Inteligência Artificial para Detecção de Emails Indesejados (spam)

Marcos Paulo Diniz
Universidade de Brasília
Departamento de Ciência da Computação
Brasília, Brasil
marcosdiniz@aluno.unb.br

***Resumo*—O trabalho busca mensurar a capacidade de aplicar um modelo para a detecção de emails não desejados. Para isso, foi impenentado o algoritmo do MLP e RBF, em busca de mensurar qual teria melhor capacidade de aprendiado e resolver o problema dos *spams*.**

***Index Terms*—E-mail, indesejado, spam, aprendizado, MLP, RBF.**

I. Introdução

Esse trabalho tem como objetivo usar o aprendizado de máquina supervisionado, por meio dos algoritmos *Perceptron Multicamadas (MLP)* e *Radial Basis Function (RBF)*, para identificar emails spams reebidos. No *dataset* utilizado haviam 4601 dados, entre spams e não spams.

Para a implementação do algoritmo foi-se usado a linguagem de programação Python (versão 3.6.5), com auxílio das bibliotecas *Sklearn, Matplotlib, Pandas e Numpy*.

Por se tratar de um algoritmo de aprendizado de maquina supervisionado, será preciso separar a base em duas partes, sendo uma para treinamento e a outra para testes, essa separação será de 90% e 10%, respectivamente. Além dessa divisão, também foi usado o modelo de validação cruzada *cross validation*, aplicando uma divisão de 10 conjuntos.

Para o bom compreendimento do trabalho realizado, é preciso, primeiramente, entender como o MLP e o RBF funcionam.

No MLP temos uma rede neural profunda composta por mais de um perceptron, com uma camada de entrada que recebe o sinal e uma de saída que faz a decisao ou predição, além de um número arbitrário de camadas implícitas responsáveis pelo processamento bruto de dados do MLP. MLPs sao usados para problemas de aprendizagem supervisionada pela capacidade de aprender a correlação de um dataset com uma base de treino com entradas e saídas. A fase de treino consiste no ajuste de parâmetros e desvios com o intuito de minimização de erros, contando com o auxílio de tecnicas de retropropagação para o ajuste dos fatores em relação ao erro.

O RBF é uma rede neural com função de base radial é um tipo de rede neural artificial em que as funções de ativação são funções de base radial. Assim como no MLP, a saída da rede é o resultado da combinação linear das funções de ativação, tendo como maior diferença o fato de a camada implícita corresponder a funções radiais, normalmente uma gaussiana. O processo de treino de uma rede de base

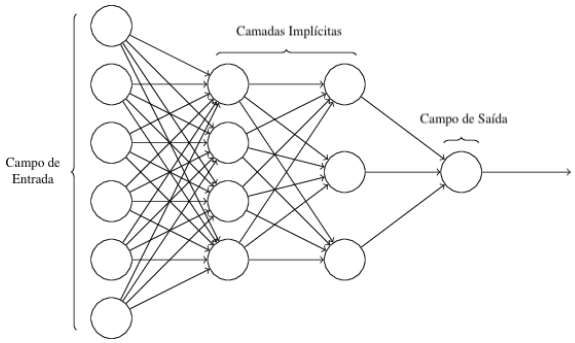

Figura 1. Exemplo de Rede Neural

radial consiste inicialmente no treino da camada oculta usando retro propagação, sendo que, para cada nó implícito devem ser encontrados seus receptores e a variancia (no caso da função radial). Para a realização dessa etapa podem ser usados metodos como agrupamento k-means, em que cada grupo e determinado pelas entradas. O número de grupos e escolhido arbitrariamente, enquanto o número de dimensoes dos centros é determinado pelo set de dados. Após isso, os vetores de pesos entre a camada de saída e a camada oculta sao ajustados de acordo com uma função objetiva.

II. Analise do Experimento

Para a realização do experimento a primeira etapa foi carregar o dataset e entender como os dados estavam distribuidos nele.

Após isso, foi seperado os conjuntos de trainamento e teste, conforme explicado anteriormente.

Tendo os dados já separados nos conjuntos de treinamento e teste, foi decidido usar a técnica do *Grid Search*, que basicamente dando uma sequencia de parâmetros ele faz a busca de quais parâmetros retornam um resultado melhor e, nesse caso, como colocamos vários parâemtros diferentes ele faz a combinação de todos esses em busca da combinação ideal com os dados disponíveis, com isso o modelo do MLP ficará melhor para o nosso *dataset*.

Os parâmetros usados foram: *activation*, *solver* e *hidden_layer_sizes*, na tabela a seguir podemos ver as variações dos dados q foram disponibilizados e bombinados entre si em busca do melhor resultado.

Tabela I
TABELA DE PARÂMETROS

activation	solver	hidden_layer_sizes
relu	lbfgs	50
tanh	sgd	100
identity	adam	150
logistic		200
		250
		300
		350
		400
		450

Após aplicar o *Grid Search* com os dados acima, o retorno do melhor filtro aplicado pode ser visto na tabela abaixo.

Tabela II
MELHORES PARÂMETROS APÓS O GRIDSEARCH

activation	tanh
hidden_layer_sizes	250
solver	adam

Com esse filtro foi possível chegar numa acuária de 94,44%, quando só pbservado o conjunto de treinamento, após passar pelo *predict*, para validação, conseguiu-se uma acurácia um pouco superior, quase chegando aos 96% de acerto. A seguir tem-se a matriz de confusão obtida.

Tabela III
MATRIZ DE CONFUSÃO PARA O KERNEL LINEAR

	Not Spam	Spam
Not Sapm	175	13
Spam	7	265

Com o resultado obtido, foi aplicado novamento um *Grid Search*, dessa vez variando somente o valor do *hidden_layer_sizes*, uma vez que o valor dado na execução anterior foi de 250, dessa vez estamos em busca de um valor entre 200 e 300, já que esses dois valores já foram tastados anteriormente e o escolhido foi o intermediário, com isso surgiu o questionamento se não teria um outro valor nesse intervalo que retornaria uma acurácia superior.

Após a execução do novo *Grid Search*, estranhamente foi retornado que o melhor valor seria 240, porém a acurácia dada doi de 94,39%, inferior a obtida anteriormente.

Em seguida, aplicamos o modelo do MLP padão, isto é, sem nenhuma modificação de parâmetros, com objetivos de ver se o uso da técnica do *GridSearch* foi realmente eficaz. A matriz de confusão obtida pode ser vista abaixo.

Tabela IV
MATRIZ DE CONFUSÃO PARA O MLP PADRÃO

	Not Spam	Spam
Not Sapm	175	14
Spam	24	248

Nesse modelo foi obtido uma acurácia pouco abaixo dos 92%. O modelo usado anterirormente conseguiu uma acurácia superior, como era esperado.

O que pode ser notado é que para casos do email não ser considerado spam, nos dois modelos foi obtido os mesmo resultados, com isso é possível entender que o algoritmo padrão tem dificuldade na classificação de spams, sendo melhorado com a alteração de alguns de seus parâmetros.

Após os resultados com o MLP, passamos aplicar o RBF usando os mesmos conjuntos de treinamento e testes. Para o RBF foi usado o algoritmo do Rob Romijnders, disponível no GitHub do autor.

Primeiramente foram feitos alguns ajustes no algoritmo dele, a fim de adaptar e aplicar para a nossa base, as alterações foram dos parâmetros usados, além de mudar o dataset para o nosso.

O RBF do Romijnders usa como principal impementador da sua rede a bibliteca do *TensorFlow*. Essa biblioteca é voltada para criação e treinamento de redes neurais para detectar e decifrar padrões e correlações, análogo à forma como nós aprendemos e raciocinamos.

Na table abaixo pode ser visto os resultados da acurácia obtidos.

Tabela V
VALORES DE ACURÁCIA

Máximo	82,10%
Médio	74,34%
Mínimo	59,47%

Nos gráficos a seguir pode-se ver com mais detalhe o resuldado obtido, bem como o comportamento ao longo da execução.

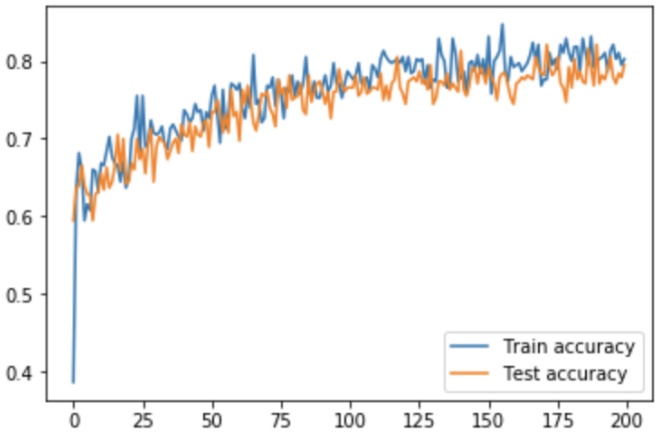

Figura 2. Acurácia do RBF

Como pode ser observado, no geral o resultado do comportamento foi bem semelhante, no entanto ficou abaixo do esperado, o gráfico parece estar um pouco deslocado, reflexo das taxas obtidas.

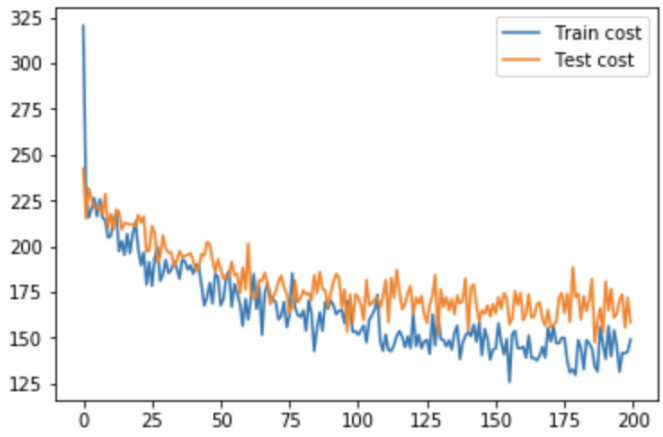

Figura 3. Custo do RBF

Com o resultado inferior do esperado, novamente foi executado o algorítmo, dessa vez aumentando o número máximo de interações de 10000 para 100000, e também o número de dados coletados, pegando todos os múltiplos de 20, invés dos multiplos de 50, como usado anteriormente.

Após essas alterações e excutado o algoritmo novamente, o resultado obtido foi melhor, como pode ser visto na tabela abaixo.

Tabela VI
VALORES DE ACURÁCIA

Máximo	87,10%
Médio	80,58%
Mínimo	56,57%

Na imagem abaixo podemos ver o gráfico da taxa de acuráxia, vale ressaltar que como foi configurado o código teria 5000 dados no gráfico, por conta da ilegibilidade (em vista da grande quantidade de dados) foi feita uma redução de para 10% dos dados, usando apenas 500.

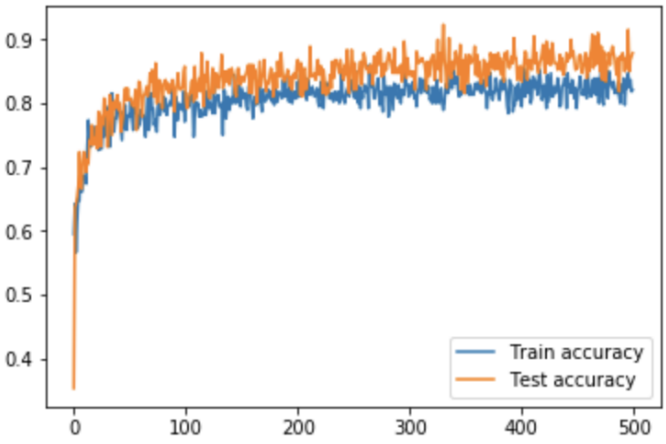

Figura 4. Taxa de acerto de teste x C

Como pode ser notado, o resultado obtido com os ajustes feitos redsultou num aumento de cerca de 6% na taxa média de acerto, fazendo com que esse valor passasse um pouco dos 80%, chegando a uma taxa de acurácia já mais aceitável.

III. CONCLUSÕES

Após a análise de todos os resultados obtidos, é notável que para se conseguir uma boa taxa de acerto com o MLP não é interessante aplicar somente o algoritmo padrão disponível na biblioteca, isto é, sem alteração dos parâmetros.

Contudo, vale ressaltar que é importante executar o MLP padrão para fins de comparação e com isso verificar as mudanças após a realização dos ajustes dos parâmetros.

Com o MLP padrão foi conseguido uma acurácia bem próxima do com as modifições, sendo esse valor acima dos 90%, o que o deixa bem confiável, mesmo sem mudar os parâmetros. Já no modelos após o *Grid Search*, a taxa de acurácia foi acima dos 95%, notávelmente mais confiável e fazendo valer aplicar a técnica de busca dos melhores parâmetros. Por fim, o último modelo do MLP usado foi, novamente, usando o *Grid Search*, dessa vez era com o objetivo de buscar uma taxa superior a obtida no anterior, tentando aperfeiçoá-lo, contudo, foi percebito uma taxa um pouco inferior, no caso o algoritmo errou um dos parâmetros. Na tabela a seguir pode-se ver os resultados obtidos com o MLP.

Tabela VII
ACUÁCIAS COM O MLP

MLP Padrão	91,75%
MLP após o GridSerachCV	94,44%
MLP retreinado com o GridSearchCV	94,39%

Já no caso do RBF aplicado para o meso dataset, foi observado um resultado bem inferiror ao esperado, no modelo similar ao proposto por Romijnders, com a bvaixa taxa de acurácia, foi feito alguns ajustes nos algoritmo, e com isso chegamos a uma taxa média de cerca de 80%, contra a taxa máxima do RBF anterior de pouco acima desse valor.

Na tabela abaixo podemos ver os resultados em ambos os testes.

Tabela VIII
VALORES DE ACURÁCIA

	Primeiro RBF	**Segundo RBF**
Máximo	82,10%	87,10%
Médio	74,34%	80,58%
Mínimo	59,47%	56,57%

Diante dos resultados, é notável que para o MLP é se tornar mais confiável é interessnte aplicar a técnica da busca pelos melhores parâmetros apresentada, porém a rebusca dos dados aplicada sobre o resultado obtido não foi eficaz no modelo, porém não deve ser descartada.

Já no caso do RBF, é interessante aplicar vários treinamentos com diferentes valores, por se tratar de um algoritmo

personálizado e feito por um autor cuja o dataset continha dados distintos, é considerável o resultado obtido não ser tão satisfatório quanto o MLP, porém conseguimos passar os 80% de acurácia, o que faz pensar que com mais tentativas de ajustes seria possível alcançar um resuldado mais satisfatório ainda.

REFERÊNCIAS

[1] Bishop, C. Pattern Recognition and Machine Learning. Springer, 2006

[2] Mitchell, T. Machine Learning. McGraw Hill, 1997.

[3] Lichman, M. (2013). UCI Machine Learning Repository. Irvine, CA: the University of California, School of Information and Computer Science.

[4] Bird, S., Klein, E., and Loper, E. (2009). Natural language processing with Python: Analyzing text with the natural language toolkit. Sebastopol, CA: O'Reilly Media.

[5] Romijndershttps, Rob https://github.com/RobRomijnders/rbfn_learnable/blob/master/rbfn_tf_main.py