descent* (SGD) é iterativo, que começa num ponto aleatório numa função e percorre de modo descendente cada nível até atingir o ponto mais baixo dessa função. Considerando que é necessário calcular a derivada da função em respeito de cada característica, então é necessário executar a equação $numero\_de\_entradas \times caracteristicas$, que pode tornar-se muito custosa a nível de recursos computacionais.

A inserção de uma medida estocástica, ou seja, aleatória, permite a seleção aleatória de uma única entrada do conjunto de dados a cada iteração de modo a reduzir o número de cálculos que é necessário efetuar.

a) Perceptron: O modelo *perceptron* é um modelo computacional genérico que recebe um *input*, real ou booleano, a que se associa um conjunto de pesos e um *bias* (*threshold*). Os pesos são somados (*weighted sum*). Se o seu valor é menor que o *threshold*, então o *perceptron* retorna o valor `1`, se não, retorna `0` [5]. O objetivo é, então, encontrar o vetor `w` que pode classificar de modo ótimo *inputs* positivos e negativos nos dados.

b) Regressão Logística: A regressão logística é um modelo estatístico usado para tarefas de classificação. É um classificador binário e por isso pode apenas prever duas classes: `0` ou `1`. É usado quando a variável que se quer prever é binária [6].

O procedimento a ter em conta na utilização deste algoritmo é o seguinte:

1) Recolher e pré-processar os dados. Dividir os mesmos em conjunto de treino e de teste;
2) Treinar o classificador. Para tal, é necessário ajustar um modelo de regressão logística ao conjunto de treino usando um algoritmo de otimização para encontrar os melhores parâmetros a aplicar no modelo;
3) Testar o classificador, ou seja, avaliar a sua prestação no conjunto de teste usando métricas como a precisão e a exatidão.

c) Support Vector Machine: As *Support Vector Machine* (SVM) são um exemplo de aprendizagem supervisionada e permitem resolver problemas de classificação binária usando geometria para fazer previsões, ao invés das habituais probabilidades. Primeiramente, os dados são agrupados num gráfico semelhantemente à regressão logística, sendo estes separados depois de forma linear (através de uma reta ou plano). Novos dados que vão aparecer são classificados consoante a área onde se encontram [7].

A SVM pode aumentar a dimensão para conseguir separar os dados de forma linear (conduzindo a uma forma ótima). Ao encontrar hiperplanos é possível maximizar a margem definida por estes, alcançando assim uma maior área para classificação.

C. Detalhes de Implementação

De modo a validar o *dataset* que está a ser utilizado, é possível verificar se as colunas que estão a ser lidas são as que estão presentes no documento `.csv`. Esta verificação está demonstrada no excerto 1. Neste excerto, `email` representa o *dataframe* lido a partir do documento `.csv` e `colunas` é um vetor que guarda as colunas que é esperado receber: `"id", "label", "text", "label_num"`.

```
datatest.validate(email.columns, colunas)
```

Listing 1. Validação do *dataframe* lido, conforme as colunas recebidas.

A reformatação do texto para facilitar o processo de aprendizagem por parte do agente está mostrado no excerto de código 2. A explicação é feita passo a passo em seguida:

- A função `filterwarnings`, da biblioteca `warnings`, implementada na linha `2`, determina que os avisos de *pattern matching* não sejam impressos na consola/terminal aquando da execução do código [8];
- A variável `palavras_stop` guarda as *stop-words* da língua inglesa (a razão está explicada na secção III-A);
- As linhas `7` e `8` fazem uso de expressões regulares para reformatar o texto: a linha `7` substitui todos os carateres especiais por um espaço e a linha `8` substitui todos os espaços contíguos por um único espaço;
- Nas linhas `10` a `12`, itera-se por cada entrada do *dataframe*, verificando cada palavra: se não for uma *stop-word*, então é mantida;
- A condição `else`, na linha `14`, é de programação defensiva, de modo a que o programa pare na eventualidade de alguma das entradas não possuir texto onde este é esperado.

```
1  def reformatar(email):
2    warnings.filterwarnings("ignore")
3    palavras_stop = set(stopwords.words("english"))
4    for indice, linha in email.iterrows():
5      if type(linha["text"]) is str:
6        texto = ""
7        linha["text"] = re.sub("[^a-zA-Z0-9\n]", " ",
       linha["text"])
8        linha["text"] = re.sub("\s+", " ", linha["text
       "])
9        linha["text"] = linha["text"].lower()
10       for palavra in linha["text"].split():
11         if not palavra in palavras_stop:
12           texto = texto + palavra + " "
13         email["text", indice] = texto
14     else:
15       print ("Nao ha descricao textual para o indice
        ", indice, ".")
16   return email
```

Listing 2. Função de reformatação do texto de cada entrada do *dataframe*.

Outra função que é importante realçar é a de fatorização do conjunto de treino, nomeadamente, os valores de entrada, como mostrado no excerto 3. A linha `2` inicializa a variável `vetorizador` com a função `TfidfVectorizer`. São especificados dois parâmetros [9]:

- `min_df`: ao construir o vocabulário, ignora termos que têm uma frequência no documento estritamente inferior ao *threshold* definido (neste caso, `10`);
- `max_features`: constrói um vocabulário que apenas considera as máximas características ordenadas por frequência de termos ao longo do *corpus* linguístico.

Em seguida, os valores de entrada são utilizados para treinar a variável `vetorizador` (linha `3`) para aprender o vocabulário e o *Inverse Document Frequency* (IDF). Após o treino, os conjuntos de treino e teste são transformados por este e retornados pela função.

```
1  def fatorizar_texto(x_treino, x_teste):
2    vetorizador = TfidfVectorizer(min_df = 10,
       max_features = 5000)
3    vetorizador.fit(x_treino.values)
4    texto_treino = vetorizador.transform(x_treino.
       values)
5    texto_teste = vetorizador.transform(x_teste.values
       )
6    return texto_treino, texto_teste
```

Listing 3. Função de fatorização do texto com recurso à função `TfidfVectorizer`, que organiza o texto em matrizes de palavras consoante a sua importância.

A aplicação do algoritmo SGD em `Python` pode ser feito através da biblioteca `sklearn`. O `SGDClassifier` permite implementar modelos lineares com aprendizagem pelo algoritmo SGD, nomeadamente, redes neuronais, regressão logística e *Support Vector Machine*s, discutidos em maior detalhe na secção II-B, cuja implementação está especificada no excerto 4. Neste classificador, o parâmetro `alpha` representa o valor constante que multiplica o termo de regularização [10].

```
SGDClassifier(alpha = 1, loss = "hinge"),
SGDClassifier(alpha = 1, loss = "log_loss"),
SGDClassifier(alpha = 1, loss = "perceptron")
```

Listing 4. Uso do classificador SGD para a implementação de SVM (`hinge`), regressão logística (`log_loss`) e redes neuronais (`perceptron`).

Os algoritmos de K Vizinhos Mais Próximos e *Naïve Bayes* Multinomial são implementados como mostrado no excerto 5. Não são definidos quaisquer parâmetros específicos, utilizando-se todos os valores *default*.

```
KNeighborsClassifier()
MultinomialNB()
```

Listing 5. Inicialização das funções que implementam os algoritmos K Vizinhos Mais Próximos e *Naïve Bayes* Multinomial.

Por fim, é importante falar da implementação da função `CalibratedClassifierCV`, que executa a calibração de probabilidades com recurso, neste caso, a regressão logística. Esta função utiliza validação cruzada para estimar os parâmetros de um classificador e, subsequentemente, calibrá-lo, como mostrado na linha `1` do excerto 6. É sobre o novo classificador calibrado que é feito o treino do agente, e a partir do qual se preveem resultados.

```
classificador_calibrado = CalibratedClassifierCV(
    classificador)
classificador_calibrado.fit(texto_treino, y_treino)
y_previsto = classificador_calibrado.predict(
    texto_teste)
```

Listing 6. Implementação de um classificador calibrado com recurso à função `CalibratedClassifierCV` e a sua aplicação no código.

D. Conclusão

A implementação de um programa compreende também as execuções intermédias e a correção de erros, de modo a garantir o bom funcionamento do código e otimizar os resultados obtidos. Após esta etapa, é possível considerar o código como concluído e utilizar os resultados obtidos.

III. Discussão de Resultados

A implementação do programa em código apresentou diversos problemas, que são apresentados com maior detalhe na secção III-A. Na secção III-B, demonstram-se os resultados obtidos e referem-se algumas conclusões iniciais.

A. Problemas Encontrados

Como o *dataset* é composto por entradas do tipo `string` (ou seja, texto), a normalização não é possível ser executada. Haveria a possibilidade de utilizar *one-hot-encoding* de modo a transformar os textos em valores numéricos, no entanto, esta transformação iria implicar perdas de dados — bem como a impossibilidade de verificar as *stop-words* e o cálculo da importância de cada palavra, como se discute em seguida.

A importância que teria de ser dada a cada palavra no texto do *e-mail* é um dos problemas que necessita de ser ultrapassado, bem como a existência de palavras demasiado comuns. As bibliotecas de `Python` fornecem alguns recursos para lidar com este problema.

O módulo `corpus` da biblioteca `nltk` tem uma função que permite lidar com *stop-words* (a função `stopwords`) [11]. *Stop-words* são palavras que, num determinado idioma, são tão comuns que não fornecem informação útil para a maior parte dos processos de análise textual. A utilização desta função permite, portanto, eliminar as palavras mais comuns que não acrescentam valor ao texto (por exemplo, para a língua inglesa, *the*, *it*, *are*, *was*, *of*).

O valor *Term Frequency - Inverse Document Frequency* (TFIDF) é uma medida estatística utilizada para indicar a importância de uma palavra num documento em relação a outro documento ou a um *corpus* linguístico [12]. A biblioteca `sklearn` contém, no módulo `feature_extraction.text`, a função `TfidfVectorizer`, que permite automatizar este processo.

Na análise prévia do *dataset*, o grupo notou a desformatação dos diferentes corpos de texto nos *e-mails* fornecidos, como, por exemplo, mais do que um espaço em branco entre as palavras. Para corrigir este problema, recorre-se a uma expressão regular para selecionar todos os espaços em branco contíguos e substitui-se por um único espaço em branco, como mostrado no excerto de código 2.

A função `SGDClassifier`, como discutido em II-C e implementado no excerto de código 4, pode ser utilizada para implementar vários algoritmos. A princípio, tentou-se utilizar um vetor contendo as palavras-chave para cada algoritmo; no entanto, como não foi possível fazer a implementação devida deste método, escolheu-se fazer a chamada manualmente para cada um dos algoritmos.

B. Resultados

De modo a obter resultados mais conclusivos, realizaram-se quatro rondas de testes. Estes testes são mutuamente exclusivos e não são extensivos; no entanto, o grupo considerou estes como sendo os de maior relevância e, por isso, apresenta-os nas tabelas seguintes.

A tabela I apresenta a precisão dos algoritmos sem a aplicação de validação cruzada sobre o conjunto de treino. Os valores para a previsão encontram-se entre `89%` e `98%`, com arredondamento por excesso. Embora os valores sejam bastante elevados, é necessário ter em consideração que a ausência de validação cruzada pode aumentar o número de falsos positivos, o que é indesejável. Note-se que o classificador que obtém melhores resultados é o de *perceptron*, com uma precisão de `97.87%`.

Tabela I
Precisão em percentagem para os cinco algoritmos aplicados, sem a aplicação de validação cruzada sobre o conjunto de treino.

Algoritmo	**Precisão (%)**
K Vizinhos Mais Próximos	96.91
Naïve Bayes Multinomial	96.39
Perceptron	97.87
Regressão Logística	89.24
SVM	88.98

A calibração de cada classificador foi outro aspeto a ser testado. Os resultados para a execução sem a calibração dos classificadores estão apresentados na tabela II e estão presentes no espetro entre as percentagens `71%` e `95%`, arredondadas por excesso. Neste caso, é o classificador de *Naïve Bayes* Multinomial a atingir a maior precisão, de `94.57%`. Ainda assim, a calibração é mantida para o código final de modo a equilibrar os resultados dos diferentes algoritmos entre si.

Tabela II
Precisão em percentagem para os cinco algoritmos aplicados, sem a aplicação de calibração sobre o classificador.

Algoritmo	**Precisão (%)**
K Vizinhos Mais Próximos	94.84
Naïve Bayes Multinomial	94.57
Perceptron	89.50
Regressão Logística	70.99
SVM	70.99

O último ponto considerado para testar foi a remoção dos carateres especiais — possíveis carateres acentuados e pontuação — de modo a verificar a influência que estes podem ter sobre o resultado final. Na tabela III encontram-se os valores das precisões Com Carateres Especiais (CCE) e Sem Carateres Especiais (SCE). Os resultados são iguais para todos os algoritmos, exceto para o de *perceptron*, cuja precisão aumenta com a ausência de carateres especiais. O valor mais

baixo pertence ao algoritmo de SVM, 85.82%, e o valor mais alto é de novo o do algoritmo de *perceptron*, 95.76%.

Tabela III
PRECISÃO EM PERCENTAGEM PARA OS CINCO ALGORITMOS APLICADOS, COM A EXISTÊNCIA DE CARATERES ESPECIAIS NO TEXTO E A SUA AUSÊNCIA.

Algoritmo	**Precisão CCE (%)**	**Precisão SCE (%)**
K Vizinhos Mais Próximos	94.66	94.66
Naïve Bayes Multinomial	94.38	94.38
Perceptron	94.94	95.76
Regressão Logística	86.19	86.19
SVM	85.82	85.82

As figuras 1 a 5 representam as matrizes de confusão relativas a cada um dos cinco algoritmos implementados no código, especificados em II-B. Cada uma das matrizes é composta por duas linhas e duas colunas, em que as linhas representam as classes originais e as colunas representam as classes previstas [2]:

- O canto superior esquerdo (linha 0, coluna 0) representa o número de verdadeiros negativos;
- O canto superior direito (linha 0, coluna 1) representa o número de falsos positivos;
- O canto inferior esquerdo (linha 1, coluna 0) representa o número de falsos negativos;
- O canto inferior direito (linha 1, coluna 1) representa o número de verdadeiros positivos.

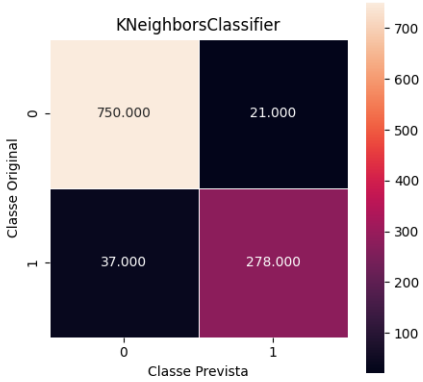

Figura 1. Matriz de confusão para a implementação do algoritmo de K Vizinhos Mais Próximos.

[2]De relembrar que, para o *dataset* em questão, se um *e-mail* é legítimo é identificado com o valor 0, se um *e-mail* é fraudulento é identificado com o valor 1.

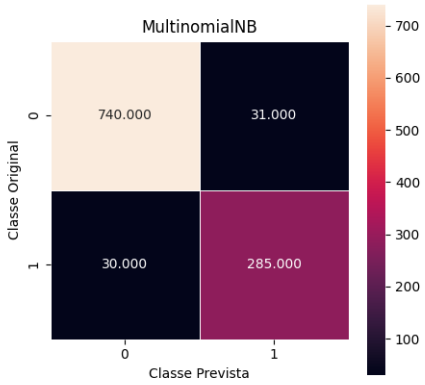

Figura 2. Matriz de confusão para a implementação do algoritmo de *Naïve Bayes* Multinomial.

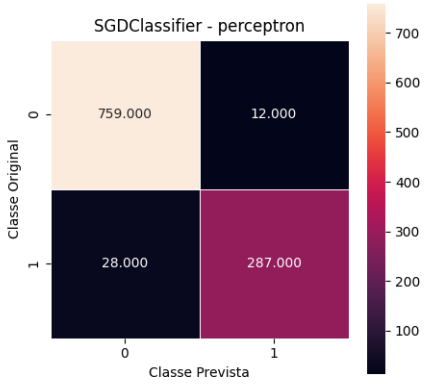

Figura 3. Matriz de confusão para a implementação do algoritmo de *Perceptron*.

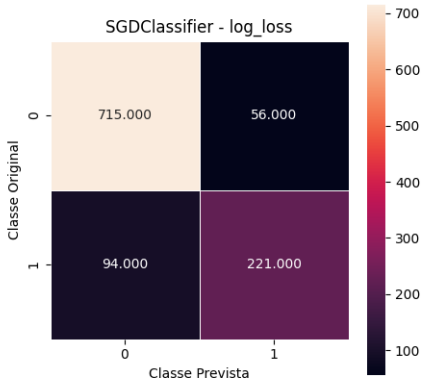

Figura 4. Matriz de confusão para a implementação do algoritmo de Regressão Logística.

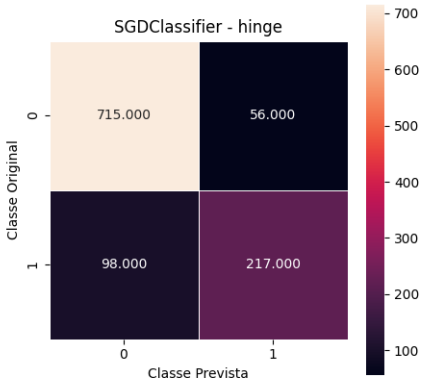

Figura 5. Matriz de confusão para a implementação do algoritmo de SVM.

C. Conclusão

A superação dos problemas é uma fase importante na escrita de código, pois é através da ultrapassagem destes que é possível melhorar o código. Os resultados obtidos permitem concluir que, de forma geral, é o classificar que implementa o algoritmo de *perceptron* a obter os resultados mais satisfatórios, com uma precisão final de cerca de `95%` e a menor quantidade de falsos positivos e falsos negativos, como é possível verificar na tabela III e na figura 3.

IV. Conclusão

A Aprendizagem Automática é um tópico cada vez mais presente e ubíquo no quotidiano da sociedade atual. Os desafios que o mundo propõe podem ser, com cada vez menor custo e investimento, resolvidos por máquinas a que se dá o privilégio de aprender e que o conseguem fazer com cada vez menor envolvimento por parte do programador.

A análise dos diferentes algoritmos de aprendizagem supervisionada para a concretização deste projeto permitiu uma compreensão mais profunda daquilo que é a ML. A implementação, na linguagem `Python`, faz recurso a bibliotecas já pertencentes e permite a obtenção de resultados, que podem ser manipulados de acordo com as necessidades do programador.

Após a implementação dos cinco algoritmos propostos, definidos na secção II-B, os resultados obtidos permitem concluir que o algoritmo que consegue atingir melhor precisão é o de *perceptron*.

Ainda assim, o trabalho pode ser complementado no futuro. Um dos pontos de melhoria será a obtenção de resultados mais extensivos, bem como a implementação de outros algoritmos de aprendizagem supervisionada — por exemplo, Florestas Aleatórias — ou a implementação de algoritmos de aprendizagem não supervisionada — por exemplo, algoritmos de *clustering*. De um modo mais ambicioso, pode-se almejar para a criação de um *dataset* completo e extenso na língua portuguesa.

Referências

[1] I. Androutsopoulos, "Spam Mails Dataset," https://www.kaggle.com/datasets/venky73/spam-mails-dataset, 2006, [Online] Último acesso a 09 de dezembro de 2022.

[2] IBM, "What is the k-nearest neighbors algorithm?" https://www.ibm.com/topics/knn, 2022, [Online] Último acesso a 26 de dezembro de 2022.

[3] Shriram, "Multinomial Naive Bayes Explained," https://www.upgrad.com/blog/multinomial-naive-bayes-explained/, 2022, [Online] Último acesso a 26 de dezembro de 2022.

[4] A. V. Srinivasan, "Stochastic Gradient Descent," https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31, 2019, [Online] Último acesso a 14 de dezembro de 2022.

[5] A. L. Chandra, "Perceptron Learning Algorithm," https://towardsdatascience.com/perceptron-learning-algorithm-d5db0deab975, 2018, [Online] Último acesso a 16 de dezembro de 2022.

[6] IBM, "What is Logistic regression?" https://www.ibm.com/topics/logistic-regression, 2022, [Online] Último acesso a 16 de dezembro de 2022.

[7] R. Gandhi, "Support Vector Machine," https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47, 2018, [Online] Último acesso a 26 de dezembro de 2022.

[8] P. S. Foundation, "warnings - Warning control," https://docs.python.org/3/library/warnings.html, 2022, [Online] Último acesso a 19 de dezembro de 2022.

[9] scikit-learn developers, "sklearn.feature_extraction.text.TfidfVectorizer," https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html, 2022, [Online] Último acesso a 19 de dezembro de 2022.

[10] ——, "sklearn.linear_model.SGDClassifier," https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier, 2022, [Online] Último acesso a 14 de dezembro de 2022.

[11] Pythonstop, "NLTK stop words," https://pythonspot.com/nltk-stop-words/, 2022, [Online] Último acesso a 19 de dezembro de 2022.

[12] B. Stecanella, "Understanding TF-ID: A Simple Introduction," https://monkeylearn.com/blog/what-is-tf-idf/, 2019, [Online] Último acesso a 19 de dezembro de 2022.