Universidade Federal de Santa Catarina
Programa de Pós-Graduação em Ciência da Computação
Inteligência Artificial Conexionista
Augusto André Souza Berndt 202000735

Exercício 9 - Self Organizing Map (SOM)

Enunciado: Leia o artigo SOM Visualization acima. Use os dados do Índice de Desenvolvimento Humano que estão no arquivo .CSV em anexo. Experimente vários tratamentos nos dados missing (p.ex.: exclusão, substituição pela média, pela mediana, imputação https://scikit-learn.org/stable/modules/generated/sklearn.impute.IterativeImputer.html), etc. Crie, configure e treine uma rede neural auto-organizável (self-organizing maps) (do MATLAB, ou do SOM_TOOLBOX ou do python - https://github.com/JustGlowing/minisom) e veja se é possível identificar os clusters e como cada país aparece no mapa de características da camada competitiva. Faça um relatório mostrando as diferentes configurações de rede e treinamento e os experimentos e plotagens que você gerou nesta tentativa de fazer uma análise de aglomerados (clusters) nos dados de entrada do dataset.

Solução: Se tentou adaptar os códigos dos seguintes links para realizar a implementação da visualização com SOM:

https://github.com/JustGlowing/minisom (modelo SOM).

https://stackoverflow.com/questions/25239958/impute-categorical-missing-values-in-scikit-learn (imputação dos dados).

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html (clusterização dos pesos dos neurônios da rede SOM).

Este exercício foi o que tive mais dificuldade para realizar entre todos os outros da disciplina. Inclusive tive dificuldade para interpretar os resultados obtidos. Eu vi o artigo indicado e acompanhei a aula pelo youtube, mas mesmo assim não tenho certeza se o que consegui implementar condiz com o esperado.

Com relação ao **tratamento dos dados**, removi as colunas 0, 1, 2 e 3. As 0, 1 e 3 são uma sequência de inteiros, praticamente igual entre todas linhas e “viciam” os resultados da rede SOM, ela acaba retornando um resultado de uma linha retilínea para os neurônios. A coluna diz respeito somente ao nome dos países nos dados e deu problema na hora de treinar a rede SOM.

```
51    cols = [0,1,2,3]
52    df.drop(df.columns[cols],axis=1,inplace=True)
```

Figura 1 - tratamento inicial dos dados.

A **imputação** que demonstrou melhor resultado foi a partir do link já mencionado acima. Esta implementação permite a imputação com dados categóricos (strings), diferente da referência indicada no exercício, no qual deu um erro por causa da coluna com os nomes dos países (a remoção desta coluna teve que ser realizada de qualquer maneira, mas se manteve esta função de escalonamento, uma vez que ela é mais robusta). A opção indicada no exercício parece ainda estar em desenvolvimento pelo SKlearn.

```
from sklearn.base import TransformerMixin
class DataFrameImputer(TransformerMixin):
    def __init__(self):
        """Impute missing values.

        Columns of dtype object are imputed with the most frequent value
        in column.

        Columns of other types are imputed with mean of column.

        """
    def fit(self, X, y=None):

        self.fill = pd.Series([X[c].value_counts().index[0]
            if X[c].dtype == np.dtype('O') else X[c].mean() for c in X],
            index=X.columns)

        return self

    def transform(self, X, y=None):
        return X.fillna(self.fill)
```

Figura 2 - função para imputação.

O **escalonamento** foi realizado de duas maneiras diferentes. O que mostrou melhor abordagem foi a por padrão da biblioteca SKlearn, com *scale.* A outra tentativa foi com uma equação encontrada na internet que realiza diferença de cada elemento entre a média da coluna sobre o desvio padrão da coluna.

```
X=scale(df)
# X = (df - np.mean(df, axis=0)) / np.std(df, axis=0)
# X = df.values
```

Figura 3 - escalonamento

Em seguida se realizou a **criação do modelo SOM** a partir do link já mencionado. Se testou diversos números de neurônios e números de iterações para treinamento. Um número grande de neurônios como de iterações de treinamento demonstrou um melhor valor de erro, como apresentado pela biblioteca.

```
#criando o modelo SOM a partir de https://github.com/JustGlowing/minisom
# size=3
som_shape = (15, 15)
from minisom import MiniSom
# som = MiniSom(size, size, len(df.columns), neighborhood_function='gaussian', sigma=1.5,random_seed=1)
# som.pca_weights_init(df.to_numpy())
# som.train_random(df.to_numpy(), 100000,verbose=True) # trains the SOM with 100 iterations
som = MiniSom(som_shape[0], som_shape[1], len(X[0]), neighborhood_function='gaussian', sigma=1.5,random_seed=1)
som.pca_weights_init(X)
som.train_random(X, 1000000,verbose=True) # trains the SOM with 100 iterations
```

Figura 4 - modelo SOM.

Apresenta-se em seguida **resultados gráficos** para os pesos gerados a partir da rede SOM. Se tentou quatro diferentes abordagens para plotar os pesos da rede, sendo que duas dessas tentativas resultaram em erros e não consegui fazê-las funcionar.

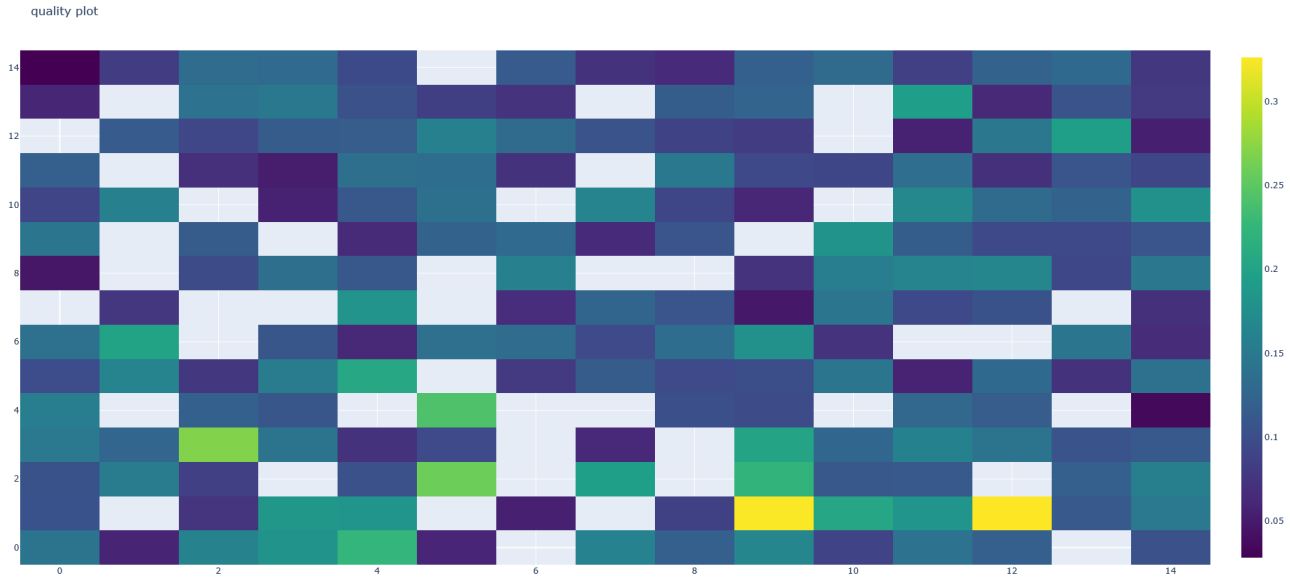

Figura 5 - gráfico com pesos da rede SOM após treinamento.

A figura a seguir demonstra um cluster para cada neurônio. Acredito que não seja a maneira mais adequada de mostrar os pesos, porém o código original que definia o modelo SOM já possuia esta saida gráfica, tive dificuldade em adaptar o código para realizar a saída correta, também tentei pedir ajuda no grupo da disciplina. Por isso, a figura anterior se faz presente e apresenta os pesos de maneira mais correta. Pode-se ver na figura anterior que existem duas regiões, uma mais a baixo com pesos maiores (mais amarelada) e outra com valores de pesos mais baixo (valores em roxo).

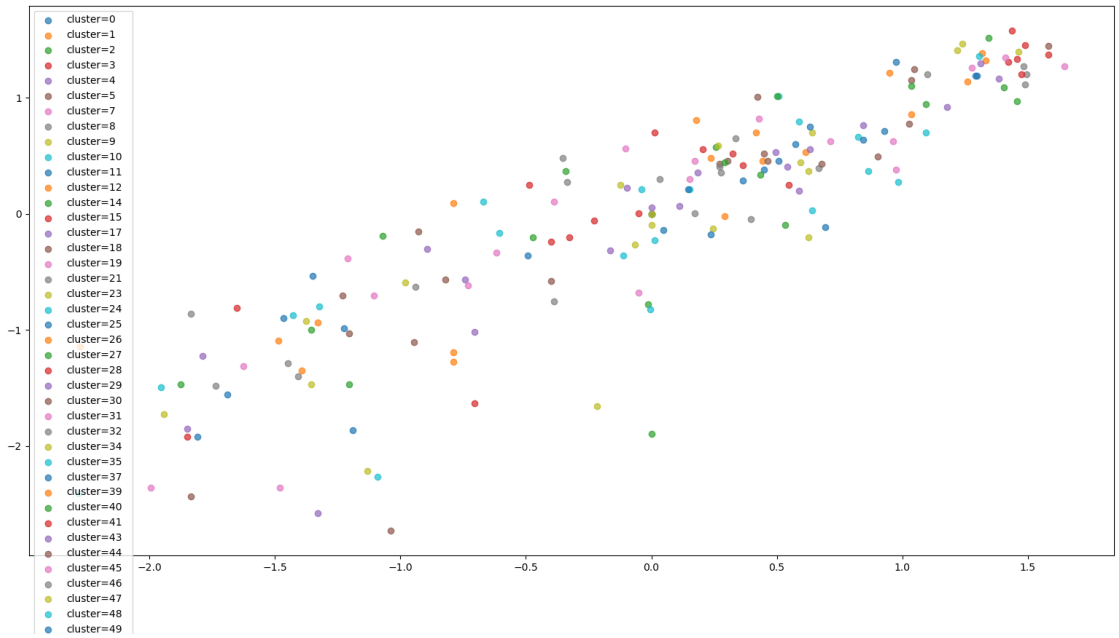

Figura 6 - gráfico com pesos da rede SOM após treinamento.

Em seguida se tentou realizar a clusterização dos pesos do modelo SOM com uma implementação KMeans. As figuras a seguir demonstram o código da tentativa e a saída gráfica obtida, tentando relacionar os pesos do modelo SOM (cluster_index) com a clusterização realizada pelo KMeans.

```
#codigo adaptado de https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
from sklearn.cluster import KMeans
kmeans= KMeans (n_clusters=3)
kmeans.fit(data,sample_weight=cluster_index)
pred=kmeans.predict(data)
print(pred)
print(data[:,0].shape)
print('data[:,0]',data[:,0])
plt.scatter(data[:,0],data[:,1], s=100,c=kmeans.labels_)
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1], s=50, marker='x',c='red')
plt.show()
```

Figura 7 - KMeans para clusterização dos pesos com SOM.

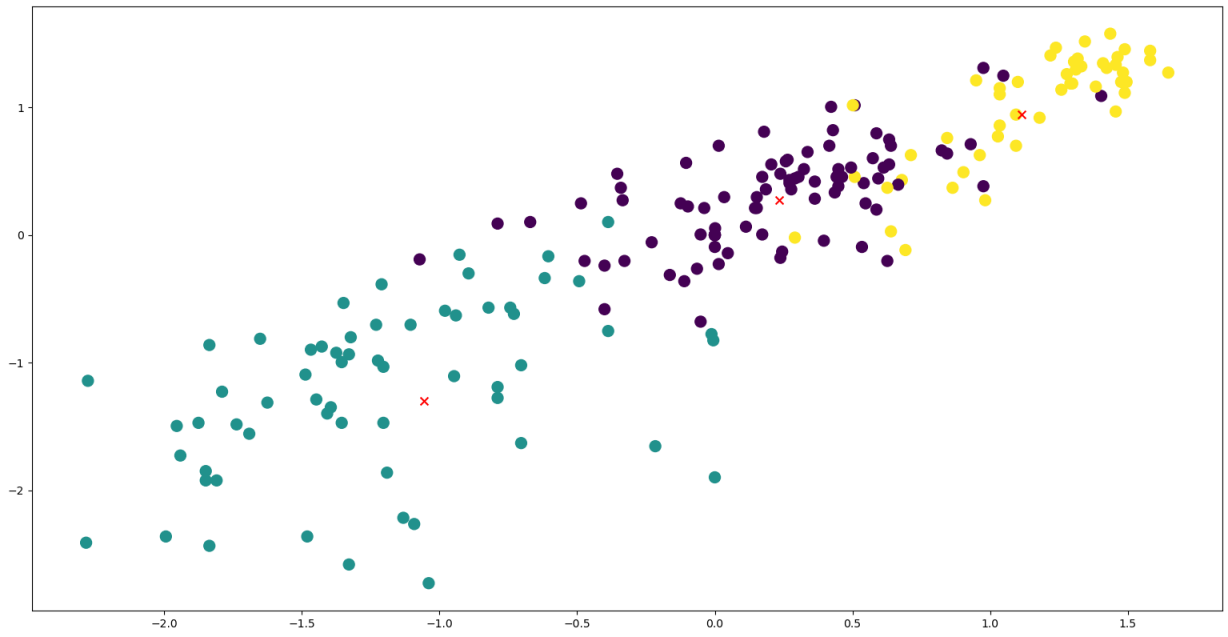

Figura 8 - Clusterização com KMeans.