2019 IEEE 35th International Conference on Data Engineering (ICDE)

Nonlinear Models Over Normalized Data

1st Zhaoyue Cheng *Department of Computer Science*
University of Toronto
cheng.zhaoyue@mail.utoronto.ca
2nd Nick Koudas *Department of Computer Science*
University of Toronto
koudas@cs.toronto.edu

***Abstract*—Machine Learning (ML) applications are proliferating in the enterprise. Increasingly enterprise data are used to build sophisticated ML models to assist critical business functions. Relational data which are prevalent in enterprise applications are typically normalized; as a result data have to be denormalized via primary/foreign-key joins to be provided as input to ML algorithms. In this paper we study the implementation of popular nonlinear ML models and in particular independent Gaussian Mixture Models (IGMM) over normalized data.**

For the case of IGMM we propose algorithms taking the statistical properties of the Gaussians into account to construct mixture models, factorizing the computation. In that way we demonstrate that we can conduct the training of the models much faster compared to other applicable approaches, without any loss in accuracy.

We present the results of a thorough experimental evaluation, varying several parameters of the input relations involved and demonstrate that our proposals both for the case of IGMM yield drastic performance improvements which become increasingly higher as parameters of the underlying data vary, without any loss in accuracy.

I. Introduction

Machine learning (ML) applications in enterprise settings are increasingly becoming mission critical. As a result numerous projects both in industry and academia integrate ML techniques in RDBMS, Spark and numerous other production systems [8], [4]. ML algorithms however have been developed on the assumption that data is readily available on a single input source, with the right formats and encodings. As such effective integrations of ML technology in mission critical systems faces a data representation mismatch. Relational data are typically normalized [1] and as a result, data have to be denormalized via primary/foreign key joins and materialized as a single (temporary) relation to be provided as an input to the ML algorithms. As a result the learning process commences after joins have been performed on the relations involved.

Consider the example of an analyst modeling customer shopping trends. The analyst builds a model utilizing customer details: Customers(CustomerID, Address, ItemID) where the ItemID is a foreign key of the item number referring to a new table that stores the items sold by the company: Items(ItemID, Price, Size, Colour, Category, ...). A join is required between the two tables because some of the information in the Items table, e.g. Price, Size and Colour are essential features of a predictive model on buying patterns. After the join, the table is materialized as a temporary table that has to be provided as an input to the various ML algorithms utilized for predictive analysis. Examples like this abound in data analytics; in a video streaming company building recommendation models one has to join user viewing history with video information, at a banking application, building fraud detection models, or conducting soft customer segmentation, requires a join of customer purchasing/spending records with merchant data, in review sites one has to associate via a join the reviewer metadata with their reviews for user modeling applications, etc.

Kumar et. al., [5], recognized these issues and proposed specific algorithms to build generalized linear models and execute various linear algebra operations by pushing ML computations through joins to base tables. That way to the extent possible for the specific class of ML models considered, they demonstrated performance advantages in certain scenarios. To the best of our knowledge with the exception of [9] and their work on Markov Chain Monte Carlo to date there has been no other work considering classes of non-linear models in this context.

Gaussian Mixture Models (GMM) abound in various modeling tasks; they are an established method to model complex data spaces with diverse multidimensional characteristics. In addition GMM are prevalent in financial analysis, quantitative finance, astronomy [6] as well as banking applications especially dealing with the returns of asset classes [7].

In this paper, for the case of GMM we focus on Independent Gaussian Mixture Models (IGMM) and compare different applicable algorithms to construct such models over normalized data. After introducing several applicable approaches, we propose algorithms pushing the computation through join for different types of Gaussians involved depending on their statistical properties, while being able to deliver the correct final model. Various trade-offs among the algorithms are analyzed.

II. Background and Preliminaries

In this section we will present material and notation necessary for the remainder of the paper.

A. *Gaussian Mixture Models (GMM)*

Assume we are given N training data points $\mathbf{x}^{(n)}, 1 \leq n \leq N$ of dimensionality d. A Gaussian Mixture Model (GMM) is a density model comprising a fixed number of Gaussian distributions used for data clustering. The distribution of a mixture of K Gaussian components is: $p(\mathbf{x}^n) =$

2375-026X/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDE.2019.00145

$\sum_{k=1}^{K} \pi_k N(\mathbf{x}^{(n)}|\mu_k, \Sigma_k)$ where π_k is the mixing parameter satisfying $\sum_{k=1}^{K} \pi_k = 1$ and

$$N(\mathbf{x}^{(n)}|\mu_k, \Sigma_k) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} e^{-\frac{1}{2}(\mathbf{x}-\mu_k)^T \Sigma^{-1}(\mathbf{x}-\mu_k)} \quad (1)$$

is the probability density function of the k^{th} Gaussian component of the mixture model.

There are a number of algorithms that can be applied for iteratively training GMM; however the Expectation Maximization Algorithm (EM) [3] is the most widely used. EM is an iterative method to identify the maximum likelihood when the model contains an unobserved latent variable. The algorithm iteratively converges to a local minimum. The EM algorithm starts with some initial estimate of the parameters and then iteratively updates the parameters until convergence. Each iteration consists of one E step and one M step.

In the E step, we fix the mean and the variance of the Gaussian distributions and re-estimate the mixing coefficients π_j for all values of k and n.

$$\gamma_k^{(n)} = p(z^{(n)} = k|\mathbf{x}) = \frac{\pi_k N(\mathbf{x}^{(n)}|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j N(\mathbf{x}^{(n)}|\mu_j, \Sigma_j)} \quad (2)$$

where $z^{(n)}$ is the hidden variable for every observation, $z \sim Categorical(\pi)$ where $\pi_k \geq 0$ and $\sum_{k=1}^{K} = 1$. Essentially $\gamma_k^{(n)}$ are our "soft" guesses for the values of $z^{(n)}$ at this step.

In the M step, we re-estimate the parameters given the current responsibilities, for all values of k and n: $\mu_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_k^{(n)} \mathbf{x}^{(n)}$
$\Sigma_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_k^{(n)} (\mathbf{x}^{(n)} - \mu_k)(\mathbf{x}^{(n)} - \mu_k)^T$
$\pi_k = \frac{N_k}{N}$ with $N_k = \sum_{n=1}^{N} \gamma_k^{(n)}$ We iteratively run E step and M step to train and update the model until convergence criteria are met. One of the commonly used criteria for checking convergence is by calculating the log likelihood: $\ln p(\mathbf{x}|\pi, \mu, \Sigma) = \sum_{n=1}^{N} ln(\sum_{k=1}^{K} \pi_k N(\mathbf{x}^{(n)}|\mu_k, \Sigma_k))$. We can stop when the log likelihood between two iterations is less than some preset threshold.

A special case of Gaussian Mixture Models that's widely used in various machine learning applications [2], is the Independent Gaussian Mixture Model. In this specific GMM an underlying assumption is that the features are statistically independent. As a result of the independent assumption in the distribution of the mixture of K Gaussian components the covariance matrix Σ_k becomes a diagonal matrix with only nonzero entries on the diagonal. Based on different studies [2] independent GMMs provide solid modeling accuracy even in cases where features are not completely statistically independent.

III. Problem description

We now introduce formally the problems we focus in this paper. Assume relation **S** with n_S tuples, and relation **R** with n_R tuples. For our learning problems, the feature vectors are split across **S** and **R**, with d_S features in $\mathbf{X}_S$ and $d_R = d - d_S$ features in $\mathbf{X}_R$.

Given two relations **S** (SID, $\mathbf{X}_S$, FK) and **R** (RID, $\mathbf{X}_R$) with a key-foreign key relationship (**S**.FK refers to **R**.RID), where $\mathbf{X}_S$ and $\mathbf{X}_R$ are feature vectors, learn a mixture of Gaussians over the result of the projected equi-join **T**(SID, [$\mathbf{X}_S$ $\mathbf{X}_R$]) $\leftarrow \pi$ (**R** $\bowtie_{RID=FK}$ **S**) such that the feature vector of a tuple in **T** is the concatenation of the feature vectors from the joining tuples of **S** and **R**. For the case of NN relation S has an additional attribute Y which is the target for learning purposes (in this case the projected schema becomes **T**(SID, Y, [$\mathbf{X}_S$, $\mathbf{X}_R$])). Table I summarizes the notation used in this paper.

Symbol	Meaning
R	Relation
S	Relation
K	number of Gaussians
T	Join Result Table
Y	target of the NN data
n_R	Number of rows in **R**
n_S	Number of rows in **S**
n	Number of rows in table **T** ($n = n_S$)
d_R	Number of features in **R**
d_S	Number of features in **S**
d	Number of features in total ($d = d_R + d_S$)
$\mathbf{x}^{(n)}$	n^{th} data point in a relation **T** (**R** join **S**)
$\mathbf{x_R}^{(n)}$	n^{th} data point in a relation **R**
$\mathbf{x_S}^{(n)}$	n^{th} data point in a relation **S**
$\mu[a:b]$	slice the vector μ from index a to b

TABLE I: Notation used in the paper

Table T which is the result of the join of **R,S** may introduce redundancy back to the data representation which normalization removed in the first place. Such redundancy introduces a lot of repeated computations during model building, which we could potential save if we manage to operate on **R,S** directly to the extend possible. In addition depending on the redundancy introduced trade-offs may exist which we will explore in our ensuing discussion.

For purposes of exposition we assume that the required joins execute in a block nested loops fashion. Our algorithms and proposals naturally generalize and are equally applicable when other types of joins are adopted such as partitioned hash joins for example.

IV. Independent GMM (IGMM)

Independent Gaussian Mixture Models (IGMM) are a special kind of GMMs used in practice [2]. In IGMMs the covariance matrix is always diagonal because IGMMs assume independence among the underlying Gaussian distributions. For IGMMs we propose a way to compute both the E step and M step over normalized data.

A. Baseline Approaches

$M - IGMM$ is s baseline approach for computing IGMM. It computes a join of the relations involved, materializes the join result and executes the EM algorithm on the output. The algorithm is summarized as Algorithm 1.

In a similar fashion, $S - IGMM$ computes a join of the relations involved on the fly without materializing the join result and executes the EM algorithm on the output. This method saves I/O in addition to being storage efficient, since we do not need to write the materialized table on the disk. This algorithm is essentially the same as Algorithm 1, however, it does not perform step 1 in Algorithm 1 which is materializing the table in the database. In addition in steps 5, 10, 16, instead

Algorithm 1 Algorithm (M-IGMM)

1: Apply join **R** and **S** and materialize the table **T** after join in the database
2: **repeat**
3: **E Step:**
4: **for** i $\leq$ number of batches **do**
5: Read batch i of **T** into memory
6: Update responsibility of data points in this batch $\gamma_k^{(n)} \leftarrow \frac{\pi_k N(\mathbf{x}^{(n)}|\mu_k,\Sigma_k)}{\sum_{j=1}^{K}\pi_j N(\mathbf{x}^{(n)}|\mu_j,\Sigma_j)}$ $\forall n \in$ batch i, $\forall k$
7: **end for**
8: **M Step:**
9: **for** i $\leq$ number of batches **do**
10: Read batch i of **T** into memory
11: Add to the sum results from this batch
12: $Sum_{\mu_k} += \sum_{n=1}^{|batch\ size\ i|} \gamma_k^{(n)}\mathbf{x}^{(n)}$
13: **end for**
14: Update $\mu_k \leftarrow \frac{1}{N_k} Sum_{\mu_k}$
15: **for** i $\leq$ number of batches **do**
16: Read batch i of **T** into memory
17: Add to the sum results from this batch
18: $Sum_{\Sigma_k} += \sum_{n=1}^{|batch\ size\ i|} \gamma_k^{(n)}||\mathbf{x}^{(n)} - \mu_k||^2$
19:
20: **end for**
21: Update $\Sigma_k \leftarrow \frac{1}{N_k} Sum_{\Sigma_k}$
22: Update $\pi_k \leftarrow \frac{N_k}{N}$ with $N_k \leftarrow \sum_{n=1}^{N}\gamma_k^{(n)}$
23: **until** Convergence

of reading data from table **T**, we perform a join of relation **R** and **S** and read directly from the join as it is computed on the fly. This is accomplished by reading **R** (in batches) and retrieving from **S** data corresponding to keys form **R**.

B. Algorithm F-IGMM

The underlying idea in algorithm $F-IGMM$ is to factorize the execution. In $F-IGMM$ we compute the E step and M step without materializing the full join of **R** and **S** in the database. Instead we push the computation closer to **S** and **R** joining in a probe based manner when needed (using keys in **R** to probe table **S**).

The algorithm is derived from Algorithm 1 as follows. $F-IGMM$ does not execute line 1 in algorithm 1 since we do not need the table after the join to complete the calculations required. Moreover, for lines 5, 10, 16, we are not reading data from the joined table **T** any more, we directly read data from table **R** and **S**. However instead of joining on the fly (as in the case of $S-IGMM$) we factorize the computation of $\gamma_k^{(n)}, \mu_k, \Sigma_k$ over the i-th batch of **R** and the part of **S** retrieved using the foreign keys of the i-th batch of **R** to probe **S**. Figure **??**(c) depicts the basic idea. The specifics of the factorization are outlined below.

1) E step: The E step involves calculating the responsibilities given current parameters, where we need to calculate the probability of each data point given the mean and variance of the Gaussian distribution as per Equations 1,2. In the calculation of Equation 1, in $N(\mathbf{x}^{(n)}|\mu_k,\Sigma_k)$ the part $\frac{1}{\sqrt{(2\pi)^d|\Sigma|}}$ is easy to calculate since data from the underlying relations is not directly involved in the calculation. When Σ is computed we can compute this part numerically. While in the calculation of $e^{-\frac{1}{2}(\mathbf{x}-\mu_j)^T\Sigma^{-1}(\mathbf{x}-\mu_j)}$, data from both tables **S** and **R** is required when computing $(\mathbf{x}^{(n)}-\mu_k)^T\Sigma_k^{-1}(\mathbf{x}^{(n)}-\mu_k)$. For the special case of IGMM, the covariance matrix entries $\sigma_{i,j}=0$ if $i \neq j$. As a result, we decompose the calculation as follows:

$(\mathbf{x}^{(n)}-\mu_k)^T\Sigma_k^{-1}(\mathbf{x}^{(n)}-\mu_k)$ =

$$[x_1^{(n)}-\mu_1, x_2^{(n)}-\mu_2, ..., x_d^{(n)}-\mu_d]^T$$

$$\times$$

$$\begin{bmatrix} \sigma_{1,1}^{-1} & 0 & \cdots & 0 \\ 0 & \sigma_{2,2}^{-1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_{m,m}^{-1} \end{bmatrix}$$

$$\times$$

$$[x_1^{(n)}-\mu_1, x_2^{(n)}-\mu_2, ..., x_d^{(n)}-\mu_d]$$

=

$$\begin{bmatrix} (x_1^{(n)}-\mu_1)^2\sigma_{1,1}^{-1} & 0 & \cdots & 0 \\ 0 & (x_2^{(n)}-\mu_2)^2\sigma_{2,2}^{-1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & (x_m^{(n)}-\mu_m)^2\sigma_{m,m}^{-1} \end{bmatrix}$$

where $\mu_k = \left[\mu_1, \mu_2, \cdots, \mu_d\right]$ is updated in algorithm 1 line 14. In order to calculate $(\mathbf{x}^{(n)}-\mu_k)^T\Sigma_k^{-1}(\mathbf{x}^{(n)}-\mu_k)$ we first compute $||x_i^{(n)}-\mu_i||^2\sigma_{i,i}^{-1}$ for $1 \leq i \leq d_S$ and store it in memory as matrix **UL** (upper left matrix). Then we calculate $||x_i^{(n)}-\mu_i||^2\sigma_{i,i}^{-1}$ for $d_S+1 \leq i \leq d$ and store it in memory as matrix **LR** (lower right matrix). Finally we combine (stitch) matrix **UL** and matrix **LR** in memory to get the full matrix of $(\mathbf{x}^{(n)}-\mu_k)^T\Sigma_k^{-1}(\mathbf{x}^{(n)}-\mu_k)$. Note that **UL** is a $d_S \times d_S$ matrix and **LR** is a $d_R \times d_R$ matrix and their combination yields a $d \times d$ matrix.

2) M step: Updating $\pi_k = \frac{N_k}{N}$ with $N_k = \sum_{n=1}^{N}\gamma_k^{(n)}$ does not involve reading data from tables **R** and **S** so we will execute the update exactly as in algorithm 1.

To update equation $\mu_k = \frac{1}{N_k}\sum_{n=1}^{N}\gamma_k^{(n)}\mathbf{x}^{(n)}$ we note that μ_k is a $1 \times d$ vector with the first d_S entries originating from **S** and the remaining d_R originating from **R**.

$$\mu_k = \begin{bmatrix} \frac{1}{N_k}\sum_{n=1}^{N}\gamma_k^{(n)}\mathbf{x}^{(n)} \\ {\scriptstyle 1\times d} \end{bmatrix} \tag{3}$$

$$= \begin{bmatrix} \frac{1}{N_k}\sum_{n=1}^{N}\gamma_k^{(n)}\mathbf{x}_S^{(n)} & \frac{1}{N_k}\sum_{n=1}^{N}\gamma_k^{(n)}\mathbf{x}_R^{(n)} \\ {\scriptstyle 1\times d_S} & {\scriptstyle 1\times d_R} \end{bmatrix} \tag{4}$$

We calculate $\frac{1}{N_k}\sum_{n=1}^{N}\gamma_k^{(n)}\mathbf{x}_S^{(n)}$ using the data from table **S** and calculate $\frac{1}{N_k}\sum_{n=1}^{N}\gamma_k^{(n)}\mathbf{x}_R^{(n)}$ using the data from table **R**, then concatenate the two vectors to obtain μ_k

To update equation $\sigma_k = \frac{1}{N_k}\sum_{n=1}^{N}\gamma_k^{(n)}||\mathbf{x}^{(n)}-\mu_k||^2$ we observe that σ_k represents the diagonal entries of the full covariance matrix Σ_k since all entries other than the diagonal are zero. Thus, since σ_k is the diagonal entry, it is a $1\times d$ vector with the first d_S entires coming from **S** and the remaining d_R coming from **R**:

$$\sigma_k = \begin{bmatrix} \frac{1}{N_k}\sum_{n=1}^{N}\gamma_k^{(n)}||\mathbf{x}^{(n)}-\mu_k||^2 \\ {\scriptstyle 1\times d} \end{bmatrix} \tag{5}$$

$$= \begin{bmatrix} \sigma_{k-d_S} & \sigma_{k-d_R} \\ {\scriptstyle 1\times d_S} & {\scriptstyle 1\times d_R} \end{bmatrix} \tag{6}$$

Experiment	n_S	n_R	d_S	d_R	Clusters
Vary d_R	10^6 and 5×10^6	1000	5	Varied	5
Vary rr	Varied	1000	5	5 and 15	5
Vary Clusters	10^6	1000	5	15	Varied

TABLE II: Synthetic Data dimensions for $IGMM$

where

$$\sigma_{k-d_S} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_k^{(n)} ||\mathbf{x}_S^{(n)} - \mu_k[1 : d_S]||^2 \quad (7)$$

and

$$\sigma_{k-d_R} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_k^{(n)} ||\mathbf{x}_R^{(n)} - \mu_k[d_S + 1 : d]||^2 \quad (8)$$

σ_{k-d_S} is a $1 \times d_S$ matrix calculated using data from table **S** and σ_{k-d_R} is a $1 \times d_R$ matrix calculated using data from table **R**. We calculate $\frac{1}{N_k} \sum_{n=1}^{N} \gamma_k^{(n)} ||\mathbf{S}^{(n)} - \mu_k[1 : d_S]||^2$ using the data from table **S** and calculate $\frac{1}{N_k} \sum_{n=1}^{N} \gamma_k^{(n)} ||\mathbf{R}^{(n)} - \mu_k[d_S + 1 : d]||^2$ using the data from table **R**, then combine the two vectors to get σ_k.

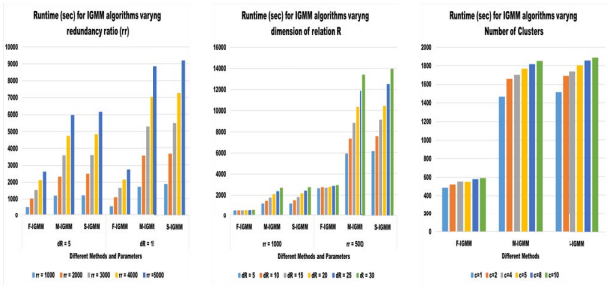
V. Experiments

In this section we present a detailed experimental evaluation of all the algorithms presented comparing their performance. We compare the runtime performance of $M - IGMM$, $S - IGMM$, $F - IGMM$ for the case of $IGMM$. There are two main parameters of interest in the underlying relations that essentially quantify the impact of normalization in terms of eliminating redundancy. These are d_R (the dimension of R) and the redundancy ratio ($rr = \frac{n_S}{n_R}$). We vary these two parameters controlling the amount of redundancy that the join introduces.

In order to be able to control the parameters of interest and vary them in a controlled way to observe trends, we utilize synthetic data sets. We generate synthetic datasets for primary-key/foreign-key joins with a wide range of attributes in the relations involved. The parameters varied are shown in Table II for $IGMM$ experiments. We generate synthetic data sampling from multiple Gaussian Distributions and add random noise in accordance to previous work [5].

All experiments were run on a cluster of machines with 16 Intel Xeon E5630 2.53 GHz cores, 96 GB RAM and 338 GB disk with CentOS 6.2. Our code is implemented in python 2.7.13 using numpy for all the matrix calculations and use psycopg2 to read and write data from PostgreSQL 9.6.

Figure 1 presents the results for the case of the IGMM algorithms varying redundancy ratio (rr) in Figure 1(a), varying the dimension of relation R (d_R) in Figure 1(b) and the number of clusters (c) in the gaussian mixture in Figure 1(c). In Figure 1(a) it is evident that as rr increases the benefits of the proposed $F - IGMM$ become increasingly larger. For $d_R = 5$ the proposed algorithm is 2.2 times faster than the other applicable approaches, which becomes 3 times faster as $d_r = 15$. The trend will persist becoming increasingly larger as d_R increases. Figure 1(b) presents the results of the same experiment varying d_R. It is evident that as d_R increases, the proposed algorithm becomes two to five times



(a) Varying rr (b) Varying d_R (c) Varying c

Fig. 1: Performance Results for IGMM algorithms varying parameters of interest

faster than the other approaches; the performance benefit will keep on increasing as we increase d_R. These trends persist as we increase rr. Finally Figure 1(c) presents the results for increasing the number of clusters (fixed rr and d_R).For these values as we vary c the proposed approach is three times faster. This benefit will increase as we vary rr, d_R. Results for real data sets are omitted due to space limitations; depending on the characteristics of the data sets they can be much larger performance benefits for real data sets than those presented.

VI. Conclusions

We proposed a set of algorithms to execute Independent Gaussian Mixture Models over normalized databases. In the future it is natural to generalize our techniques to Neural Networks, General Mixture of Gaussians and other popular deep architectures, also addressing multi-way joins.

References

[1] C. Beeri, P. A. Bernstein, and N. Goodman. A sophisticate's introduction to database normalization theory. In *Proceedings of the Fourth International Conference on Very Large Data Bases - Volume 4*, VLDB '78, pages 113–124. VLDB Endowment, 1978.

[2] C. M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007.

[3] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[4] A. Kumar, M. Boehm, and J. Yang. Data management in machine learning: Challenges, techniques, and systems. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 1717–1722, New York, NY, USA, 2017. ACM.

[5] A. Kumar, J. F. Naughton, and J. M. Patel. Learning generalized linear models over normalized data. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 1969–1984, 2015.

[6] K. J. Lee, L. Guillemot, Y. L. Yue, M. Kramer, and D. J. Champion. Application of the gaussian mixture model in pulsar astronomy - pulsar classification and candidates ranking for the fermi 2fgl catalogue. *Monthly Notices of the Royal Astronomical Society*, 424(4):2832–2840, 2012.

[7] K. P. Murphy. *Machine learning - a probabilistic perspective*. Adaptive computation and machine learning series. MIT Press, 2012.

[8] N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich. Data management challenges in production machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 1723–1726, New York, NY, USA, 2017. ACM.

[9] S. Rendle. Scaling factorization machines to relational data. In *Proceedings of the VLDB Endowment*, volume 6, pages 337–348. VLDB Endowment, 2013.