Quantile Hashing : Uma abordagem escalável baseada em *Hashing* sensível a localidade para o problema dos *k*-vizinhos mais próximos

Paulo C. Neto, Gustavo R.L. Silva, Antônio P. Braga

Departamento de Engenharia Eletrônica, Universidade Federal de Minas Gerais - Av. Antônio Carlos 6627, 31270-901, Belo Horizonte, MG, Brasil,
paulo-cirino@ufmg.br,gustavolacerdas@ufmg.br,adpbraga@ufmg.br,
http://www.paulocirino.com

Abstract. Este artigo apresenta um novo método de classificação de dados denominado QH *Quantile Hashing*, cujo o objetivo reduzir os custos computacionais associados ao k-NN, eliminando a necessidade da computação das distâncias por meio de um método ingênuo de *Locality Sensitive Hashing*(LSH). Um grande volume de dados (10^5 amostras) foi utilizado nos testes e o método proposto, QH, apresentou uma excelente escalabilidade quando comparado ao k-NN tradicional. A implementação do método QH é direcionada a dois objetivos: processar grande volume de dados em menor tempo e manter a qualidade encontrada pela versão original do k-NN.

Keywords: *Machine Learning*, *Classification*, *Nearest Neighbor*, *Locality Sensitive Hashing*

1 Introdução

Este trabalho apresenta uma metodologia direcionada a problemas de classificação com grandes volumes de dados. Nesse contexto, este artigo tem como objetivo desenvolver um método de classificação baseado nos quantis denominado QH-*Quantile Hashing*. O QH têm como objetivo aproximar a metodologia de classificação baseada em k-vizinhos mais próximos, substituindo o cálculo da distância entre amostras por uma tabela *Hashing* e os k-vizinhos por uma região de interesse. Esse trabalho se difere das metodologias atuais que utilizam *hashing* para aproximar os vizinhos mais próximos pois não trabalha com projeções de vetores como [1–4] e tampouco com o *hashing* para condensação de dimensionalidade como [5–9].

Os resultados encontrados pela implementação do QH, apresentam uma melhoria de tempo considerável quando comparados à versão original do k-NN, sem a perda da acurácia dos resultados. O restante do artigo esta organizado da seguinte forma. Na Seção 2, são apresentados os estudos relevantes na área. Na Seção 3, são descritos os detalhes de implementação do método proposto. Na

Seção 4, são apresentados os experimentos feitos e os resultados encontrados. Por fim, na seção 5, as conclusões e trabalhos futuros.

2 Trabalhos Relacionados

A análise de grandes volumes de dados em método de classificação, normalmente, esbarra em problemas relacionados à escalabilidade dos métodos existentes [10], [11]. Nas próximas seções serão apresentados a definição do método k-NN e o funcionamento de funções de *hashing* sensíveis à localidade. Destaca-se que ambos os conceitos forneceram o embasamento teórico para o método proposto.

2.1 *K*-vizinhos mais próximos

Um classificador baseado no método do vizinho mais próximo (NN) [12] pode ser descrito da seguinte forma: suponha que $S^n = \{x_1, \cdots, x_n\}$ seja um conjunto de dados rotulados Y e que $x' \in S^n$ é a amostra mais próxima a um ponto de teste x. Então a regra do vizinho mais próximo para classificação de x corresponde à associação de x ao rótulo atribuído a x' [13]. A regra NN, por sua vez, pode ser expandida para os k vizinhos mais próximos, sendo chamada de k-NN. Nessa situação, o rótulo do ponto de teste x é a moda dos rótulos de Y associados aos seus k vizinhos mais próximos em S^n.

Uma das dificuldades na implementação do k-NN é o seu elevado custo computacional. A complexidade desse algoritmo em relação ao aspecto de tempo de execução é $O(n \times d)$, em que n é o número de amostras do conjunto de treinamento e d o número de dimensões. Outro fator que eleva o custo computacional é o consumo de memória RAM para armazenar o cálculo da matriz de distância dos dados de treinamento [13]. Algumas propostas para tentar diminuir o custo computacional do K-NN sao encontradas na literatura [14, 15]. Além das propostas citadas, o autor Brian Kulis [1] apresenta uma função de *hashing* sensível à localidade aplicada às funções de *kernel*. Essa estratégia permite que a busca por similaridade do k-NN seja realizada em tempo sublinear, o que garante uma alta escalabilidade do método proposto quando aplicado a grandes volumes de dados.

2.2 Função de *Hashing* Sensível à Localidade

O princípio de uma função de *hashing* sensível à localidade baseia-se na ideia de que, se dois pontos estiverem próximos entre si, logo após a aplicação de operações de *hashing*, esses dois pontos permanecerão próximos [7, 9]. Essa propriedade tem o intuito de garantir que a probabilidade de obter chaves iguais seja muito maior para pontos próximos uns dos outros do que para aqueles distantes.

Indyk et.al [7, 9] apresentam funções de *hashing* sensíveis à localidade para o caso em que os pontos encontram-se em um espaço binário de Hamming. Com o intuito de superar as limitações de se trabalhar com o espaço binário, os autores Datar *Et al.* [2] e Andoni *Et al.* [4] generalizaram o método, com grande sucesso para o espaço Euclidiano.

Normalmente, funções de *hashing* sensíveis à localidade são mapeadas da seguinte forma: $h_{a,b}(v) : \mathcal{R}^d \rightarrow \mathcal{N}$, uma vez que a amostra v, de dimensão d, é mapeada em um ou mais valores inteiros. Isso é feito de tal forma que, a partir da escolha de três valores de a, considerado um parâmetro independente escolhido por meio de uma distribuição estável [16], b é um valor aleatório entre $[0, r]$, e r é um limiar para definir a proximidade entre pontos e aumentar a probabilidade de obterem chaves iguais. A chave da função *hashing* $h_{a,b}(v)$ é obtida com [3] :

$$h_{a,b}(v) = \left\lfloor \frac{a \cdot v + b}{r} \right\rfloor \tag{1}$$

3 Metodologia

Nesta seção, serão descritos os detalhes de implementação do método proposto *QuantileHashing*-QH.

3.1 Representação do Espaço

A principal contribuição deste trabalho consiste na substituição do cálculo da distância entre as amostras por uma função de *hashing* baseada nos quantis, para descobrir os vizinhos mais próximos. O ganho em complexidade temporal é obtido substituindo a complexidade $O(n \times d)$ do cálculo de distâncias pela complexidade de inserção e busca $O(1)$ da tabela *hashing* mais $O(d)$ da função de mapeamento.

O método QH parte do princípio de que é possível criar uma função de *hashing* que gera chaves iguais para dois pontos espacialmente próximos uns dos outros e chaves diferentes para dois pontos distantes. A ideia é criar um *hashing* imperfeito que reduz pontos diferentes a uma mesma chave quando estes estão próximos. Assim, é possível dividir o espaço amostral em regiões e atribuir para cada região uma chave única, conforme apresentado nas Figuras 1 e 2.

Com o objetivo de atribuir uma única chave a cada região, a divisão do espaço para cada dimensão foi feita através de quantis. Nesse caso, propõe-se uma forma ingênua de divisão, ingênua por não considerar a distribuição dos dados, e sim apenas a divisão dos dados em janelas de tamanhos constantes. Dessa forma, é possível, para cada dimensão, obter uma única chave que representa um quantil

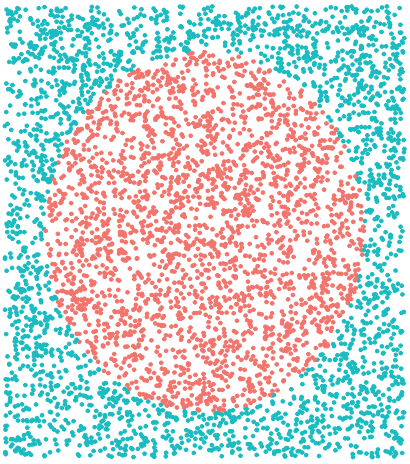

Fig. 1: Amostra de exemplo.

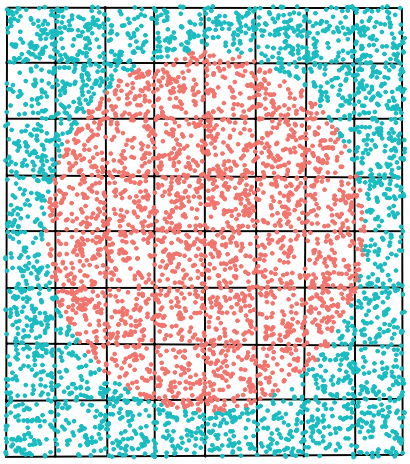

Fig. 2: Divisão do espaço.

ou intervalo de dados. Após a obtenção dos quantis para cada dimensão, a operação restante consiste em combinar os quantis de forma única, conforme Figura 3.

	1	2	3	4
1	(1,1) = 1	(2,1) = 2	(3,1) = 3	(4,1) = 4
2	(1,2) = 5	(2,2) = 6	(3,2) = 7	(4,2) = 8
3	(1,3) = 9	(2,3) = 10	(3,3) = 11	(4,3) = 12
4	(1,4) = 4	(3,4) = 13	(3,4) = 14	(4,4) = 16

Fig. 3: Divisão do espaço amostral em regiões.

Assim, munido de uma função de *hashing* que mapeia todos os pontos em uma mesma região do espaço para uma única chave, é possível utilizar uma tabela *hashing* que aproxima todas as inserções na posição dessa chave com os vizinhos mais próximos.

A metodologia proposta necessita de apenas de um parâmetro, Q_{Size}, que representa o tamanho da janela ou região em cada dimensão onde serão considerados os vizinhos mais próximos. Uma outra abordagem para definir esse parâmetro, pode ser por meio da escolha do número de partições em cada dimensão N_{Splits}.

Esse parâmetro pode ser igual ou diferente para cada dimensão, de forma que ele controla a precisão do método. Utilizando janelas muito pequenas, poucos pontos estarão presentes em cada região. Por outro lado, utilizando janelas muito grandes, as regiões terão pontos que não são necessariamente vizinhos mais próximos. As Figuras 4, 5, 6 e 7 ilustram como o parâmetro Q_{Size} influencia a superfície de decisão dos dados da Figura 1.

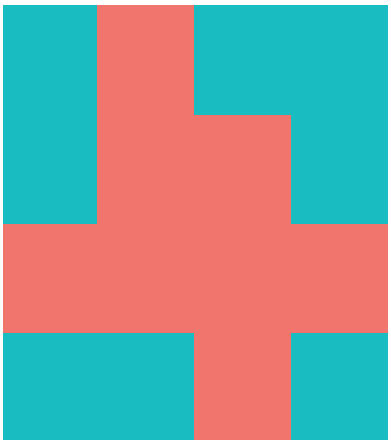

Fig. 4: Superfície de Decisão para $QSize = 0.25$

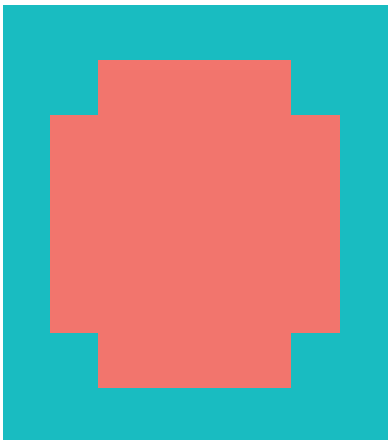

Fig. 5: Superfície de Decisão para $QSize = 0.125$

3.2 Método Proposto - QH *QuantileHashing*

O método proposto baseia-se em duas etapas, sendo que a primeira consiste na construção de uma tabela *hashing* com o auxílio das amostras de treinamento e a segunda é composta por uma etapa de classificação. As proximas seções descreverão passo a passo a abordagem proposta.

Codificação Baseada em Quantil
Dado um conjunto de dados rotulados para treinamento, $\mathbb{X} = \{X_1, \cdots, X_m, \cdots, X_M\}$, que possui M variáveis, define-se $x_i = [x_{i1}, x_{i2}, \cdots, x_{iM}], \forall i \in [1, 2, \cdots, N]$ como uma das N amostras. Além disso, para cada x_i existe um rótulo y_i, tal que $\mathbb{Y} = \{y_1, \cdots, y_n, \cdots, y_N\}$. Escolhida uma amostra genérica x_i, temos que seu quantil q_i para variável j pode ser definido como :

$$q_{ij} = \lfloor \frac{x_{ij} - \max(X_j)}{Q_{size_j}} \rfloor \tag{2}$$

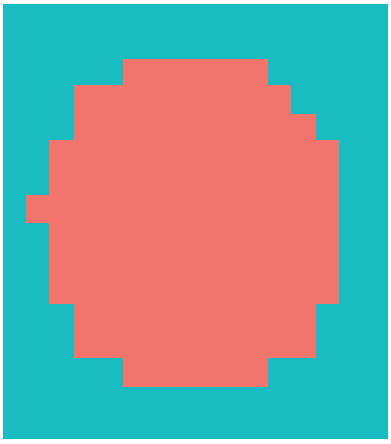

Fig. 6: Superfície de Decisão para $QSize = 0.0625$

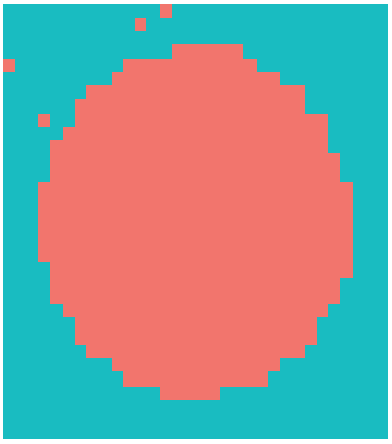

Fig. 7: Superfície de Decisão para $QSize = 0.03125$

Em que Q_{size_j} é um parâmetro que pode ser informado diretamente ou indiretamente através do número de partições, N_{splits}. A equação 3 apresenta a formulação:

$$N_{splits_j} \rightarrow Q_{size_j} = \frac{\max(X_j) - \min(X_j)}{2^{N_{splits_j}}} \tag{3}$$

Definindo V_{primo} como um vetor de número primos, é possível encontrar a chave da amostra genérica x_i como :

$$C_i = \prod_{j=1}^{m} (V_{primo}^{q_{ij}+1}) \tag{4}$$

Dado duas amostras genéricas x_p e x_q, a propriedade básica desejada em $f(x)$ é atendida tal que:

$$x_p \sim x_q \Leftrightarrow f(x_p) = f(x_q) \Leftrightarrow C_p = C_q \tag{5}$$

onde:

$$x_p \sim x_q \Rightarrow q_{pj} = q_{qj}, \forall j \in [1, 2, ..., M] \tag{6}$$

Além da propriedade básica garantida, surge uma nova característica dessa representação que é, descobrir chaves em regiões vizinhas. A definição de região de vizinhança pode ser entendida como a região presente em um quantil que se encontre ao lado de outro quantil. Tal definição é o mesmo que:

$$(q_{pj} = q_{qj}, \forall j \in [1, \cdots, m-1, m+1, \cdots, M])$$
$$\wedge$$
$$(q_{pj} = q_{qj} \pm 1) \Rightarrow C_p = C_q \cdot V_{primo}[m \pm 1]$$

Etapa de Treinamento

As equações (2), (3) e (4) apresentadas anteriormente fornecem o embasamento teórico para o Algoritmo 1 **QHTreinamento** proposto neste trabalho. O Algoritmo 1 **QHTreinamento** consiste em preencher uma tabela hashing H, sendo que na posição de cada chave existe um vetor de tamanho igual aos números de rótulos únicos W do vetor Y. Esse procedimento pode ser demonstrado da seguinte forma: dado uma chave, caso sua posição esteja vazia, é criado um vetor de zeros de tamanho W. Com o vetor já criado, a posição y referente ao rótulo é incrementada de uma unidade.

Algorithm 1: QHTreinamento.

Input : Conjunto de dados $\mathbb{X} = \{X_1, \cdots, X_m, \cdots, X_M\}$
Vetor $\mathbb{Y} = \{y_1, \cdots, y_n, \cdots, y_N\}$
Vetor $Q_{size} = \{q_1, \cdots, q_m, \cdots, q_M\}$

Output: Tabela *Hashing* **H**

$MIN_j \leftarrow min(X_j)$
for x *in* X **do**
  $q_j \leftarrow \lfloor \frac{x_j - MIN_j}{Q_{size}j} \rfloor$
  $C \leftarrow \prod_{j=1}^{m} ({V_{primo}}^{q_j+1})$
  if $\exists! H\{C\}$ **then**
    $H\{C\} = 0_W$
  end
  $H\{C\}_y \leftarrow H\{C\}_y + 1$
end

Etapa de Classificação

Após a finalização da etapa de treinamento, o método segue para a etapa de predição das amostras não rotuladas Algoritmo 2 **QHClassificador** . Esse momento de classificação consiste na computação da chave de uma amostra e também da busca na tabela *hashing* da posição referente à chave definida, com objetivo de descobrir qual a classe dominante nessa posição.

Algorithm 2: QHClassificador.

Input : Amostra $\mathbf{X} \leftarrow \{x_1, \cdots, x_m, \cdots, x_M\}$
Tabela *hashing* **H**
Vetor de mínimos por dimensão MIN
Vetor $Q_{size} = \{q_1, \cdots, q_m, \cdots, q_M\}$
Output: Predição $\hat{\mathbf{y}}$

$q_j \leftarrow \lfloor \frac{x_j - MIN_j}{Q_{size}j} \rfloor$
$C \leftarrow \prod_{j=1}^{m} (V_{primo}[j+1]^{q_j})$
if $\exists H\{C\}$ **then**
  $\hat{\mathbf{y}} \leftarrow \arg\max_y (H\{C\}_y)$
else
  $C \leftarrow$ chavesVizinhas(C)
  $\hat{\mathbf{y}} \leftarrow \arg\max(H\{C\})$
end
return $\hat{\mathbf{y}}$

Pro fim o Algoritmo 3 **chavesVizinhas** é proposto para o tratamento de exceção nas situações em que não são encontrados vizinhos. Tal fato ocorre quando não há uma colisão na tabela. Nem sempre existirá, no conjunto de treinamento, pontos na mesma região de todos os pontos do conjunto não rotulado. Assim, essa função permite uma busca em largura nas regiões vizinhas da região não preenchida.

Algorithm 3: chavesVizinhas.

Input : Chave da amostra **C**
Número de dimensões **M**
Output: Conjunto de chaves vizinhas **C**

for *c in C* **do**
  for *j from 1 to M* **do**
    $C \leftarrow C \cup c \cdot V_{primo}[j+1] \cup \frac{c}{V_{primo}[j+1]}$
  end
end
return **C**

4 Resultados

O método QH foi implementado utilizando a linguagem de programação *Python 2.7.8* e sua biblioteca *numpy 1.13.1* [17]. Para os experimentos foram implementadas 2 rotinas de testes, uma para comparar a acurácia do método do k-NN com o método aproximado QH, outra para comparação da medição empírica do

tempo de execução. Os experimentos foram realizados de forma serial em um servidor que possui dois processadores físicos Intel Xeon E5-2620 de 2 GHz, com um total de 24 *cores*, 92 GB de memória RAM.

O tempo de execução foi avaliado com a utilização de oito bases de dados sintéticas, com a seguinte variação de tamanho: 1.000, 2.000, 5.000, 10.000, 20.000, 40.000, 60.000, 80.000 e 100.000 amostras nos conjuntos de treinamento e teste. Os experimentos foram executados trinta vezes e o resultado apresentado corresponde à média das execuções.

4.1 Experimentos

Esta seção apresenta os resultados encontrados para o método QH em comparação ao k-NN. A Tabela 1 apresenta a acurácia dos algoritmos k-NN e QH com as seguintes configurações:

- k-NN com o o valor de k igual a 1, 3, 5 e 7.
- QH com o valor de Q_{size} igual a 2, 3, 4, 5 e 6.

Os valores em negrito destacam o melhor desempenho do método QH quando comparado ao k-NN. Baseado nos resultados encontrado na Tabela 1 um teste

Table 1: Avaliação da acurácia do método QH quando comparado com o k-NN.

Dataset	**k-NN-1**	**k-NN-3**	**k-NN-5**	**k-NN-7**	**QH-2**	**QH-3**	**QH-4**	**QH-5**	**QH-6**
Cassini	100.00	100.00	100.00	100.00	91.83	97.88	**99.99**	99.98	99.93
Circle	99.54	99.52	99.51	99.52	75.28	94.92	95.95	97.75	**98.89**
Circle3D	97.34	97.55	97.60	97.66	84.23	90.44	**94.50**	80.33	56.11
Normals2D	88.35	90.46	91.06	91.34	86.96	90.58	91.74	**91.77**	91.32
Simplex	100.00	100.00	100.00	100.00	**99.97**	99.86	99.22	95.14	74.25
Smiley	100.00	100.00	100.00	100.00	99.65	**99.99**	99.97	99.92	99.68
Spirals	100.00	100.00	100.00	100.00	78.76	99.74	**99.98**	99.94	99.76
XOR	99.67	99.67	99.66	99.66	**99.99**	99.99	99.99	99.99	99.99

ANOVA de 3 fatores (Base de Dados, Metodo e Parâmetro) foi aplicado. Neste teste a hipótese nula é que os fatores não influenciam na acurácia. Na Tabela 2 o p-valor do fator Método (0.0877) é maior que 0.05, então podemos dizer que falhamos em rejeitar a hipótese nula e que não existem evidências suficientes para dizer que algum dos métodos influenciaram na acurácia.

Table 2: Teste ANOVA de três fatores para os resultados da acurácia.

	Graus de Liberdade	Soma de Quadrados	Quadrado Médio	Estatística F	P valor)
Base de Dados	3	482.64	160.88	13.01	$3\ 10^{-5}$
Método	1	39.18	39.18	3.17	0.0877
Parametro	7	236.86	33.84	2.74	0.0308

A Figura 8 apresenta a comparação do tempo de execução entre os algoritmos QH com o k-NN.

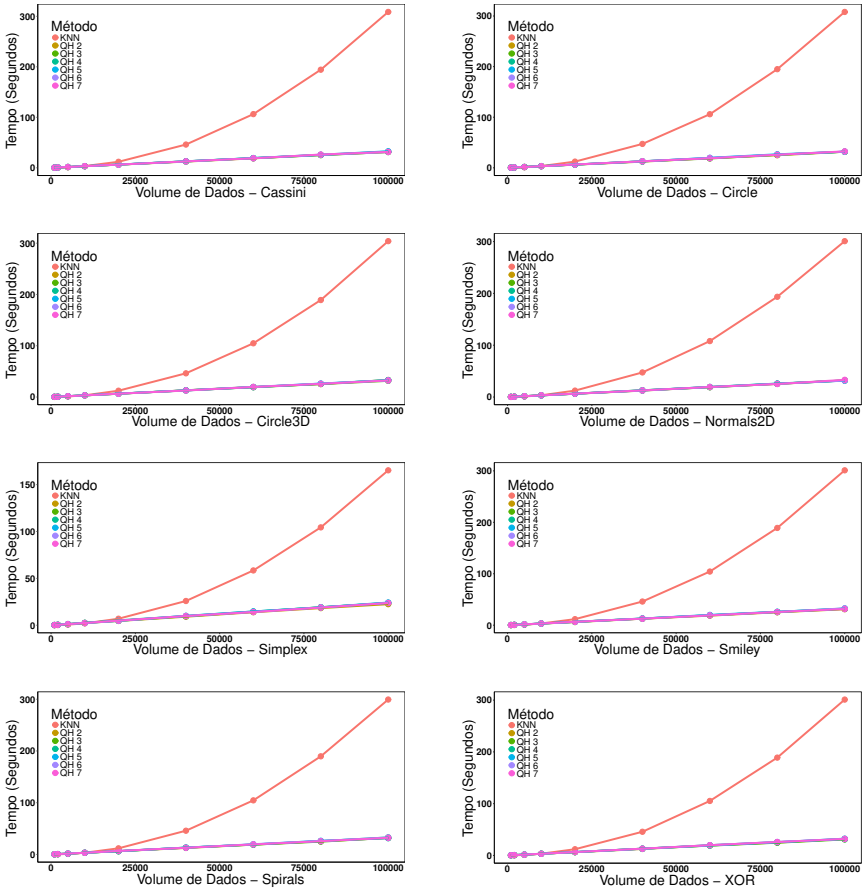

Fig. 8: Comparação do tempo de execução entre o k-NN e o método proposto com diferentes número de partições, volume de dados e *Data Sets*.

Os resultados encontrados apresentam uma vantagem muito expressiva para o método proposto por este trabalho. No geral, o QH se mostrou uma ordem de complexidade mais rápido que o método original k-NN.

5 Conclusões

Este artigo apresentou um novo método de classificação escalável denominado QH-*QuantileHasing*. A principal contribuição deste trabalho consistiu na substituição do cálculo da distância entre as amostras por uma função de *hashing* baseada nos quantis, no intuito de descobrir os vizinhos mais próximos.

Os resultados encontrados demonstram que o QH é competitivo com o k-NN sob o aspecto da acurácia dos resultados e demonstra uma superioridade em uma ordem de complexidade mais rápido que o método original k-NN.

Do ponto de vista de aplicação, o QH é uma boa opção para problemas cujo o objetivo é uma aproximação do KNN e existem um grande número de amostras para serem classificadas em pouco tempo, ou ainda em um contexto de streamming onde a classificação deve ser feita de forma rápida e contínua.

Uma possibilidade de trabalho futuro é utilizar a etapa de detecção do vizinho mais próximo do QH para acelerar a solução de problemas, tais como, seleção de características ou agrupamentos hierárquicos.

References

[1] Kulis, B., Grauman, K.: Kernelized locality-sensitive hashing. IEEE Transactions on Pattern Analysis and Machine Intelligence **34**(6) (2012) 1092–1104

[2] Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the Twentieth Annual Symposium on Computational Geometry. SCG '04, New York, NY, USA, ACM (2004) 253–262

[3] Slaney, M., Casey, M.: Locality-sensitive hashing for finding nearest neighbors [lecture notes]. IEEE Signal Processing Magazine **25**(2) (March 2008) 128–131

[4] Andoni, A., Indyk, P.: Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. In: 2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06), IEEE (2006) 459–468

[5] Zhu, X., Zhang, L., Huang, Z.: A sparse embedding and least variance encoding approach to hashing. IEEE Transactions on Image Processing **23**(9) (Sept 2014) 3737–3750

[6] Buhler, J.: Efficient large-scale sequence comparison by locality-sensitive hashing. Bioinformatics **17**(5) (2001) 419–428

[7] Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. Proceedings of the 25th International Conference on Very Large Data Bases (1999) 518–529

[8] Har-Peled, S., Indyk, P., Motwani, R.: Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality. Theory of computing (2012)

[9] Indyk, P., Motwani, R.: Approximate nearest neighbors: Towards removing the curse of dimensionality. Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing (1998) 604–613

[10] Aggarwal, C.C.: Data classification: algorithms and applications. CRC Press (2014)

[11] Bekkerman, R., Bilenko, M., Langford, J.: Scaling up machine learning: Parallel and distributed approaches. Cambridge University Press (2011)

[12] Cover, T., Hart, P.: Nearest neighbor pattern classification. IEEE Transactions on Information Theory **13**(1) (jan 1967) 21–27

[13] Larose, D.T.: K-nearest neighbor algorithm. Discovering Knowledge in Data: An Introduction to Data Mining (2005) 90–106

[14] Toussaint, G., Toussaint, G.: Proximity Graphs for Nearest Neighbor Decision Rules: Recent Progress. PROGRESS", PROCEEDINGS OF THE 34 TH SYMPOSIUM ON THE INTERFACE (2002) 17—-20

[15] Wilson, D.R., Martinez, T.R.: Reduction Techniques for Instance-Based Learning Algorithms. Machine Learning **38**(3) (2000) 257–286

[16] Zolotarev, V.: One-dimensional stable distributions. (1986)

[17] van der Walt, S., Colbert, S.C., Varoquaux, G.: The NumPy Array: A Structure for Efficient Numerical Computation. Computing in Science & Engineering **13**(2) (mar 2011) 22–30