Discriminant Analysis on a Stream of Features

Jan Motl[1][0000−0002−0388−3190] and Pavel Kordík[1][0000−0003−1433−0089]

Czech Technical University in Prague

Abstract. Online learning is a well-established problem in machine learning. But while online learning is commonly concerned with learning on a stream of samples, this article is concerned with learning on a stream on features. A modified quadratic discriminant analysis (QDA) is proposed because it is fast, capable of modeling feature interactions, and it can still return an exact solution. When a new feature is inserted into a training set, the proposed implementation of QDA showed a 1000-fold speed up to scikit-learn QDA. Fast learning on a stream of features provides a data scientist with timely feedback about the importance of new features during the feature engineering phase. In the production phase, it reduces the cost of updating a model when a new source of potentially useful features appears.

Keywords: Incremental learning · Online learning · Sequential learning · Stream learning

1 Introduction

A common issue in machine learning is the need to update machine learning models based on changing data. This issue can be simplified by assuming that new samples will be available over time. However, this article is concerned with the orthogonal problem — the fast updating of models with the arrival of new features (see Figure 1).

Interestingly, the authors of this article did not find any publicly available implementation of a classifier, which would satisfy following requirements:

1. it is faster to update the model with a new feature than to train the model from scratch,
2. is more accurate than naive Bayes, which can be straightforwardly updated to learning on a stream of features,
3. and is faster to train than logistic regression, random forest, and gradient boosted trees.

The proposed update of the quadratic discriminant analysis (QDA) to learning on a stream of features satisfies all these requirements.

1.1 Quadratic Discriminant Analysis

The authors extend QDA because it is a simple (but nontrivial) model that can be solved analytically. Furthermore, QDA is capable of modeling interactions

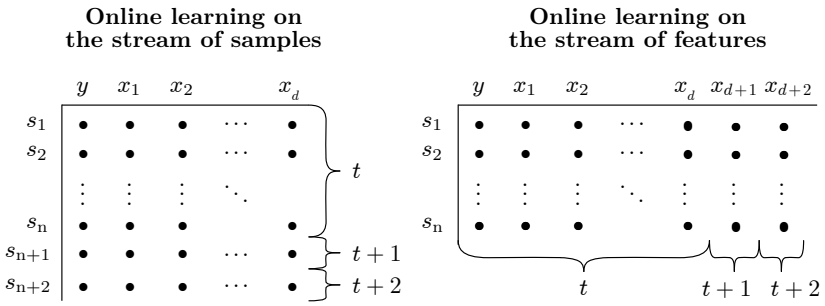

Fig. 1. Difference between learning from a stream of samples (left) and a stream of features (right). In both cases, we have n samples and d features at time t. However, at time $t + 1$, we either have one more sample (left) or one more feature (right).

between features without explicitly defining them in the data preprocessing step. This property differentiates QDA from linear discriminant analysis (LDA). Nevertheless, it is easy to reduce QDA into LDA by using the same covariance matrix estimate for each class. LDA is known to have a good tradeoff between accuracy and runtime [26], is more efficient than logistic regression [9], and can be used for semi-supervised learning (e.g., by shrinking the class conditional covariance matrix estimates toward the shared covariance matrix).

The paper is structured as follows. First, the related literature is discussed in Section 2. Then online QDA is proposed in Section 3. Empirical comparisons of the obtained accuracy and speed up are in Section 4. The paper closes with a discussion of the applications of the online QDA in Section 5 and the conclusion in Section 6.

2 Related work

Feature stream processing was introduced by [31], where it was used for feature selection. The current state-of-the-art algorithm in this field is online streaming feature selection (OSFS) [27]. OSFS works as a feature-selection filter, which evaluates incoming features. Only when a new feature is evaluated by OSFS to be relevant and non-redundant, the feature is passed to a conventional downstream model, which is retrained from scratch. However, the time to retrain the downstream model remains an unsolved bottleneck [29].

Examples of other feature selection methods that work on a stream of features are a scalable and accurate online approach for feature selection (SAOLA) [30], online stream feature selection method based on mutual information (OSFSMI) [23], online stream feature selection method with self-adaptive sliding window (OSFSASW) [28], geometric online adaption (GOA) [25], and streaming feature selection considering feature interaction (SFS-FI) [32].

Our approach is a departure from the feature selection mindset; it focuses on *updating an offline classifier into an online classifier*.

This update approach was already successfully applied on least squares support vector machine by [21], on a one-layer artificial neural network by [3], and on Bayes classifier by [19].

2.1 Unrelated work

The proposed implementation of QDA can be described as an *incremental algorithm*. However, this term can have several meanings, and we feel the need to explicitly state what our implementation is not, to avoid confusion.

A considerable amount of literature is associated with updating LDA when new samples arrive (see references in [8, Table 1]). However, this article deals with an orthogonal problem when new features arrive. In addition, the proposed QDA does not return an approximate solution (e.g., by iteratively approximating the first few eigenvectors [5]) but returns an *exact solution*.

3 QDA Algorithm

In the following text, matrices are denoted by capital letters, e.g., $Z \in \mathbb{R}^{n \times d}$, vectors by boldface characters, e.g., $\boldsymbol{z} \in \mathbb{R}^d$, and scalars by lowercase characters, e.g., $z \in \mathbb{R}$. n denotes the count of rows and d denotes the count of columns. Unless stated otherwise, vectors are column-vectors, i.e., $\boldsymbol{z}^\intercal \boldsymbol{z}$ is a scalar. Vectors and matrices are indexed with square brackets, e.g., $Z[i, j]$ for i^{th} row and j^{th} column, to unite the notion in mathematical equations and pseudocode. For brevity, broadcasting along the vertical is permitted, e.g., $Z - \boldsymbol{z} = Z - 1_n \boldsymbol{z}^\intercal$, where 1_n is a vector of ones.

A single data instance $\boldsymbol{s}$ can be scored with QDA using the following equation:

$$Z_k(\boldsymbol{s}) = -\frac{1}{2}(\boldsymbol{s} - \mu_k)^\intercal \Sigma_k^{-1}(\boldsymbol{s} - \mu_k) - \frac{1}{2}\ln|\Sigma_k| + \ln p_k. \tag{1}$$

where μ_k is the estimated mean value of instances in class k, Σ_k^{-1} is the inverse of the covariance matrix for the class k, $|\Sigma_k|$ is the determinant of the covariance matrix Σ_k, and p_k is the prior probability of the class k. Instance $\boldsymbol{s}$ is then classified into the class with the highest Z value.

3.1 Covariance

The class-conditional sample covariance matrix $\Sigma_{t,k}$ at time t is estimated from the class-conditional data matrix $X_{t,k}$ with

$$\Sigma_{t,k} = \frac{1}{n_k - 1}(X_{t,k} - \mu_{t,k})(X_{t,k} - \mu_{t,k})^\intercal, \tag{2}$$

where $\mu_{t,k}$ is a class-conditional vector of the feature means, and n_k is the class-conditional sample count. When a new feature $\boldsymbol{x}_{t+1}$ is appended into the data

matrix, X_t becomes X_{t+1}. The readily available $\Sigma_{t,k}$ is then updated to $\Sigma_{t+1,k}$ with:

$$\begin{aligned} \boldsymbol{x}^*_{t+1,k} &= \boldsymbol{x}_{t+1,k} - \bar{\boldsymbol{x}}_{t+1,k}, \\ X^*_{t+1,k} &= [X^*_{t,k}, \boldsymbol{x}^*_k], \\ \Sigma_{t+1,k} &= \left[\Sigma_{t,k}, \tfrac{1}{n_k - 1}(X^*_{t+1,k} \cdot \boldsymbol{x}^*_{t+1,k})\right], \end{aligned} \tag{3}$$

where the covariance matrices is stored in a packed (triangular) format because covariance matrices are always symmetric. For convenience, the * superscript marks the centered matrices through the text. $\bar{\boldsymbol{x}}_{t+1,k}$ is the estimated mean value (a scalar) of vector $\boldsymbol{x}_{t+1,k}$.

3.2 Inverse

Inverse of a matrix, as given in Equation (1), should generally never be explicitly calculated because faster and more numerically stable methods exist [6]. Suppose that we want to solve $\boldsymbol{x} = A^{-1}\boldsymbol{y}$, where $\boldsymbol{x}$ is the unknown vector. Whenever matrix A is symmetric and positive definite, $\boldsymbol{x}$ can be efficiently obtained via the Cholesky decomposition `chol` [13, Section 4.2.3] followed by backward substitution `backward` [13, Section 3.1.6]:

$$\begin{aligned} R &= \texttt{chol}(A), \\ \boldsymbol{x} &= \texttt{backward}(R, \boldsymbol{y}). \end{aligned} \tag{4}$$

The Cholesky factorization decomposes matrix A into an upper triangular matrix R such that the product of R and its transpose $R^\intercal$ yields A:

$$A = RR^\intercal. \tag{5}$$

Backward substitution then solves the problem from bottom to top (hence, the name):

$$\begin{aligned} \boldsymbol{x}[d] &= \boldsymbol{y}[d]/R[d,d], \\ \boldsymbol{x}[i] &= \left(\boldsymbol{y}[i] - \sum_{j=i+1}^{d} R[i,j]\boldsymbol{x}[j]\right)/R[i,i], \quad i \in [1, d-1]. \end{aligned} \tag{6}$$

Herein, an important factor is that both the Cholesky decomposition and backward substitution can be efficiently updated when a new column is added to matrix A. The Cholesky decomposition can be updated with `cholinsert`, which is a built-in function in Octave or Julia. Rank-one update of backward substitution is visually represented in Figure 2. Since the sample covariance matrices are always symmetric and positive definite when the features are linearly independent, we can use the Cholesky decomposition in QDA. Further discussions regarding the steps to be taken if this assumption is violated are detailed in Section 3.6.

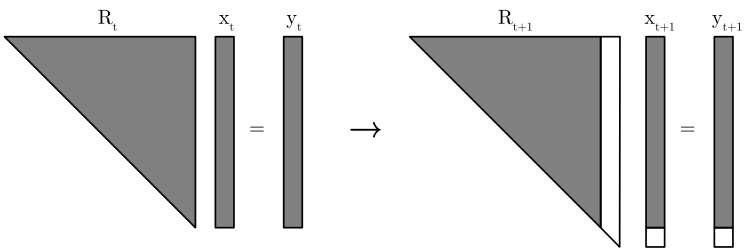

Fig. 2. Updating the system of equations by appending a single column into triangular matrix R

3.3 Determinant

The determinant of matrix A can be obtained as the product of the squared diagonal elements of $\texttt{chol}(A)$:

$$|A| = \prod_{i=1}^{d} R[i,i]^2, \tag{7}$$

where d is the size of R.

Whenever the logarithm of the determinant is needed, as in Equation (1), it is possible to avoid unnecessary overflows by summing the logarithms of the diagonal elements of $\texttt{chol}(A)$:

$$\ln|A| = \ln \prod_{i=1}^{d} R[i,i]^2 = 2 \cdot \sum_{i=1}^{d} \ln(R[i,i]). \tag{8}$$

Because `cholinsert` does not change the values of the old Cholesky decomposition R_t (it simply appends a new column), it is possible to update the logarithm of the determinant in real time:

$$\ln|A_{t+1}| = \ln|A_t| + 2 \cdot \ln(R_{t+1}[t+1, t+1]), \tag{9}$$

where $R_{t+1}[t+1, t+1]$ is the bottom right number in triangular matrix R_{t+1}.

3.4 Vectorization

Equation (1) is given only for a single sample. To score all new samples at once, first note that:

$$\boldsymbol{b}^\intercal A^{-1} \boldsymbol{b} = \boldsymbol{b}^\intercal (RR^\intercal)^{-1} \boldsymbol{b} \tag{10}$$

$$= \boldsymbol{b}^\intercal (R^{-1})^\intercal R^{-1} \boldsymbol{b} \tag{11}$$

$$= (R^{-1}\boldsymbol{b})^\intercal (R^{-1}\boldsymbol{b}), \tag{12}$$

where $\boldsymbol{b}$ is a vector, R is the Cholesky decomposition of A. From this, we obtain a scoring function for matrix X:

$$Z_k(x) = -\frac{1}{2}\sum_d (R_k^{-1}(X-\mu_k)) \circ (R_k^{-1}(X-\mu_k)) - \frac{1}{2}\ln|\Sigma_k| + \ln p_k. \quad (13)$$

where $\circ$ is element-wise multiplication, μ_k is the class-conditional mean of the training data, R_k is the Cholesky decomposition of the Σ_k covariance matrix, and $R_k^{-1}X$ is solved with backward substitution.

3.5 Online version

The QDA update process is described in Algorithm 1, where `cholinsert` is a built-in function as implemented in Octave or Julia. In addition, `solveinsert` updates the solution of the triangular system of equations with multiple right-hand sides using the algorithm described in Algorithm 2 (backward substitution, but scaled to work with a matrix X instead of vector $\boldsymbol{x}$).

Algorithm 1: Updating QDA when a new feature is inserted.

Input: new feature, $\boldsymbol{x}_{t+1}$; label conditional sample count, m; label conditional logarithm of prior probability, lp; label conditional feature means, μ_t; label conditional centered features, X_t^*; label conditional Cholesky decomposition, R_t; and label conditional logarithm of determinant, ld_t; t is the count of features, k is the class

Output: $\mu_{t+1}, X^*_{t+1}, R_{t+1}, ld_{t+1}, Z_{t+1}$

foreach $k \in K$ **do**

- $\mu_{t+1,k} = [\mu_{t+1,k}, \bar{\boldsymbol{x}}_{t+1,k}]$ `/* Mean vector */`
- $X^*_{t+1,k} = [X^*_{t,k}, \boldsymbol{x}^*_{t+1,k} - \bar{\boldsymbol{x}}_{t+1}]$ `/* Centered matrix */`
- $\boldsymbol{\sigma}_k = \frac{1}{m_k - 1} \cdot \boldsymbol{x}^*_{t+1,k} \cdot X^*_{t+1,k}$ `/* Covariance vector */`
- $R_{t+1,k} =$ `cholinsert`$(R_{t,k}, \boldsymbol{\sigma}_k)$ `/* Cholesky decomposition */`
- $ld_{t+1,k} = ld_{t,k} + 2 \cdot \ln(R_{t+1,k}[t+1, t+1])$ `/* Log determinant */`
- $A_{t+1,k} =$ `solveinsert`$(A_{t,k}, R_{t+1,k}, X^*_{t+1})$ `/* Solve equations */`
- $Z_{t+1,k} = lp_k - \frac{1}{2}\sum(A_{t+1,k} \circ A_{t+1,k}) + ld_{t+1,k}$ `/* Discrimination score */`

end

Algorithm 2: Function `solveinsert` updates the solution of a triangular system of equations with multiple right-hand side for $A \cdot R = X$.

Input: A_t, R_{t+1}, X_{t+1}, where t is the count of features

Output: A_{t+1}

$A_{t+1} = [A_t, (X_{t+1}[:, t+1] - A_t[:, 1:t] \cdot R_{t+1}[1:t, t+1])/R_{t+1}[t+1, t+1]]$

3.6 Regularization

One of the key problems of QDA is how to reliably estimate the covariance matrices. Equation (2) presents an empirical estimate of the sample covariance matrices. However, these sample covariance matrices suffer from a high variance of the parameter estimates as the count of the parameters to estimate grows quadratically with the number of features d and linearly with the number of the classes k.

When the count of the samples is small relative to d and k, it is frequently beneficial to assume that the covariance matrices are identical and use a single shared covariance matrix Σ_t in place of $\Sigma_{t,k}$:

$$\Sigma_t = \frac{(n_1 - 1)\Sigma_{t,1} + (n_2 - 1)\Sigma_{t,2} + \ldots + (n_k - 1)\Sigma_{t,k}}{(n_1 - 1) + (n_2 - 1) + \ldots + (n_k - 1)}, \tag{14}$$

where n with the index is the number of samples in the class. This variant of QDA is known as linear discriminant analysis (LDA).

Another common issue with QDA is the invertibility of the sample covariance matrices [17]. This can be remedied with covariance shrinking toward the identity matrix [18]:

$$\Sigma_{t,k} = (1 - \lambda)\Sigma_{t,k} + \lambda I, \tag{15}$$

where I is the identity matrix of the size $\Sigma_{t,k}$ and λ is a shrinkage coefficient, $\lambda \in (0, 1)$.

For an improved predictive accuracy, regularized discriminant analysis (RDA) [11] can be used, where the covariance matrix is a linear combination of the shared covariance matrix, the class conditional covariance matrix, and the identity matrix:

$$\Sigma_{t,k} = (1 - \alpha - \lambda)\Sigma_t + \alpha\Sigma_{t,k} + \lambda I, \tag{16}$$

where α is a tunable weight, $\alpha \in (0, 1 - \lambda)$.

4 Experiments

4.1 Speed

The proposed implementation of QDA is evaluated on the OpenML-CC18 benchmarking suite [2]. This suite includes 72 datasets, from which datasets that include nominal features or missing values were excluded, as the implementation does not directly support them.

Two algorithms were compared: QDA learning from scratch with scikit-learn 0.24.0 QDA as the reference versus the proposed QDA update by feature insert. For each of these two scenarios, the training and scoring times were recorded on each of the datasets. Brier scores (a calibration measure for classification) were also measured to validate that the quality of the predictions is equivalent. Since the Brier scores were indeed identical between the implementations, we do not report them.

Because some datasets have collinear features, as indicated by the high correlation coefficients in the last column in Table 1, regularized covariance matrices: $(1 - \lambda) \sum_k + \lambda I$ were used, where $\lambda = 0.02$. A constant regularization was used, as the aim of the experiment was not to find the best possible regularization coefficients but to compare runtimes.

Median runtimes from 15 iterations are provided in Table 1. The proposed updatable QDA was always faster than scikit-learn QDA. Notably, there was a 1000-fold increase in speed in five cases. Based on the code profiling, SVD decomposition is the bottleneck in scikit-learn implementation. For the description of how to use SVD for discriminant analysis, see for example [17]. The code for the result replication is available at github.com/janmotl/qda.

Table 1. Training and scoring times in seconds for scikit-learn QDA and the proposed updatable QDA, when the last feature is inserted. Dataset metadata: count of samples, features, classes, and the maximal absolute Person's correlation coefficient between two different features in the dataset. Predictions from scikit-learn and the proposed QDA are identical. The datasets are ordered by the obtained speed up.

dataset	scikit-learn	proposed	speed up	samples	features	classes	corr
mfeat-morphological	0.015	0.014	1.10	2 000.000 0	6.000 0	10.000 0	0.997
balance-scale	0.004	0.004	1.15	625.000 0	4.000 0	3.000 0	0.000
wilt	0.006	0.004	1.38	4 839.000 0	5.000 0	2.000 0	0.096
diabetes	0.005	0.003	1.71	768.000 0	8.000 0	2.000 0	0.554
vehicle	0.010	0.005	1.77	846.000 0	18.000 0	4.000 0	1.000
blood-transfusion	0.005	0.003	1.79	748.000 0	4.000 0	2.000 0	1.000
segment	0.022	0.010	2.21	2 310.000 0	19.000 0	7.000 0	1.000
banknote-authentication	0.007	0.003	2.44	1 372.000 0	4.000 0	2.000 0	0.779
climate-model	0.006	0.003	2.45	540.000 0	20.000 0	2.000 0	0.111
phoneme	0.010	0.004	2.45	5 404.000 0	5.000 0	2.000 0	0.832
kc2	0.008	0.003	2.94	522.000 0	21.000 0	2.000 0	1.000
wdbc	0.009	0.003	3.50	569.000 0	30.000 0	2.000 0	1.000
steel-plates-fault	0.042	0.012	3.61	1 941.000 0	27.000 0	7.000 0	1.000
letter	0.334	0.090	3.71	20 000.000 0	16.000 0	26.000 0	0.985
jungle_chess_2pcs	0.081	0.021	3.92	44 819.000 0	6.000 0	3.000 0	0.902
pendigits	0.097	0.023	4.18	10 992.000 0	16.000 0	10.000 0	0.986
pc1	0.013	0.003	5.00	1 109.000 0	21.000 0	2.000 0	1.000
kc1	0.019	0.003	6.30	2 109.000 0	21.000 0	2.000 0	1.000
qsar-biodeg	0.019	0.003	7.06	1 055.000 0	41.000 0	2.000 0	0.992
mfeat-zernike	0.090	0.013	7.08	2 000.000 0	47.000 0	10.000 0	1.000
wall-robot-navigation	0.049	0.007	7.31	5 456.000 0	24.000 0	4.000 0	0.963
pc3	0.025	0.003	8.02	1 563.000 0	37.000 0	2.000 0	1.000
texture	0.153	0.019	8.07	5 500.000 0	40.000 0	11.000 0	1.000
satimage	0.105	0.011	9.40	6 430.000 0	36.000 0	6.000 0	0.996
pc4	0.026	0.003	9.64	1 458.000 0	37.000 0	2.000 0	1.000
mfeat-karhunen	0.127	0.013	9.82	2 000.000 0	64.000 0	10.000 0	0.957
analcatdata_authorship	0.052	0.005	10.38	841.000 0	70.000 0	4.000 0	0.971
GesturePhaseSegmentation	0.136	0.012	11.17	9 873.000 0	32.000 0	5.000 0	0.994
mfeat-fourier	0.172	0.013	13.17	2 000.000 0	76.000 0	10.000 0	0.968
optdigits	0.243	0.017	14.42	5 620.000 0	64.000 0	10.000 0	0.993
first-order-theorem-proving	0.184	0.011	17.11	6 118.000 0	51.000 0	6.000 0	1.000
numerai28.6	0.765	0.033	23.32	96 320.000 0	21.000 0	2.000 0	0.986
spambase	0.091	0.004	23.62	4 601.000 0	57.000 0	2.000 0	1.000
ozone-level-8hr	0.085	0.003	29.33	2 534.000 0	72.000 0	2.000 0	1.000
semeion	0.624	0.012	50.07	1 593.000 0	256.000 0	10.000 0	0.981
mfeat-factors	0.686	0.014	50.67	2 000.000 0	216.000 0	10.000 0	1.000
mfeat-pixel	0.796	0.010	83.44	2 000.000 0	240.000 0	10.000 0	0.994
cnae-9	0.986	0.011	88.31	1 080.000 0	856.000 0	9.000 0	1.000
isolet	11.465	0.051	224.41	7 797.000 0	617.000 0	26.000 0	1.000
madelon	1.893	0.004	515.26	2 600.000 0	500.000 0	2.000 0	0.999
har	9.689	0.014	670.19	10 299.000 0	561.000 0	6.000 0	1.000
Fashion-MNIST	101.438	0.086	1 177.68	70 000.000 0	784.000 0	10.000 0	0.996
Devnagari-Script	731.853	0.484	1 511.97	92 000.000 0	1 024.000 0	46.000 0	0.997
mnist_784	145.914	0.091	1 600.96	70 000.000 0	784.000 0	10.000 0	1.000
Bioresponse	40.582	0.004	9 068.45	3 751.000 0	1 776.000 0	2.000 0	0.998
CIFAR_10	2 130.479	0.084	25 385.96	60 000.000 0	3 072.000 0	10.000 0	0.998

4.2 Accuracy

Because we are generally not only interested in the runtime but also in the accuracy of the classifiers, the accuracy of QDA is compared to a few selected classifiers. Due to the lack of classifier implementations that work on a stream of features, all classifiers are evaluated in offline use case.

Classifier selection Because the focus of this article is on learning on a stream of features, naive Bayes (NB) and k-nearest neighbors (kNN) are included as potential competitors. As an approximation[1] of the online one-layer neural network as described in [3] we use logistic regression with L1 and L2 regularization (LR) [33]. Furthermore, random forest (RF) by [4], and gradient boosted trees (GBT) by [12] are included as examples of common offline classifiers. For all these algorithms, we use implementations from scikit-learn 0.24.0. As variants of QDA, we also include LDA, which uses a shared covariance matrix, and regularized discriminant analysis (RDA) by [11], which uses a linear combination of the shared covariance matrix and the class conditional covariance matrix. For an exhaustive comparison of QDA to other classifiers, see Delgado's empirical comparison of QDA to other 178 classifiers on 121 datasets [10].

Methodology We use the same dataset as in Section 4.1 and estimate the testing ROC-AUC with 10-fold cross-validation. Because kNN (with Euclidean distance) requires normalized features, features are normalized with z-score normalization for all the classifiers. The classifier hyperparameters were initialized to the recommended settings by [22] and greedily optimized based on the average testing ROC-AUC over all datasets. The used hyperparameters are listed in Table 2.

Table 2. The used classifier hyperparameters. The classifiers are described in Section 4.2.

Classifier	Parameters
GBT	n_estimators=500, learning_rate=0.1, max_features='log2'
kNN	n_neighbors=29
LDA	shrinkage=0.0001
LR	C=1.5, penalty='elasticnet', l1_ratio=0.5
NB	var_smoothing=10e-12
QDA	shrinkage=0.02
RDA	shrinkage=0.0001, alpha=0.2
RF	n_estimators=300, max_features=0.25, criterion='entropy'

Results The results are in Table 3. However, the Devnagari-Script, Fashion-MNIST, mnist_784, and numerai28.6 are absent in the comparison because too

[1] We contacted the authors but we were told that the implementation is not public.

many algorithms didn't finish on these datasets in 3 hours. QDA underperforms on har, madelon, pc4, phoneme, and wall-robot-navigation datasets as QDA is incapable of modeling XOR-like problems because each class is represented by only a single prototype. To illustrate the character of these datasets, we cite a part of wall-robot-navigation dataset description: "The wall-following task and data gathering were designed to test the hypothesis that this apparently simple navigation task is indeed a non-linearly separable classification task"[2]. The problem is neither separable with a quadratic decision border. On the other end, decision tree based algorithms (RF, GBT) are capable of representing XOR-like problems and perform well on these datasets. A possible remedy of this QDA deficiency is to use multiple prototypes per class as in mixture discriminant analysis by [14] or map the data into a high-dimensional space as in generalized discriminant analysis by [1]. However, evaluation of these alternatives is out of the scope of this article.

On the other end, QDA performs well on mfeat-zernike and vehicle datasets, which are both separable with quadratic decision boundaries.

Statistical significance Statistically significant differences between the classifiers are depicted in Figure 3 following the procedure by [7]. NB is significantly worse than any other evaluated classifier. RF is the best. QDA, kNN, LDA, RDA, LR, and GBT are comparable.

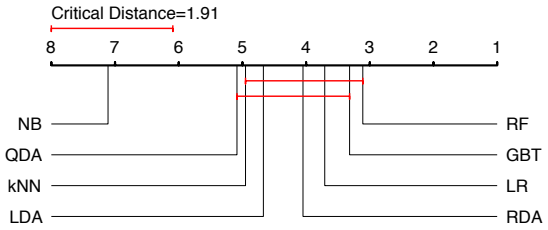

Fig. 3. Comparison of the classifiers based on ROC-AUC. Groups of classifiers that are not significantly different (at p=0.01) are connected.

Pareto optimality The comparison of the average classifier ROC-AUC over datasets in Table 3 versus the average classifier total runtime on the cross-validation (training and scoring time) is depicted in Figure 4. LDA and RDA are not only Pareto optimal, but are also on the knee-point of the Pareto frontier.

[2] https://www.openml.org/d/1497

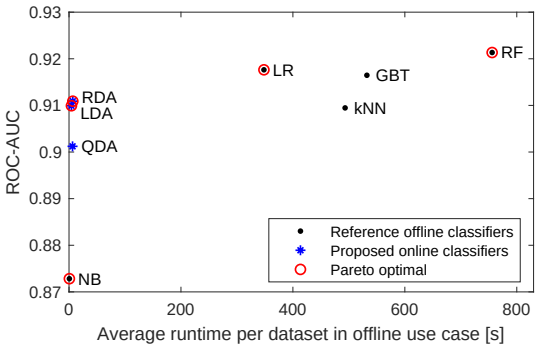

Fig. 4. Classifier ROC-AUC and cross-validation runtime tradeoff. Regularized discriminant analysis (RDA) and linear discriminant analysis (LDA) are on the knee-point. The runtime is for the *offline* use case (all features are available at once). If we evaluated the classifiers on a stream of features, the *average* runtime of some of the offline classifiers would be in days.

Table 3. Testing ROC-AUC from cross-validation.

Dataset	NB	kNN	LR	LDA	RDA	QDA	RF	GBT
Bioresponse	0.575	0.805	0.845	0.749	0.749	0.639	0.880	0.867
GesturePhaseSegmentation	0.670	0.706	0.743	0.735	0.684	0.674	0.763	0.747
analcatdata_authorship	0.994	1.000	1.000	1.000	1.000	0.997	0.999	1.000
balance-scale	0.889	0.916	0.964	0.951	0.955	0.971	0.714	0.872
banknote-authentication	0.938	1.000	1.000	1.000	1.000	1.000	1.000	1.000
blood-transfusion	0.801	0.733	0.947	0.932	0.927	0.877	0.548	0.610
climate-model	0.952	0.919	0.956	0.948	0.944	0.858	0.949	0.926
cnae-9	0.926	0.985	0.996	0.968	0.968	0.968	0.989	0.997
diabetes	0.816	0.813	0.830	0.830	0.831	0.812	0.833	0.809
first-order-theorem-proving	0.608	0.753	0.680	0.678	0.698	0.679	0.810	0.805
har	0.959	0.993	0.997	0.998	0.998	0.993	0.999	0.999
isolet	0.988	0.997	0.999	0.999	0.999	0.997	0.999	0.999
jungle_chess_2pcs	0.800	0.860	0.790	0.791	0.799	0.815	0.785	0.813
kc1	0.788	0.769	0.794	0.790	0.797	0.794	0.757	0.745
kc2	0.827	0.812	0.825	0.818	0.825	0.816	0.780	0.707
letter	0.957	0.998	0.980	0.967	0.982	0.996	1.000	1.000
madelon	0.644	0.622	0.585	0.587	0.587	0.527	0.913	0.703
mfeat-factors	0.993	0.998	0.998	0.999	0.999	0.997	0.999	0.999
mfeat-fourier	0.966	0.970	0.978	0.976	0.980	0.979	0.984	0.980
mfeat-karhunen	0.996	0.996	0.996	0.997	0.999	0.998	0.998	0.998
mfeat-morphological	0.946	0.959	0.966	0.965	0.964	0.963	0.956	0.958
mfeat-pixel	0.985	0.998	0.997	0.996	0.996	0.998	0.999	0.999
mfeat-zernike	0.960	0.978	0.982	0.979	0.981	0.980	0.969	0.968
optdigits	0.972	0.999	0.999	0.998	0.996	0.994	1.000	1.000
ozone-level-8hr	0.817	0.864	0.896	0.869	0.877	0.846	0.872	0.868
pc1	0.720	0.804	0.853	0.832	0.819	0.793	0.819	0.793
pc3	0.772	0.804	0.801	0.830	0.824	0.795	0.856	0.819
pc4	0.843	0.872	0.907	0.894	0.890	0.894	0.946	0.945
pendigits	0.979	0.999	0.997	0.990	0.997	0.999	1.000	1.000
phoneme	0.818	0.921	0.812	0.814	0.825	0.842	0.965	0.940
qsar-biodeg	0.822	0.905	0.919	0.904	0.912	0.908	0.917	0.920
satimage	0.956	0.986	0.977	0.972	0.976	0.975	0.991	0.991
segment	0.957	0.984	0.978	0.973	0.976	0.978	0.996	0.995
semeion	0.956	0.993	0.995	0.991	0.991	0.992	0.996	0.998
spambase	0.922	0.949	0.961	0.943	0.936	0.938	0.978	0.983
steel-plates-fault	0.884	0.884	0.893	0.895	0.899	0.894	0.907	0.902
texture	0.968	0.999	1.000	1.000	1.000	1.000	1.000	1.000
vehicle	0.776	0.903	0.945	0.943	0.944	0.956	0.934	0.938
wall-robot-navigation	0.820	0.918	0.890	0.847	0.868	0.870	0.999	0.999
wdbc	0.985	0.991	0.996	0.993	0.995	0.990	0.991	0.995
wilt	0.841	0.933	0.956	0.966	0.960	0.958	0.985	0.988
Avg (larger is better)	0.873	0.909	0.918	0.910	0.911	0.901	0.921	0.916
Rank (smaller is better)	7.110	4.951	3.707	4.671	4.049	5.085	3.110	3.317

5 Discussion

5.1 Feature selection

The ability to quickly insert new features can be used, for example, to speed up forward variable selection with LDA, as implemented in `greedy.wilks()` [24].

5.2 Limitations

The online version of QDA inherits all the disadvantages of offline QDA, namely high sensitivity to outliers and difficulties with the accurate estimation of the covariance matrices, which can be addressed with robust statistical estimates [16] and covariance shrinking [18], respectively. Missing values in the estimate of the covariance matrix can be handled as described by [20]. And a mixture of numerical and nominal features can be handled with one of the methods described by [15].

6 Conclusion

The key to learning on a stream of features with QDA is to use the Cholesky decomposition and `cholinsert` function in place of the matrix inverse. This method, together with an efficient update of the covariance matrices, reduced the computational complexity of the QDA update when a new feature was added, from $\mathcal{O}(nd^2+d^3)$ to $\mathcal{O}(nd+d^2)$. This is equivalent to the reduction by d (the number of features). The proposed QDA returns identical predictions to scikit-learn QDA while being 1000 times faster. In the accuracy comparison, the need to regularize the estimated covariance matrices was ientified. The proposed regularized discriminant analysis on a stream of features is statistically significantly more accurate than naive Bayes (at p=0.01) and comparable to regularized logistic regression, k-nearest neighbors, and gradient boosted trees while in runtime it is much closer to naive Bayes.

Acknowledgments

We are grateful to Karel Klouda, Oliver Schulte, and anonymous reviewers for constructive comments on earlier drafts of this article. This research was supported by Grant Agency of the Czech Technical University in Prague (SGS17/210/OHK3/3T/18) and Czech Science Foundation (GAČR 18-18080S).

References

1. Baudat, G., Anouar, F.: Generalized discriminant analysis using a kernel approach. Neural Comput. **12**(10), 2385–2404 (2000). https://doi.org/10.1162/089976600300014980

2. Bischl, B., Casalicchio, G., Feurer, M., Hutter, F., Lang, M., Mantovani, R.G., van Rijn, J.N., Vanschoren, J.: OpenML Benchmarking Suites. arXiv pp. 1–6 (2017)
3. Bolon-Canedo, V., Fernández-Francos, D., Peteiro-Barral, D., Alonso-Betanzos, A., Guijarro-Berdiñas, B., Sánchez-Maroño, N.: A unified pipeline for online feature selection and classification. Expert Syst. Appl. **55**, 532–545 (2016). https://doi.org/10.1016/j.eswa.2016.02.035
4. Breiman, L.: Random forest. Mach. Learn. **45**(5), 1–35 (1999). https://doi.org/10.1023/A:1010933404324
5. Dagher, I.: Incremental PCA-LDA algorithm. CIMSA (4), 97–101 (2010). https://doi.org/10.1109/CIMSA.2010.5611752
6. Davis, T.A.: Algorithm 9xx: FACTORIZE: An object-oriented linear system solver for MATLAB. ACM Trans. Math. Softw. **39**(4), 1–18 (2013). https://doi.org/10.1145/2491491.2491498
7. Demšar, J.: Statistical Comparisons of Classifiers over Multiple Data Sets. J. Mach. Learn. Res. **7**, 1–30 (2006). https://doi.org/10.1016/j.jecp.2010.03.005
8. Dhamecha, T.I., Singh, R., Vatsa, M.: On incremental semi-supervised discriminant analysis. Pattern Recognit. **52**, 135–147 (2016). https://doi.org/10.1016/j.patcog.2015.09.030
9. Efron, B.: The efficiency of logistic regression compared to normal discriminant analysis. J. Am. Stat. Assoc. **70**(352), 892–898 (1975). https://doi.org/10.1080/01621459.1975.10480319
10. Fernández-Delgado, M., Cernadas, E., Barro, S., Amorim, D.: Do we Need Hundreds of Classifiers to Solve Real World Classification Problems? J. Mach. Learn. Res. **15**, 3133–3181 (2014)
11. Friedman, J.H.: Regularized discriminant analysis. J. Am. Stat. Assoc. **84**(405), 165–175 (1989). https://doi.org/10.1080/01621459.1989.10478752
12. Friedman, J.H.: Greedy function approximation: A gradient boosting machine. Ann. Stat. **29**(5) (oct 2001). https://doi.org/10.1214/aos/1013203451
13. Golub, G.H., Loan, C.F.V.: Matrix Computations. Johns Hopkins, 4th edn. (2013)
14. Hastie, T., Tibshirani, R.: Discriminant Analysis by Gaussian Mixtures. J. R. Stat. Soc. Ser. B **58**(1), 155–176 (1996). https://doi.org/10.1111/j.2517-6161.1996.tb02073.x
15. Huberty, C.J., Wisenbaker, J.M., Smith, J.D., Smith, J.C.: Using Categorical Variables in Discriminant Analysis. Multivariate Behav. Res. **21**(4), 479–496 (oct 1986). https://doi.org/10.1207/s15327906mbr2104{_}7
16. Jiang, J.h., Chen, Z.p., Xu, C.j., Yu, R.q.: Robust Linear Discriminant Analysis for Chemical Pattern Recognition. J. Chemom. **13**(January 1998), 3–13 (1999). https://doi.org/10.1002/(SICI)1099-128X(199901/02)13:1<3::AID-CEM524>3.0.CO;2-R
17. Kalina, J., Valenta, Z., Tebbens, J.D.: Computation of Regularized Linear Discriminant Analysis. Comput. Stat. Int. Conf. pp. 128–133 (2015)
18. Ledoit, O., Wolf, M.: Improved estimation of the covariance matrix of stock returns with an application to portfolio selection. J. Empir. Financ. **10**(5), 603–621 (dec 2003). https://doi.org/10.1016/S0927-5398(03)00007-0
19. Liyanage, Y.W., Zois, D.S., Chelmis, C.: On–the–Fly Joint Feature Selection and Classification. arXiv pp. 1–12 (2020)
20. Lounici, K.: High-dimensional covariance matrix estimation with missing observations. Bernoulli **20**(3), 1029–1058 (2014). https://doi.org/10.3150/12-BEJ487
21. Ojeda, F., Suykens, J.A., De Moor, B.: Variable selection by rank-one updates for least squares support vector machines. In: 2007 Int. Jt. Conf. Neural Networks. pp. 2283–2288. IEEE (aug 2007). https://doi.org/10.1109/IJCNN.2007.4371314

22. Olson, R.S., Cava, W.L., Mustahsan, Z., Varik, A., Moore, J.H.: Data-driven advice for applying machine learning to bioinformatics problems. In: Biocomput. 2018. pp. 192–203. World Scientific (jan 2018). https://doi.org/10.1142/9789813235533{_}0018
23. Rahmaninia, M., Moradi, P.: OSFSMI: Online stream feature selection method based on mutual information. Appl. Soft Comput. J. **68**, 733–746 (2018). https://doi.org/10.1016/j.asoc.2017.08.034
24. Roever, C., Raabe, N., Luebke, K., Ligges, U., Szepannek, G., Zentgraf, M., Ligges, M.U.: The klaR package (2006)
25. Sekeh, S.Y., Ganesh, M.R., Banerjee, S., Corso, J.J., Hero, A.O.: A Geometric Approach to Online Streaming Feature Selection. arXiv (2019)
26. Tjen-Sien, L., Wei-Yin, L., Shih, Y.S.: A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-three Old and New Classification Algorithms. Mach. Learn. **229**, 203–229 (1992). https://doi.org/10.1023/A:1007608224229
27. Wu, X., Yu, K., Wang, H., Ding, W.: Online Streaming Feature Selection. Int. Conf. Mach. Learn. pp. 1159–1166 (2010)
28. You, D., Wu, X., Shen, L., Deng, S., Chen, Z., Ma, C., Lian, Q.: Online Feature Selection for Streaming Features Using Self-Adaption Sliding-Window Sampling. IEEE Access **7**, 16088–16100 (2019). https://doi.org/10.1109/ACCESS.2019.2894121
29. Yu, K., Ding, W., Simovici, D.A., Wu, X.: Mining emerging patterns by streaming feature selection. KDD pp. 60–68 (2012). https://doi.org/10.1145/2339530.2339544
30. Yu, K., Wu, X., Ding, W., Pei, J.: Towards Scalable and Accurate Online Feature Selection for Big Data. Proc. - IEEE Int. Conf. Data Mining, ICDM **2015-Janua**(January), 660–669 (2015). https://doi.org/10.1109/ICDM.2014.63
31. Zhou, J., Foster, D., Stine, R., Ungar, L.: Streaming feature selection using alpha-investing. KDD pp. 384–393 (2005). https://doi.org/10.1145/1081870.1081914
32. Zhou, P., Li, P., Zhao, S., Wu, X.: Feature Interaction for Streaming Feature Selection. IEEE Trans. Neural Networks Learn. Syst. pp. 1–12 (2020). https://doi.org/10.1109/TNNLS.2020.3025922
33. Zou, H., Hastie, T.: Addendum: Regularization and variable selection via the elastic net. J. R. Stat. Soc. Ser. B (Statistical Methodol. **67**(5), 768–768 (nov 2005). https://doi.org/10.1111/j.1467-9868.2005.00527.x