ML-KNN Algorithm Based on Nearest Neighbor Distance Weighting (Postprint)

Authors: Lu Kai, Xu Hua

Date: 2019-01-28T00:00:00+00:00

Abstract

In big data environments, the issue of high time complexity in the K-Nearest Neighbor multi-label algorithm (ML-KNN) becomes particularly pronounced; moreover, ML-KNN fails to consider the impact of the k nearest neighbors on the final classification outcome. To address these issues, this research first clusters the training set, then identifies the training data cluster closest to each test instance to serve as a new training dataset; subsequently, it computes distance weights for the nearest neighbor samples and employs these weights to characterize the influence of both the nearest neighbor and other neighbors on the prediction results; finally, it classifies test samples using a novel objective function. Experiments conducted on datasets including images and Web page text data demonstrate that the proposed algorithm achieves superior classification performance while substantially reducing time complexity.

Full Text

Preamble

Vol. 37 No. 4
Application Research of Computers
ChinaXiv Partner Journal

ML-KNN Algorithm Based on Nearest Neighbor Distance Weight

Lu Kai, Xu Hua+
(School of Internet of Things Engineering, Jiangnan University, Wuxi, Jiangsu 214122, China)

Abstract: In big data environments, the high time complexity of the multi-label K-nearest neighbor algorithm (ML-KNN) becomes particularly pronounced. Moreover, ML-KNN fails to consider the differential impact of the k nearest neighbors on the final classification results. To address these

issues, this paper first clusters the training set and then identifies the training data cluster closest to the test set as the new training dataset. Next, it calculates distance weights for the nearest neighbors and uses these weights to characterize the influence of both the nearest neighbor and other neighbors on prediction outcomes. Finally, a new objective function classifies unseen samples. Experiments on image and Web page text datasets demonstrate that the proposed algorithm achieves better classification performance while substantially reducing time complexity.

Keywords: multi-label classification; ML-KNN; clustering; nearest neighbor; distance weight

CLC Number: TP301.6
doi: 10.19734/j.issn.1001-3695.2018.09.0738

0 Introduction

The rapid development of the Internet has led to exponential growth in both data scale and information volume, rendering traditional single-label classification inadequate for current classification demands. Consequently, multi-label classification has emerged as a focal point in data classification research. In conventional single-label classification [1], each training sample corresponds to only one label from a finite, mutually exclusive label set. In multi-label classification [2], each instance may be associated with multiple labels simultaneously, which fundamentally differs from multi-class classification [3]—the latter predicts a single class from more than two options and remains essentially a single-label problem.

Multi-label classification methods generally fall into three categories: problem transformation, algorithm adaptation, and ensemble methods [4,5]. Problem transformation converts a multi-label problem into one or more single-label problems, with the most widely used approach treating each label prediction as an independent binary classification task. For each distinct label λ, a binary classifier h_λ is trained: the original dataset is transformed into $|\mathcal{L}|$ datasets D_λ, where λ indicates the presence of the label and $\neg\lambda$ indicates its absence. This method is known as Binary Relevance (BR) [6,7]. Algorithm adaptation extends traditional classifiers to directly handle multi-label problems, with ML-KNN being a typical example. Ensemble methods combine multiple base multi-label classifiers to achieve superior performance.

Recent research on ML-KNN has yielded promising results. Reference [8] proposed a hierarchical ML-KNN method by introducing auxiliary labels to compensate for ML-KNN's failure to consider label correlations, thereby improving performance. Reference [9] employed a prototype selection algorithm to reduce training data volume and derived a label ranking based on inter-label correlations, discarding many irrelevant prototypes. Reference [10] utilized Linear

Discriminant Analysis (LDA) for feature extraction to reduce data dimensionality before training classifiers and capturing label correlations. Reference [11] computed conditional probabilities between each label pair in the label set, then ranked these probabilities for labels to be predicted against already-predicted labels, multiplying the maximum value with corresponding labels and combining it with maximum a posteriori estimation to construct the multi-label classification model. These studies primarily focus on label correlations, as does reference [12]. While label correlations enhance performance, large datasets often entail numerous labels, inevitably increasing computational overhead. Additionally, big data contexts naturally suggest distributed technologies like Hadoop and Spark; reference [13] proposed a Spark-based parallel multi-label KNN algorithm. However, the cost of distributed environments required by Spark is non-negligible in practical production. The algorithm proposed in this paper achieves dual objectives—reducing time complexity and improving performance—by leveraging clustering and introducing nearest neighbor distance weights.

Received: 2018-09-19
Revised: 2018-11-08
Funding: Ministry of Education-H3C Group "Cloud-Data Convergence" Fund Project (2017A13055)
Author Biographies: Lu Kai (1994-), from Yancheng, Jiangsu, Master's student, research interests include big data and machine learning; Xu Hua (1978-), female (corresponding author), from Wuxi, Jiangsu, Associate Professor, Master's supervisor, Ph.D., research interests include computational intelligence, workshop scheduling, and big data (joanxh2003@163.com).

---

1 Preliminaries

1.1 Multi-Label Classification

A multi-label classification problem can be mathematically described as follows: Given a d-dimensional sample space $\mathcal{X} = \mathbb{R}^d$ and a finite label set $\mathcal{L} = \{l_1, l_2, ..., l_q\}$, the training set can be represented as $\mathcal{D} = \{(\mathbf{x}_i, L_i) \mid 1 \leq i \leq n\}$, where $\mathbf{x}_i \in \mathcal{X}$ is a d-dimensional feature vector and $L_i \subseteq \mathcal{L}$ is the label set associated with $\mathbf{x}_i$. The goal is to learn a multi-label classifier $h : \mathcal{X} \to 2^{\mathcal{L}}$ that predicts the label set for unseen instances.

1.2 ML-KNN Algorithm Overview

ML-KNN [14] is a lazy multi-label learning algorithm based on the traditional KNN algorithm and maximum a posteriori (MAP) principle. Its core idea is that a sample's label set can be determined by its nearest neighbors. Given a test sample $\mathbf{x}$, let its label set be $L(\mathbf{x})$. ML-KNN first identifies k nearest neighbors of $\mathbf{x}$ in the training dataset, then counts the number of neighbors belonging to each label l, denoted as $E_l^{\mathbf{x}}$. The posterior probability for each label l is calculated as follows:

$$P(H_l^{\mathbf{x}} \mid E_l^{\mathbf{x}}) = \frac{P(H_l^{\mathbf{x}}) \cdot P(E_l^{\mathbf{x}} \mid H_l^{\mathbf{x}})}{P(E_l^{\mathbf{x}})}$$

where $H_l^{\mathbf{x}}$ represents the event that label l belongs to $\mathbf{x}$'s true label set. The final prediction rule is:

$$y_l(\mathbf{x}) = \begin{cases} 1, & P(H_l^{\mathbf{x}} \mid E_l^{\mathbf{x}}) \geq 0.5 \\ 0, & P(H_l^{\mathbf{x}} \mid E_l^{\mathbf{x}}) < 0.5 \end{cases}$$

where $y_l(\mathbf{x}) = 1$ indicates that label l is in $\mathbf{x}$'s predicted label set, and $y_l(\mathbf{x}) = 0$ indicates it is not.

---

2 Improved Algorithm

In any dataset, a sample's k nearest neighbors should theoretically share some similarity with the sample's own label set, with this similarity varying according to the distance between neighbors and the sample. Greater distances correspond to lower similarity. However, ML-KNN's calculation process does not account for this relationship. To address this limitation, this paper simultaneously considers the influence of both the k nearest neighbors and the single nearest neighbor, using weights to quantify these effects. Based on Equation (1), we derive a new classification function:

$$P(H_l^{\mathbf{x}} \mid E_l^{\mathbf{x}}) = w \cdot P(H_l^{\mathbf{x}} \mid E_l^{\mathbf{x}}) + (1 - w) \cdot P(H_l^{\mathbf{x}} \mid NN_l^{\mathbf{x}})$$

where w represents the weight converted from the distance between $\mathbf{x}$ and its nearest neighbor, serving as the weight for the nearest neighbor in our improved algorithm; $P(H_l^{\mathbf{x}} \mid NN_l^{\mathbf{x}})$ denotes the posterior probability based on the nearest neighbor's label information; and $NN_l^{\mathbf{x}}$ indicates whether the nearest neighbor of $\mathbf{x}$ contains label l, taking values of only 0 or 1.

Equation (3) shows that the key to our algorithm is determining the weight w. Three primary methods exist for converting distance to weight: inverse function, subtraction function, and Gaussian function. For a distance d, the inverse function is simplest: $w = 1/(d + c)$, where constant c prevents infinite values when d approaches zero. However, this approach assigns excessively large weights to nearest neighbors while causing weights to decrease too rapidly as d increases. The subtraction function similarly introduces a constant c: $w = c - d$ if $d \leq c$, which avoids over-weighting nearest neighbors but inevitably decreases to zero. The Gaussian function overcomes these limitations by converting distance to weight using:

$$w = a \cdot e^{-2d^2/c^2}$$

where a and c are constants, and all distances are Euclidean. The Gaussian function avoids the potential problem of excessive neighbor weighting while ensuring weights decrease smoothly with increasing distance but never reach zero. Therefore, this paper adopts the Gaussian approach.

The detailed steps of our improved algorithm are as follows:

Algorithm 1: Improved ML-KNN Algorithm

Input: Training data, test data
Output: Classification results

1. Use the k-means clustering algorithm to partition the training data into r clusters with centers $R_1, R_2, ..., R_r$.
2. Use the k-means clustering algorithm to partition the test data into t clusters with centers $T_1, T_2, ..., T_t$.
3. For each test cluster T_i where $i \in \{1, 2, ..., t\}$:
   - For each training cluster R_j where $j \in \{1, 2, ..., r\}$:
     - Calculate the distance $D(T_i, R_j)$ between cluster centers.
   - Find the nearest training cluster: $R_{\text{New}} = \arg\min_j D(T_i, R_j)$.
4. For each test sample $\mathbf{u}$ in T_i:
   - Use R_{New} as the new training dataset.
   - Find $\mathbf{u}$'s k nearest neighbors in R_{New}.
   - Calculate the distance d from $\mathbf{u}$ to its nearest neighbor.
   - Convert distance d to weight w using Equation (4).
5. For each label $l \in \mathcal{L}$:
   - Calculate the posterior probability $P(H_l^{\mathbf{u}} \mid E_l^{\mathbf{u}})$ using Equation (3).
6. For each test sample $\mathbf{u}$:
   - If $P(H_l^{\mathbf{u}} \mid E_l^{\mathbf{u}}) \geq 0.5$, include l in $\mathbf{u}$'s predicted label set.
7. Output the final predictions.

According to Algorithm 1, we first apply K-means to cluster both training and test data [15], then select the nearest training cluster for each test cluster. This approach reduces data scale and ML-KNN's computational load, thereby decreasing time complexity. Additionally, the nearest training cluster exhibits greater similarity to the test data compared to other training data, improving various multi-label evaluation metrics. Finally, we classify the data using our improved ML-KNN algorithm.

---

3 Experimental Results and Analysis

For convenience, we denote our proposed algorithm as DML-KNN and the improved algorithm from reference [16] as IML-KNN. Both algorithms were implemented in MATLAB, a powerful tool for algorithm simulation.

3.1 Evaluation Metrics

We employ five standard multi-label evaluation metrics:

a) Hamming Loss: Measures the frequency of misclassified label pairs—either when a label belonging to the sample is not predicted, or when a label not belonging to the sample is predicted.

$$\text{hloss} = \frac{1}{m}\sum_{i=1}^{m}\frac{|Y_i \Delta h(\mathbf{x}_i)|}{Q}$$

where Q is the number of labels, Y_i is the true label set, $h(\mathbf{x}_i)$ is the predicted label set, and Δ denotes symmetric difference.

b) Coverage: Evaluates the average number of labels needed in the ranked list to cover all possible true labels for a sample.

$$\text{coverage} = \frac{1}{m}\sum_{i=1}^{m}\max_{l\in Y_i}\text{rank}(\mathbf{x}_i, l) - 1$$

where $\text{rank}(\mathbf{x}_i, l)$ returns the rank of label l for sample $\mathbf{x}_i$.

c) One-error: Assesses how often the top-ranked label is not in the sample's true label set. If the top-ranked label belongs to the true label set, the value is 0; otherwise, it is 1.

$$\text{one-error} = \frac{1}{m}\sum_{i=1}^{m}[\![\arg\max_{l\in\mathcal{L}} f(\mathbf{x}_i, l) \notin Y_i]\!]$$

where $f(\mathbf{x}_i, l)$ is the decision function and $[\![\cdot]\!]$ is the indicator function.

d) Ranking Loss: Evaluates the average fraction of incorrectly ordered label pairs.

$$\text{rloss} = \frac{1}{m}\sum_{i=1}^{m}\frac{|\{(l, l') \mid f(\mathbf{x}_i, l) \leq f(\mathbf{x}_i, l'), (l, l') \in Y_i \times \overline{Y}_i\}|}{|Y_i| \cdot |\overline{Y}_i|}$$

where $\overline{Y}_i$ is the complement of Y_i in the full label set.

e) Average Precision: Measures the average fraction of true labels ranked above a particular label.

$$\text{avgprec} = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{|Y_i|}\sum_{l\in Y_i}\frac{|\{l' \in Y_i \mid \text{rank}(\mathbf{x}_i, l') \leq \text{rank}(\mathbf{x}_i, l)\}|}{\text{rank}(\mathbf{x}_i, l)}$$

For the first four metrics, lower values indicate better performance; for average precision, higher values are better.

3.2 Datasets

Table 1 summarizes four datasets from KEEL: the email dataset *enron*, image dataset *scene*, yeast cell dataset *Yeast*, and Web page text dataset *delicious*. Label cardinality indicates the average number of labels per sample, while label density is label cardinality divided by the total number of labels.

Table 1: Experimental Dataset Information

Dataset	Samples	Features	Labels	Cardinality	Density
enron	1702	1001	53	3.38	0.064
scene	2407	294	6	1.07	0.179
Yeast	2417	103	14	4.24	0.303
delicious	16105	500	983	19.02	0.019

3.3 Experimental Results and Analysis

Based on the four datasets from Section 3.2 and the five evaluation metrics from Section 3.1, we assessed our algorithm's performance. Algorithm 1 involves several key parameters: number of nearest neighbors k, Gaussian function parameters a and c, training cluster count r, and test cluster count t. For ML-KNN, k should be neither too small (insufficient neighbor information) nor too large (excessive computation). Through extensive experimentation, we set $k = 20$.

Tables 2-5 show experimental results for the three algorithms across four datasets with different a and c values.

Table 2: Results on enron Dataset

Algorithm	$a = 10, c = 1/2$	$a = 8, c = 1/3$	$a = 10, c = 1/2$
	hloss	rloss	one-error
ML-KNN	0.058	0.178	0.312
IML-KNN	0.054	0.165	0.298
DML-KNN	**0.049**	**0.152**	**0.276**

Table 3: Results on scene Dataset

Algorithm	$a = 8, c = 1/5$	$a = 10, c = 1/2$	$a = 8, c = 1/3$
	hloss	rloss	one-error
ML-KNN	0.092	0.156	0.287
IML-KNN	0.088	0.148	0.265

Algorithm	$a = 8, c = 1/5$	$a = 10, c = 1/2$	$a = 8, c = 1/3$
DML-KNN	**0.081**	**0.134**	**0.241**

Table 4: Results on Yeast Dataset

Algorithm	$a = 10, c = 1/6$	$a = 8, c = 1/4$	$a = 10, c = 1/2$
	hloss	rloss	one-error
ML-KNN	0.198	0.167	0.234
IML-KNN	0.189	0.158	0.221
DML-KNN	**0.176**	**0.143**	**0.203**

Table 5: Results on delicious Dataset

Algorithm	$a = 10, c = 1/5$	$a = 8, c = 1/3$	$a = 10, c = 1/2$
	hloss	rloss	one-error
ML-KNN	0.017	0.312	0.421
IML-KNN	0.016	0.298	0.405
DML-KNN	**0.014**	**0.267**	**0.376**

The results show that DML-KNN performs best on *enron* when $a = 10, c = 1/2$, on *scene* when $a = 8, c = 1/5$, on *Yeast* when $a = 10, c = 1/6$, and on *delicious* when $a = 10, c = 1/5$. Notably, DML-KNN demonstrates significant advantages in time complexity, particularly on the large *delicious* dataset, where it achieves substantial improvements across all five performance metrics while dramatically reducing computation time.

Furthermore, as shown in Algorithm 1, different cluster numbers critically affect performance by determining the training set for each test cluster. Tables 6-9 illustrate DML-KNN's performance under varying cluster counts when a and c are fixed.

Table 6: Effect of Different r, t on enron

Configuration	hloss	rloss	one-error	coverage	avgprec	Time(s)
$r = 2, t = 2$	**0.049**	**0.152**	**0.276**	**29.18**	**0.551**	**45.2**
$r = 3, t = 2$	0.051	0.158	0.281	29.67	0.547	43.8
$r = 3, t = 3$	0.053	0.161	0.289	30.12	0.543	41.5

Table 7: Effect of Different r, t on scene

Configuration	hloss	rloss	one-error	coverage	avgprec	Time(s)
$r = 2, t = 2$	**0.081**	**0.134**	**0.241**	**1.58**	**0.841**	**28.7**
$r = 3, t = 3$	0.084	0.139	0.248	1.63	0.837	26.4
$r = 4, t = 3$	0.087	0.142	0.253	1.68	0.833	24.9

Table 8: Effect of Different r, t on Yeast

Configuration	hloss	rloss	one-error	coverage	avgprec	Time(s)
$r = 2, t = 2$	**0.176**	**0.143**	**0.203**	**5.34**	**0.789**	**58.4**
$r = 3, t = 2$	0.181	0.149	0.211	5.42	0.784	55.7
$r = 3, t = 3$	0.185	0.153	0.218	5.51	0.781	52.3

Table 9: Effect of Different r, t on delicious

Configuration	hloss	rloss	one-error	coverage	avgprec	Time(s)
$r = 11, t = 4$	**0.014**	**0.267**	**0.376**	**287.4**	**0.348**	**892.5**
$r = 13, t = 3$	0.015	0.271	0.381	291.2	0.345	876.3
$r = 14, t = 4$	0.015	0.273	0.384	293.8	0.343	869.7

The results indicate that $r = 2, t = 2$ works best for *enron*, *scene*, and *Yeast*, while $r = 11, t = 4$ is optimal for *delicious*. The cluster counts must be chosen appropriately: if r is too small, time complexity reduction is minimal; if r is too large, accuracy may degrade despite time savings. For t, which determines iteration count and computational load, values should not be excessively large.

Figures 1-6 [FIGURE:1-6] visually demonstrate DML-KNN's superiority. Since coverage and time vary significantly across datasets, we present these comparisons using percentage improvements.

---

4 Conclusion

To address the high time complexity of traditional ML-KNN, this paper first clusters training and test data to reduce data scale and computational load, thereby decreasing time complexity. Additionally, we consider that the k nearest neighbors exert varying influence on predictions based on their distances, quantifying this effect through nearest neighbor distance weighting. By combining clustering with distance weighting, we propose an improved ML-KNN algorithm. Experimental results demonstrate that our algorithm achieves superior classification performance and outperforms the original algorithm across multiple multi-label evaluation metrics.

References

[8] Qi Hongwei, Zhou Yanquan, Guo Qi. A hierarchical ML-KNN method for complex emotion analysis on customer reviews [C]// Proc of International Conference on Mechatronics Engineering and Information Technology. 2016: 6.

[9] Calvo-Zaragoza J, Valero-Mas J J, Rico-Juan J R. Improving KNN multi-label classification in Prototype selection scenarios using class proposals [J]. Pattern Recognition, 2015, 48 (5): 1608-1622.

[10] Li Zhiqiang. An improved ML-KNN multi-label classification model based on feature dimensionality reduction [C]// Proc of International Conference on Computer, Mechatronics and Electronic Engineering. [S.l.]: Science and Engineering Research Center, 2016: 4.

[11] Tan Hefeng, Liu Zhengyi. Multi-label K-nearest neighbor algorithm by exploiting label correlation [J]. Journal of Computer Applications, 2015, 35 (10): 2761-2765.

[12] Yang Xiaodan, Zhou Lihua, Wang Lizhen. An improved ML-KNN approach based on coupled similarity [C]// Proc of Asia-Pacific Web Conference. Berlin: Springer International Publishing, 2016: 77-89.

[13] Wang Jin, Xia Cuiping, OuYang Weihua, et al. Parallel multi-label K-nearest neighbor algorithm based on Spark [J]. Computer Engineering & Science, 2017, 39 (2): 227-235.

[14] Zhang Minling, Zhou Zhihua. ML-KNN: a lazy learning approach to multi-label learning [J]. Pattern Recognition, 2007, 40 (7): 2038-2048.

[15] Deng Zhenyun, Zhu Xiaoshu, Cheng Debo, et al. Efficient KNN classification algorithm for big data [J]. Neurocomputing, 2016, 195 (C): 143-148.

[16] Zeng Yong, Fu Haoming, Zhang Yuping, et al. An improved ML-KNN algorithm by fusing nearest neighbor classification [J]. DEStech Transactions on Computer Science and Engineering, 2016 (aics): 8196.

Note: Figure translations are in progress. See original paper for figures.