Evaluating KNN, LDA and QDA Classification for embedded online Feature Fusion

Andreas Starzacher and Bernhard Rinner
Institute of Networked and Embedded Systems, Klagenfurt University, Austria
Lakeside Park B02b, 9020 Klagenfurt, [andreas.starzacher, bernhard.rinner]@uni-klu.ac.at

Abstract

In this paper we evaluate k-nearest neighbor (KNN), linear and quadratic discriminant analysis (LDA and QDA, respectively) for embedded, online feature fusion which poses strong limitations on computing resources and timing. These algorithms are implemented on our multisensor data fusion (MSDF) architecture and are applied to traffic monitoring, i.e., classifying vehicles using distributed image, acoustic and laser sensors.

We performed several tests of the algorithms on our embedded platform and evaluated CPU performance and memory consumption for training as well as classification. The results obtained are very promising for further use, especially of LDA and QDA for embedded online fusion at feature-level.

Keywords: multisensor data fusion, embedded system, traffic monitoring

1. Introduction

Multisensor data fusion (MSDF) is a well-known technique to combine information originating from multiple homogeneous and heterogeneous sensors. The main benefit of MSDF is to achieve significant advantages over single source data, i.e., to improve robustness and confidence, to extend spatial and temporal coverage as well as to reduce ambiguity and uncertainty of the processed sensor readings. There is a huge number of different fusion algorithms known in literature ranging from estimation methods such as the famous Kalman Filter, to all kinds of classification and statistical inference methods such as Bayesian statistics and Dempster-Shafer theory of evidence [1], [2].

MSDF is typically performed at three different levels of abstraction [3]. *Raw-data fusion* is used to correlate data from sensors which measure the same physical parameters. *Feature-based fusion* combines features that have been extracted from the raw data of the individual sensors. Finally, *decision fusion* integrates preliminary results derived from the individual sensor data. Fusion at the top level provides an assessment of the observed scene such as delivering the identity of an observed object. Faceli [1] presents a related categorization into complementary, competitive and cooperative fusion.

In this paper we focus on *embedded, online* MSDF where sensory data is processed on dedicated processing nodes in a streaming manner. The major challenges here are the limited resources on the embedded processing node (computing power, memory capacity etc.) and strict timing requirements on the fusion methods in order to keep pace with the steady data stream. Embedded, online MSDF is an important enabling technology for many applications such as environmental monitoring, smart environments and pervasive computing. Our target application is traffic monitoring (e.g. [4]) where we want to reliably derive important traffic parameters, such as lane occupancy, speed, vehicle classification and tracking, by fusing data from heterogeneous, distributed sensors. We currently exploit image, acoustic and laser sensors. Fusion is performed in a distributed, embedded, multi-tier architecture [5].

This paper evaluates and compares the algorithms *k-nearest neighbor (KNN)*, *linear* and *quadratic discriminant analysis (LDA, QDA)* for embedded, online MSDF. These algorithms are well-known classification algorithms, however they can also be applied to feature-level fusion. Our evaluation especially focuses on the limited timing and resource capabilities of the embedded platforms. This extends our previous work on embedded MSDF [6], [7] where we applied support vector machines (SVM) for embedded MSDF.

The remainder of the paper is structured as follows: Sections 2 and 3 briefly present our sensor network structure and three-layered MSDF architecture approach. Section 4 describes the selected algorithms for embedded feature fusion. Experimental results and evaluations of KNN, LDA and QDA on different simulated training sets on our embedded platform are presented in section 5. Section 6 concludes the paper with a summary and an outlook.

2. Sensor Network

The overall structure of our sensor network is shown in Fig. 1. It consists of the following components: multiple sensor nodes (SN_i), a single center node (CE) and a "fusion backbone to center" node (FBC) which is actually a dedicated SN sending fused data (decision) to the CE. Thus, the FBC performs the final fusion processing, i.e., deriving the final decision about a specific task which may be object detection and identification.

Each SN, CE and FBC has a couple of attributes and methods defining its state and functionality. Attributes are for example an ID for unique identification, internal description and timestamps (logging actions of a SN). Methods that each sensor node has implemented are for example: send, receive, fusion methods and methods interfacing external devices (e.g., stop lights in traffic monitoring environments).

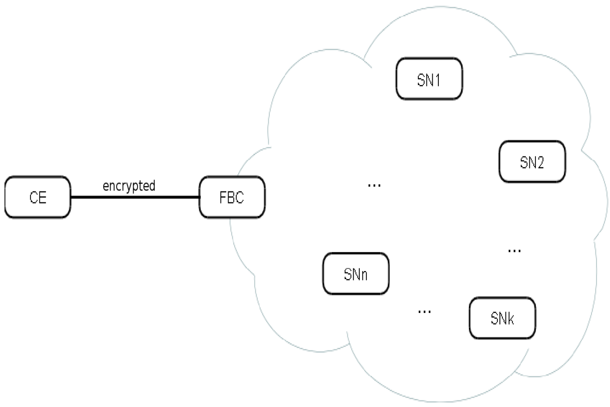

Fig. 1: Overall sensor network structure

The SN in our network can be organized in clusters which perform different fusion tasks on specific regions of the overall supervised area such as monitoring an intersection area. Currently, the assignment of SN to clusters is done a-priori. The clusters in the sensor network are labeled as PFC_i in the second layer of our fusion architecture (more details are given in section 3-B).

3. Fusion Architecture

Our fusion architecture consists of three layers as depicted in Fig. 2. The main reason for choosing a layered architecture approach is to abstract and encapsulate the numerous processing steps into autonomous processing units. Each layer is structured in a task-specific way and responsible for a predefined processing task, hence contributing individually to the whole fusion process.

A. Layer 1

Layer 1 performs two main tasks: (1) capturing raw data of the different sensors and (2) performing spatial and temporal alignment of the captured data. We call this layer the hardware layer, because it has to deal with interfacing the underlying hardware with the homo-/heterogeneous sensory devices.

B. Layer 2

Layer 2 corresponds to the inter-cluster fusion tasks within the sensor network. Furthermore, this layer may be seen as the first fusion layer in our architecture that performs competitive (redundant) and complementary integration within the PFC_i. *Competitive integration (fusion)* means combining independent sensor data that represent measurements of the same property [8]. As a result, the objective of competitive integration is mainly to reduce uncertainty and resolve potential conflicts concerning the sensor readings. However, if sensors measure different features of a certain object of interest, it is called *complementary integration*. This kind of integration creates a more complete model in terms of enriching knowledge about the specific universe of discourse [9] (e.g., front- and backshot images of a certain vehicle) [9]. In this layer

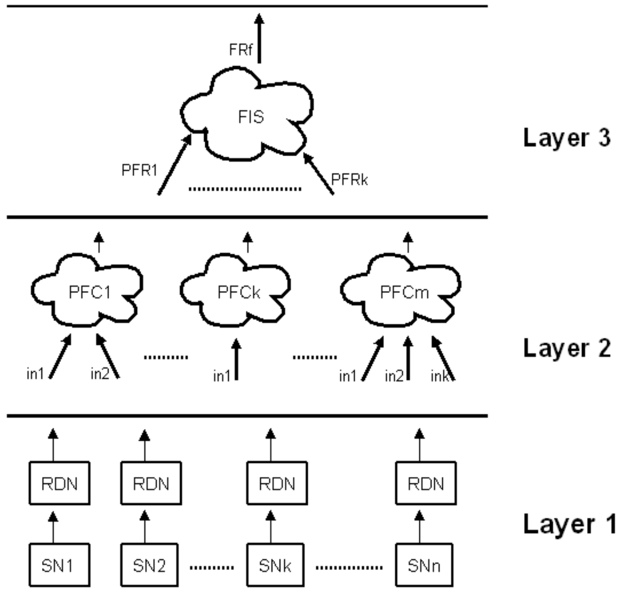

Fig. 2: Three-layered fusion architecture. Legend: SN_i (sensor node), RDN_i (raw data acquisition and normalization), in_i (observed data from layer 1), PFC_i (partial fusion clusters), PFR_i (partial fusion results), FIS (fuzzy inference system), FR_f (final decision). Explanation of the different components given in section 3.

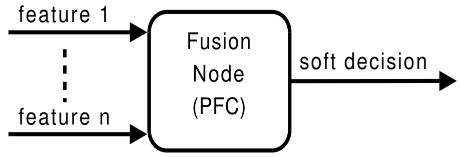

Fig. 3: Fusion process. Input features originate from homo-/heterogeneous sensors and are fused appropriately forming a soft decision.

KNN, LDA and QDA may be used for heterogeneous feature fusion. Intra-cluster fusion results into partial fusion results (PFR_i) representing a preliminary decision which we call a soft decision (see Fig. 3). The PFR_i of each cluster is sent to the third layer for final (hard) decision-fusion.

It is possible that each cluster consists only of a single SN. In this case no intra-cluster fusion takes place and layer 2 can be omitted.

C. Layer 3

Within layer 3 (second fusion layer) the PFR_i are fused by means of *cooperative integration* at decision level resulting in a final fusion result FR_f (hard decision). Well-known methods to perform high-level fusion are fuzzy logic, statistical methods such as Bayesian inference or (its generalization) Dempster-Shafer theory. In our case we may apply fuzzy logic algorithms to fuse all PFR_i. *Cooperative integration* is the process of combining autonomous measurements to obtain additional information about the whole universe of discourse. Briefly, this means that extra information is achieved from several sensors that would not have been available with each single sensor alone. An example of this type of integration may be the calculation of disparity maps using stereo vision

TABLE 1: OVERVIEW OF PROS AND CONS OF KNN, LDA AND QDA

Method	Pros	Cons
KNN	notable classification results no (re)training phase distance metrics error probability bounded	time consuming classification time memory utilization finding optimal k
LDA	linear decision boundary fast classification easy to implement	Gaussian assumptions training time complex matrix ops
QDA	quadratic decision boundary fast classification classification more accurate outperforms KNN and LDA	Gaussian assumptions training time complex matrix ops

[10]. The outcome of the third layer is the final decision (FR_f). That SN which is elected to act as the FBC sends the FR_f to the CE for further post-processing by responsible executives (e.g., law enforcement or accounting departments).

There are a lot of different fusion modeling approaches [11], [3], [12], [6]. The main objective of our MSDF architecture is to keep the overall structure simple and to develop autonomous layers (modules) which can be developed separately and substituted if required.

4. SELECTED ALGORITHMS

To perform intra-cluster fusion in layer 2, we decided to implement a non-weighted *k*-nearest neighbor algorithm with common majority vote as classification rule as well as linear and quadratic discriminant analysis approaches. Major advantages and disadvantages of these algorithms are listed in Table 1. We have chosen these basic algorithms in statistical classification to extend our previous work on embedded MSDF [6], [7]. Furthermore, KNN, LDA and QDA are well-known and show promising results concerning classification rates (e.g., low false positives rate), but also concerning CPU performance and memory consumption for the training and classification step on our embedded platform.

A. K-Nearest Neighbor

K-nearest neighbor classification is a very simple and well-known technique that has been extensively studied. In the seminal work of Cover et al. [13] the nearest neighbor decision rule is explained in detail. They stated that for any number of classes, the error probability of the nearest neighbor rule has twice the Bayes probability of error as its supremum. Hence, it may be generalized that half of the total information needed for classification purpose (in an infinite sample set) is contained in the nearest neighbor.

The performance of KNN is determined by two factors [14]. First, it is crucial to find an appropriate k which is a non-trivial problem. In general, large k's are less affected by noise, and class boundaries achieve smoother shapes. Adopting an optimal k from one application to a different one is nearly infeasible. The second factor that influences the performance is the distance metric. There are a lot of approaches found in literature in order to enhance the performance of KNN algorithms, e.g., [15], [16], [17], [18].

KNN gives promising classification rates when applied to large data sets. For online fusion KNN may only be applied to relatively small training sets due to the high computational effort in calculating the distances. Therefore, KNN is not applicable for critical realtime systems if huge training samples are involved. Whenever new data x has to be classified, all distances between x and the training data have to be calculated. As distance metric we have implemented Minkowski distance L_m (see eq. 1) which allows for flexible change of distance metrics (e.g., L_1 Manhattan and L_2 Euclidean distance) and Mahalanobis distance (eq. 2).

$$L_m(p,q) = (\sum_{i=1}^{n}(|p_i - q_i|)^m)^{\frac{1}{m}}, \tag{1}$$

$$M_k(p,q) = \sqrt{(p_i - q_i)^T \Sigma_k^{-1}(p_i - q_i)} \tag{2}$$

In eq. 1 and 2 n is the number of features, p, q are n-dimensional feature vectors and Σ^{-1} the inverse covariance matrix of class k.

To avoid KNN to be only applicable to rather small sets of training samples, we might modify our algorithm to a Voronoi-based KNN approach as proposed by Kolahdouzan and Shahabi [19]. In this case, a Voronoi diagram with respect to the training data available is calculated only once (training). Each data (generator) generates a Voronoi polygon. New data x is classified according to the class label of the generator which generated the Voronoi polygon in which x is located. In contrast to the common KNN approach Voronoi-based KNN has to perform retraining (re-calculating the Voronoi diagram) if new training samples are to be added to the existing training set.

B. Linear and Quadratic Discriminant Analysis

Linear discriminant analysis [20] may be used to reduce the dimensionality of data and for classification purposes. Many applications have taken advantage of LDA [21]. LDA projects data onto a lower-dimensional space. This space provides maximum class separability [22]. Features that are derived from LDA are linear combinations of the original ones.

In classical LDA the optimal projection of features is achieved as follows: The within-class distance is minimized and, contrary, the between-class distance is maximized. This obviously results in a maximum class separability. An important observation concerning two-class classification is that LDA is equivalent to linear regression. Thus, LDA can be formulated as a least squares problem [21]. Quadratic discriminant analysis is quite similar to LDA, but it allows for quadratic decision boundaries between classes. There are also lots of different variants of QDA such as the Bayesian quadratic discrimint analysis. Srivastava et al. [23] present theoretical as well as algorithmic contributions to Bayesian estimation for QDA and describe several variants of QDA.

We perform feature fusion simulations within the binary-class problem framework even though our algorithm implementations allow for multi-class classification as well. The estimated discriminant functions $\hat{d}_k(x)$ for LDA and QDA are given in equation 3 and 4, respectively. The assumptions for LDA and QDA hold that the training data follow a multivariate normal distribution. With LDA all classes are assumed to have the same covariance matrices, whereas with QDA the covariance matrices are assumed to be different for each class. Furthermore, it is claimed that prior probabilities of class-membership are known or can be estimated beforehand.

$$\hat{d}_k(x) = \log \hat{p}_k - \frac{1}{2}\hat{\mu}_k^T \hat{\Sigma}^{-1} \hat{\mu}_k + x^T \hat{\Sigma}^{-1} \hat{\mu}_k \qquad (3)$$

The discriminant functions for LDA are specified by equation 3 where $k = 1...\#classes$, x is the new feature vector that is to be classified, $\hat{p}_k$ the estimated prior probability of class k, $\hat{\mu}_k$ the estimated mean of class k and $\hat{\Sigma}^{-1}$ the estimated inverse pooled covariance matrix.

$$\hat{d}_k(x) = \log \hat{p}_k - \frac{1}{2}\log|\hat{\Sigma}_k| - \frac{1}{2}(x - \hat{\mu}_k)^T \hat{\Sigma}_k^{-1}(x - \hat{\mu}_k) \quad (4)$$

The discriminant functions for QDA are specified by equation 4 where k and x are as in eq. 3. $\hat{\Sigma}_k$ ($\hat{\Sigma}_k^{-1}$) is the estimated (inverse) covariance matrix of class k.

The classification rule as given in eq. 5 is the same for both LDA and QDA.

$$\hat{D}(x) = k^* :\Leftrightarrow k^* = \arg\max_k \hat{d}_k(x) \qquad (5)$$

The classification rule for LDA and QDA is very intuitive. The major computational effort though is the training phase, meaning the computation of the discriminant functions and their parameters (see section 5). Once the training phase is completed, new data x can be classified simply by solving the appropriate discriminant function for each class k and applying the classification rule (eq. 5).

In section 5 we will see that the computational bottleneck of the algorithms lies in the training phase which is the estimation of all the parameters needed for the discriminant functions. Concerning computational efforts, the classification process is independent of the size of the training set. Obviously, the classification task is performed using only the discriminant function values. Thus, if the training set is too large for being trained on a dedicated embedded device, the training phase may be performed off-line (e.g., on back office workstations). In our case we performed the training as well as the classification task on our embedded platform described in section 5-A.

5. Experiments

In this section we present and discuss several performance tests and results concerning the execution time of KNN, LDA and QDA for training and classification and resource capacity utilization of the embedded platform. Additionally, a comparison between these algorithms is accomplished and discussed as well.

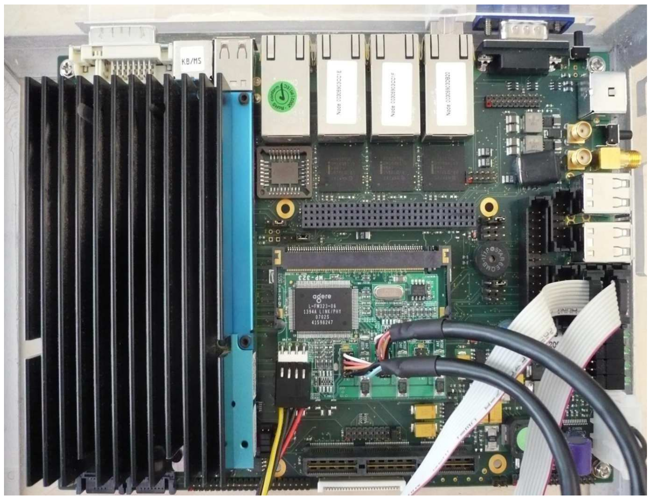

Fig. 4: High-performance embedded platform

Before we present our experimental results, we describe our embedded platform on which we performed all of our simulated tests (section 5-A).

A. High-performance embedded platform

As test and evaluation platform we use a MICROSPACE EBX (*MSEBX945*), an embedded computer board from *DigitalLogic AG* (see Fig. 4) serving as a multi-sensor data fusion platform. It has a compact EBX single-board construction (146mm $\times$ 203mm) with several interfaces such as RS-232, LAN 100MB, FireWire over MiniPCI and USB. The *SMX945-L7400* CPU module includes an Intel Core 2 Duo with 2 $\times$ 1500 MHz and a 667 MHz FSB. Besides interfaces, the MSEBX945 board has the following main characteristics: 2048MB DRAM, USB 2.0, RS232C, COM-Interface, 10/100BASE-T, 1GB-LAN PCIe, MiniPCI slot and PS/2 interface. The total power consumption is approximately between 12–15 W.

Currently, we successfully interfaced the following sensors: *Noptel CM3-30* (single-beam laser for distance measurements and altitude profile generation), acoustic sensor (for mono/stereo audio recordings), *Baumer FWX14-K08* camera (for taking single shots and continuous recording) and *ELV ST-2232* (an environmental sensor for recoring LUX, Celsius/Fahrenheit, dB). Basically, the sensor interface reflects the implementation of the first layer in our MSDF architecture.

B. Performance evaluations

Our test and evaluation focus lies on testing the execution time of KNN, LDA and QDA for training and classification, the scalability concerning number of training data and features used and the resource capacity utilization of the embedded platform during execution of the algorithms. The number of training samples and features are varied throughout the tests to evaluate which numbers are feasible for later online fusion on our embedded board. KNN is always performed with

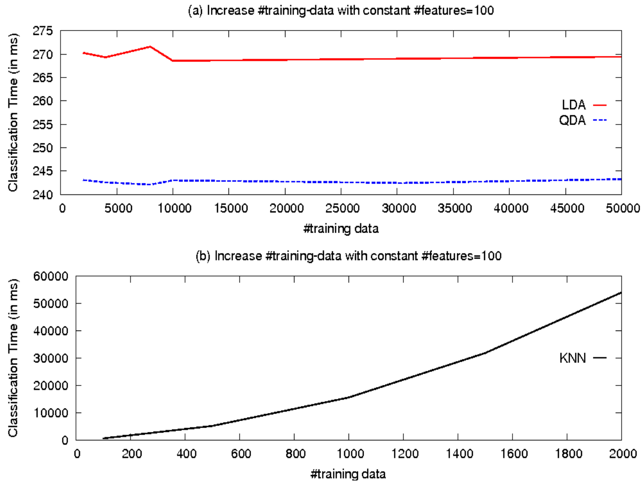

Fig. 5: Increasing number of training data with number of features set to 100 (constant)

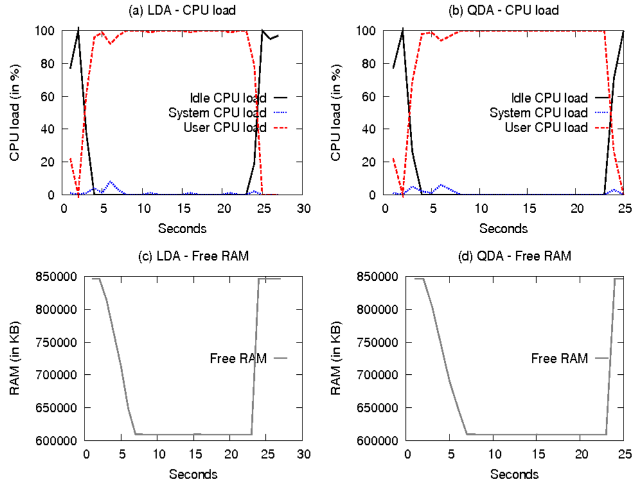

Fig. 6: LDA and QDA - CPU load and RAM usage (n_{ts} = 100000) where CPU and RAM parameters are recorded starting approx. $2s$ before and stopping approx. $2s$ after executing the algorithms (hence the "transitions")

Minkowski distance L_2, namely the Euclidean distance. Mahalanobis distance is too expensive and not feasible for online fusion on our platform. For testing resource capacity utilization we have chosen to analyze idle, system and user CPU load as well as free RAM available during classification. All our data is simulated by sampling from Gaussian distributions to fulfill one of the assumptions of LDA and QDA (see section 4-B).

TABLE 2: KNN STATISTICS

n_{ts}	t_{init}	$t_{\bar{c}}$	t_{var}	t_{ct}	t_{tot}
100	3.288	5.51487	0.0015	551.487	554.775
500	16.357	51.0812	11.0781	5108.12	5124.48
1000	30.263	155.216	1.5836	15521.6	15551.9
1500	45.13	318.344	58.0989	31834.4	31879.5
2000	63.383	540.49	599.161	54049	54112.4

First, we evaluated the initialization and classification performance of KNN, LDA and QDA by increasing the number of training data for each class from small to large values whereas the number of features is kept constant to 100 (scalability concerning various sizes of training data). The results are summarized in Tables 2, 3 and 4 where n_{ts} is the number of training samples available for each class (all classes have same n_{ts}), t_{init} represents the execution time for the training (initialization) phase (in case of KNN this is solely the time for generating test data), $t_{\bar{c}}$ gives the empirical mean execution time for classifying a single data, t_{var} is the execution time variance and t_{ct} is the total execution time for classifying 100 test data. Each test data is a vector comprising 100 features. The sum of t_{init} and t_{ct} equals t_{tot} (total execution time of training and classifying 100 new test data with training data of size n_{ts} for each class). Time is measured in milliseconds.

As expected, LDA and QDA outperform KNN only with respect to classification time (as KNN actually has no training). A graphical comparison of the classification time for each algorithm is given in Fig. 5 (see Table 2 with n_{ts} = 2000 and Table 3 and 4 with n_{ts} = 50000). As a first observation, KNN is not suitable for online fusion if the training data gets very large — even if the number of features is relatively small. For example, with $n_{ts} = 2000$, KNN spends $540.49ms$ to classify a single test data, whereas LDA only spends $2.70278ms$ and QDA $2.26568ms$, respectively. Additionally, LDA and QDA spend most of the time for training, i.e., the estimation of the parameters. Therefore, if online training is not required, we may perform training offline at a high-performance workstation. Fig. 6 (a) and (b) exemplifies the CPU load and memory utilization with discriminant analysis. The CPU load is very high, whereas the memory utilization is neglectable (as with KNN).

TABLE 3: LDA STATISTICS

n_{ts}	t_{init}	$t_{\bar{c}}$	t_{var}	t_{ct}	t_{tot}
2000	488.898	2.7028	0.0005	270.278	759.176
4000	913.976	2.6831	0.0004	268.304	1182.28
10000	2131.87	2.6856	0.0003	268.561	2400.43
50000	10420.3	2.6793	0.0004	267.931	10688.2
100000	21338.9	2.67792	0.00033	267.792	21606.7

TABLE 4: QDA STATISTICS

n_{ts}	t_{init}	$t_{\bar{c}}$	t_{var}	t_{ct}	t_{tot}
2000	414.945	2.2657	0.0007	226.567	641.513
4000	809.703	2.2407	0.00079	224.07	1033.77
10000	1991.07	2.4401	0.00059	244.01	2235.07
50000	10218.6	2.423	0.000779	242.3	10473.2
100000	20956	2.4618	0.00048	246.18	21202.2

Second, we focused on the evaluation of the scalability concerning the number of features. As $t_{\bar{c}}$ of KNN with large n_{ts} and lots of features is much greater than of LDA and QDA (with factors of several hundred ms), we discuss only LDA and QDA. We fixed n_{ts} to 1000 during execution and successively increased the number of features. Fig. 7 shows the classification time needed to classify new data consisting of different numbers of features (50, 100, 200 and 400). Even in case of test data consisting of 400 features, the mean

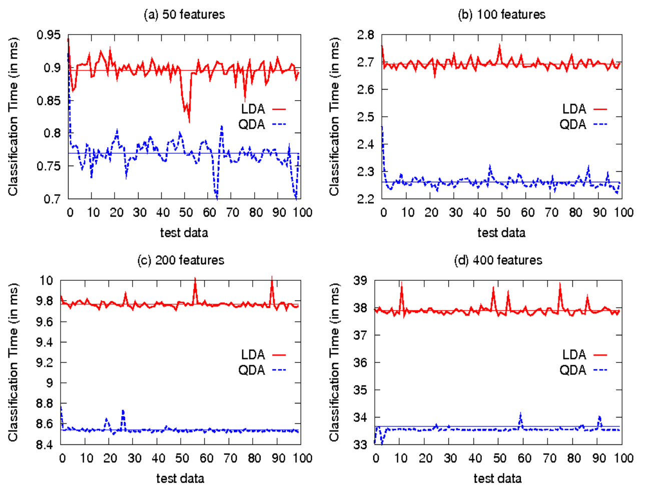

Fig. 7: Increasing number of features with datasets of 1000 entries (constant)

classification time spent is quite promising for applying LDA and QDA to online MSDF problems (LDA $t_{\bar{c}} = 37.9ms$ and QDA $t_{\bar{c}} = 33.7ms$). Additionally, it is worth mentioning that QDA slightly outperforms LDA in two aspects. First, the classification time $t_{\bar{c}}$ of QDA was always less than of LDA and second, the false positives rate is also less than with LDA (actually this is a matter of statistical classification theory and not discussed in this paper).

As a result, LDA and QDA may be used for further online feature fusion. This is because of the scalability concerning the number of training data and features, promising classification times and low complexity concerning implementation. Since KNN may only be used with relatively small n_{ts}, k and numbers of features, we will rather not use KNN for online feature fusion. This is because KNN needs to have large training data to achieve acceptable classification results and in a MSDF process, however, we must be able to handle huge datasets and large numbers of features.

6. Conclusion

The objective of this paper was to evaluate the feasibility of KNN, LDA and QDA for embedded online feature fusion. These evaluations extended our previous work on embedded MSDF where we applied SVMs. The algorithms are implemented on our MSDF architecture and applied to traffic monitoring. We described our architecture and selected feature fusion algorithms in detail. In order to evaluate the feasibility of using these algorithms for online feature fusion, we performed several tests on our embedded platform. Within these tests we evaluated CPU performance and memory consumption for training as well as classification. Additionally, we investigated the scalability concerning number of training data and features for each algorithm. The results obtained turned out to be very promising for further use of LDA and QDA for embedded online feature fusion.

Acknowledgment

This research is supported by the Austrian Research Promotion Agency under the FIT-IT[visual computing] grant 813399.

References

[1] K. Faceli, A.C.P.L.F. de Cavalho, and S.O. Rezende. Combining intelligent techniques for sensor fusion. *Applied Intelligence*, 20(3):199–213, 2004.

[2] D. Fasbender, V. Obsomer, J. Radoux, P. Bogaert, and P. Defourny. Bayesian data fusion: spatial and temporal applications. In *MultiTemp2007*, pages 001–006, Provinciehuis Leuven, Belgium, 2007.

[3] J. Llinas and D.L. Hall. An introduction to multi-sensor data fusion. In *ISCAS-2008*, pages 537–540, Monterey, CA, 1998.

[4] Andreas Klausner, Allan Tengg, and Bernhard Rinner. Vehicle Classification on Multi-Sensor Cameras using Feature- and Decision-Fusion. In *Proceedings of the ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC 2007)*, pages 67–74, Vienna, Austria, September 2007.

[5] Andreas Starzacher and Bernhard Rinner. An Embedded Multi-Sensor Data Fusion Framework for Enhancing Vision-based Traffic Monitoring. In *Proceedings of the ACM/IEEE Conference on Distributed Smart Cameras (PhD Forum)*, pages 400–401, Vienna, Austria, sep 2007.

[6] Andreas Klausner, Allan Tengg, and Bernhard Rinner. Distributed multilevel data fusion for networked embedded systems. *IEEE Journal on Selected Topics in Signal Processing*, 2(4):538–555, 2008.

[7] Andreas Klausner, Stefan Erb, and Bernhard Rinner. DSP Based Acoustic Vehicle Classification for Multi-Sensor Real-Time Traffic Surveillance. In *Proceedings of the 15th European Signal Processing Conference (EUSIPCO 2007)*, pages 1916–1920, Poznan, Poland, September 2007.

[8] W. Elmenreich. *Sensor Fusion in Time-Triggered Systems*. PhD thesis, Technische Universität Wien, Austria, 2003.

[9] H. Wu. *Sensor Data Fusion for Context-Aware Computing Using Dempster-Shafer Theory*. PhD thesis, Carnegie Mellon University, USA, 2003.

[10] H. Ruser and F.P. León. Informationsfusion - Eine Übersicht. *Technische Messen, Oldenburg*, 74(3):093–102, 2007.

[11] J. Esteban, A. Starr, R. Willetts, P. Hannah, and P. Bryanston-Cross. A review of data fusion models and architectures: towards engineering guidelines. 14(4):273–281, 2005.

[12] M. Bedworth and J. O'Brien. THEThe omnibus model: a new model of data fusion? *IEEE Aerospace and Electronic Systems Magazine*, 15:030–036, 2000.

[13] T.M. Cover and P.E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):021–027, 1967.

[14] M. Latourrette. Toward an ecplanatory similarity measure for nearest-neighbor classification. In *ECML-2000*, pages 238–245, London, UK, 2000.

[15] T. Hastie and R. Tibshirani. Discriminant adaptive nearest neighbor classification. *Pattern Analysis and Machine Intelligence*, 18(6):607–616, 1996.

[16] C. Domeniconi, J. Peng, and D. Gunopulos. Locally adaptive metric nearest-neighbor classification. *Pattern Analysis and Machine Intelligence*, 24(9):1281–1285, 2002.

[17] K.Q. Weinberger, J. Blitzer, and L.K. Saul. Distance metric learning for large margin nearest neighbor classification. In *NIPS-2005*, pages 1473–1480, Cambridge, Massachusetts, 2006.

[18] Y. Song, J. Huang, D. Zhou, H. Zha, and C. Lee Giles. IKNN: informative k-nearest neighbor pattern classification. In *PKDD-2007*, pages 248–264, Warsaw, Poland, 2007.

[19] M. Kolahdouzan and C. Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *VLDB-2004*, volume 30, pages 840–851, Toronto, Canada, 2004.

[20] R. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.

[21] J. Ye. Least squares linear discriminant analysis. In *ICML-2007*, volume 227, pages 1087–1093, Corvalis, Oregon, 2007.

[22] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*, volume 2. Wiley, 2001.

[23] S. Srivastava, M.R. Gupta, and B.A. Frigyik. Bayesian Quadratic Discriminant Analysis. *The Journal of Machine Learning Research*, 8:1277–1305, 2007.