A Nonlinear Classifier Based on Factorization Machines Model

Xiaolong Liu, Yanming Zhang, and Chenglin Liu

National Laboratory of Pattern Recognition (NLPR)
Institute of Automatic, Chinese Academy of Science
No. 95, Zhongguancun East Road, Beijing, 100190, China
{xiaolong.liu,ymzhang,liucl}@nlpr.ia.ac.cn

Abstract. Polynomial Classifier (PC) is a powerful nonlinear classification method that has been widely used in many pattern recognition problems. Despite its high classification accuracy, its computational cost for both training and testing is polynomial with the dimensionality of input data, which makes it unsuitable for large-scale problems. In this work, based on the idea of factorization machines (FMs), we propose an efficient classification method which approximates PC by performing a low-rank approximation to the coefficient matrix of PC. Our method can largely preserve the accuracy of PC, while has only linear computational complexity with the data dimensionality. We conduct extensive experiments to show the effectiveness of our method.

Keywords: Polynomial Classifier, Factorization Machines model, Low-rank Approximation.

1 Introduction

Classification is one of the most fundamental problems in machine learning, and plays a central role in many applications, such as characters recognition [1], visual object classification [2], and text classification [3]. Actually, classifier design has always been a focus of the machine learning community, and obtained great development in the past decades.

According to the form of the classification function, classification methods can be roughly divided into two categories: linear methods and nonlinear methods. Linear classification methods, such as perceptron [4], linear discriminant analysis (LDA) [5] and linear Support Vector Machines (SVM) [6], are efficient in both training and testing, and thus are ready for large-scale applications which becomes more and more common nowadays. However, due to the limited representation ability of linear functions, these methods are often abused for their low accuracies. On the other hand, nonlinear methods, such as multilayer perceptron (MLP) [7], polynomial classifier (PC) [8], kernel SVM [9], are much more powerful and can always obtain better accuracies than linear methods. However, their computational cost for both training and testing is much higher, and cannot fulfill the requirements of many applications. For example, the cost of kernel SVM is at least $O(n^2)$ for training and $O(n_{sv})$ (n_{sv} is the number of support vectors) for testing.

S. Li et al. (Eds.): CCPR 2014, Part I, CCIS 483, pp. 1–10, 2014.

Due to its high performance, PC is a very popular nonlinear classification method in pattern recognition and machine learning. However, for a d-order PC, it has $O(p^d)$ parameters to learn and takes $O(p^d)$ to perform one evaluation operation. Thus, it is slow in both training and testing. In this work, we focus on speeding up PC, while preserving its accuracy as much as possible. Essentially, based on the idea of Factorization Machines (FMs) [10], our approach approximates the coefficient matrix in polynomial function by a low-rank matrix. Using the factorization, our classifier has only $O(kdp)$ parameters (k is the dimensionality of the factorization matrix) and the cost for predicting one sample is also $O(kdp)$.

FMs [10] were originally proposed in the context of Recommender Systems, and have obtained great success in many real-world applications. The current study of FMs is mainly on two aspects: improving the accuracy of FMs by adding specific context-aware information [11, 12], and fast learning algorithms [13]. As far as we know, there is no study about the application of FMs for classifier design.

The paper is organized as follows: Section 2 gives a detailed introduction to the FMs model, including its definition, relationship with PC and the evaluation cost. Section 3 proposes the method for applying the FMs in the classification problem and our learning algorithm. Experiments results are shown in Section 4 to verify the effectiveness of our method.

2 Factorization Machines Model

In this section, we introduce the FMs model in detail and discuss its computational cost for evaluation operation.

2.1 FMs Model [10]

The model equation for a factorization machine of degree $d = 2$ is defined as:

$$\hat{y}(x) = w_0 + \sum_{i=1}^{p} w_i x_i + \sum_{i=1}^{p} \sum_{j=i+1}^{p} \left\langle v_i, v_j \right\rangle x_i x_j \,, \tag{1}$$

where p is the dimensionality of input x, and $\langle \cdot, \cdot \rangle$ denotes the inner product of two vectors. $w_0 \in \mathbb{R}$, $w \in \mathbb{R}^p$, $V \in \mathbb{R}^{p \times k}$ are model parameters which can be estimated from the training set. w_0 is the global bias, w_i is the weight of the feature x_i, and $\hat{w}_{i,j} := \left\langle v_i, v_j \right\rangle$ is the weight of the second-order feature $x_i x_j$.

2.2 Relationship with PC

The decision function of a second-order PC can be written as:

$$\hat{y}(x) = w_0 + w'x + \frac{1}{2} x'Wx \tag{2}$$

where W is a symmetric matrix of size p-by-p. Due to the second-order terms, the evaluation of PC needs $O(p^2)$ operations. On the other hand, it is easy to show the FMs model defined in Eq. (1) can be written as:

$$\hat{y}(x) = w_0 + w'x + \frac{1}{2}x'(VV' - diag(VV'))x \tag{3}$$

where $V = [v_1, v_2, ..., v_p]' \in \mathbb{R}^{p \times k}$, and $diag(VV')$ is a matrix of size p-by-p with its diagonal equals to the diagonal of VV' and all the off-diagonal elements equal to 0.

Thus, from the perspective of matrix approximation, the core idea of FMs is to approximate a symmetric matrix W by $VV' - diag(VV')$. And as we will show immediately, by utilizing this matrix factorization form, a FMs model can be evaluated in $O(kp)$, instead of $O(p^2)$ as in the second-order PC. When the dimensionality of V is low ($k \ll p$), FMs will significantly faster than PC.

2.3 Evaluation Cost

In this section, we show that the evaluation of a FMs model can be done in linear time $O(kp)$, which is the main advantage of our method. We reformulate the third term in Eq. (1) as:

$$\begin{aligned}
&\sum_{i=1}^{p}\sum_{j=i+1}^{p}\langle v_i, v_j\rangle x_i x_j \\
&= \frac{1}{2}\sum_{i=1}^{p}\sum_{j=1}^{p}\langle v_i, v_j\rangle x_i x_j - \frac{1}{2}\sum_{i=1}^{p}\langle v_i, v_j\rangle x_i x_i \\
&= \frac{1}{2}\left(\sum_{i=1}^{p}\sum_{j=1}^{p}\sum_{f=1}^{k} v_{i,f} v_{j,f} x_i x_j - \sum_{i=1}^{p}\sum_{f=1}^{k} v_{i,f} v_{i,f} x_i x_i\right) \\
&= \frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{i=1}^{p} v_{i,f} x_i\right)^2 - \sum_{i=1}^{p} v_{i,f}^2 x_i^2\right)
\end{aligned} \tag{4}$$

This equation has only linear complexity for both k and p, thus its computation complexity is $O(kp)$.

2.4 d-way Factorization Machines

The 2-way FMs described above can be smoothly generalized to a d-way FMs:

$$\hat{y}(x) = w_0 + \sum_{i=1}^{p} w_i x_i + \sum_{l=2}^{d}\sum_{i_1=1}^{p}\cdots\sum_{i_l=i_{l-1}+1}^{p}\left(\prod_{j=1}^{l} x_{i_j}\right)\left(\sum_{f=1}^{k_l}\prod_{j=1}^{l} v_{i_j,f}^{(l)}\right) \tag{5}$$

where the weight matrices for the l-order product are factorized by the PARAFAC model [14] with the parameters below $V^{(l)} \in \mathbb{R}^{p \times k_l}, k_l \in \mathbb{N}_0^+$.

The straight forward computation complexity for Eq. (5) is $O(k_d p^d)$. But with the same arguments as in section 2.3, we can show it can be computed in linear $O(k_d dp)$.

3 Training FMs for Classification

In this section, we present how to apply FMs models to classifier design.

3.1 Objective Function

We propose to optimize the following objective function to train a FMs classifier:

$$OPTREG(S, \lambda) = \arg\min_{\Theta} \left(\sum_{(x,y) \in S} l(\hat{y}(x \mid \Theta), y) + \sum_{\theta \in \Theta} \lambda_\theta \theta^2 \right) \tag{6}$$

where $\hat{y}(x \mid \Theta)$ is the FMs classifier, Θ are the classifier parameters, and $l(\hat{y}(x \mid \Theta), y)$ is the loss function that measures the prediction error of the FMs classifier on x. Thus, the first term of the objective function is to minimize the prediction error of the classifier on the training set, and the second term is to regularize the classifier from over-fitting.

For a binary classification problem ($y \in \{-1,1\}$), we adapt the negative log-likelihood as the loss function, which is defined as:

$$l(\hat{y}(x \mid \Theta), y) = -\ln \sigma(\hat{y}(x \mid \Theta) \cdot y) \tag{7}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$. For a multi-class problem, we solve it by the standard one-vs-all strategy.

3.2 Learning Algorithm

In this work, we use stochastic gradient descent (SGD) algorithm for training a FMs classifier.

3.3 Stochastic Gradient Descent (SGD)

FMs can be optimized with SGD [10] which has low computation and storage complexity.

The algorithm iterates over all the samples in the training dataset and performs updates on the model parameters.

$$\theta \leftarrow \theta - \eta\left(\frac{\partial}{\partial\theta} l(\hat{y}(x \mid \Theta), y) + 2\lambda_\theta \theta\right) \tag{8}$$

where $\eta \in \mathbb{R}^+$ is the learning rate for gradient descent. The gradient of the negative log-likelihood loss function is:

$$\frac{\partial}{\partial\theta} l(\hat{y}(\mathrm{x} \mid \Theta), \mathrm{y}) = \frac{\partial}{\partial\theta} - \ln \sigma(\hat{y}(x \mid \Theta) \cdot y) = (\sigma(\hat{y}(x \mid \Theta) \cdot y) - 1)\, y \frac{\partial}{\partial\theta} \hat{y}(x \mid \Theta) \tag{9}$$

3.4 Complexity and Hyper-Parameters

The SGD algorithm for FMs has a linear computational and constant storage complexity. For one iteration over all training cases, the runtime of SGD is $O(knp)$.

For SGD algorithm, there are several critical hyper-parameters.

- Leaning rate η: The convergence of SGD depends largely on η. If η is chosen too big, the SGD doesn't converge, and too small, convergence is slow.
- Regularization parameter λ: The generalization capabilities of FMs depends largely on λ, and in this paper we use grid search to chose the proper λ for SGD algorithm. As there are several regularization parameters, the grid has exponential size and thus this search is very time-consuming. To make the search more feasible, the number of regularization parameters is reduced to only two: λ^w for w_0 and w, and λ^v for factorization matrix V.
- Initialization σ: The model parameters of FMs are initialized with non-constant values which are sampled from a zero-mean normal distribution with standard deviation σ. Typically small values are used for σ.

4 Experiments

To examine the classification performance of FMs model, we compared it with couple of commonly used classifiers on 7 datasets.

4.1 Compared Classifiers

We compare the FMs with PCA-PNC (polynomial network classifier with dimensionality reduction by principal component analysis) [15], linear-SVM, and poly-SVM (SVM with polynomial kernel function).

The PCA-PNC is a subspace-feature-based PNC that can efficiently reduce the computation complexity of PNC and perform fairly well in practice [15]. First, the input feature vector is projected onto an m-dimensional principal subspace ($m < p$); then, the network is computed on the subspace features; finally, the input pattern is classified to the class of maximum output.

For SVM, liblinear [16] and libSVM [17] are used as implementations of linear-SVM and poly-SVM. And the kernel function used for poly-SVM in our experiments is $K(x_i, x_j) = (\gamma x_i' x_j + r)^d, \gamma > 0$, where $d = 2$.

The linear-SVM and poly-SVM are typical linear and polynomial classifiers on full feature space; comparing with them, we can evaluate the classification capability of FMs. FMs and PCA-PNC have similar structures, so we want to compare the performance of them. Table 1 below lists the training and test computation complexity of them.

Table 1. Training and test computation complexity

Classifier	Training	Test
FMs (bi-class)	$O(knp)$ [10]	$O(kp)$ [10]
PCA-PNC (multi-class)	$O(p^3) + O(m^2 n)$	$O(pm) + O(m^2 M)$
linear-SVM (bi-class)	$O(pn)$ [6]	$O(p)$ [6]
poly-SVM (bi-class)	$O(pn^2)$ [6]	$O(pn_{sv})$ [6]

Where p is the original dimensionality of feature vectors; n is the sample number of train data subset; M is the number of classes; k is the factorization matrix dimensionality of FMs; m is the subspace dimensionality of PAC-PNC; and n_{sv} is the number of support vectors of SVM.

4.2 Datasets

We select 7 datasets, 5 of them are from the UCI Machine Learning Repository [18] and 2 from LibSVM datasets [19], as summarized in Table 2. We select the multi-class datasets that have at least 10 features.

Most of the data sets have been partitioned into standard training and test subsets. For the others, we arrange the samples in random order and evaluate in 5-fold cross-validation; and we give the average value and standard deviation of the classification accuracy on each of them.

Table 2. Summary of 7 datasets. The right column shows the dimensionality of factorization matrix for FMs or the subspace dimensionality for PCA-PNC (multiple of m_1).

#	Name	#class	#feature	#train	#test	m_1
1	Waveform	3	21	5,000	5-fold	2
2	Vehicle	4	18	846	5-fold	2
3	Segment	7	19	2,100	210	3
4	Letter	26	16	16,000	4,000	3
5	Splice	2	60	1,000	2,175	2
6	Isolet	26	617	6,238	1,559	10
7	Gisette	2	5,000	6,000	1,000	10

4.3 Experiment Design

The PCA-PNC is a network with M output nets, so it can be directly applied to multi-class classification. FMs, linear-SVM and poly-SVM are all bi-class classifiers, and we adopt one-vs-all strategy to make them available for multi-class classification.

For PCA-PNC, we set the dimensionality of subspace as $m = l \cdot m_1$, $l = 1,...,5$. m_1 depends on the dataset, and has been listed in the right columns of Table 2. For FMs, we set the dimensionality of factorization matrix as $k = l \cdot m_1$, $l = 1,...,5$, corresponding to PCA-PNC. For SVMs, the feature vectors are uniformly scaled.

4.4 Experiments Results

1. FMs and PCA-PNC

The classification accuracy (%) of FMs and PCA-PNC on test subset of the 7 datasets is shown in Table 3. For each dataset, the accuracy of PCA-PNC and FMs on variable dimensionalities is listed in two rows. And we can see, FMs can give test accuracy comparable to PCA-PNC.

Table 3. Test accuracies (%) of FMs and PCA-PNC on 7 dataset

Dataset	PCA= m_1	PCA= $2m_1$	PCA= $3m_1$	PCA= $4m_1$	PCA= $5m_1$
	factor= m_1	factor= $2m_1$	factor= $3m_1$	factor= $4m_1$	factor= $5m_1$
Waveform	63.78	86.40	86.92	**87.12**	86.98
	83.92±1.50	84.36±1.27	85.32±1.01	85.70±0.77	85.90±0.86
Vehicle	53.19	67.38	71.75	76.24	78.37
	78.37±2.91	78.61±2.49	78.85±2.49	79.32±2.89	**80.15±2.88**
Segment	69.14	80.48	83.76	85.86	85.71
	91.10	91.33	92.00	92.29	**92.33**
Letter	26.73	58.73	75.78	84.67	88.20
	78.20	84.33	87.33	89.60	**91.33**
Splice	70.30	77.89	82.94	84.23	84.18
	85.47	85.52	85.56	85.61	**85.66**
Isolet	80.18	91.28	93.52	94.16	95.19
	94.18	94.24	94.78	95.02	**95.45**
Gisette	95.80	97.30	97.60	**98.10**	98.10
	97.10	97.40	97.60	98.00	98.10

For Waveform, when the dimensionality is low, FMs give a higher accuracy than PCA-PNC; when the dimensionality goes up, PCA-PNC surpasses FMs gradually. This can be explained that when the dimensionality is low, the linear terms of FMs with full dimensionality make FMs more expressive; but when the dimensionality goes up, the PCA-PNC shows its power.

On the other datasets, FMs always give higher accuracy than PCA-PNC, especially on lower dimensionality, however, when the dimensionality goes up, the gap is gradually narrowing.

Table 4. Highest accuracier of FMs, PCA-PNC, linear-SVM and poly-SVM

Dataset	FMs	PCA-PNC	linear-SVM	poly-SVM
Waveform	85.90±0.86	**87.12**	86.76	82.37
Vehicle	80.15±2.88	78.37	78.01	**81.21**
Segment	**92.33**	85.86	91.24	92.14
Letter	91.33	88.20	67.50	**94.43**
Splice	85.66	84.18	83.63	**88.32**
Isolet	95.45	95.19	94.61	**96.28**
Gisette	**98.10**	**98.10**	**98.10**	98.75
Average Acc	89.85	88.15	85.69	**90.50**

2. FMs, PCA-PNC, and SVM

The highest accuracies of FMs, PCA-PNC, linear-SVM and poly-SVM on the 7 datasets are compared in Table 4, and the average classification accuracy of all the classifiers is in the last row.

We can find that in most datasets, the FMs give classification accuracy between linear (linear-SVM) and polynomial (poly-SVM) classifiers. Comparing the accuracy of FMs and PCA-PNC, it is evident that the classification accuracy of FMs is higher than PCA-PNC on most datasets.

4.5 Computation Complexity of FMs

To compare the predict computation complexity of different classifiers, we chose three datasets (Vehicle, Splice, and Isolet) from Table 2, The predict computation time (ms) of FMs ($factor = 5m_1$), PCA-PNC ($m = 5m_1$), linear-SVM and poly-SVM on the test subset is listed in Table 5.

Table 5. The predict computation time of FMs, PCA-PNC, linear-SVM and poly-SVM

Dataset	#test	#dim $5m_1$/SVM	FMs	PCA-PNC	linear-SVM	poly-SVM	
						#sv	time
Vehicle	169	10/18	4	6	2	667	11
Splice	2175	10/60	67	27	20	481	220
Isolet	1559	50/617	6230	654	800	3106	20410

For all the three datasets, the computation time of FMs and PCA-PNC are much smaller than poly-SVM. This can be explained as the reduced dimensionality of FMs and PCA-PNC speed up the predict operation.

We can find that there is no fixed relationship between the computation time of FMs and PCA-PNC, and this seems contradictory to what we have discussed in Table 1---the computation complexity of FMs is linear, and PCA-PNC is polynomial. We would take a closer look at the computation complexity of FMs and PCA-PNC.

For a dataset whose dimensionality is p and number of classes is M, the computation complexity of the FMs with $factor = k$, is supposed to be $(p + k(2p+1))M$; and that of the PCA-PNC is $pm + \frac{1}{2}m(m+1) + (m + \frac{1}{2}m(m+1))M$ (where $m = k$). So the practical computation time of FMs and PCA-PNC largely depend on p and M. In most cases, $m \ll p$ is fulfilled, and this makes the practical computation time of FMs comparable, even larger than PCA-PNC when $m = k$.

4.6 Comparison of Classification Performance

To compare the performance of FMs and PCA-PNC, we plot them in Fig. 1, where the horizontal axis is the amount of multiplication calculation for a feature vector, and the vertical axis is the classification accuracy on the test subset.

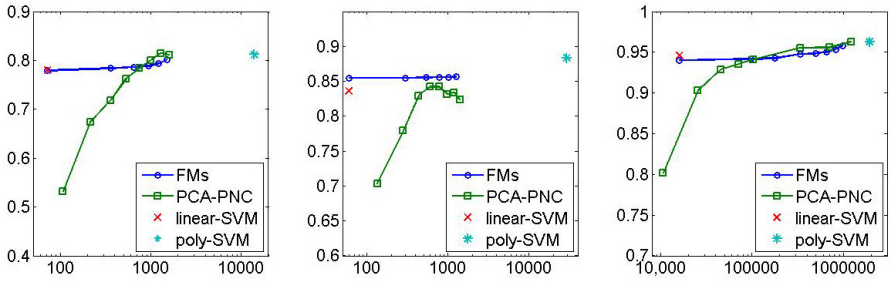

Fig. 1. Performance of FMs and PCA-PNC on Vehicle, Segment, and Isolet

The figures in Fig. 1 from left to right are in turn of Vehicle, Splice, and Isolet datasets. From the figures in Fig. 1, we can conclude that the FMs classifier can give classification accuracy between linear-SVM and poly-SVM. And FMs give much higher classification accuracy than PCA-PNC on low factorization dimensionality; when the dimensionality goes up, they are still comparable.

5 Conclusion

In this paper, we attempt to apply Factorization Machines model to general classifier design. Then we evaluate the classification performance of FMs, PCA-PNC, linear-SVM, and poly-SVM on 7 datasets. And we can now carefully give the following conclusions about FMs and the other three classifiers.

1. FMs can give classification accuracy between linear (linear-SVM) and polynomial (poly-SVM) classifiers, while its computation complexity is linear.

2. FMs can always give higher classification performance than PCA-PNC, especially on low dimensionality, however, when the dimensionality goes up, the gap is gradually narrowing.

References

1. Cheriet, M., Kharm, N., Liu, C.L., et al.: Character recognition systems: A guide for students and practitioners. John Wiley & Sons (2007)
2. Everingham, M., Van Gool, L., Williams, C.K.I., et al.: The pascal visual object classes (voc) challenge. International Journal of Computer Vision 88(2), 303–338 (2010)
3. Joachims, T.: Text categorization with support vector machines: Learning with many relevant features. In: Nédellec, C., Rouveirol, C. (eds.) ECML 1998. LNCS, vol. 1398, pp. 137–142. Springer, Heidelberg (1998)
4. Freund, Y., Schapire, R.E.: Large margin classification using the perceptron algorithm. Machine Learning 37(3), 277–296 (1999)
5. Lzenman, A.J.: Linear Discriminant Analysis. Springer, New York (2008)
6. Burges, C.J.C.: A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery 2(2) (1998)
7. Pal, S.K., Mitra, S.: Multilayer perceptron, fuzzy sets, and classification. IEEE Transactions on Neural Networks 3(5), 683–697 (1992)
8. Liu, C.L., Sako, H.: Class-specific feature polynomial classifier for pattern classification and its application to handwritten numeral recognition. Pattern Recognition 39(4), 669–681 (2006)
9. Schölkopf, B., Smola, A.J.: Learning with kernels: Support vector machines, regularization, optimization, and beyond. MIT Press (2002)
10. Rendle, S.: Factorization machines. In: IEEE 10th International Conference on Data Mining, ICDM (2010)
11. Rendle, S., Gantner, Z., Freudenthaler, C., et al.: Fast context-aware recommendations with factorization machines. In: Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (2011)
12. Rendle, S.: Factorization machines with libFM. ACM Transactions on Intelligent Systems and Technology (TIST) 3(3), 57 (2012)
13. Robbins, H., Monro, S.: A stochastic approximation method. The Annals of Mathematical Statistics, 400–407 (1951)
14. Harshman, R.A.: Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multimodal factor analysis. In: UCLA Working Papers in Phonetics (1970)
15. Liu, C.-L.: Polynomial network classifier with discriminative feature extraction. In: Yeung, D.-Y., Kwok, J.T., Fred, A., Roli, F., de Ridder, D. (eds.) SSPR&SPR 2006. LNCS, vol. 4109, pp. 732–740. Springer, Heidelberg (2006)
16. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: LIBLINEAR: A library for large linear classification. The Journal of Machine Learning Research 9, 1871–1874 (2008)
17. Chang, C.C., Li, C.J.: LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology (TIST) 2(3) (2011)
18. UC Irvine Machine Learning Repository, `http://archive.ics.uci.edu/ml/` (accessed June 9)
19. LIBSVM Data, `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/` (accessed June 9)