Randomização de Centros e Raios para Redes Neurais RBF por meio da métrica L_∞

Felipe Bartelt, *2016026841*

***Abstract*—This paper presents a way of randomly selecting centers and radii for RBF-type neural networks. The method for randomizing the centers is based on modifications to the scalable `k-means++`, known as `k-means‖`, a method proposed for initializing clusters before `k-means` method is used. The supremum metric, L_∞, is used instead of the Euclidean metric for all clustering distance calculations and radii determination, for which two approaches are proposed, one defines equal radii for all radial basis functions and the other defines specific radii for each function. The results, obtained using four datasets, show that the method performs effectively when the radii are different, despite having a finite limit on the number of centers, unlike the method with equal radii, which has no limit but performs worse.**

***Index Terms*—RBF, neural networks, random, clustering, supremum metric.**

***Resumo*—Esse artigo apresenta uma forma de seleção aleatória de centros e raios para redes neurais do tipo RBF. O método de randomização dos centros se baseia em modificações quanto ao *scalable* `k-means++`, conhecido como `k-means‖`, um método proposto para a inicialização de *clusters* antes do uso do `k-means`. Utiliza-se a métrica do supremo, L_∞, ao invés da métrica euclidiana para todos os cálculos de distância do *clustering* e determinação dos raios, para o qual se propõe duas abordagens, uma define raios iguais para todas as funções de base radial e a outra define raios específicos para cada função. Os resultados, obtidos por meio de quatro *datasets*, mostram que o método se apresenta eficaz quando os raios são diferentes, apesar de ter limite finito de número de centros, diferentemente do método com raios iguais, que não possui limite, porém tem desempenho pior.**

***Index Terms*—RBF, redes neurais, aleatória, clustering, métrica do supremo.**

I. Introdução

REDES neurais do tipo RBF (*Radial Basis Functions Neural Networks* - RBFNNs) [1] são redes de camada intermediária única, caracterizadas por funções radiais em seus neurônios, cujas respostas são combinadas linearmente para a obtenção da saída da rede. De maneira geral uma RBFNN pode ser descrita pela Equação 1

$$f(\mathbf{x},\theta) = \sum_{i=1}^{p} w_i h_i(\mathbf{x},\mathbf{z}_i) + w_0, \tag{1}$$

onde $\mathbf{w} = [w_0, w_1, \ldots, w_p]$ é o vetor de parâmetros do neurônio de saída, $\mathbf{z}_i$ é o vetor que contém todos os parâmetros da função de ativação radial do neurônio i da camada intermediária e $\theta = [\mathbf{w}, \mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_p]$ o vetor que contém a concatenação de todos os vetores de parâmetros da rede.

Normalmente, representa-se redes RBF por meio de funções Normais. Dessa forma, pode-se representar a resposta de um neurônio i da camada intermediária com base na matriz de covariância Σ_i e vetor de médias μ_i, conforme a Equação 2.

$$h_i(\mathbf{x},\mathbf{z}_i) = \frac{1}{\sqrt{(2\pi)^n |\Sigma_i|}} e^{\left(-\frac{1}{2}(\mathbf{x}-\mu_i)^T \Sigma_i^{-1}(\mathbf{x}-\mu_i)\right)}, \tag{2}$$

onde n é o número de variáveis da entrada.

Considerando-se independência entre as variáveis, pode-se modificar a Equação 2 a forma da Equação 3.

$$h_i(\mathbf{x},\mathbf{z}_i) = \frac{1}{\sqrt{(2\pi)^n |r^2\mathbf{I}|}} e^{\left(-\frac{1}{2}(\mathbf{x}-\mu_i)^T (r^2\mathbf{I})^{-1}(\mathbf{x}-\mu_i)\right)}, \tag{3}$$

onde $\mathbf{I}$ é a matriz identidade de dimensão apropriada e r os desvios padrão das variáveis, chamado de raio da função.

Nota-se que os parâmetros de uma rede RBF são determinados, de certa forma, a priori. Necessita-se, além da definição do número de centros, de um método para alocação de centros e definição dos raios das funções radiais à ela pertencentes.

Um dos métodos mais comuns e, muitas vezes, menos custoso computacionalmente, é a alocação dos centros das RBFNNs por meio do método de *clustering* `k-means` [2], tomando-se os raios por meio da matriz de covariância das variáveis ou por meio de distâncias relativas entre os centros definidos ou distância entre centros e amostras.

Pode-se conjecturar, com base no Teorema de Cover [3] e, por consequência, no funcionamento das redes ELM [4], que é possível garantir a separabilidade do problema no espaço da camada intermediária por meio da alocação aleatória dos centros e raios para redes RBF. Dessa forma, propõe-se um método de seleção aleatória para treinamento de RBFNNs.

II. Revisão de Literatura

NA literatura, não há muitos estudos quanto à redes RBF com determinação aleatória de seus parâmetros. Sabendo-se então que a determinação aleatória de parâmetros sem um certo tipo de heurística teria um desempenho ruim, buscou-se estudar sobre métodos de inicialização para *clustering*, de forma a permitir um melhor entendimento quanto à melhor forma de se obter *clusters* iniciais e, por consequência, centros para RBFNNs, aleatoriamente.

Seja $X = \{x_1, \ldots, x_n\}$ o conjunto de pontos de dimensão d e k o número de *clusters* inteiro e positivo. Seja $\|x_i - x_j\|$ a distância euclidiana entre x_i, x_j. Para um ponto x e um subconjunto $Y \subseteq X$ de pontos, a distância é definida como $d(x, Y) = \min_{y \in Y} \|x - y\|$. Seja o centroide de um subconjunto $Y \subseteq X$ de pontos ser dado por (4)

$$\text{centroide}(Y) = \frac{1}{|Y|} \sum_{y \in Y} y \tag{4}$$

Seja $\mathcal{C} = \{c_1, \ldots, c_k\}$ um conjunto de pontos e $Y \subseteq X$, define-se o custo de Y com respeito a $\mathcal{C}$ por (5)

$$\phi_Y(\mathcal{C}) = \sum_{y \in Y} d^2(y, \mathcal{C}) = \sum_{y \in Y} \min_{i=1,\ldots,k} \|y - c_i\|^2 \quad (5)$$

Assim, o objetivo do `k-means` é obter um conjunto de centros $\mathcal{C}$ tal que minimize o custo $\phi_X(\mathcal{C})$. Tomando ϕ^* como o custo ótimo do `k-means`, chama-se o conjunto $\mathcal{C}$ de centros uma aproximação-α se $\phi(\mathcal{C}) \leq \alpha\phi^*$. Encontrar ϕ^* é NP-difícil [5] e portanto, uma vez que o `k-means` se baseia na seleção aleatória de centros e depende da separabilidade do conjunto para encontrar *clusters* bons, muitas vezes faz-se diversas iterações do método e toma-se os *clusters* que apresentam melhor separabilidade.

Devido à não garantia de boa separabilidade pelo `k-means` e necessidade de diversas iterações, Arthur e Vassilvitskii [6] propuseram o algoritmo `k-means++`, uma forma de inicialização melhor do `k-means`, tornando possível uma aproximação-$(8 \log k)$ de ϕ^*, até mesmo para o *cluster* inicial. Esse método se baseia na seleção de centros, um a um, com base em um viés.

Como a seleção de um centro i pelo `k-means++` depende do centro $i-1$, esse método não possui boa escalabilidade. Como forma de contornar esse problema, Bahmani et al. [7] propuseram o algoritmo `k-means`$\|$, com intuito de tornar o método anterior paralelizável, utilizando um fator de sobre-amostragem λ, tal que, ao contrário do `k-means++`, toma-se λ centros por iteração. Ainda, o método proposto permite que sejam feitas menos iterações para se encontrar solução análoga.

Um pseudo-código do algoritmo `k-means`$\|$ é mostrado no Algoritmo 1. Bahmani et al. demonstram que o número de pontos esperado em $\mathcal{C}$ após o término das iterações é $\lambda \log \psi$, tipicamente superior a k. Para garantir que só existam k *clusters*, o passo 7 pondera todos os centros encontrados pelo número de pontos do conjunto X que estão mais perto de um centro arbitrário do que de qualquer outro. O passo 8 por sua vez reagrupa os centros com base na ponderação feita, normalmente para esse passo é utilizado o `k-means++`, utilizando o vetor de pesos obtidos como probabilidades de seleção dos centros.

Algoritmo 1: Inicialização k-means$\|(k, \lambda)$

1 $\mathcal{C} \leftarrow$ amostrar um ponto aleatoriamente de X
2 $\psi \leftarrow \phi_X(\mathcal{C})$
3 **para** $O(\log \psi)$ vezes **faça**
4 $\quad \mathcal{C}' \leftarrow$ amostre λ pontos $x \in X$ independentemente com probabilidade $p_x = \frac{d^2(x,\mathcal{C})}{\phi_X(\mathcal{C})}$
5 $\quad \mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}'$
6 **fim para**
7 Para $x \in \mathcal{C}$, tome w_x como o número de pontos em X mais perto de x do que qualquer outro ponto em $\mathcal{C}$
8 Reagrupe os pontos ponderados em $\mathcal{C}$ em k clusters

Pelos resultados do artigo, nota-se que com somente 5 iterações o algoritmo tem resposta excelente e tem uma seleção melhor para $\lambda = 2 \cdot k$.

III. Metodologia

A fim de se obter um algoritmo razoavelmente bom para a determinação aleatória de centros e raios, decidiu-se modificar levemente o `k-means`$\|$, Algoritmo 1. Dessa forma, ter-se-ia um algoritmo de alocação de centros pseudo-aleatório, uma vez que o método propõe uma melhor forma de inicialização de centros e não a obtenção dos melhores, sendo necessário o uso do `k-means` posteriormente para tal. Os raios serão baseados em características obtidas pelo método de *clustering* utilizado.

Intuitivamente, todos os algoritmos citados se baseiam em um agrupamento de dados por meio de hiperesferas que englobam a maior quantidade de amostras possível, segundo uma função de custo baseada na métrica euclidiana. Interpretando-se a quantidade máxima de amostras como o volume das hiperesferas às quais correspondem os *clusters* e munido do fato de que o maior hipervolume para hiperesferas unitárias é alcançado na quinta dimensão, decaindo para as dimensões seguintes, pode-se por indução mostrar que a razão entre o hipervolume do hipercubo unitário circunscrito à hiperesfera e o hipervolume da hiperesfera unitária passa a aumentar, de forma não linear, conforme se aumenta as dimensões do problema, a partir da quinta dimensão, ou seja, um hipercubo comporta cada vez mais amostras em relação à hiperesfera. Assim, tomando-se o homeomorfismo entre um quadrado e um círculo ao se utilizar a métrica do supremo,L_∞, conjectura-se possível englobar o máximo de amostras possível por um *cluster* ao se utilizar a métrica do supremo para n dimensões, obtendo-se um hipercubo como superfície de agrupamento. Dessa forma, a função de custo implementada é dada por (6)

$$\Xi_Y(\mathcal{C}) = \sum_{y \in Y} d_\infty(y, \mathcal{C}) = \sum_{y \in Y} \min_{i=1,\ldots,k} (\max |y - c_i|) \quad (6)$$

Propõe-se também utilizar os resultados obtidos por Bahmani et al. e considerar o número de iterações constante e igual a 5, com $\lambda = 2 \cdot k$. Mantém-se o passo 7 do Algoritmo 1 e executa-se o passo 8 conforme o seguinte raciocínio: seja $\zeta = \lfloor k/2 \rfloor + 1$ o piso da metade do número de *clusters* desejados mais 1, toma-se os ζ centros de $\mathcal{C}$ com maior peso w calculado e armazena-os em um conjunto $\mathcal{D}$; agrupa-se os centros não selecionados de $\mathcal{C}$ em $k - \zeta$ subconjuntos, não necessariamente de tamanhos iguais, de forma ordenada e toma-se a média de cada subconjunto, armazenando esses resultados em um conjunto $\mathcal{E}$; ao final toma-se a união de $\mathcal{D}$ e $\mathcal{E}$ como o conjunto de centros finais $\mathcal{F}$. Ou seja $\mathcal{F} = \mathcal{D} \cup \mathcal{E} = \{c_1, \ldots, c_\zeta\} \cup \{\overline{\{c_{\zeta+1}, \ldots\}}_1, \ldots, \overline{\{\ldots, c_{10k}\}}_{k-\zeta}\}$. O pseudocódigo do algoritmo de seleção de centros implementado, que será chamado `k-means`$_\infty$ ao longo do texto, é apresentado em Algoritmo 2.

Por meio dos centros obtidos pelo `k-means`$_\infty$, pode-se definir o método de seleção dos raios. Propôs-se implementar dois métodos com intuitos diferentes, ambos baseados na Equação 3. O primeiro define um raio único para todas as funções radiais, tomado como a distância mínima entre dois centros quaisquer, sob a métrica L_∞, de forma a evitar o máximo de sobreposições de funções possível. O segundo busca estimar um raio para cada função de base radial, toma-se

Algoritmo 2: Inicialização k-means$_\infty(k)$

$\mathcal{C} \leftarrow$ amostrar um ponto aleatoriamente de X
para i=1 até 5 **faça**
  $\mathcal{C}' \leftarrow$ amostre $2 \cdot k$ pontos $x \in X$ independentemente com probabilidade $p_x = \frac{d_\infty(x,C)}{\sum_X(C)}$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}'$
fim para
Para $x \in \mathcal{C}$, tome w_x como o número de pontos em X mais perto, pela métrica L_∞, de x do que qualquer outro ponto em $\mathcal{C}$
Ordene os pontos $x \in \mathcal{C}$ com base no peso w_x de forma decrescente
$\mathcal{D} \leftarrow \zeta$ primeiros pontos ordenados
$\mathcal{E} \leftarrow$ médias dos $k - \zeta$ subconjuntos de $\mathcal{C} \setminus \mathcal{D}$
$\mathcal{F} \leftarrow \mathcal{D} \cup \mathcal{E}$
retorne $\mathcal{F}$

assim para cada *cluster* C_i, com centro c_i, a distância máxima, sob a métrica L_∞, entre seu centro e todas as amostras $x \in X$ pertencentes à C_i. Dessa forma, far-se-á, também, uma análise quanto à eficácia de raios diferentes para cada função de base radial que compõe uma RBFNN.

IV. Resultados

VISANDO avaliar a qualidade do algoritmo proposto, comparou-se o resultado obtido por redes RBF utilizando o método `k-means` para seleção de centros, que será chamada de **RBFkm**, com os resultados obtidos pelas RBFNN utilizando o método proposto, `k-means`$_\infty$ para seleção de centros com raios diferentes para cada função radial, que será chamada de **RBFmdr**. Ainda, comparou-se os resultados obtidos pela RBFmdr com os resultados obtidos pela RBF utilizando o `k-means`$_\infty$, porém com raios únicos para toda função de base radial, denominada **RBFmsr**. A RBFkm utiliza matrizes de covariância para a definição de raios, conforme a Equação 2, assim não se considerou independência entre variáveis para nenhum *dataset*. A métrica de avaliação escolhida foi a acurácia e, para cada banco de dados, iterou-se, para cada número de centros escolhidos, 10 vezes, de forma a se obter uma acurácia média e desvio padrão para cada hiperparâmetro testado. O número de iterações não é suficiente para se afirmar com certeza a qualidade de uma ou outra rede, porém, devido ao custo computacional, optou-se por esse número de iterações, que permite, de certa forma, analisar o comportamento das redes analisadas. Ambas as redes utilizam função de ativação $\tanh(\cdot)$ na camada de saída.

Utilizou-se 4 *datasets* de problemas de classificação: Breast Cancer, para o qual se fez pré-processamento, removendo-se as *features* com pouca influência, analisado por meio da correlação de Pearson, cujos índices são $[2, 3, 9, 11, 12, 13, 14, 18, 19, 22, 23]$, além da normalização dos dados por meio de *standardization*; Statlog(Heart), para o qual também se fez pré-processamento, removendo-se as *features* de índice $[4, 5]$ e utilizando *standardization*; Iris, sem pré-processamento; Digits, também sem pré-processamento. Um resumo dos dados de cada *dataset* é apresentado na Tabela I. Com exceção do Statlog, que foi retirado direto do repositório da UCI, todos os *datasets* foram carregados a partir do módulo `scikit-learn`.

Tabela I: Características dos *datasets*

Dataset	N° amostras	Dimensão	N° Classes
Breast Cancer	569	30	2
Statlog	270	13	2
Iris	150	4	3
Digits	1797	64	10

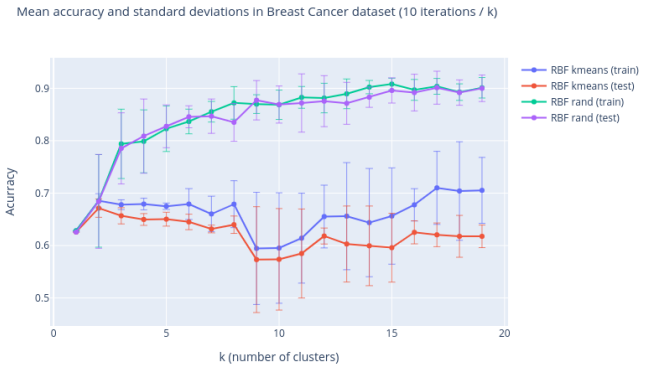

Figura 1: Acurácias médias e desvios padrão, dentro de 10 iterações, para RBFkm (RBF kmeans) e RBFmdr (RBF rand) no *dataset* Breast Cancer

Todos os problemas de classificação foram tomados como binários, representados por valores -1 e 1, devido à função de ativação escolhida. Dessa forma, além de se modificar as classes nulas, necessitou-se transformar dois bancos de dados em problemas binários. O *dataset* Iris apresenta três classes distintas, tomou-se então a classificação do problema como 'setosa' e 'não-setosa', ou seja, pretende-se apenas reconhecer se a planta é ou não da espécie setosa, que corresponde à primeira classe do conjunto. O *dataset* Digits apresenta amostras de dígitos escritos à mão, tendo assim 10 classes distintas, que foram separadas em duas classes: 'ímpar' e 'par', de forma a tentar treinar uma RBFNN para distinguir dígitos manuscritos pares e ímpares.

Para todos os testes, dividiu-se, igualmente, as amostras em conjunto de treino e teste, que correspondem a 70% e 30%, respectivamente, das amostras totais.

Como primeiro teste, avaliou-se o desempenho da RBFkm ("RBF kmeans" nos gráficos) e RBFmdr ("RBF rand" nos gráficos) para o *dataset* Breast Cancer, variando-se o número de centros entre 1 e 19. Percebe-se na Figura 1 o melhor desempenho da RBFmdr, atingindo uma acurácia média de 90% a partir de 14 centros. O comportamento da RBFkm é longe do ideal, tendo acurácia máxima de 70% nos dados de treino e 67% nos dados de teste, especula-se que isso se deve ao uso de matrizes de covariância como método de definição dos raios e à normalização utilizada, uma vez que o desvio padrão da acurácia de treino aumenta conforme se aumenta o número de centros. Nota-se ainda que enquanto a acurácia da RBFmdr aparenta crescer de forma quase linear, a acurácia da RBFkm tende a se estabilizar em torno de 63%.

Em seguida, avaliou-se, utilizando-se o mesmo intervalo de

Figura 2: Acurácias médias e desvios padrão, dentro de 10 iterações, para RBFkm (RBF kmeans) e RBFmdr (RBF rand) no *dataset* Statlog

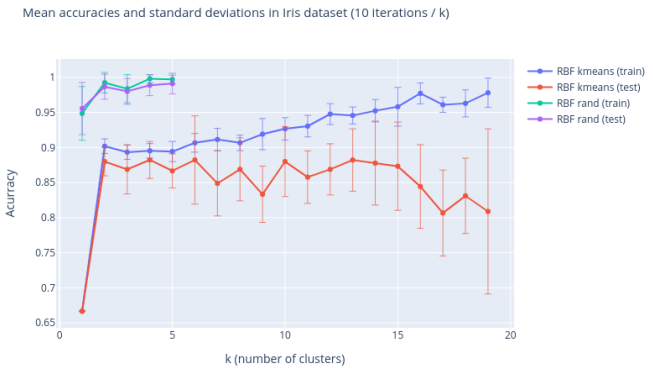

Figura 3: Acurácias médias e desvios padrão, dentro de 10 iterações, para RBFkm (RBF kmeans) e RBFmdr (RBF rand) no *dataset* Iris

centros, o comportamento de ambas as redes para o banco de dados Statlog, Figura 2. Nota-se novamente um desempenho pior para a RBFkm, que tende a ter acurácias cada vez piores nos dados de teste, atingindo o máximo 72% para 2 centros. A RBFmdr se inicia, com apenas um centro, de forma aceitável, apresentando assim 70% de acurácia. O máximo de acurácia obtido pela RBFmdr foi de 81% para 7 centros, tornando-se pior à medida que se aumenta o número de centros. Reitera-se a suposição do comportamento da RBFkm se deve à normalização utilizada unida ao uso de matrizes de covariância para seleção dos raios.

O banco de dados analisado em seguida, Iris, é mais simples, assim nota-se maiores acurácias para ambas as redes, Figura 3. Por meio desse banco de dados, descobriu-se um fenômeno não previsto, a RBFmdr tem um limite de centros possível, no caso de Iris, ele não consegue alocar mais de 5 centros, não havendo, assim, dados para os demais números de centros. Nota-se que o método utilizado pela RBFmdr permite uma excelente acurácia com apenas 1 centro, em torno de 95%, obtendo máxima acurácia com 5 centros, apresentando quase 100% de acurácia. A RBFkm tem acurácia de 67% para um único centro, que aumenta significativamente ao se adicionar mais um centro. Apesar de sua acurácia de treino continuar aumentando, a acurácia nos dados de teste alcança seu máximo 88% para 6 centros, decaindo para números maiores de centros.

Por último, comparou-se o desempenho das redes no *dataset* Digits, Figura 4. Para esses dados, ambas as redes obtiveram péssimo desempenho, pouco melhor do que simplesmente sempre prever toda e qualquer amostra como uma classe específica. Entretanto, nota-se que a RBFkm obteve resposta melhor, tanto nos dados de treino quanto dados de teste, apresentando acurácia máxima para 11 centros, que decai a partir desse número. A RBFmdr não aparenta melhorar para nenhum número de centros, o que pode significar que ela classifica toda amostra como uma classe específica. Percebe-se assim, que avaliar imagens vetorizadas, ao menos de dígitos manuscritos, não é uma função recomendável para redes do tipo RBF, muito menos para o método proposto.

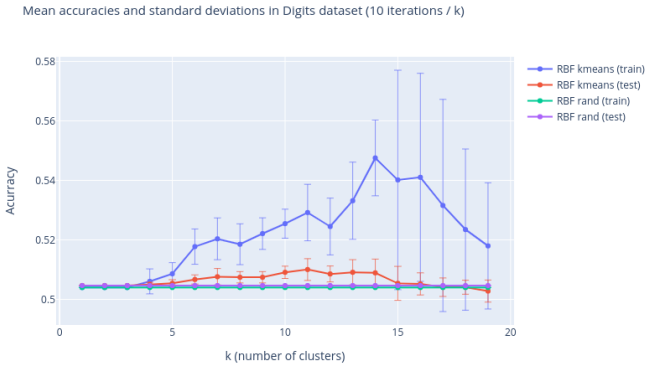

Figura 4: Acurácias médias e desvios padrão, dentro de 10 iterações, para RBFkm (RBF kmeans) e RBFmdr (RBF rand) no *dataset* Digits

A fim de se investigar a influência de definição de raios únicos ou diferentes para cada função de base radial, avaliou-se, para os mesmos conjuntos de dados e intervalos de centros, o comportamento da RBFmsr (”Same radius” nos gráficos), que será comparado com o comportamento da RBFmdr (”Diff radius” nos gráficos) apenas.

Para o conjunto Breast Cancer, Figura 5, nota-se um melhor comportamento para a RBFmdr, para todos os números de centros analisados, obtendo uma diferença de acurácias máximas em torno de 10%. Assim, para essa base de dados, avalia-se como mais eficaz a alocação de um raio para cada função de base radial, ao invés de um raio único para todas.

Para o conjunto Statlog, Figura 6, nota-se comportamento análogo. A RBFmdr apresenta comportamento melhor que a RBFmsr, porém, ao final, enquanto a acurácia da RBFmdr tende a diminuir, a RBFmsr aparenta ter capacidade de aumentar sua acurácia, uma vez que a acurácia de treino está em crescimento e a acurácia de testes não aparenta estar em um ponto de inflexão. Entretanto, não é possível dizer se há capacidade da RBFmsr ultrapassar a acurácia obtida pela RBFmdr.

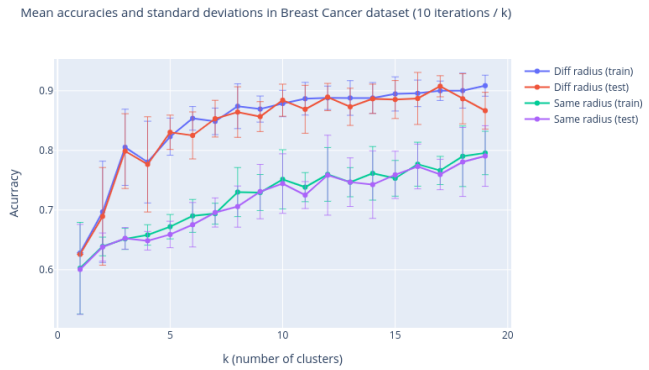

Figura 5: Acurácias médias e desvios padrão, dentro de 10 iterações, para RBFmdr (Diff radius) e RBFmsr (Same radius) no *dataset* Breast Cancer

Figura 6: Acurácias médias e desvios padrão, dentro de 10 iterações, para RBFmdr (Diff radius) e RBFmsr (Same radius) no *dataset* Statlog

Figura 7: Acurácias médias e desvios padrão, dentro de 10 iterações, para RBFmdr (Diff radius) e RBFmsr (Same radius) no *dataset* Iris

Figura 8: Acurácias médias e desvios padrão, dentro de 10 iterações, para RBFmdr (Diff radius) e RBFmsr (Same radius) no *dataset* Digits

Para o conjunto Iris, Figura 7, percebe-se que a RBF com raios únicos para toda função radial, RBFmsr, tem capacidade de alocação de maior número de centros, apesar do gráfico ser truncado em 9 centros para melhor visualização. A RBFmsr apresenta a vantagem de atingir exatamente 100%, enquanto a outra tem probabilidade não nula de conseguir acurácia um pouco menor, quando treinada com 5 centros.

Para o conjunto Digits, Figura 8, ambas as redes tiveram o mesmo comportamento, com exceção de quando o número de centros é 1, para o qual a RBFmsr tem acurácia pior.

Em suma, é possível afirmar que a seleção de raios diferentes para cada função radial apresenta grandes vantagens. Definindo-se o raio com base no número de amostras englobadas, é possível otimizar o funcionamento das redes RBF, uma vez que necessitam de menos número de centros para obter acurácias satisfatórias. Um ponto negativo, ao menos para a implementação tomada, é o fato de haver um número finito de centros possíveis para a função, uma vez que seu raio é baseado no número de amostras englobadas e torna-se impossível definir um *cluster* que não representa nenhuma amostra de treino. A única vantagem vista para a alocação de centros iguais para toda função é radial, é contornar o problema que surge da implementação da outra, como os raios iguais são definidos pela distância entre os centros, torna-se praticamente infinito o número de *clusters* possíveis de serem definidos.

V. Conclusão

O método proposto se mostrou deveras promissor. Para a maioria dos conjuntos de dados analisados, obteve-se melhores resultados ao utilizá-lo quando comparado à RBF com utilização do `k-means` e matrizes de covariância. Entretanto, devido ao desempenho inesperadamente pior da RBFNN padrão, que se especula ser devido à normalização utilizada junto do uso de matrizes de covariância, não se pode afirmar que o método proposto permite sempre melhores resultados do que uma RBFNN comum, sendo necessários outros testes sem utilizar normalização e escolhendo outro método para determinação dos raios.

A utilização da métrica L_∞ se mostrou eficaz para a seleção de centros, obtendo resultados excelentes de acurácia até mesmo para casos com somente um único centro, o que

pode indicar que realmente permite a maximização do volume de amostras englobado pelos *clusters*. A decisão de não se utilizar o `k-means++`, no passo final do `k-means‖`, e tomar a média de subconjuntos dos centros sobre-amostrados não se mostrou prejudicial, restando analisar, posteriormente, qual das abordagens resulta em melhores resultados.

Para o método proposto, alocar raios diferentes para cada função de base radial faz com que as acurácias obtidas sejam melhores com menor custo computacional, uma vez que necessita-se mais centros para a RBF com raios iguais ter acurácia próxima. A vantagem para o uso de raios iguais para a RBFNN proposta repousa sobre o fato de permitir alocação de, supõe-se, praticamente infinitos centros, o que pode ser crucial para a classificação de alguns conjuntos de dados. Ainda assim, a limitação do número de centros não aparentou ser prejudicial, salvo para o banco de dados de dígitos manuscritos, que não se mostra ideal para análise por meio de qualquer RBFNN. Pensa-se que para base de dados com muitas amostras, o funcionamento do método proposto com raios diferentes, RBFmdr, é melhor, uma vez que se torna possível definir mais *clusters*.

Mostrou-se também que redes RBF, seja utilizando o método proposto ou o mais comum, não são bons classificadores para imagens vetorizadas, porém, há a possibilidade de terem bom desempenho caso seja possível manter as informações locais das imagens, como bordas e quinas. Supõe-se que justamente essas informações tornem possível o uso de funções radiais no final de redes convolucionais para visão computacional.

REFERÊNCIAS

[1] D. Broomhead and D. Lowe, "Multivariable functional interpolation and adaptive networks," *Complex Systems*, vol. 2, pp. 321–355, 1988.

[2] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, pp. 129–137, 03 1982. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1056489

[3] T. M. Cover, "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," *IEEE Transactions on Electronic Computers*, vol. EC-14, pp. 326–334, 06 1965.

[4] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: a new learning scheme of feedforward neural networks," *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, vol. 2, pp. 985–990, 07 2004.

[5] D. Aloise, A. Deshpande, P. Hansen, and P. Popat, "Np-hardness of euclidean sum-of-squares clustering," *Machine Learning*, vol. 75, pp. 245–248, 01 2009.

[6] D. Arthur and S. Vassilvitskii, "k-means++: the advantages of careful seeding," in *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[7] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, "Scalable k-means++," *Proceedings of the VLDB Endowment*, vol. 5, pp. 622–633, 03 2012.

APÊNDICE

Algoritmo 3: Classe da RBF padrão (com k-means)

```
import numpy as np
from sklearn.cluster import KMeans

class km_rbf:
    def __init__(self, x_train, y_train, hidden_dim, indep = False):
        self.n_clusters = hidden_dim
        km = KMeans(hidden_dim).fit(x_train)
        self.centers = km.cluster_centers_
        self.labels = km.labels_
        self.w, self.cov_list = self._train(x_train, y_train, hidden_dim, indep)

    def _h_response(self, x, centers, cov_mat):
        m = x.shape[0]
        dist = x - centers

        if cov_mat.ndim < 2:
            cov_mat = np.eye(m) * cov_mat

        norm_factor = 1/np.sqrt((2*np.pi)**m * np.linalg.det(cov_mat))
        h = norm_factor * np.exp(-0.5 * dist @ np.linalg.inv(cov_mat) @ dist.T)

        return h

    def _train(self, x_train, y_train, hidden_dim, indep=False):
        N = x_train.shape[0]
        m = x_train.shape[1]
        cov_list = []
        H = np.zeros((N, hidden_dim))

        for p in range(hidden_dim):
            p_samples = (self.labels == p).nonzero()

            if indep:
                cov_mat = np.diag(np.var(x_train[p_samples], axis=0)) + 0.001*np.diag(np.ones(m))
            else:
                cov_mat = np.cov(x_train[p_samples], rowvar=False) + 0.001*np.diag(np.ones(m))

            cov_list.append(np.copy(cov_mat))
            H[:,p] = np.array([self._h_response(x_sample, self.centers[p], cov_mat)
                               for x_sample in x_train])

        H_aug = np.append(np.ones((N, 1)), H, axis=1)
        w = np.linalg.pinv(H_aug) @ y_train

        return (w, cov_list)

    def eval(self, x_test):
        N = x_test.shape[0]
        h = np.zeros((N, self.n_clusters))

        for p in range(self.n_clusters):
            cov_mat = np.copy(self.cov_list[p])
            h[:,p] = np.array([self._h_response(x_sample, self.centers[p], cov_mat)
                               for x_sample in x_test])

        h = np.append(np.ones((N, 1)), h, axis=1)
        y_hat = h @ self.w

        return y_hat
```

Algoritmo 4: Classe da RBF proposta (com k-means$_\infty$)

```
import numpy as np

class rnd_rbf:
    def __init__(self, x_train, y_train, hidden_dim, metric, same_radii = True, mean = True, r=5, l=-1):
        self.n_clusters = hidden_dim
        self.metric = metric
        self.centers, self.labels = self._rnd_clustering(x_train, mean = mean, r = r, l = l)
        self.radii = self._get_radii(x_train, same_radii)
        self.w = self._train(x_train, y_train, hidden_dim)
```

```
    def distance(self, X, C, metric = 'euclidean'):
        if metric == 'euclidean':
            dist = np.sum((X - C[:, None])**2, axis=2)
        elif metric == 'supremum':
            dist = (np.max(np.abs(X - C[:, None]), axis=2))
        elif metric == 'manhattan':
            dist = (np.sum(np.abs(X - C[:, None]), axis=2))
        elif metric == 'cosine':
            dist = (C @ X.T)**2/(np.sum(X**2, axis=1) * np.sum(C**2, axis=1)[:, None])
        return dist

    def _prob_kmp(self, dist, cost):
        prob = np.min(dist, axis=0) / cost
        return prob

    def _cost_kmp(self, dist):
        return np.sum(np.min(dist,axis=0))

    def _sample_kmp(self, X, prob, l):
        X = X.copy()
        rng = np.random.default_rng()
        samples = rng.choice(X,l, p=prob)

        return samples

    def _h_response(self, x, centers, cov_mat):
        m = x.shape[0]
        dist = x - centers

        if cov_mat.ndim < 2:
            cov_mat = np.eye(m) * cov_mat

        cov_mat = cov_mat + 0.001*np.diag(np.ones(m))
        norm_factor = 1/np.sqrt((2*np.pi)**m * np.linalg.det(cov_mat))
        h = norm_factor * np.exp(-0.5 * dist @ np.linalg.inv(cov_mat) @ dist.T)

        return h

    def _rnd_clustering(self, samples, mean = True, r=5, l=-1):
        X = samples.copy()
        N = samples.shape[0]
        rng = np.random.default_rng()
        C_set = X[rng.integers(N, size=1), :]

        if l == -1:
            l = 2*self.n_clusters

        for _ in range(r):
            dist = self.distance(X, C_set, self.metric)
            cost = self._cost_kmp(dist)
            prob = self._prob_kmp(dist, cost)
            C_temp = self._sample_kmp(X, prob, l)
            C_set = np.r_[C_set, C_temp]

        dist = self.distance(X, C_set, self.metric)
        closest = np.zeros(dist.shape)
        closest[np.argmin(dist, axis=0), range(dist.shape[1])] = 1
        count = np.sum(closest, axis=1)
        best_c = np.argsort(count)[::-1]
        if mean and self.n_clusters > 2:
            a = C_set[best_c].copy()
            fixed = int(np.floor(self.n_clusters/2))+1
            temp = np.array_split(a[fixed::], self.n_clusters-fixed)
            m = np.array([np.mean(chunk, axis=0) for chunk in temp])
            C_set = np.r_[a[:fixed], m]
        else:
            C_set = C_set[best_c[:self.n_clusters],:]
        labels = np.argmin(self.distance(X, C_set, self.metric), axis=0)

        return (C_set, labels)

    def _get_radii(self, X, same = True):
        C = self.centers.copy()
        X = X.copy()

        if same:
            radius = np.min(np.mean((self.distance(C, C, self.metric)), axis=0))
```

```
            radii = radius * np.ones((self.n_clusters, 1))
        else:
            radii = []
            for label in range(self.n_clusters):
                radius = np.max(self.distance(X[np.nonzero(self.labels == label)], C[label, None],
                                self.metric))
                radii.append(radius)
        radii = np.array(radii)

        if self.metric == 'euclidean':
            radii = np.sqrt(radii)

        return radii

    def _train(self, x_train, y_train, hidden_dim):
        N = x_train.shape[0]
        H = np.zeros((N, hidden_dim))

        for p in range(hidden_dim):
            H[:,p] = np.array([self._h_response(x_sample, self.centers[p], self.radii[p])
                               for x_sample in x_train])

        H_aug = np.append(np.ones((N, 1)), H, axis=1)
        w = np.linalg.pinv(H_aug) @ y_train

        return w

    def eval(self, x_test):
        N = x_test.shape[0]
        H = np.zeros((N, self.n_clusters))

        for p in range(self.n_clusters):
            cov_mat = np.copy(self.radii[p])
            H[:,p] = np.array([self._h_response(x_sample, self.centers[p], cov_mat)
                               for x_sample in x_test])

        H = np.append(np.ones((N, 1)), H, axis=1)
        y_hat = H @ self.w

        return y_hat
```

Algoritmo 5: Módulo *utils*

```
import numpy as np
import plotly.graph_objects as go

def normalize_features(X, mean, std):
    Xtemp = np.copy(X)
    Xtemp = Xtemp - mean
    Xtemp = Xtemp / std
    return Xtemp

def delete_features(X, feat_idx):
    # Returns matrix X with features indexes in feat_idx ignored
    Xtemp = np.copy(X)
    Xtemp = np.delete(Xtemp, feat_idx,1)
    return Xtemp

def eval_accuracy(y_hat, y, nn_type = 0):
    # Divides quadratic error by 4 if activation function is tanh
    N = np.shape(y)[0] * (4 ** nn_type)
    return (1 - ((y-y_hat).T @ (y-y_hat)) / N).ravel()

def split_set(X, y_sample, train_prop = 0.7):
    N = X.shape[0]

    # Get indexes corresponding to each class
    idx1 = [idx for idx, val in enumerate(y_sample.flatten()) if val==1]
    idx0 = sorted(list(set(range(0,N)) - set(idx1)))
    N0,N1 = len(idx0), len(idx1)
    N_train0, N_train1 = round(train_prop*N0), round(train_prop*N1)
    # Randomize indexes
    np.random.default_rng().shuffle(idx0)
    np.random.default_rng().shuffle(idx1)

    # Select samples for training and testing
```

```
    x_train = X[np.append(idx0[0:N_train0], idx1[0:N_train1]),:]
    x_test = X[np.append(idx0[N_train0::], idx1[N_train1::]),:]
    y_train = y_sample[np.append(idx0[0:N_train0], idx1[0:N_train1]),:]
    y_test = y_sample[np.append(idx0[N_train0::], idx1[N_train1::]),:]

    return (x_train, x_test, y_train, y_test)

def plot_accuracy_std(cluster_rng, accs, stds, data_names, title):
    """
        Plots scatter plot of accuracies and standard deviations for multiple data

        Parameters:
        -----------
            cluster_rng: Tuple
                specifies cluster numbers range as (c_min, c_max)
            accs: Tuple
                tuple of accuracies tuple ((acc1_train, acc1_test), (acc2_train, acc2_test)...)
            stds: Tuple
                tuple of standard deviation tuple ((std1_train, std1_test), (std2_train, std2_test)...)
            data_names: Tuple
                tuple with plot names ('method1', 'method2')
            title: String
                text that will be showed after default - 'Mean accuracy and standard deviation <title>'

        Returns:
        --------
            fig: graph_object
                graph_object figure made with plotly

    """
    fig = go.Figure()
    xmin, xmax = cluster_rng

    for acc, std, name in zip(accs, stds, data_names):
        fig.add_trace(go.Scatter(x = np.arange(xmin, xmax), y=acc[0], error_y=dict(type='data',
                                 array=std[0],visible=True, thickness=0.7),
                                 name = '{} (train)'.format(name)))
        fig.add_trace(go.Scatter(x = np.arange(xmin, xmax), y=acc[1], error_y=dict(type='data',
                                 array=std[1],visible=True, thickness=0.7),
                                 name = '{} (test)'.format(name)))

    fig.update_layout(title = {'text':'Mean accuracies and standard deviations {}'.format(title),
                      'font_size':15}, width=800, height=500)
    fig.update_yaxes(title = 'Acurracy')
    fig.update_xaxes(title = 'k (number of clusters)')

    return fig
```

Algoritmo 6: Script usado para testes (somente é mostrado para o dataset Breast Cancer uma vez que os demais são análogos)

```
from rbf.km_rbf import km_rbf
from rbf.rnd_rbf import rnd_rbf
import numpy as np
from sklearn.datasets import load_iris, load_breast_cancer, load_digits
import plotly.graph_objects as go

breast_cancer = load_breast_cancer()
X_samples = breast_cancer['data']
y_sample = np.reshape(breast_cancer['target'], (-1,1))
ignored_idx = [2,3,9,11,12,13,14,18,19,22,23]

X = utils.delete_features(X_samples, ignored_idx)
X_mean = np.mean(X, axis = 0)
X_std = np.std(X, axis = 0)
X = utils.normalize_features(X, X_mean, X_std)
y_sample[y_sample==0] = -1

x_train, x_test, y_train, y_test = utils.split_set(X, y_sample)

# Itera RBF com k-means
accs, stds = [], []
for p in range(1,20):
    acc, acct = [], []
    for _ in range(10):
        x_train, x_test, y_train, y_test = utils.split_set(X, y_sample)
        km = km_rbf(x_train, y_train, p)
```

```
        y_hat_train = np.sign(km.eval(x_train))
        y_hat_test = np.sign(km.eval(x_test))
        acc.append(utils.eval_accuracy(y_hat_train, y_train, 1))
        acct.append(utils.eval_accuracy(y_hat_test, y_test, 1))
    accs.append((np.mean(acc), np.mean(acct)))
    stds.append((np.std(acc), np.std(acct)))
accs = (tuple(map(list, zip(*accs))),)
stds = (tuple(map(list, zip(*stds))), )

# Itera RBF aleatorio
accs_2, stds_2 = [], []
for p in range(1,20):
    acc, acct = [], []
    for _ in range(10):
        x_train, x_test, y_train, y_test = utils.split_set(X, y_sample)
        km = rbfrnd = rnd_rbf(x_train, y_train, p, 'supremum', same_radii = False, mean = True)
        y_hat_train = np.sign(km.eval(x_train))
        y_hat_test = np.sign(km.eval(x_test))
        acc.append(utils.eval_accuracy(y_hat_train, y_train, 1))
        acct.append(utils.eval_accuracy(y_hat_test, y_test, 1))
    accs_2.append((np.mean(acc), np.mean(acct)))
    stds_2.append((np.std(acc), np.std(acct)))
accs_2 = (tuple(map(list, zip(*accs_2))),)
stds_2 = (tuple(map(list, zip(*stds_2))), )

accs_test = accs +accs_2
stds_test = stds +stds_2

fig = utils.plot_accuracy_std((1,20), accs_test, stds_test, ['RBF kmeans', 'RBF rand'],
                              'in Breast Cancer dataset (10 iterations / k)')
fig.show(width = 800, heigth = 500)
```