# Relatório do Trabalho II
# Reconhecedor de sementes com uma rede neural Kohonen

**Ivair Puerari**[1]

[1]Universidade Federal Fronteira Sul (UFFS)
Chapecó – SC – Brasil

[2]Campus Chapecó – Ciência da Computação – Inteligência Artificial
27 de junho 2018

puerariivair@gmail.com

## 1. Problema

O trabalho consiste em desenvolver um reconhecedor de sementes utilizando-se de uma rede neural, do tipo Kohonen. Redes Neurais Kohonen são um tipo de Mapas Auto-Organizáveis (SOM), não supervisionadas baseadas em aprendizado competitivo.

Fundamentalmente, os neurônios de um SOM, como forma de aprendizado, competem entre si para decidir qual dos neurônios sera ativado a partir de um dado de entrada, assim, reconhecendo aquele neurônio vencedor como representante daquele padrão. Logo, obtem-se a capacidade de organizar dados em agrupamentos, onde cada agrupamento reconhece um tipo de padrão que os dados demonstram.

Com isso, o objetivo é realizar o agrupamento dos padrões de dados sobre o conjunto de dados que representam classes de sementes, disponibilizado pela University of california, Irvine (UCI), em http://archive.ics.uci.edu/ml/datasets/seeds. O conjunto possui 210 exemplos, onde cada exemplo é constituído com sete parâmetros geométricos da estrutura de uma semente, mensurados em valores reais contínuos, que caracterizam cada exemplo para uma classe de semente, sendo que, no total, o conjunto de dados representa três classes de sementes:

1. Kama
2. Rosa
3. Canadiam

## 2. Preparação dos dados

O conjunto de dados sementes, como dito anteriormente, é formado por 210 exemplos classificados em três classe, cada classe representado por 70 exemplos dentro do conjunto de dados. Um exemplo dentro do conjunto de dados é formado por sete características que representam a classe de uma semente e um ultimo valor sendo a classe em si, a resposta para aquelas características.

| Area | Perimeter | Compactness | lenght of kernel | width of kernel | coefficient | groove | label |
|---|---|---|---|---|---|---|---|
| 15.26 | 14.84 | 0.871 | 5.763 | 3.312 | 2.221 | 5.22 | 1 |
| 17.63 | 15.98 | 0.8673 | 6.191 | 3.561 | 4.076 | 6.06 | 2 |
| 13.07 | 13.92 | 0.848 | 5.472 | 2.994 | 5.304 | 5.395 | 3 |

**Tabela 1.**Tabela com os exemplos do conjunto de dados $seeds\_dataset.txt$

É possível visualizar todos os exemplos do conjunto de dados no arquivo $seeds\_dataset.txt$.

O conjunto de dados sera dividido em dois conjuntos, conjunto de treino e conjunto de teste. O conjunto de treino sera formado por 70% dos dados, enquanto o conjunto de teste sera formado pelo os 30% restantes, assim, criando a capacidade da rede neural de melhor generalizar os exemplos a ele mostrado. Apenas as sete características serão usadas no conjunto de teste e treino, a label com os valores da classe sera omitida, para que a rede neural descubra os padrões de modo competitivo entre os seus neurônios.

Ao visualizar no arquivo $seeds\_dataset.txt$ ira notar que o conjunto de dados vem ordenado, ou seja, todos os 70 primeiros exemplos se referem a classe 1, Kama, e assim sucessivamente até os últimos 70 exemplos pertencendo a classe 3, Canadian. O qual interfere na construção do modelo de rede neural, pois, para uma melhor capacidade de aprendizado da rede neural,bem como, mensurar o desempenho do modelo que sera construída, como foi dito anteriormente, é necessário dividir o conjunto de dados previamente em dois conjuntos, conjunto de treino e conjunto de teste. O motivo é se o conjunto fosse dividido como os dados se despõem no momento, em 70% para treinamento e 30% para teste, os conjunto de treino iria ter em sua grande maioria exemplos da classe 1 e 2 enquanto no conjunto de teste apenas exemplos da classe 3. O que vem de contra a ideia de criação de um modelo com capacidade de encontrar padrões, assim, o primeiro tratamento nos dados que deve ser realizado, se refere há um melhor balanceamento na disposição dos exemplos dentro do conjunto de dados.

Para isso, foi utilizado a função $np.random.suffle(X[])$ da biblioteca $numpy$, onde é passado o conjunto de dados como parâmetro da função, retornando uma matriz com o conjunto de dados embaralhado aleatoriamente. Há de se notar que a aleatoriedade da função np.random.suffle poderá criar conjunto de dados com tamanho diferentes em cada inicialização, logo, foi criado uma função $process()$ para salvar a base de dados gerada pela função em um arquivo $datasuffle\_.txt$. Assim é possível salvar um conjunto de dados e aplica-lo diversas vezes em seu modelo, setando o valor de flag da variável $SUFFLE\_FLAG$ igual á 1 e atribuindo a variavel $VERSION$ para um numero da versão que sera gerado o novo conjunto de dados, exemplo $datasuffle1.txt$ que sera salvo no diretorio que se encontra o programa, ou toda vez que inicializar, ira aplicar um novo formato de conjunto de dados ao modelo, com a variável $SUFFLE\_FLAG$ igual á 0.

Uma propriedade desse conjunto de dados, são os valores nele contido, valores reais contínuos. É possível visualizar um exemplo para cada classe na **Tabela 1** onde a escala de valores entre as caracteristicos estão entre 0.000001 e 21.99999 variando em diferentes escalas. Dados como esses dispostos podem influenciar nos resultados de aprendizado, assim, foi desenvolvido a função $featureScaling(X)$ , que tem como objetivo transformar os valores do conjunto de dados em escalas controladas entre 0 e 1.

$$x' = \frac{x - \bar{x}}{\sigma} \tag{1}$$

A função $featureScaling(X)$ segue a **equação 1** como base. É feito a subtração para cada característica pela média do exemplo que a característica pertence, dividido pelo desvio padrão do conjunto de dados. Assim todos os valores do conjunto de dados

estará em uma escala entre 0 e 1. Logo, o conjunto de dados esta pronto para ser dividido em treino e teste em suas devidas proporções estáveis.

## 3. Algoritmo e Bliblioteca SOM

Para construção do modelo, foi utilizado a biblioteca pyhton Minisom. Minisom é baseada em Numpy, e implementa Mapas auto organizáveis (SOM), sua instalação é pratica, somente necessário o comando de linha $pip\ install\ minisom$, disponível em https://github.com/JustGlowing/minisom.

Após divisão do conjunto treino e teste, a biblioteca minisom usa os dados organizados em uma matriz Numpy.Logo, para isso foi utilizado a função np.loadtxt() para carregar os dados, já, para o formato numpy direto do arquivo que contém os dados, assim cada linha da matriz ira corresponder a um exemplo. O arquivo único e principal de desenvolvimento do modelo é o $Kohonen.py$, neste arquivo esta contido as funções de processamento de dados, bem como, a construção do modelo, contendo a chamada da biblioteca $from\ minisom\ import\ MiniSom$, bem como, $matplotlib$ para construir gráficos que irão auxiliar na solução.

Logo, para construção do modelo, a biblioteca minisom, fornece a função $Minisom(parameters)$ Os parâmetros exigidos pela função podem ser visualizados na

```
class MiniSom(object):
    def __init__(self, x, y, input_len, sigma=1.0, learning_rate=0.5,
                 decay_function=None, neighborhood_function='gaussian',
                 random_seed=None):
        """Initializes a Self Organizing Maps.

```

**Figura 1. Representação da função de construção do modelo SOM.**

**Figura 1**, $x$ e $y$ são as dimensões do mapa SOM, que serão utilizadas para instanciar uma matriz numpy, representando o mapa. As dimensões determinam a quantidade de neurônios que a rede ira ter, quanto mais neurônios existem , mais possível o mapeamento se torna, mas a complexidade computacional da fase de treinamento da rede aumenta também. Após pesquisas, foi utilizado como numero de neurônios a equação encontrada no artigo [G. Akçapnar and Cosgun 2014] que pode ser vista na **Equação 2** onde M sera o numero de neurônios e N é o numero de exemplos do conjunto de dados.

$$M = 5\sqrt{N} \tag{2}$$

O parâmetro $input\_len$ é numero de características do conjunto de dados, e que o SOM sera exposto, ou seja, 7, pelo numero de características que o conjunto de dados do problema tem. $Sigma$ é o fator variante da função de vizinhança, função vizinhança representada pelo parâmetro $neighborhood\_function$ que pode ser escolhido como uma função gaussiana ou mexican hat wavelet. O $learning\_rate$ a taxa de aprendizagem do SOM que junto com o parâmetro $sigma$ a cada iteração é decrementada segundo uma função que pode ser atribuída no parâmetro $decay\_function$, onde ira se adequar ao mapa a cada iteração. O ultimo parâmetro $random\_seed$ que é um numero gerado aleatoriamente, utilizado para estimativas de probabilidade,mas,se não dado como entrada,

a cada geração de um novo modelo, um numero diferente é gerado, e consequentemente resultados diferentes, sendo assim, aconselhável atribui-lo um valor inteiro fixo, como forma de mensurar o desempenho da rede.

Dito isso, e apresentado a função de construção do modelo, é necessário inicializar os pesos dos neurônios, para isso, utiliza-se a função $random\_weights\_init(data)$, que gera um numero aleatório entre 0 e o maior índice do conjunto de dados, assim, gera-se um índice aleatório de um exemplo, onde utiliza-se o exemplo selecionado como peso inicial para um neurônio da rede e assim sucessivamente até todos os neurônios forem inicializados. O próximo passo é treinar a rede, a biblioteca minisom oferece dois modos de treinamento, primeiro o $train\_random(n_iteration, data)$, onde a função seleciona um exemplo randomicamente dentro do conjunto de dados, para entrada, sendo este o que sera associado ao neurônio vencedor, e assim, sucessivamente, até não existir números não retirados entre 0 e o maior índice do conjunto de dados, ou seja, até todos os exemplos forem selecionados. O segundo é o $train(n_iteration, data)$ que pode ser dito como, a função convencional, pois, todos os exemplos serão selecionados na ordem que despõem dentro do conjunto de dados.

Ao final, é possível saber qual o neurônio vencedor para um dado exemplo, utilizando-se da função $winner(data[i])$ que ira retornar a posição i,j do neurônio associado a aquele dado, ou seja, a aquele padrão de exemplo, que tem como alvo uma classe de semente.

## 4. Construção e Testes

O conjunto de dados, após o processamento dos dados, é divido em dois conjuntos, treino e teste, como $Xtr$ e $Xte$, mais, suas respectivas labels, como $ytr$ e $yte$. O conjunto de treino é formado por 70% dos exemplos contra 30% dos exemplos formando o conjunte de teste, que pode ser visualizado na **Figura 2**. Como dito anteriormente, é possível exportar um conjunto de dados embaralhado e escalável entre 0 e 1. E com isso, foi gerado três formatos de conjunto de dados, com embaralhamentos diferentes, logo, também conjunto de treino e testes diferentes. Assim ao total, os testes de construção do modelo ocorreram em base, em três formatos de conjuntos de treino e teste diferentes.

Para verificar a ocorrência de classes dentro de cada divisão do conjunto, foi implementada a função $occurrence(y)$ onde retorna um dicionario com a label e quantidade de ocorrências para cada classe do problema, bem como, gera um gráfico com os valores.

As dimensões do SOM, bem como, a quantidade neurônios, levou em base a **Equação 1.**, como a equação retorna apenas o numero de neurônios, é necessário usar uma função raiz quadrada,$np.sqrt()$ e arredonda-lá para um numero inteiro, o resultado é o numero de dimensões do mapa, assim, SOM sera uma matriz 6x6 com 36 neurônios. Ao construir o modelo é necessário escolher valores para os parâmetros $sigma$ e $learning\_rate$ e que os parâmetros se adéquem a cada iteração utilizando-se de uma função de decaimento que pode ser atribuída ao parâmetro $decay_function$. Logo, primeiro foi criado a função de decaimento, onde a própria biblioteca disponibiliza:

$$lambdax, current\_iteration, max\_iter : x/(1 + current\_iteration/max\_iter)$$

x sera $sigma$ ou $learning\_rate$, current_iteration o numero da iteração no momento e max_iter a quantidade de iterações do modelo. Assim, é possível escolher valores

para $sigma$ e $learning\_rate$, bem como, o numero de iterações do modelo. Os valores de $sigma$ foram escolhidos entre [0.8 - 1] e para $learning\_rate$ entre [0.1 - 0.5] e com 1000 sendo o numero de iterações do modelo. Alguns exemplos de testes realizados com o terceiro conjunto de dados, para diferentes combinações de configuração:

| Iterações | sigma | rate_learnig | Média de acertos |
|---|---|---|---|
| 1000 | 0.8 | 0.5 | 80.85% |
| 1000 | 0.8 | 0.2 | 87.29% |
| 1000 | 1 | 0.5 | 82.06% |
| 1000 | 1 | 0.2 | 82.45% |
| 1000 | 1 | 0.3 | 82.45% |
| 1000 | 0.9 | 0.3 | 83.96% |
| 1000 | 0.8 | 0.3 | 85.96% |
| 1000 | 0.8 | 0.4 | 81.08% |
| 1000 | 1 | 0.4 | 84.33% |

É possível visualizar que a melhor configuração é com $sigma$ e $learning\_rate$, e'iniciar com valores 0.8 e 0.2 respectivamente e decrementado a cada iteração, segunda a função atribuída a $decay_function$.

Para treinar a rede foi utilizado a função $train_random()$ junto com os parâmetros $Xtr$ e $n_iters$ com valor 1000, escolhido anteriormente. Após o treinamento da rede, é feito a chamada de função para $plot\_som()$ , que foi implementada , que tem como parâmetros o conjunto de treino $Xtr$ e suas labels $ytr$, $plot\_som()$ tem como finalidade, marcar cada neurônio vencedor com um padrão de entrada para os dados de treinamento. Assim, ao chamar a função $winner()$ e passar como parâmetro um exemplo, o neurônio na posição i,j retornado, é marcado com a label do exemplo que ela foi a vencedora, e assim sucessivamente para todos os exemplos e neurônios vencedores. Ao final é retornado uma matriz com os neurônios associados e não associados às classes, referenciado como $maps_train$. Com a matriz de neurônios marcados, é necessário classifica-los, levando

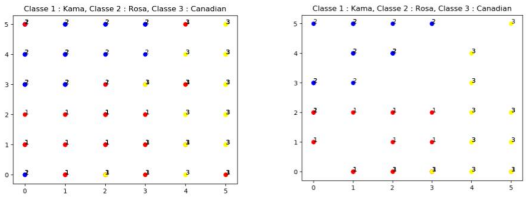


**Figura 2. Gráficos da matriz de neurônios de treinamento e teste com o terceiro conjunto.**

em conta o conjunto de teste previamente separado. Para isso foi implementado a função, $classify()$ que tem como parâmetros, o conjunto de dados de teste $Xte$, as labels $yte$ e matriz de neurônios $maps_train$. A função tem como finalidade, verificar se para cada exemplo de entrada , é associado corretamente ao neurônio que corresponde a aquele padrão. Para isso, cada exemplo contido em $Xte$ é passado para a função $winner()$ o neurônio i,j retornado, é verificado em sua matriz de neurônios, onde está marcado com a sua label, retornada pela função anterior, se, a label do exemplo corresponde ao padrão já associado a aquele neurônio, se não, é marcado um erro na matriz de erros $error[]$, e assim sucessivamente até todos os exemplos do conjunto de teste terminarem.

Ao final, é calculado a porcentagem de acertos, sendo o numero de acertos dividido pelo quantidade de ocorrências, assim gerando a acurácia (acertos) do modelo para cada classe, e a media das acurácias, sendo a soma das acurácias para as três classes.

```
Classe 1 - Kama: 20  ocorrencias com  acerto: 90.00 %.
Classe 2 - Rosa: 22  ocorrencias com  acerto: 90.91 %.
Classe 3 - Canadian: 21  ocorrencias com  acerto: 80.95 %.
Média de acertos 87.29 %.
```

**Figura 3. Saída do terminal.**

## 5. Resultados

A primeira separação de dados **Figura 2 e 3**, houve a ocorrência de exemplos para o conjunto de treino , em 44 para classe 1, 49 para classe 2 e 54 para classe 3, já o seu conjunto de teste os exemplos ficaram em 26 para classe 1, 21 para classe 2 e 16 para classe 3. É possível notar um certo desequilíbrio em relação a quantidade de ocorrências para cada classe, ao gerar um modelo SOM e classifica-lo, utilizando-se desta configuração, levando com conta todos os outros parâmetros já decididos, ou seja, no melhor ambiente de teste, essa divisão conseguiu apenas uma média de acertos 81.73%, com 57.69% para classe 1 Kama, 100.00% para classe 2 Rosa e 87.50% para classe 3 Canadian, pode ser conferido na **Figura 4.**

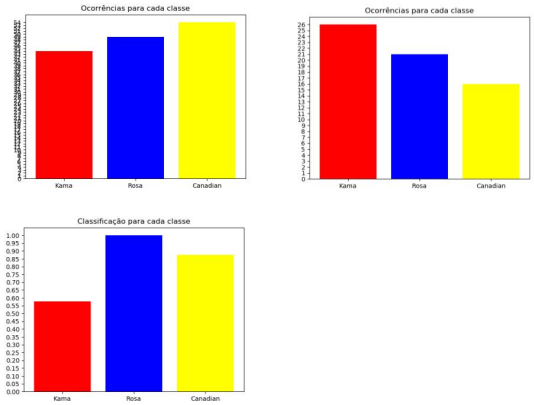


**Figura 4. Gráficos do conjunto 1.**

Na **Figura 3** há uma melhor distribuição de certo modo entre as classes, sendo a classe 3 a classe que varia para mais, com um maior numero de exemplos. É possível visualizar que também alcançou um melhor desempenho em relação aos seus exemplos classificados, com 100% deles sendo associados aos neurônios corretamente. Com uma distribuição ainda melhor, variando apenas em um exemplo para cada classe, na **Figura 4** houve também uma melhor distribuição em classificação para cada classe, com Kama 90.00%, Rosa 90.91% e Canadian 80.95%.

É possível verificar que para diferentes formatos, o modelo consegue classificar os dados em suas respectivas classes em sua maioria. Mas, como proposito de trabalho a resolver o problema de encontrar o melhor modelo para classificação das três classes de sementes, sendo assim, o terceiro conjunto de dados com uma média de 87.29% e classificando em ambas as classes muito bem, levando em consideração esse conjunto de dados, assim teve a melhor acurácia.

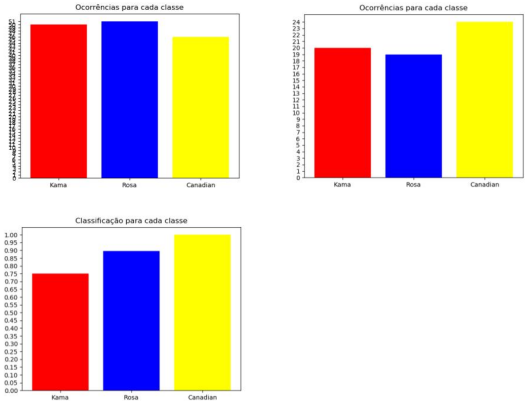


**Figura 5. Gráficos do conjunto 2.**

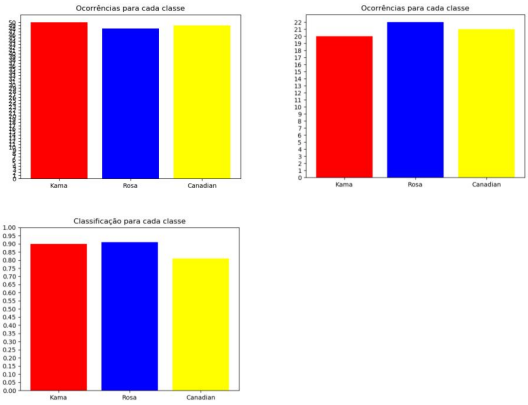


**Figura 6. Gráficos do conjunto 3.**

## Referências

G. Akçapnar, A. A. and Cosgun, E. (2014). Investigating students' interaction profile in an online learning environment with clustering. IEEE 14th International Conference on Advanced Learning Technologies.