Density Preserving Sampling (DPS) for error estimation and model selection

Marcin Budka, *Member, IEEE,* and Bogdan Gabrys, *Senior Member, IEEE*

Abstract—Estimation of the generalization ability of a classification or regression model is an important issue, as it indicates expected performance on previously unseen data and is also used for model selection. Currently used generalization error estimation procedures like cross–validation (CV) or bootstrap are stochastic and thus require multiple repetitions in order to produce reliable results, which can be computationally expensive if not prohibitive. The correntropy–based Density Preserving Sampling procedure (DPS) proposed in this paper eliminates the need for repeating the error estimation procedure by dividing the available data into subsets, which are guaranteed to be representative of the input dataset. This allows to produce low variance error estimates with accuracy comparable to 10 times repeated cross–validation at a fraction of computations required by CV. The method can also be successfully used for model ranking and selection. This paper derives the Density Preserving Sampling procedure and investigates its usability and performance using a set of publicly available benchmark datasets and standard classifiers.

Index Terms—Error estimation, model selection, sampling, cross–validation, bootstrap, correntropy.

1 Introduction

Estimation of the generalization ability of a classification or regression model is an important issue in the machine learning field, especially that it is independent of the actual model used. Generalization accuracy estimates are not only used as indicators of the expected performance of the developed classifier or regressor on previously unseen data, but are also commonly used for model ranking and selection [1].

In contrast to the large number of various regression and classification methods currently in use, there is only a handful of model independent generalization error estimation techniques. The most popular of them are cross–validation [2] dating back to 1968, and bootstrap [3] developed in the 1979. These techniques, and especially cross–validation are being used even more willingly and blindly after the publication of a seminal paper by Kohavi in 1995, presenting a comparative study of bootstrap and cross–validation [4], and currently estimated to have more than 1500 citations according to Harzing's Publish or Perish citation retrieval system [5].

The basic idea, shared by all generalization error estimation methods, is to reserve a subset of available data to test the model after it has been trained using the remainder of the dataset. The main difference between various techniques is the way the generalization error is calculated, the size of the subset reserved for testing or whether the procedure is repeated multiple times or not. They have however something in common, and that is the way in which the testing subset is generated – random sampling. Although the stochastic nature of bootstrap and cross–validation ensures that in the limit they would both converge to a true value, this may also lead to large variations in the estimate between consecutive runs, making the results unreliable. This effect can be alleviated to a large extent by repeating both procedures multiple times, which however significantly increases the computational demands.

A good test set should be independent of the training data and representative of the population from which it has been drawn. While random sampling meets the first requirement, it does not guarantee the representativeness of the test set. In order to address this issue, stratified sampling approaches have been developed [4], which try to increase the representativeness at the expense of independence, and are able to achieve better results than their non–stratified counterparts.

Inspired by the success of stratified sampling approaches, in this paper we propose a density preserving sampling procedure (DPS), which further sacrifices the independence of the test set to enforce its representativeness. The method achieves this goal by optimizing the correntropy, a recently developed, non–parametric similarity measure of the probability density functions [6], and as shown in the experimental section produces accurate generalization estimates requiring a fraction of computations when compared to cross–validation.

This paper is an extended version of [7] and is organized as follows. In Section 2 the problem of estimation of generalization error is introduced, together with standard estimation techniques and criteria of their evaluation. Section 3 describes the concept of Information Theoretic Learning including the online manipulation of Renyi's quadratic entropy and the definition of correntropy. In Section 4, the novel Density Preserving Sampling procedure is derived. The experimental results,

• *M. Budka and B. Gabrys are with the Computational Intelligence Research Group, Bournemouth University, School of Design, Engineering and Computing, Poole House, Talbot Campus, Fern Barrow, Poole BH12 5BB, United Kingdom.*
Email: mbudka@bournemouth.ac.uk / bgabrys@bournemouth.ac.uk

including evaluation of DPS in terms of bias and variance as well as its usability for model selection are given in Section 5. Finally, the discussion and conclusions can be found in Sections 6 and 7 respectively.

2 Generalization error estimation

Generalization error is the error a predictive model will make on novel, previously unseen data, generated from the same distribution as the data used to develop the model [1]. Low generalization error is thus a sign of a good match between the model and the problem, and lack of overfitting [8].

It is impossible to obtain a closed form solution for calculation of the generalization error or even for calculation of tight bounds for the error, in all but the simplest cases [9]. The only practical solution is to estimate the generalization error from all available i.i.d. (independently and identically distributed) data samples by splitting them into training and validation sets [10]. For the error estimate to be meaningful both these datasets should be representative of the true distribution, so the way in which the data is split plays a crucial role.

2.1 Hold–out and random subsampling

The simplest and the least computationally expensive way to estimate the generalization error is the hold–out method [11], in which the data is split randomly into two parts: the training set and the hold–out set, in a priori chosen proportions. The model is then trained using the training dataset and its error on the hold–out data becomes an estimate of the generalization error. The obvious drawback of the hold–out method is that unless both datasets are large enough (which is a vague term by itself), different estimates will be obtained from one run to another. A workaround, known as random subsampling [10], repeats the hold–out procedure multiple times and averages the results. This procedure however still has some disadvantages as it does not guarantee that all instances will at some point be used for training nor that none of the classes will be over/under–represented in the hold–out set [4]. In order to circumvent these issues, more advanced resampling techniques have been developed. Yet, the holdout method is still being used when dealing with large datasets, as in this case other techniques quickly become untractable. It is also sometimes assumed that more advanced resampling techniques are simply not needed for large amounts of data.

2.2 Cross–validation

Cross–validation is a widely used standard statistical technique for estimation of model generalization ability, applied with a great success to both classification and regression problems [8], [1]. In k–fold cross–validation the whole available dataset is first randomly divided into k approximately equal subsets. Each of these subsets or folds is then in turn put aside as validation data, a model is built using the remaining $k-1$ subsets and tested using the validation subset. The estimate of the generalization error is then calculated as a mean value of all validation errors, while the standard deviation of the validation error can be used to approximate the confidence intervals of obtained error estimate. The whole procedure thus requires development of exactly k models. Since the results obtained in the setting described above are also likely to vary from one run to another, the procedure is usually repeated multiple times for various random splits and the results are averaged.

The most often used variants of cross–validation are:

- Leave–one–out cross–validation in which a single instance is used as a validation set each time. This produces unbiased error estimates but with high variance and can be computationally prohibitive for large datasets.
- Repeated 10–fold cross–validation, which often is a good compromise between speed and accuracy.
- Repeated 2–fold cross–validation, which is an approximation of the bootstrap method [11].

In order to improve the accuracy of the estimates obtained, a stratified cross–validation approach is used in practice, which samples the data in a way that approximates the percentage of each class in every fold [4]. For regression problems, stratified cross–validation produces folds with equal mean values of the target variable [10].

2.3 Bootstrap

Bootstrap is a second commonly used generalization error estimation procedure [8], [1], especially useful when dealing with small datasets [11]. Given an input dataset of size m, the method performs uniform sampling with replacement to produce a training set of the same size. The instances not picked during the sampling procedure form the test set. The probability of each instance ending up in the test set is thus given by:

$$\left(1-\frac{1}{m}\right)^{m} \approx e^{-1} \approx 0.368 \tag{1}$$

Due to the fact that the probability of each instance being picked for training is $1-0.368=0.632$, the method is also often called the '0.632 bootstrap' [11]. Since the error estimate obtained using test data only would be overly pessimistic (only about 63.2% of instances are used for training every time), the generalization error estimate is calculated using weighted test and training errors:

$$error = \frac{1}{b}\sum_{i=1}^{b}\left(0.632 \times e_{test} + 0.368 \times e_{all}\right) \tag{2}$$

where b is the number of bootstrap samples and e_{all} is the error on all samples from the original dataset. In general, the more times the whole process is repeated, the more accurate the estimate. A comprehensive comparative study of cross–validation and bootstrap has been described in [4].

2.4 Bias and variance of error estimation methods

The bias of an error estimation method is the difference between the expected value of the error and the estimated value [4]. For an unbiased estimator, this difference is equal to zero. Bias can also be either positive or negative. In the former case, the estimate is said to be overly optimistic, as the estimated error is lower than the expected error. Negative bias on the other hand leads to overly pessimistic error estimates.

Low bias on its own does not guarantee good performance of the model. There is another important parameter – the variance, which measures the variability of the error estimate from one run to another. In the case of subsampling methods discussed in this paper, the variability is usually approximated by the expected standard deviation of a single accuracy estimation run [4]. A good generalization error estimator should thus have low bias and low variance. Unfortunately in practice it is usually difficult to achieve both at the same time, leading to so called bias–variance trade–off [8].

3 Information Theoretic Learning (ITL)

Information Theoretic Learning is a procedure of adapting the parameters of a learning machine using information theoretic criterion [12]. The goal of learning can be stated as exploration and exploitation of redundancies from a single or multiple sources of information the learning machine is exposed to. This shifts the problem towards quantification and manipulation of redundancy, which thus makes Shannon's information theory the ultimate framework of machine learning [13].

Application of the information theory to learning problems is however not straightforward. The main issue is the omnipresent 'learning from exemplars' paradigm, while the information theory in its traditional form is only able to deal with probability density functions given in an analytic form [13]. Although it is possible to use numerical approximation, it quickly becomes intractable as the dimensionality of the input space grows [12]. Information Theoretic Learning framework should thus allow for both non–parametric estimation and manipulation of entropy and various divergence measures, enabling training of both linear and nonlinear mappers by transferring as much information as possible from the training data into parameters of the system [13].

3.1 Renyi's quadratic entropy

Entropy is a measure of the uncertainty associated with a random variable. It quantifies the average information content that is missing due to the unknown value of the variable and is the main criterion used in ITL approaches. The higher the entropy, the more information can be gained by discovering the value of the variable.

Denoting by $p(y)$ the probability density function of y, Shannon's differential entropy is given by:

$$H_S(y) = \int p(y) \log \frac{1}{p(y)}\, dy = -\int p(y) \log p(y)\, dy \quad (3)$$

Calculation of Shannon's entropy requires the density function to be given in an analytic form. There are however other definitions of entropy that can be used in the ITL framework and Renyi's entropy is one of them. The definition of Renyi's entropy of order α for a continuous random variable is given by:

$$H_{R_\alpha}(y) = \frac{1}{1-\alpha} \log \int p(y)^\alpha\, dy \quad (4)$$

Renyi's entropy involves calculation of the integral of the power of PDF rather than integral of the logarithm as in the case of Shannon's counterpart, which is much easier to estimate [13]. Moreover, Shannon's entropy is the limiting case of Renyi's entropy when $\alpha \to 1$. For practical applications the choice of $\alpha = 2$ is a good compromise between robustness and computational complexity ($O(n^2)$) [13], which leads to the definition of Renyi's quadratic entropy:

$$H_{R_2} = -\log \int p(y)^2\, dy \quad (5)$$

The most important property of Renyi's entropy from the point of view of ITL is that the extrema of H_S and H_R overlap [12], so both definitions are equivalent for the purpose of entropy optimization.

The above criterion is however still useless without a good estimate of the probability density function, which fortunately can be obtained and efficiently integrated into (5) by using the Parzen window density estimator [1]. Denoting by $G(y,\ \sigma^2 I)$ a spherical Gaussian kernel centered at y with a diagonal covariance matrix $\sigma^2 I$, the PDF can be estimated as follows:

$$p(y) = \frac{1}{N} \sum_{i=1}^{N} G(y - y_i,\ \sigma^2 I) \quad (6)$$

Substituting (6) into (5) and using the convolution property of the Gaussian kernel yields:

$$H_{R_2}(y) = -\log \int p(y)^2\, dy = -\log V(y) \quad (7)$$

$$\begin{aligned} V(y) &= \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} \int G(z - y_i,\ \sigma^2 I)\, G(z - y_j,\ \sigma^2 I)\, dz \\ &= \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} G(y_i - y_j,\ 2\sigma^2 I) \end{aligned} \quad (8)$$

Renyi's entropy of order α calculates the interactions between α–tuplets of samples, so the higher the value of α, the more information about the structure of the dataset can be extracted [13] but the computational complexity – $O(n^\alpha)$ – quickly becomes prohibitive.

If some imaginary particles were placed on top of each data sample a potential field would be created, since $G(y_i - y_j,\ 2\sigma^2 I)$ is always positive and decays exponentially with the square of the distance between y_i and y_j [12]. The samples can thus be referred to as Information Particles while $V(y)$, which is an averaged

sum of all pairs of interactions and represents the total potential energy of the dataset – Information Potential. By analogy to classical physics the gradient of potential energy is a force, which would drag the particles to a state with minimum potential if they were free to move. This behaviour can be used for training of adaptive systems with forces taking the place of the injected error and used for adjusting parameters of the model [13].

3.2 Auto– and cross–correntropy

A Generalized Correlation Function (GCF) for a stochastic process x_t has been defined in [14] as:

$$V_X(t_1, t_2) = E[\phi(x_{t_1}), \phi(x_{t_2})] = E[k(x_{t_1}, x_{t_2})] \quad (9)$$

where E stands for the expected value, ϕ denotes some kernel induced transformation and k is a kernel function, assumed to be Gaussian from now on. It has been proven, that the GCF estimator not only conveys information about autocorrelation but also about the structure of the dataset, as its mean value for non–zero lags converges asymptotically to the estimate of the Information Potential calculated using Renyi's quadratic entropy [14]. For this reason the function has been named auto–correntropy and is a preferred choice over traditional methods also due to taking advantage of all even moments of the PDF.

The idea of auto–correntropy has been further developed in [6] for a general case of two arbitrary random variables. The new measure, named cross–correntropy (or correntropy) is defined for variables X and Y as:

$$V_{XY}(X, Y) = E[\phi(X), \phi(Y)] = E[k(X, Y)] \quad (10)$$

The correntropy can be used as a measure of similarity between X and Y but only in the neighbourhood of the joint space. This results from the restriction of Gaussian kernels, which have high values only along the $x \approx y$ line with exponential fall off otherwise. The size of this neighbourhood is therefore controlled by the kernel width parameter σ. As a result, correntropy can also be defined as the integral of the joint probability density along the line $x = y$:

$$V_{XY}(X, Y) \approx \int p(x, y) \mid_{x=y=u} du \quad (11)$$

The joint PDF can be estimated from the data using the Parzen window method:

$$p(x, y) \approx \frac{1}{N} \sum_{i=1}^{N} G(x - x_i, \sigma^2 I) G(y - y_i, \sigma^2 I) \quad (12)$$

By integrating the above along the $x = y$ line and using the convolution property of Gaussian functions again, the estimate of correntropy is finally obtained as:

$$V_{XY}(X, Y) \approx \frac{1}{N} \sum_{i=1}^{N} G(x_i - y_i, 2\sigma^2 I) \quad (13)$$

The correntropy can thus be regarded as the PDF of equality of two variables in the neighbourhood of the joint space, of the size determined by the kernel width parameter σ [6], [15]. The measure has many interesting properties and one of them is that for independent X and Y it can be approximated by the Information Potential formula similar to (7) and named Cross Information Potential [15]. Correntropy has been successfully used as a localized, outlier–resistant similarity measure for many supervised learning applications [16], [15], [17], [18].

4 Density Preserving Sampling (DPS) procedure

Both cross–validation and bootstrap, described in sections 2.2 and 2.3 are stochastic methods. The immediate consequence is that the results can vary a lot from one run to another and there is no guarantee that the datasets obtained by splitting the original data are representative, which is a necessary condition for obtaining accurate error estimates. For this reason, in order to obtain reliable results, averaging over multiple iterations is required. In general, the more times the procedure is repeated the better, as in the limit both methods will converge to the true error values. For $k-$fold cross–validation using $m-$element dataset this could mean averaging over all $\binom{m}{m/k}$ possibilities of choosing m/k instances out of m (the so called 'complete cross–validation' [4]), which quickly becomes untractable. There is however another, often overlooked possibility – intelligent sampling aiming at producing only representative splits.

From statistics, a random sample is considered representative if its characteristics reflect those of the population from which it is drawn [19]. Since these characteristics are reflected by the probability density function, the more similar the distribution of the sample to the distribution of the population, the more representative this sample is. The correntropy described in section 3.2 can be used to measure the similarity between two distributions and thus to measure the 'representativeness' of the sample. Moreover, it is also possible to use correntropy as an optimization criterion, guiding the sampling process in order to split a given dataset into two or more maximally representative subsets.

4.1 Estimation of correntropy for unsorted datasets with different cardinalities

Equation 13 defines correntropy between two random variables or datasets X and Y as the value of a Gaussian kernel centered at $(x_i - y_i)$ averaged over all N instance pairs. There are thus three requirements for calculation of correntropy to be possible: the datasets (1) must be ordered, (2) must have the same dimensionality and (3) must have the same number of objects. While the second requirement is irrelevant for sampling, as each subset of objects necessarily needs to have the same dimensionality as the set from which it has been selected, the remaining two requirements may pose a problem.

For some applications like e.g. supervised learning, all the above requirements are met automatically – if

X denotes the output of a mapper and Y denotes the target value, $|X| = |Y|$ and x_i is the prediction of y_i. In sampling however in general one cannot expect the instances to be ordered, which means that it is not obvious on the difference of which instances to center the Gaussians. Moreover, the datasets may have different cardinalities e.g. when one wants to calculate the correntropy between the original dataset and its subset.

To address the ordering issue the following approach is adapted. For every instance $x_i, i \in (1..N)$, the Gaussian is centred at $(x_i - y_j)$, such that:

$$j = \underset{j}{\operatorname{argmin}} \parallel x_i - y_j \parallel, j \in (1..N) \tag{14}$$

where $\parallel \cdot \parallel$ denotes the Euclidean norm. In other words y_j is selected to be as close to x_i as possible. Both x_i and y_j are then removed from their respective sets and the procedure is repeated until all instances are exhausted. The generalized, instance ordering insensitive formula for calculation of correntropy thus becomes:

$$V_{XY}(X,Y) \approx \frac{1}{N}\sum_{i=1}^{N} G(x_i - y_j, 2\sigma^2 I)$$
$$j = \underset{j}{\operatorname{argmin}} \parallel x_i - y_j \parallel, \quad j \in J_{avail} \tag{15}$$

where the set J_{avail} contains indices of y which haven't yet been used, to ensure that each y_k is used only once.

When the datasets have different cardinalities, that is without loss of generality if $N_X > N_Y$, the approach outlined above will terminate after N_Y instances are processed. To avoid this, a new dataset Y_N is created by duplicating the original Y dataset $\lceil N_X/N_Y \rceil$ times. Correntropy is then calculated between X and Y_N and the calculation will terminate after exactly N_X steps.

For the correntropy values to be more comparable for different experiments, we scale $V_{XY}(X,Y)$ to fit into the $0..1$ range, by dividing each $G(x_i - y_j, 2\sigma^2 I)$ in the sum in Equation 15 by $G(0, 2\sigma^2 I)$. Every correntropy value given in the rest of this paper has been scaled. Note however, that the correntropies should be compared with caution as their absolute difference can be made almost arbitrarily large by manipulating the Parzen window width parameter σ. For this reason the correntropy values given should be seen as ranks of various models/solutions on an ordinal scale.

4.2 Correntropy based sampling procedure

In this section we propose a correntropy–based, hierarchical, binary density preserving splitting procedure. The correntropy given by Equation 13 is a function differentiable with respect to both x_i and y_i, which unfortunately is not the case for the generalized function given by Equation 15. Moreover, none of them is differentiable with respect to the indices i and j, which are the only variables that can be manipulated within the splitting process. Gradient driven optimization procedure is thus not straightforward hence we have reverted to a greedy, locally optimal approach.

Since correntropy is being estimated by a scaled sum of Gaussians, it reaches a maximum when all components of the sum reach their maximal values. In case of a single Gaussian function, the maximum is reached at 0 and since the function is piecewise monotonic and symmetric, the closer x_i and y_j are in Equation 15, the higher $V_{XY}(X,Y)$ will be. This immediately suggests an iterative, binary splitting procedure of a dataset Z into datasets X and Y, which at each step selects two instances z_i and z_j so that:

$$i, j = \underset{i,j}{\operatorname{argmin}} \parallel z_i - z_j \parallel \tag{16}$$

and then adds them to the sets X and Y, so that $X = X \cup z_i$ and $Y = Y \cup z_j$ or the other way round, removing them from dataset Z at the same time. The above procedure aims at directly maximizing $V_{XY}(X,Y)$, that is the correntropy between the two new datasets. Due to the way correntropy is calculated for sets with various cardinalities however, it also indirectly maximizes $V_{XZ}(X,Z)$ and $V_{YZ}(Y,Z)$. As a result, newly obtained datasets are splits with distributions maximally similar to each other and to the distributions of the original dataset. To obtain more than 2 splits, the procedure can be repeated by splitting datasets X and Y again, which will produce 4 splits and so on. The total number of splits is thus always a power of 2.

The instances z_i and z_j can be added to the sets X and Y arbitrarily or not. In our approach we have devised a procedure in which the two objects are distributed in a way that maximizes the average coverage of the input space by both splits. Denoting by d_{kV} the average Euclidean distance between instance z_k and all instances in set V, the rules are:

$$d_{iX} + d_{jY} \geq d_{jX} + d_{iY} \Rightarrow X = X \cup z_i,\ Y = Y \cup z_j \tag{17}$$
$$d_{iX} + d_{jY} < d_{jX} + d_{iY} \Rightarrow X = X \cup z_j,\ Y = Y \cup z_i \tag{18}$$

For classification problems, the splitting procedure can be executed in either supervised or unsupervised mode. In the former case, the algorithm takes advantage of the class labels supplied with the data by considering each class in turn and in separation from the rest. In other words the dataset is being split class by class. We refer to this approach as DPS–S. In the unsupervised mode, the class labels are ignored, so the procedure is purely density–driven and has been called DPS–U. Similar remark applies to estimation of correntropy, which can also be calculated in a class–wise (supervised) or class–less (unsupervised) mode.

In current implementation, if the classes are too small to be divided into a given number of subsets, DPS–S automatically falls–back to DPS–U. Since the splitting procedure is hierarchical, every dataset can be divided using the supervised approach up to some point, after which the unsupervised procedure will take over. Note, that this a different behavior than described in [7], where for the sake of experiments classes with less than 16 objects been dropped from the datasets.

The computational complexity of the DPS approach is of the order $O(N^2/2)$ in the unsupervised case, as the most time consuming operation is calculation of pairwise distances between all N instances in a form of a symmetric distance matrix. For supervised DPS, the complexity is of order $O(\sum_i N_i^2/2)$, where N_i is the cardinality of the i^{th} class. This is however negligible when compared to the complexity of most training algorithms.

5 EXPERIMENTS

The experiments have been conducted on 27 publicly available datasets using a total of 16 different classifiers. The datasets used come from the UCI Machine Learning Repository [20], the ELENA database [21] and the PRTools Pattern Recognition Toolbox for MATLAB [22]. The details of datasets are given in Table 1. The star symbol in the '#obj/attr' column denotes the number of instances actually used in the experiment, sampled randomly from the whole, much bigger dataset in order to keep the experiments computationally tractable. The classifiers used are implemented within the PRTools toolbox and their list is given in Table 2.

The experiments were designed to (a) compare the error estimation accuracy of cross–validation (CV) and density preserving sampling approaches (DPS), (b) test the stability of both error estimators, (c) test applicability of DPS to the classifier selection process, (d) check, if it is possible to reliably estimate the generalization error using a single DPS fold only, thus reducing the computational requirements by another order of magnitude, and (e) examine the behavior of DPS in the context of ensemble models, in comparison to cross–training.

We have followed a similar approach to that outlined in [4]. For each dataset a stratified random subsampling procedure has been repeated 100 times, resulting in 100 random divisions of the dataset into a training part $(2/3)$ and independent test data $(1/3)$. The training part was then used to estimate the generalization error using CV and DPS for each classifier, while the independent test part has been used to calculate the 'true' generalization error, once again for each classifier in turn. The true generalization error then served to calculate the bias of each estimate, while the generalization error estimates of a single estimation run have been used to calculate the variance. Finally, the results have been averaged over all 100 runs of the random subsampling procedure.

The CV estimate has been calculated within a 10 times repeated 8–fold cross–validation scheme. We provide the average results for all 10 iterations as well as the result of the best and worst single run in terms of bias/variance to emphasize how wrong the things can go with CV.

Three 8–fold DPS estimates are also given – DPS–S (using class label information), DPS–U (ignoring class label information) and DPS–SU (averaged over the two[1]).

1. The averaging method used here is different than the one used in [7]. Namely we are now averaging the errors of classifiers trained on DPS–S and DPS–U folds rather than combining their outputs.

TABLE 1
Dataset details

abbr	name	source	#obj/attr	#class
azi	Azizah dataset	PRTools	291/8	20
bio	Biomedical diagnosis	PRTools	194/5	2
can	Breast cancer Wisconsin	UCI	569/30	2
cba	Chromosome bands	PRTools	1000*/30	24
chr	Chromosome	PRTools	1143/8	24
clo	Clouds	ELENA	1000*/2	2
cnc	Concentric	ELENA	1000*/2	2
cnt	Cone–torus	[23]	800/3	2
dia	Pima Indians diabetes	UCI	768/8	2
ga2	Gaussians 2d	ELENA	1000*/2	2
ga4	Gaussians 4d	ELENA	1000*/4	2
ga8	Gaussians 8d	ELENA	1000*/8	2
gla	Glass identification data	UCI	214/10	6
ion	Ionosphere radar data	UCI	351/34	2
iri	Iris dataset	UCI	150/4	3
let	Letter images	UCI	1000*/16	26
liv	Liver disorder	UCI	345/6	2
pho	Phoneme speech	ELENA	1000*/5	2
sat	Satellite images	UCI	1000*/36	6
seg	Image segmentation	UCI	1000*/19	7
shu	Shuttle	UCI	1000*/9	7
son	Sonar signal database	UCI	208/60	2
syn	Synth–mat	[24]	1250/2	2
tex	Texture	ELENA	1000*/40	11
thy	Thyroid gland data	UCI	215/5	3
veh	Vehicle silhouettes	UCI	846/18	4
win	Wine recognition data	UCI	178/13	3

TABLE 2
Classifier list

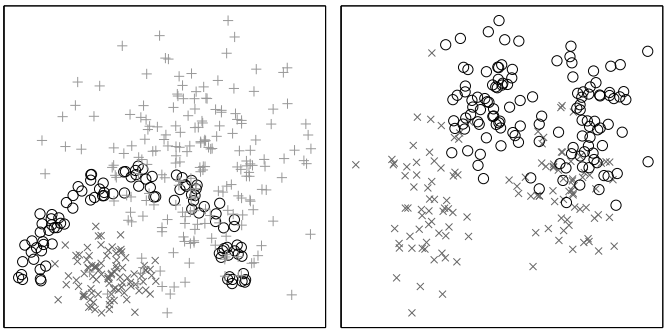
name	description
fisherc	Fisher's Linear Classifier
ldc	Linear Bayes Normal Classifier
loglc	Logistic Linear Classifier
nmc	Nearest Mean Classifier
nmsc	Nearest Mean Scaled Classifier
quadrc	Quadratic Discriminant Classifier
qdc	Quadratic Bayes Normal Classifier
udc	Uncorrelated Quadratic Bayes Normal Classifier
klldc	Linear Classifier using KL expansion
pcldc	Linear Classifier using PC expansion
knnc	K–Nearest Neighbor Classifier
parzenc	Parzen Density Based Classifier
treec	Decision Tree Classifier
naivebc	Naive Bayes Classifier
nusvc	Support Vector Classifier with linear kernels
rbnc	Radial Basis Function Neural Network Classifier

5.1 Toy problems

The analysis starts with two synthetic datasets first used in [23] and [24]. The datasets have been chosen as they are both two–dimensional, which allows for visualisation of the results and have been extensively used in our previous studies due to their well known properties.

5.1.1 Cone–torus dataset

Cone–torus is a synthetic 2 dimensional dataset consisting of 3 classes, with data points generated from 3 differently shaped distributions: a cone, half a torus, and a normal distribution. A scatter plot of the dataset is given in Figure 1(a). Figure 2 depicts scatter plots of 8 DPS–S folds, while in Figure 3, 8 CV folds generated



(a) Cone–torus (b) Synth–mat

Fig. 1. Synthetic datasets

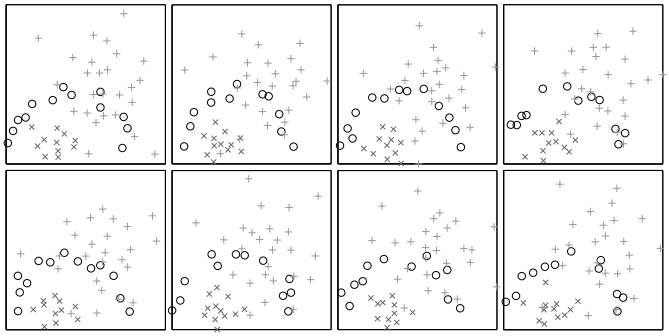


Fig. 2. Cone–torus – scatter plots of 8 DPS–S folds

during a single random run are given. Note, that in case of DPS, the classes tend to preserve their shapes – the half torus for example is clearly visible in 7 out of 8 folds, while for CV only in 4 or 5. This is also well reflected by the mean value of correntropy between all 8 folds and the original dataset, which is 0.81 for DPS and 0.71 for CV averaged over 10 runs ($\sigma = 0.12$).

The decision boundaries for the qdc classifier trained on each of 8 folds in turn, superimposed on the original dataset have been given in Figure 4. The black solid line represents the boundaries of a classifier trained using the DPS–S folds, while the blue dotted line shows the boundaries for a single CV run. Notice, that for DPS the decision boundaries generally do not change their shape from one fold to another, as opposed to CV, where the boundaries seem very unstable and can change radically.

5.1.2 *Synth–mat dataset*

The Synth–mat dataset is a 2 dimensional mixture of 4 normal distributions and has been presented in Figure 1(b). Both classes have bimodal distribution – there are two Gaussians in each of them. The mean value of correntropy between all 8 folds and the original dataset is 0.75 for DPS and 0.66 for CV averaged over 10 runs ($\sigma = 0.12$). Since the scatter plots of all DPS and CV folds were already presented for the Cone–torus dataset and not much changes here, only the decision boundaries of a classifier trained as previously have been given in Figure

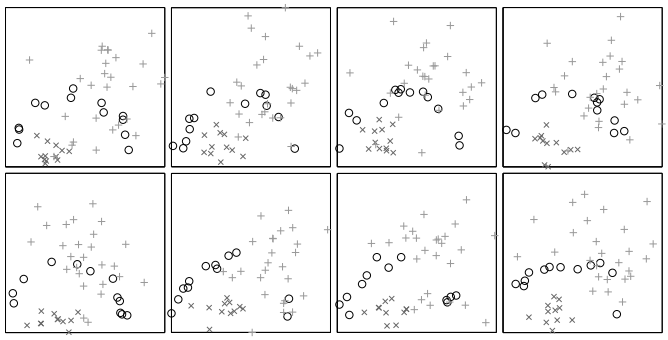


Fig. 3. Cone–torus – scatter plots of 8 CV folds

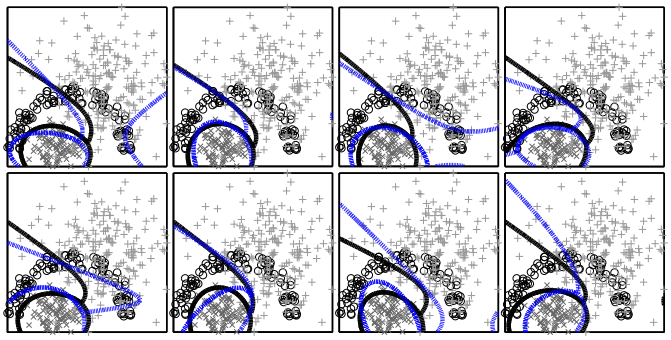


Fig. 4. Cone–torus – decision boundaries for qdc trained on DPS–S (solid line) and CV (dotted line) folds

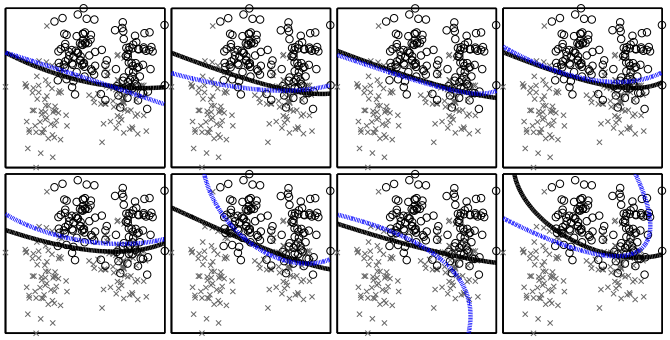


Fig. 5. Synth–mat – decision boundaries for qdc trained on DPS–S (solid line) and CV (dotted line) folds

5. Once again the boundaries appear stable for DPS and differ a lot from one CV fold to another.

5.2 Benchmark datasets

5.2.1 *Correntropy*

Figure 6 presents the values of averaged correntropy between the original dataset and 8 folds generated using DPS and CV, for all 27 datasets used in the experiment. Note, that although the correntropy has been normalized to the $0 \div 1$ range, according to our earlier argument the values represent an ordinal scale. Also, the Gaussian kernel width used for each dataset has been chosen to optimize the correlation between bias and correntropy, as described in Section 5.2.5.

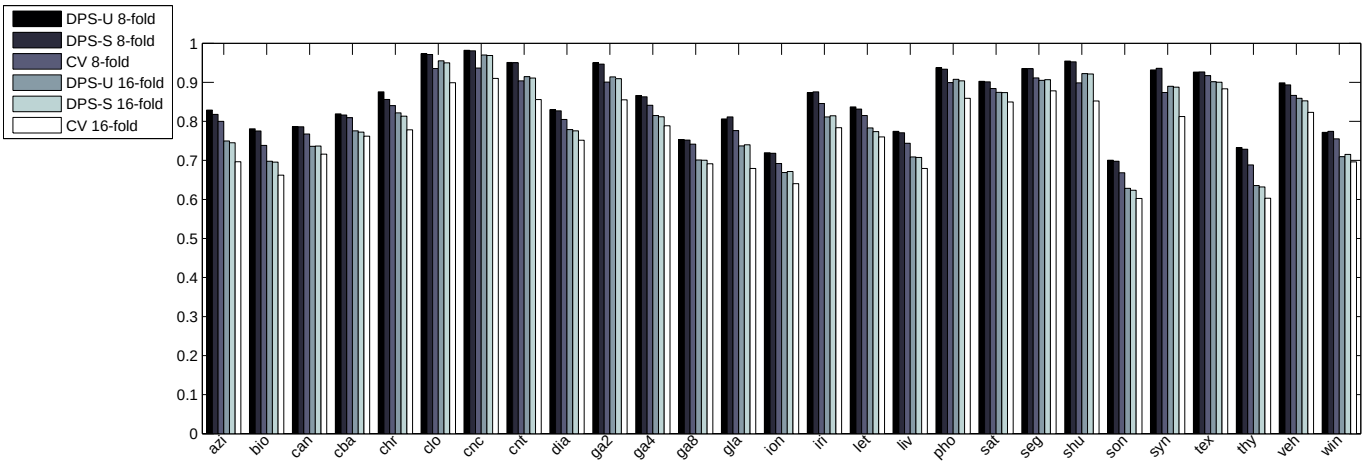


Fig. 6. Mean correntropy between each fold and the original dataset

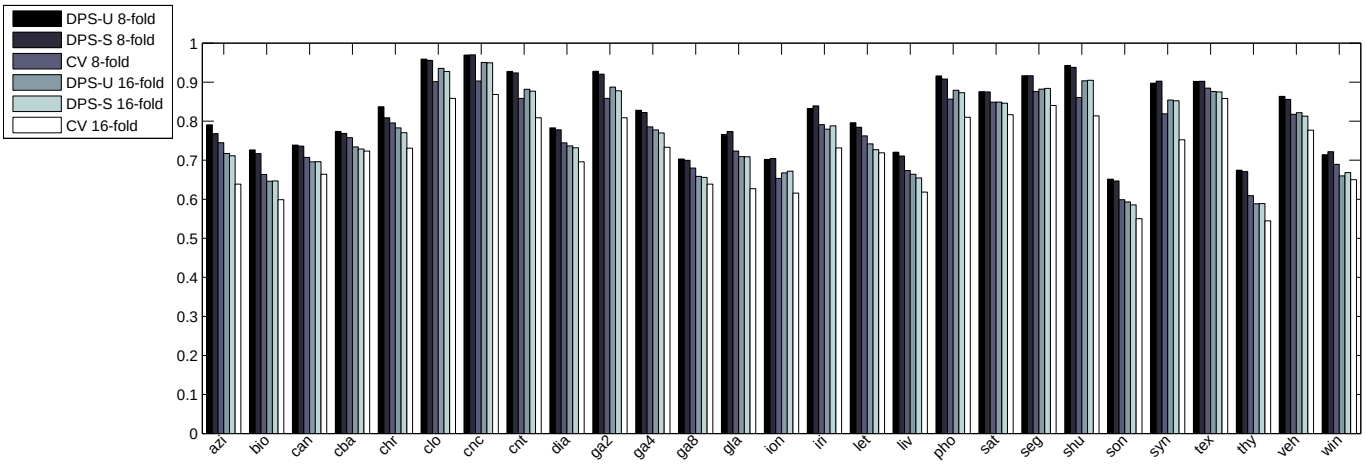


Fig. 7. Mean between–fold correntropy

The correntropy between the DPS folds and the original dataset is always higher than in the case of the CV folds, regardless of the number of folds (8 or 16). This is not surprising since the DPS splits have been obtained by maximization of correntropy. The picture is very similar for the between–fold correntropy depicted in Figure 7, where DPS is again an unquestionable leader.

5.2.2 Bias

The mean absolute bias for both DPS and CV can be seen in Figures 8 and 9. The DPS approach has a bias comparable to the mean CV result, with slight advantage of the latter for roughly half of the datasets. Note however, that the DPS estimates are never as biased as the worst–case CV scenario, yet the result has been achieved with 10 times less computations.

A summary of the results can be found in Table 3, where a mean value and standard deviation of bias (and variance) across all datasets and classifiers for each error estimation method has been given. Both DPS–U and DPS–S have on average the same bias with a tiny difference in its standard deviation. DPS–SU on the other hand comes very close to the repeated cross–validation, which is a result of combining both supervised and unsupervised methods. Note, that this combination does not require additional computations in order to obtain the splits, as all pairwise within–class distances form a subset of all pairwise distances for the whole dataset, which are calculated anyway by the unsupervised DPS. All DPS approaches also have mean bias and standard deviation lower than the worst–case CV scenario.

5.2.3 Variance

The variance of error estimates can be seen in Figure 10 (averaged over all classifiers) and Figure 11 (averaged over all datasets). Out of all three DPS approaches, once again DPS–SU demonstrates the best performance with average variance lower by 0.0130 than the best–case CV scenario (Table 3), while DPS–S performs at the level best–case CV and DPS–U still outperforms 10 times repeated cross–validation. Note, that both in terms of variance, DPS–S outperforms DPS–U and is additionally computationally cheaper (see Section 4.2). As a result good error estimation can be achieved with roughly 10%

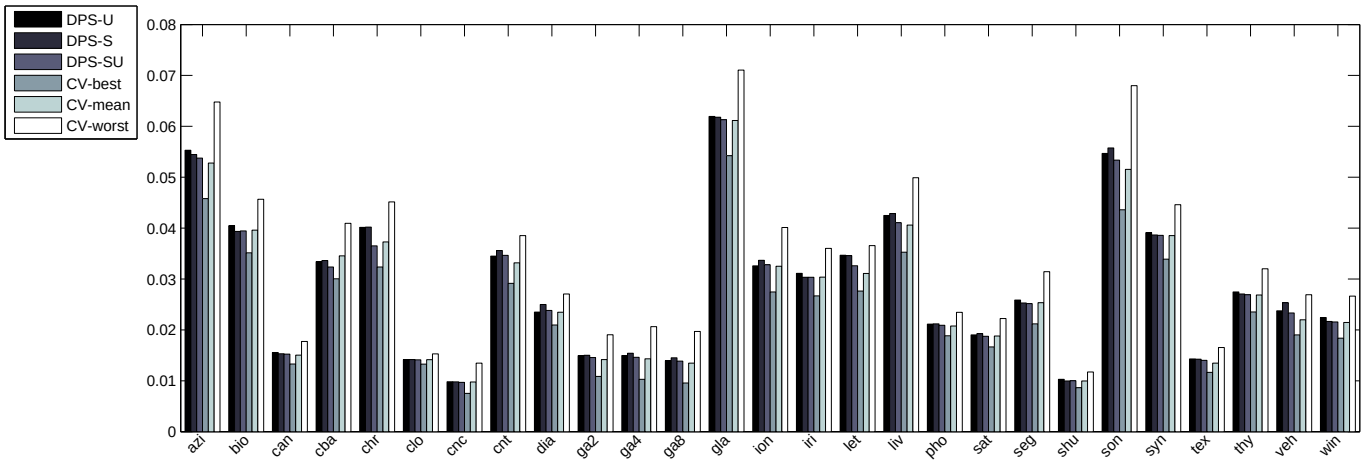


Fig. 8. Mean absolute bias (averaged over all classifiers)

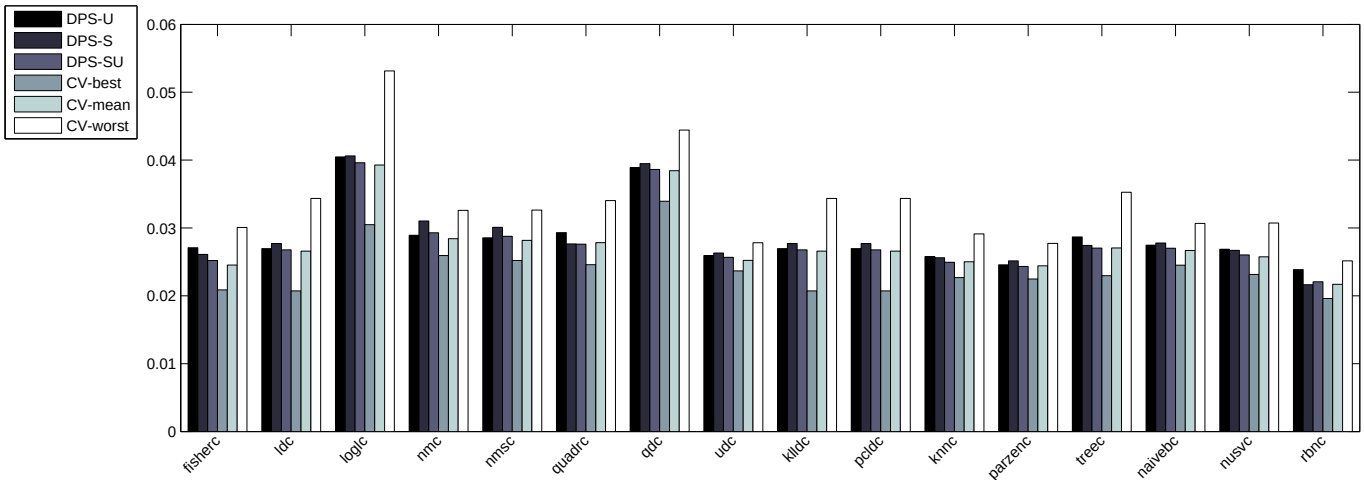


Fig. 9. Mean absolute bias (averaged over all datasets)

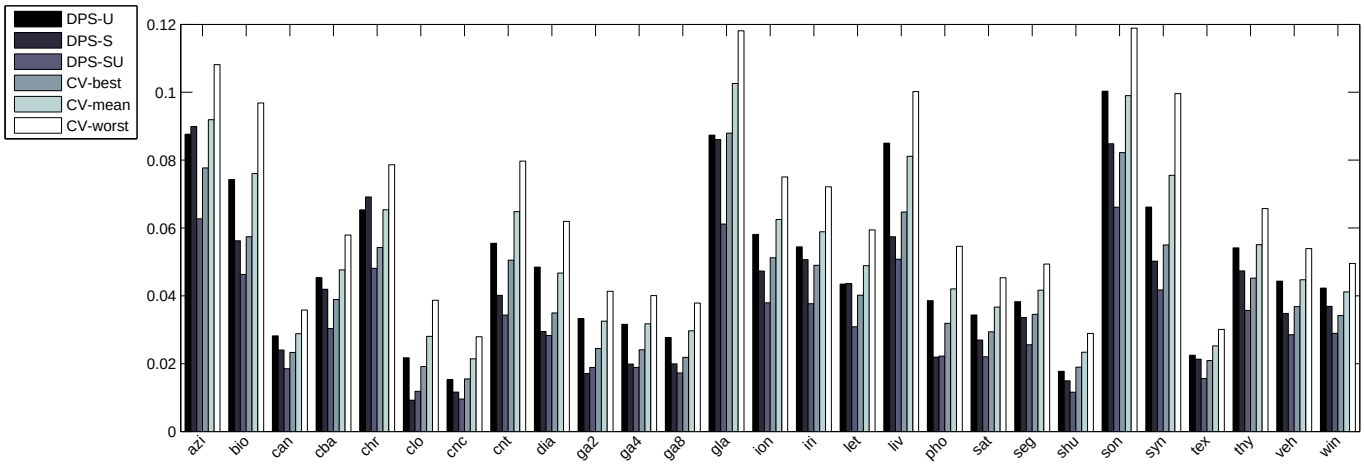


Fig. 10. Standard deviation of error estimate (averaged over all classifiers)

of computations required by 10 times repeated CV. For best results however one should resort to DPS–SU, which seems to stabilize the error estimates but requires about 20% of the computational time of CV.

5.2.4 *Classifier selection*

Selection of a single best model from a group of available models is an important problem in machine learning.

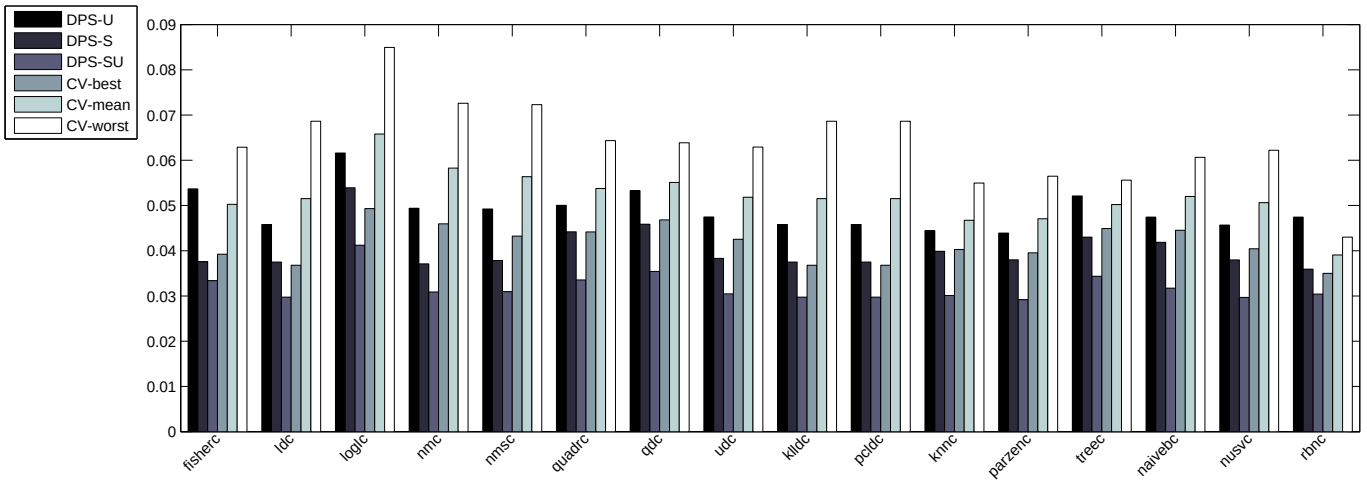


Fig. 11. Standard deviation of error estimate (averaged over all datasets)

TABLE 3
Bias and variance summary

	DPS U	DPS S	DPS SU	CV best[a]	CV mean	CV worst
BIAS-mean	0.0281	0.0281	0.0275	0.0239	0.0273	0.0326
BIAS-stdev	0.0201	0.0202	0.0198	0.0164	0.0194	0.0242
VAR-mean	0.0597	0.0504	0.0394	0.0524	0.0627	0.0743
VAR-stdev	0.0332	0.0327	0.0229	0.0304	0.0345	0.0397

a. 'CV best' denotes the best cross–validation run out of 10 for each dataset/classifier pair in terms of lowest bias/variance. For CV the division of data which produced the lowest bias did not in general produce the lowest variance. Similar remarks apply to 'CV worst'.

A typical selection criterion is the generalization error estimated using cross–validation. The ranking of top 3 classifiers according to both CV and DPS for all datasets was given in Table 4. Note, that the overall ranking for all datasets is exactly the same for both error estimators, and reflects the ranks based on the true generalization error. The differences are however apparent when the results for each dataset are examined separately.

The last three rows in the table denote the number of datasets out of 27, for which the true top classifier was included in top 1, top 2 and top 3 classifiers according to each error estimation method. For CV, the best classifier has been correctly identified 19 times and has been included in the top 2 and top 3 classifiers 25 times. For the best DPS approach (DPS–SU) the numbers are very similar – 20, 23 and 25.

The correlation coefficients for different error estimates and the true generalization error have been given in Table 5. As shown, all tested error estimators are strongly correlated with the true error, and although there are some small differences, the correlation coefficient is never lower than 0.959.

5.2.5 Correlation between correntropy and bias

The ability to estimate the generalization error using a single DPS fold only would allow to reduce the computa-

TABLE 4
Ranking of top 3 classifiers

dataset	true	DPS–U	DPS–S	DPS–SU	CV
azi	11 14 12	11 12 2	11 12 14	11 12 2	11 12 14
bio	6 7 8	6 7 8	6 7 8	6 7 8	6 7 8
can	12 2 9	12 2 9	15 12 2	15 12 2	12 15 2
cba	12 2 9	12 2 9	12 2 9	12 2 9	12 2 9
chr	14 8 11	14 8 11	14 8 2	14 8 11	14 8 11
clo	15 1 2	1 2 3	15 1 2	15 1 2	15 1 2
cnc	1 15 3	1 15 2	1 15 5	1 15 5	1 15 5
cnt	16 12 13	16 12 7	12 16 13	16 12 13	16 12 13
dia	1 3 2	2 9 10	1 11 2	1 2 9	1 3 2
ga2	1 2 3	15 4 2	2 3 9	2 3 9	3 1 2
ga4	1 2 3	1 4 2	15 5 2	15 1 2	15 4 3
ga8	16 3 1	16 1 5	16 5 4	16 5 1	16 15 5
gla	3 15 2	2 9 10	2 9 10	2 9 10	2 9 10
ion	14 11 7	14 7 11	14 13 7	14 13 7	14 7 11
iri	2 9 10	2 9 10	2 7 9	2 9 10	2 9 10
let	11 12 7	12 11 2	12 11 2	12 11 2	12 11 2
liv	1 3 2	1 3 2	11 1 3	1 3 11	1 3 11
pho	11 16 12	16 12 11	11 12 16	11 16 12	11 12 16
sat	11 12 2	11 12 14	11 12 2	11 12 2	11 12 7
seg	11 3 2	3 11 2	11 2 9	11 3 2	3 11 2
shu	13 1 16	13 16 1	13 1 16	13 16 1	13 1 16
son	12 11 15	12 11 14	12 11 14	12 11 14	12 11 2
syn	14 12 16	14 12 16	12 14 16	14 12 16	12 14 16
tex	2 9 10	2 9 10	2 9 10	2 9 10	2 9 10
thy	8 15 7	15 6 7	7 8 11	15 7 8	15 8 11
veh	7 6 3	6 7 2	7 6 2	7 6 2	7 6 2
win	7 6 2	6 1 4	4 6 7	6 4 7	6 7 4
overall	2 9 10	2 9 10	2 9 10	2 9 10	2 9 10
in top 1	27/27	17/27	17/27	20/27	19/27
in top 2	27/27	20/27	23/27	23/27	25/27
in top 3	27/27	21/27	24/27	25/27	25/27

TABLE 5
Correlation between true generalization error and error estimates

correlation	DPS–U	DPS–S	DPS–SU	CV
per dataset (8 folds)	0.9594	0.9657	0.9676	0.9710
per classifier (8 folds)	0.9967	0.9975	0.9976	0.9973
per dataset (16 folds)	0.9640	0.9646	0.9671	0.9695
per classifier (16 folds)	0.9964	0.9969	0.9969	0.9969

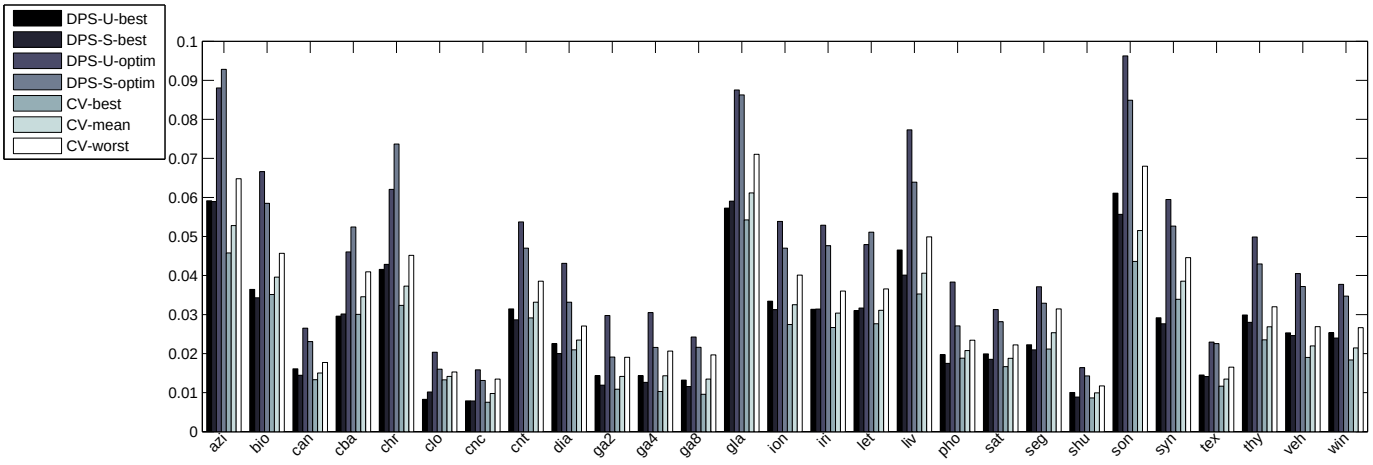


Fig. 12. Bias of DPS error estimate calculated using a single fold (averaged over all classifiers)

tional cost of the estimation procedure by another order of magnitude, when compared to 10 times repeated CV. Figure 12 depicts the bias of the estimate calculated using a single DPS fold, which has been chosen on the basis of the lowest bias itself ('DPS–best'). Although in practice this kind of selection procedure is useless, it shows that the method has some potential as for most datasets the bias is comparable with the one obtained using 10 times repeated cross–validation or even the best–case CV scenario. The problem however is how to choose the appropriate DPS fold. The value of correntropy seems to be an obvious choice. Note however, that there is no principled way of selecting the width σ of the Gaussian kernel for estimation of correntropy and the estimated value can vary greatly depending on the choice of σ. We have therefore decided to check the correlation between bias and the values of correntropy. The experiment was conducted for 8 and 16 DPS–S folds and the results can be seen in Figure 13. Note, that for the sake of calculating the correntropy, σ was chosen using an exhaustive search in order to optimize the correlation. In other words, the results given in Figure 13 represent the best–case scenario, for the most optimal kernel width, which in practice is not known a priori. As it can be seen, the correlation varies from about -0.1 to -0.6 depending on the dataset. The bias of an estimate obtained using a single DPS fold chosen on the basis of highest correntropy is always higher even in comparison to the worst–case CV scenario bias ('DPS–optim' in Figure 12). The estimate of correntropy is only slightly to moderately correlated with the bias of the error estimate, even for an optimal choice Gaussian kernel width. As a result it cannot be used to select a single best fold which would minimize the bias, although some other divergence measures might be appropriate for this task.

5.2.6 Combining classifiers

In this experiment a simple ensemble model based on the majority voting rule was built. It is believed, that the classifiers used in a combination should be diverse, which enforces complementarity of the ensemble members [25]. One way to enforce this diversity is cross–training, a technique based on cross–validation, which combines all models obtained during a single or repeated CV run. For this experiment the two synthetic datasets described in Section 5.1 have been used. Both datasets were split into 8 folds using DPS–S and CV and then for each classifier listed in Table 2 an ensemble model was built by combining 8 models trained on all but one fold in turn and using the majority voting rule. For CV this procedure has been repeated 10 times. Each combination was then tested on an independent test set. In order to monitor performance of the combinations, a single control model trained using all 8 folds was also used.

The results have been depicted in Figures 14 and 15. In most cases, combinations based on DPS folds do not improve on the performance of a single control model. This was expected, as for each classifier all 8 ensemble members should be very similar, since they were all trained using representative subsets of data. For the combinations based on CV, some improvement can be observed even in the worst case scenario.

In order to illustrate this issue, discrete error distribution plots showing the probability of various numbers of ensemble members being in error at the same time have been given in Figures 16 and 17. The classifiers used to produce these plots (qdc and treec) have been chosen primarily for illustrative purposes. The area of the shaded region in each figure represents the error of the combined model. Usually the number of models in majority voting is chosen to be odd, so that there are no ties. In our case there are however 8 models, so the ties were resolved randomly.

Note, that for DPS most of the mass is concentrated in the corners of the plots, meaning that the classification decisions are taken unanimously in most cases, proving that the classifiers are indeed very similar.

In case of CV the situation is different. In Figure 16

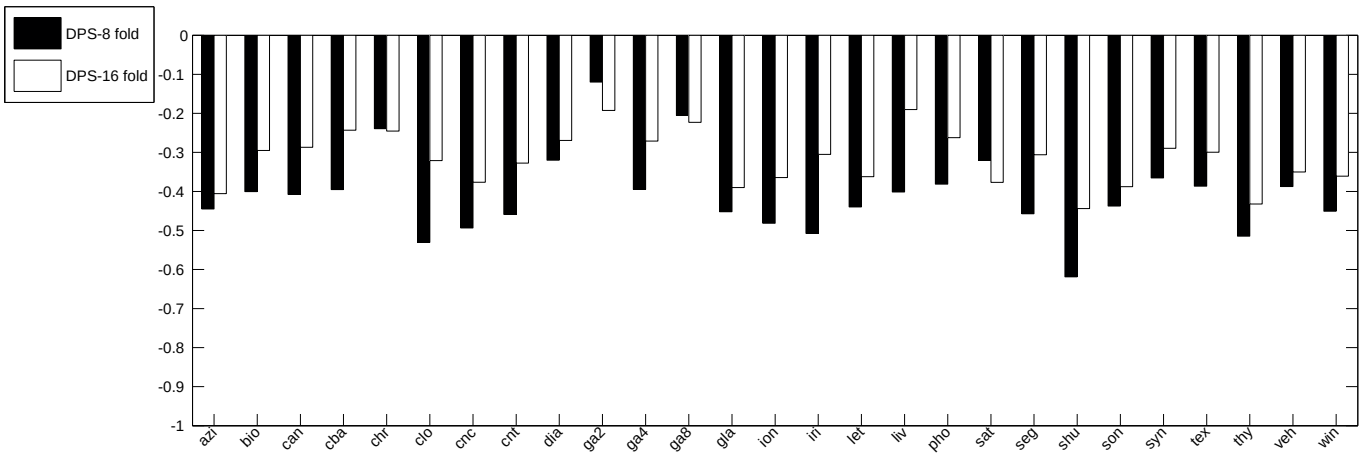


Fig. 13. Correlation between bias and correntropy

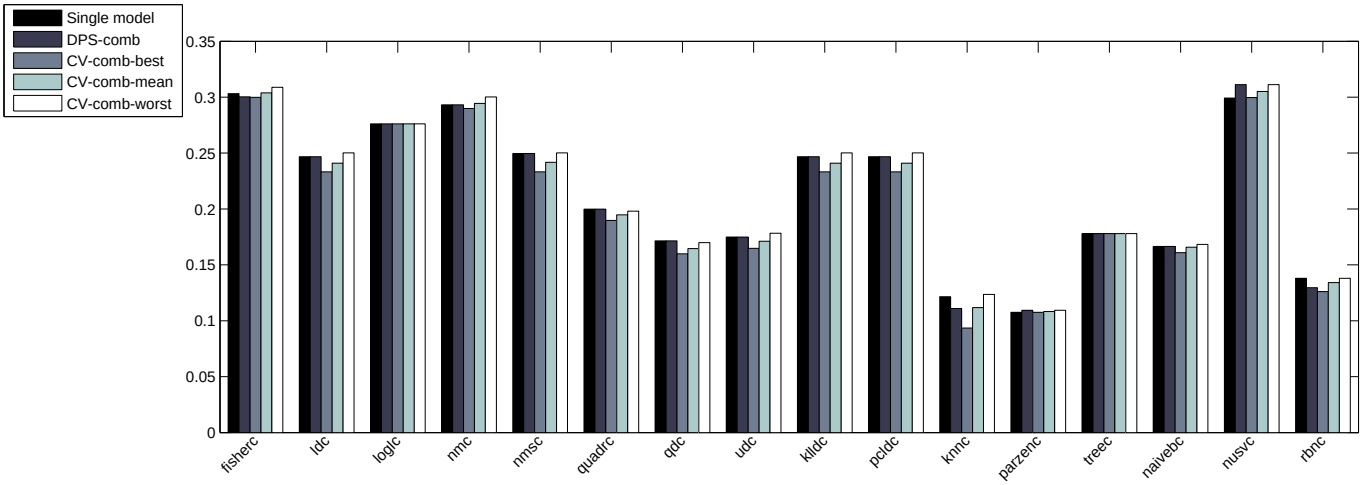


Fig. 14. Single model v. combination errors for Cone–torus dataset

for example some mass is scattered all over the plot, meaning that there are situations when the classifiers tend to disagree thus demonstrating complementarity. As it can be seen, the stochastic nature of CV has in this case a very positive effect on the performance by introducing diversity to the ensemble. This result also confirms, that if the goal is to select a single best model, it is much safer to use DPS as this minimizes the risk of choosing a bad model due to the stability of decision boundaries as discussed before. From the point of view of diversity required for ensemble models however, this feature of DPS becomes a disadvantage and it's usually much better to use a stochastic method instead.

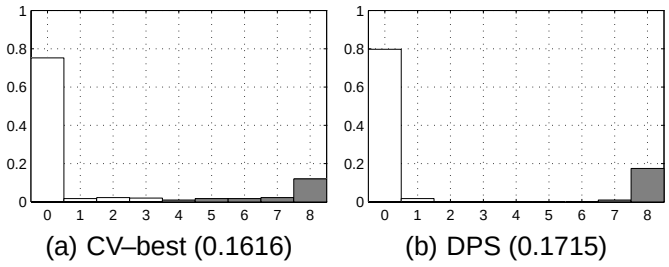


Fig. 16. Discrete Error Distributions for Cone–torus dataset and qdc (error rates given in brackets)

6 Discussion

The presented Density Preserving Sampling procedure is a very attractive alternative for the commonly used cross–validation technique for a number of reasons.

For the purpose of the generalization error estimation, k–fold cross–validation is without a doubt the most widely and commonly used technique, due to its universal character, simplicity and effectiveness. Its stochastic nature however requires the estimation to be repeated multiple times for different random divisions of the data, in order to circumvent the risk of obtaining the best/worst–case scenario estimate, which as demonstrated in this paper can be highly biased and can have a large variance. The need for running the procedure multiple times makes it computationally expensive, forcing the researchers to seek compromise elsewhere,

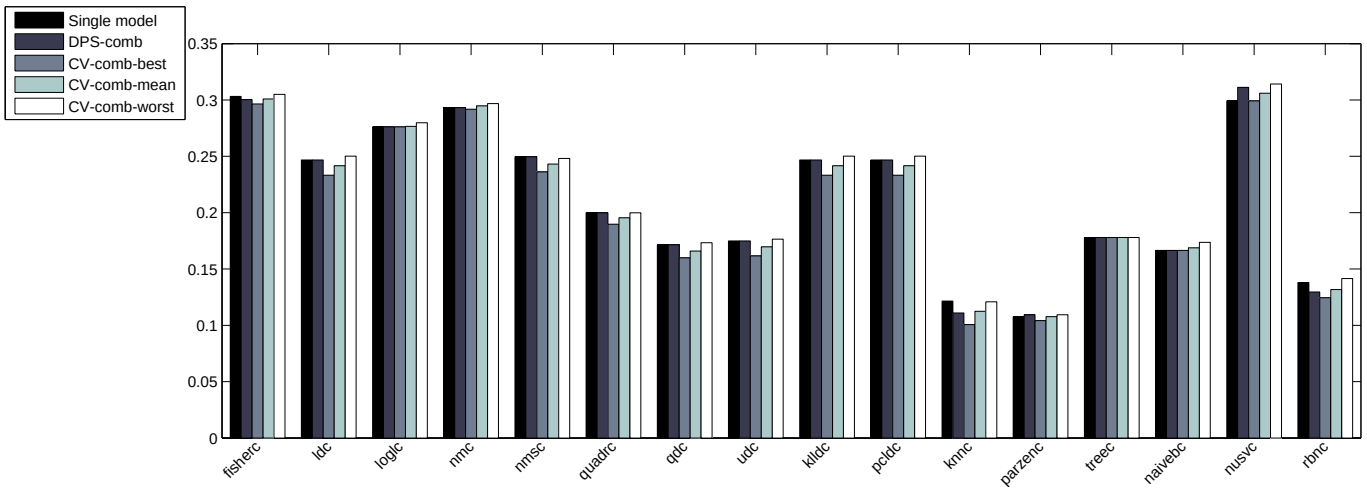
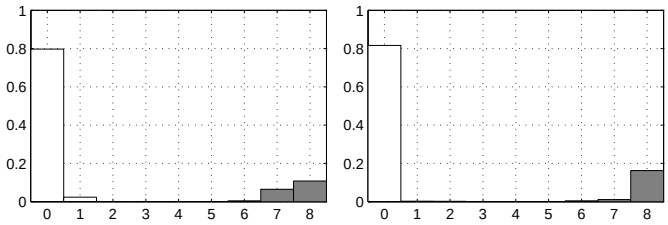


Fig. 15. Single model v. combination errors for Synth–mat dataset



(a) CV–best (0.1790) (b) DPS (0.1790)

Fig. 17. Discrete Error Distributions for Synth–mat dataset and treec (error rates given in brackets)

for example by not calculating the full gradient during optimization or taking other shortcuts, which negatively influence the solution of the problem. The DPS procedure proposed in this paper is however deterministic. It thus does not need to be repeated in order to improve the quality of the error estimate, at the same time producing results comparable to repeated cross–validation when it comes to bias, and superior to CV in terms of the variance of obtained estimates. Yet it all happens at 5–10 times lower computational cost.

Another related application area of CV is parameter estimation. Since for some models the objective function is not differentiable wrt. all its parameters, the optimization procedure must resort to a search in the parameter space. One example of such situation is the k–NN classifier, for which the number of nearest neighbours k is usually being set by testing a number of possible values using cross–validation. In such case, as the search itself might be very costly depending on the dimensionality of the search space, the cross–validation is usually not being repeated in order to save computations. As before, due to the non–deterministic nature of CV, this can lead to suboptimal decisions based on highly biased performance estimates (worst–case scenario). Note, that it also applies to other algorithms requiring calculation of performance estimates repeated many times like e.g. feature selection. The benefit of using DPS rather than CV in these scenarios can be tremendous.

In case of some machine learning methods it is a common practice to cross–train multiple models and select the best performing one. The cross–training procedure is analogous to cross–validation, with the difference that the obtained models instead of being discarded, are considered as candidates for a final solution. This applies especially to models like decision trees, which cannot be retrained using the full dataset due to their instability. The danger here is the combination of a relatively unstable error estimation procedure (see plots of the decision boundaries in Figures 4 and 5) with an unstable learning method, which in an unfavorable case may lead to selection of one of the worst models rather than the best. On the other hand, models trained using various DPS splits will likely be much more similar to each other, as shown in Section 5.2.6, minimizing the risk and cost of incorrect choice.

The final possible application of random sampling procedures we want to discuss here is early stopping, a technique widely used in training of universal approximators to prevent overfitting. In this approach a randomly selected subset of the data is used for continuous monitoring of model performance during training, in order to stop it when the validation error starts to increase, which is a sign of overfitting. The risk of using unrepresentative validation set is obvious in this case. Although the behavior of using DPS in conjunction with early stopping has not been addressed in this paper, it forms an interesting and promising research direction.

7 Conclusions

The correntropy–based density–preserving data sampling (DPS) procedure developed and investigated in this paper is an interesting alternative for widely used cross–validation technique in many applications. Unlike CV, DPS is a deterministic method, which eliminates the need for multiple repetitions of the sampling procedure

to obtain reliable results, considerably reducing the computational burden.

The main property of the proposed method is that it aims to produce only representative splits, which has many implications outlined in the previous section. The experiments conducted using a comprehensive set of publicly available benchmark datasets and a number of standard classifiers have revealed that:

- For generalization error estimation, DPS is slightly more biased than 10 times repeated cross–validation but has much lower variance, often lower than the best–case CV scenario. The DPS bias in all cases is also much lower than in the worst–case CV scenario.
- The decision boundaries of a classifier trained using DPS folds are much more stable than in the case of a single cross–validation folds, which is the result of representativeness of the subsets generated by DPS. The stability of models trained on various DPS divisions of the dataset has been confirmed in experiments involving ensemble models.
- For model ranking and selection, DPS is at least as good as 10 times repeated cross–validation, at much lower computational cost.

Further research will focus on application of DPS to early stopping, gradient driven optimization of the DPS objective function and investigating usability of other divergence measures for selection of a single fold to be used for error estimation, effectively reducing the computational requirements of repeated cross–validation by another order of magnitude.

REFERENCES

[1] R. Duda, P. Hart, and D. Stork, *Pattern Classification 2nd ed.* Wiley–Interscience, 2000.

[2] T. Cover, "Learning in Pattern Recognition." Tech. Rep., 1968.

[3] B. Efron, "Bootstrap methods: another look at the jackknife," *The Annals of Statistics*, vol. 7, no. 1, pp. 1–26, 1979.

[4] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *IJCAI*, 1995, pp. 1137–1145.

[5] A. Harzing, "Publish or Perish," www.harzing.com/pop.htm.

[6] W. Liu, P. Pokharel, and J. Principe, "Correntropy: A Localized Similarity Measure," *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pp. 4919–4924, 2006.

[7] M. Budka and B. Gabrys, "Correntropy-based density-preserving data sampling as an alternative to standard cross-validation," in *IJCNN'2010. International Joint Conference on (Accepted)*. IEEE, 2010.

[8] C. Bishop, *Neural Networks for Pattern Recognition*. Oxford University Press, USA, 1995.

[9] A. Antos, L. Devroye, and L. Gyorfi, "Lower bounds for Bayes error estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 7, pp. 643–645, 1999.

[10] A. Jain, R. Duin, and J. Mao, "Statistical pattern recognition: A review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 4–37, 2000.

[11] S. Weiss and C. Kulikowski, *Computer systems that learn: classification and prediction methods from statistics, neural nets, machine learning, and expert systems*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1991.

[12] J. Principe, D. Xu, and J. Fisher, *Information theoretic learning*. New York: Wiley, 2000, ch. 7, pp. 265–319.

[13] J. Principe, D. Xu, Q. Zhao, and J. Fisher, "Learning from Examples with Information Theoretic Criteria," *The Journal of VLSI Signal Processing*, vol. 26, no. 1, pp. 61–77, 2000.

[14] I. Santamaría, P. Pokharel, and J. Principe, "Generalized correlation function: Definition, properties, and application to blind equalization," *IEEE Transactions on Signal Processing*, vol. 54, no. 6, pp. 2187–2197, 2006.

[15] W. Liu, P. Pokharel, and J. Principe, "Correntropy: properties and applications in non-Gaussian signal processing," *IEEE Transactions on Signal Processing*, vol. 55, no. 11, pp. 5286–5298, 2007.

[16] ——, "Error Entropy, Correntropy and M-Estimation," in *Machine Learning for Signal Processing, 2006. Proceedings of the 2006 16th IEEE Signal Processing Society Workshop on*, 2006, pp. 179–184.

[17] K. Jeong and J. Principe, "Enhancing the correntropy MACE filter with random projections," *Neurocomputing*, vol. 72, no. 1-3, pp. 102–111, 2008.

[18] S. Seth and J. Principe, "Compressed signal reconstruction using the correntropy induced metric," in *Acoustics, Speech and Signal Processing, ICASSP 2008. IEEE International Conference on*.

[19] Y. Dodge, D. Cox, D. Commenges, A. Davison, and P. Solomon, *The Oxford dictionary of statistical terms*. Oxford University Press, USA, 2006.

[20] A. Asuncion and D. Newman, "UCI machine learning repository," 2007. [Online]. Available: http://www.ics.uci.edu/∼mlearn/MLRepository.html

[21] "ELENA database," 1995. [Online]. Available: http://www.dice.ucl.ac.be/mlg/DataBases/ELENA/REAL/

[22] R. Duin, P. Juszczak, P. Paclik, E. Pekalska, D. de Ridder, D. Tax, and S. Verzakov, "Pr-tools 4.1, a matlab toolbox for pattern recognition," 2007, http://prtools.org.

[23] L. Kuncheva, *Fuzzy classifier design*. Physica Verlag, 2000.

[24] B. Ripley, *Pattern recognition and neural networks*. Cambridge Univ Pr, 1996.

[25] D. Ruta and B. Gabrys, "Analysis of the correlation between majority voting error and the diversity measures in multiple classifier systems," *Proceedings of the 4th International Symposium on Soft Computing*, pp. 1824–025, 2001.