Kernel Factory: An Ensemble of Kernel Machines

Michel Ballings[1]

Dirk Van den Poel[2]

December 2012

2012/825

[1] Researcher, Ghent University
[2] Corresponding author: Prof. Dr. Dirk Van den Poel, Professor of Marketing Analytics / analytical Customer Relationship Management, Faculty of Economics and Business Administration, dirk.vandenpoel@ugent.be; The R software is available for free download from: http://cran.r-project.org/web/packages/kernelFactory/index.html

D/2012/7012/58

KERNEL FACTORY:

AN ENSEMBLE OF KERNEL MACHINES

Michel Ballings, Dirk Van den Poel
Faculty of Economics and Business Administration, Department of Marketing, Ghent University, Tweekerkenstraat 2, B-9000 Ghent, Belgium, http://www.crm.ugent.be, Michel.Ballings@UGent.be, Dirk.VandenPoel@UGent.be

Abstract

We propose an ensemble method for kernel machines. The training data is randomly split into a number of mutually exclusive partitions defined by a row and column parameter. Each partition forms an input space and is transformed by a kernel function into a kernel matrix *K*. Subsequently, each *K* is used as training data for a base binary classifier (Random Forest). This results in a number of predictions equal to the number of partitions. A weighted average combines the predictions into one final prediction. To optimize the weights, a genetic algorithm is used. This approach has the advantage of simultaneously promoting (1) diversity, (2) accuracy, and (3) computational speed. (1) Diversity is fostered because the individual *K*'s are based on a subset of features and observations, (2) accuracy is sought by optimizing the weights with the genetic algorithm, and (3) computational speed is obtained because the computation of each *K* can be parallelized. Using five times two-fold cross validation we benchmark the classification performance of Kernel Factory against Random Forest and Kernel-Induced Random Forest (KIRF). We find that Kernel Factory has significantly better performance than Kernel-Induced Random Forest. When the right kernel is specified Kernel Factory is also significantly better than Random Forest. In addition, an open-source R-software package of the algorithm (kernelFactory) is available from CRAN.
Keywords: Kernel Factory, Ensemble Learning, Classification, Machine learning, Genetic Algorithm, Random Forest

1 Introduction

In the last decade, kernel- based methods have become very popular for classification, regression and pattern recognition (Üstün, Melssen & Buydens 2006). It has been shown that classifiers can be improved by mapping input space $\mathcal{X}$ into feature space $\mathcal{H}$ (Jäkel, Schölkopf & Wichmann 2007). However, this mapping can increase the number of dimensions substantially to the point that analysis becomes problematic. Kernel methods are attractive in that they can create the aforementioned mapping and temper the dimensionality explosion problem by increasing the number of dimensions only linearly with the size of the original data (Shawe-Taylor & Cristianini 2004).
Although the dimensionality of the data scales only linearly with the original data when kernel methods are applied, data can grow very large. Hence, researchers have to resort to machine learning techniques that can handle a large number of predictors, such as Random

Forest. In that context, it is only recently that Kernel- Induced Random Forest (KIRF) has been introduced (Fan 2009). The advantage of KIRF over support vector machines (SVM) is that, in contrast to the latter, the former can handle remaining non-linearities in $\mathcal{H}$. Nevertheless, data can grow so large that classifier accuracy suffers due to the lower probability of selecting informative features. It can even grow unwieldy making analysis infeasible.
In an attempt to alleviate these computational efficiency and classifier accuracy problems, we propose an ensemble of kernel machines: Kernel Factory[3]. In addition, Kernel Factory has the advantages of increased computational speed and ensemble member diversity.
The remainder of this article is organized as follows. Section 2 reviews Random Forest, the kernel trick, and KIRF. Section 3 describes the proposed method Kernel Factory in detail. In Section 4 we present the methodology and results of an empirical study in which we benchmark Kernel Factory against Random Forest and KIRF. Section 5 provides a discussion and conclusion of the results. Finally, Section 6 offers avenues for future research and limitations.

2 Kernels and Random Forest

In this section we first elaborate on Random Forest. Second, we discuss the attractiveness of kernels. Third, we discuss the combination of both: Kernel-Induced Random Forests.

2.1 Random Forest

Binary recursive partitioning (BRP) is a method that grows decision trees, also referred to as classification and regression trees (CART) (Breiman, Friedman, Olshen & Stone 1984). The BRP algorithm starts by predicting a criterion variable by creating a binary partitioning of the data based on one predictor. The algorithm proceeds recursively by, within a parent partition, creating two child-partitions of the data. This partioning is based on another predictor or another split value of the same predictor variable that was used to create the parent partition (Merkle & Shaffer 2011). At each partitioning step, the predictor that produces the purest division of data is selected and the algorithm stops when, for example, a minimum partition size, a specific impurity or an amount of partitions is reached.
BRP is also used in Random Forest (Breiman 2001). Instead of growing one tree, Random Forest grows, and averages over, an ensemble of trees. Each tree is grown using an independent bootstrap sample for which at each partitioning step of a tree a subset of variables is randomly selected as splitting candidates (Breiman 2001).
Literature shows that Random Forest is one of the best-performing classification techniques available (Luo, Kramer, Goldgof, Hall, Samson, Remsen & Hopkins 2004). Moreover, it is very robust and consistent and does not overfit (Breiman 2001). Furthermore, the algorithm has reasonable computing times (Buckinx & Van den Poel 2005) and the procedure is easy to implement: only two parameters are to be set (number of trees and number of predictors) (Larivière & Van den Poel 2005; Duda, Hart & Stork 2001).

2.2 Kernels and the kernel trick

[3] Random Forest is an ensemble of decision trees, with a forest being a collection of trees. This inspired us to label the new method Kernel “Factory”, with a factory being a collection of machines.

Consider the binary classification problem in the left pane of Figure 1. A binary recursive partitioning tree would partition the data in two by using $x_1 >= 3$ as split rule in the root node and $x_1 <= 3$ in the child node. As such, to obtain a perfect classification the variable x_1 needs to be selected twice (x_2 has no discriminatory power in this case).
By applying feature map Φ that transforms the input space (x_1, x_2), by taking all second-degree unordered monomials, a dimension can be found that reduces the tree size (see right pane of Figure 1).
$\Phi: \mathbb{R}^2 \rightarrow \mathbb{R}^3$
$\Phi(x_1,x_2)=(z_1,z_2,z_3)=(x^2_1,x^2_2,x_1x_2)$ (1)

In feature space, a binary recursive partitioning tree would partition the data by only one split, as opposed to two splits in input space, using $z_1 >= 8$ as the split rule.

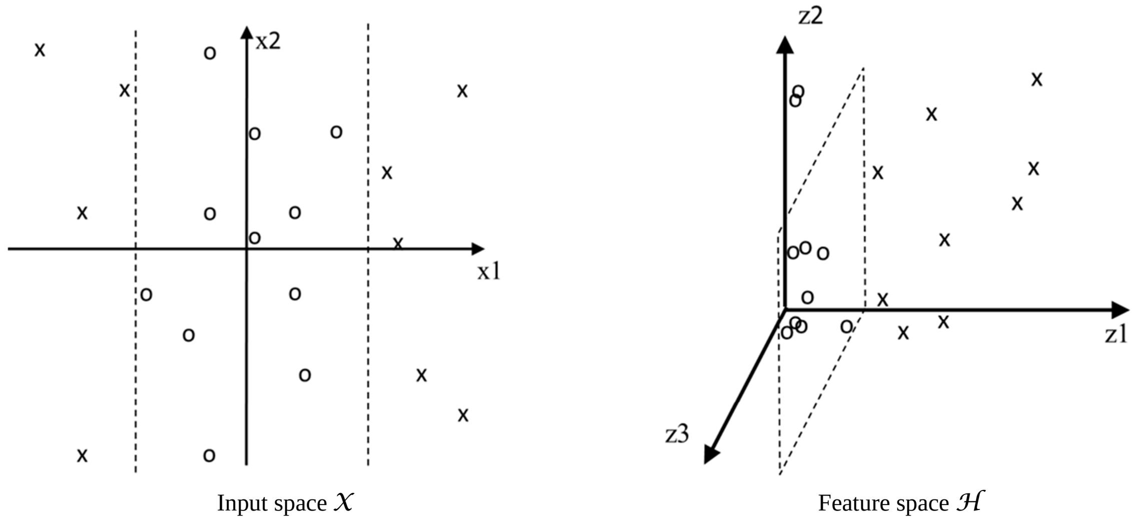

Figure 1. The feature map Φ transforms the input space (x_1, x_2), by taking all second-degree unordered monomials, to the feature space (z_1, z_2, z_3) where only one binary split rule is required to obtain the same result. (Figure adapted from Schölkopf & Smola 2002; note that actually $z_3= \sqrt{2}\, x_1x_2$ and not x_1x_2 but this will create a similar plot with the same decision boundary)
Because of this reduction in tree size, the performance of Random Forest can be improved. In Random Forests, out of a the total set of predictor variables, at each node a subset of candidate variables is selected at random and the candidate that produces the best split is used to split the node (Breiman 2001). Hence, because the probability is lower that the variable is selected twice as a candidate split variable, as opposed to once, the individual trees are bound to be stronger in feature space, as such decreasing the forest error rate (Breiman 2001).
While working in feature space has its advantages, it clearly has its disadvantages as well. Although the classification performance is improved thanks to elimination of the need to select the variable in our example more than once, there are now more predictors, decreasing the probability of selection for the candidate set (it has to be noted that the size of the subset is dependent on the size of the total set).
Consider the formula (2) (Schölkopf & Smola 2002) that shows that the dimensionality N in feature space can easily explode given monomials of degree d, making analysis prohibitive.

$$N_H = \binom{d+N_x-1}{d} = \frac{(d+N_x-1)!}{d!(N_x-1)!} \qquad (2)$$

,where $\mathcal{X}$ denotes input space and $\mathcal{H}$ feature space
In our example d=2 and $N_{\mathcal{X}}$=2 which results in $N_{\mathcal{H}}$=3. For d=2 and $N_{\mathcal{X}}$=150 dimensionality $N_{\mathcal{H}}$ amounts to 11,325. If in the latter case the degree increases by one (d=3) then $N_{\mathcal{H}}$ =573,800.

Calculating the inner product offers a solution by limiting the feature explosion so that the complexity of a classifier increases only linearly with the size of the input data. For two observations $\vec{x}_i$ and $\vec{x}_j$ (i,j=1,2,3,…l) defined by two variables (x_1, x_2) (Jäkel, Schölkopf & Wichmann 2007):

$$\begin{aligned}
&\langle \Phi(x_{i1}, x_{i2}), \Phi(x_{j1}, x_{j2}) \rangle \\
&= \langle (x_{i1}^2, x_{i2}^2, \sqrt{2}x_{i1}x_{i2}), (x_{j1}^2, x_{j2}^2, \sqrt{2}x_{j1}x_{j2}) \rangle \\
&= x_{i1}^2 x_{j1}^2 + x_{i2}^2 x_{j2}^2 + 2x_{i1}x_{j1}x_{i2}x_{j2} \\
&= (x_{i1}x_{j1} + x_{i2}x_{j2})^2 \\
&= \langle (x_{i1}, x_{i2}), (x_{j1}, x_{j2}) \rangle^2 \qquad (3)
\end{aligned}$$

The result in (3) shows that the inner product in feature space equals the inner product to the power of d in input space (see Schölkopf & Smola 2002 for proof). This is attractive because whereas the computational effort in feature space scales with the number of dimensions (see formula 2), in input space it scales with the number of observations (Jäkel, Schölkopf & Wichmann 2007). In the case of a monomial feature map, it is not required to map the observations *i* and *j* to feature space to compute the inner product: it is sufficient to calculate the inner product in the input space and take it to the power of d (Jäkel, Schölkopf & Wichmann 2007).

The combination of an inner product and a feature map Φ defines a kernel (4), short for kernel function, k($\vec{x}_i$, $\vec{x}_j$). The fact that kernels enable us to obtain the same superior classification performance as in feature space, for a much lower computational cost in input space, is called the kernel trick.

$$k(\vec{x_i}, \vec{x_j}) = \langle \Phi(\vec{x_i}), \Phi(\vec{x_j}) \rangle \qquad (4)$$

The example above uses a polynomial kernel. The polynomial kernel including tuning parameters and two other widely used kernels are displayed in Table 1 (Shawe-Taylor & Cristianini 2004; Park, Liu, Ye, Jeong, & Jeong 2012). The choice of the kernel function is largely dependent upon the data and can generally be determined by cross-validation (Fan 2009).

Table 1: Some examples of kernels

Linear kernel	$k(\vec{x_i}, \vec{x_j}) = \langle \vec{x}_i, \vec{x}_j \rangle$
Gaussian kernel	$k(\vec{x_i}, \vec{x_j}) = \exp(-\frac{\lVert \vec{x}_i - \vec{x}_j \rVert^2}{2\sigma^2})$
Polynomial kernel	$k(\vec{x_i}, \vec{x_j}) = (\gamma\langle \vec{x}_i, \vec{x}_j \rangle + r)^d$

$d, r \in \mathbb{N};\ \gamma \in \mathbb{R}^+$

In the next section we discuss a combination of kernels and Random Forest: Kernel-Induced Random Forest (Fan 2009).

2.3 Kernel-Induced Random Forest

Kernel-Induced Random Forest (KIRF) (Fan 2009) differs from Random Forest in that, during the training phase, the former selects a random subgroup of observations and uses their kernel functions as splitting candidates, while the latter selects a random subgroup of variables as splitting candidates (Fan 2009).
During the scoring phase, each splitting observation $\vec{x}_i$ (i=1,2,3,…,l) can be used with all observations that need to be scored $\vec{x}_k$ (k=1,2,3,…,m) in a kernel function $k(\vec{x}_i,.)$ resulting in what can be conceived of as a novel feature $k_i(\vec{x}_k)=k(\vec{x}_i, \vec{x}_k)$. For l splitting observations, there will be l such novel, kernel-induced features, $k_i(.)$, i=1,2,3,…,l (Fan 2009). These novel, kernel-induced, features can then be employed by a binary recursive space partitioning tree in order to get pure divisions of the criterion variable in the terminal node.
Although KIRF has been shown to have excellent performance (Fan 2009), the algorithm has the disadvantage that although the size of the kernel matrix *K* scales only linearly with the number of observations in the original data, *K* can quickly grow too large for Random Forest and subsequently KIRF. Kernel Factory alleviates this problem by randomly splitting the training data into a number of mutually exclusive partitions.
This approach reduces the probability of surpassing limitations of software packages by several orders of magnitude (e.g., for R the maximum size of an object is 2^{31}-1 elements (R Core Team (2012)) and increases computational speed by several orders of magnitude (because kernel matrices can now be computed in parallel more easily).

3 Kernel Factory

The training phase starts by partitioning the data into mutually exclusive row and column partitions followed by scaling. Each partition forms an input space and is transformed by a kernel function into a kernel matrix *K* (categorical features are not used in the computation of the *K* but are added afterwards). Subsequently, each *K* is used as training data for a base binary classifier (Random Forest).
The kernel function (polynomial, linear, or radial basis) can be (1) user specified, (2) randomly chosen per partition, or (3) determined by sequentially applying all kernels to the first partition, assessing predictive performance on a hold- out validation set, and selecting the best performing kernel.
This process results in a number of predictions equal to the number of partitions. We use a weighted average to combine the predictions into one final prediction in which the weights are optimized using a genetic algorithm (GA). For the fitness score we employ the predictive performance (area under the receiver operating characteristic curve) on a hold out sample. In extant literature, several studies have used a GA to weight these predictions of the ensemble members before combination. To the best of our knowledge Yao & Liu (1998) were the first to use a GA in this context (for a neural network ensemble). Later, Zhou, Wu & Tang (2002) show that using the evolved weights to make up the neural network ensemble yields superior performance to normal averaging (see also Kim, Street, & Menczer 2002 for a similar approach). Finally, Sylvester & Chawla (2005) use a GA to combine trees. All studies point to the beneficial effects of GAs in ensemble formation on predictive performance.
In sum, Kernel Factory promotes diversity by the partitioning step and fosters accuracy by the weight optimization step.

Pseudo code for the estimation phase is provided in Figure 2.

```
Training

Input
  • Number of column partitions (cp)
  • Number of row partitions (rp)
  • Method: polynomial kernel function (pol), linear kernel function (lin), radial basis kernel function (rbf), random
    choice (ran={pol, lin, rbf}), burn- in choice of best function (burn={pol, lin, rbf })
  • Input space $\mathcal{X}$=($x_1$,…,$x_c$) with class labels (Y={0,1}) where $c$ is the number of features and let $r$ be the number of
    objects
Randomize order of rows and columns of $\mathcal{X}$

Divide $\mathcal{X}$: 80% of $r$ into $\mathcal{X}_{\text{training}}$ and 20% of $r$ into $\mathcal{X}_{\text{validation}}$

Scale both $\mathcal{X}_{\text{training}}$ and $\mathcal{X}_{\text{validation}}$ : for every feature: $\frac{x_c}{range(x_{c,training})}$

For $\mathcal{X}_{\text{training}}$: create rp times cp partitions of equal size ($p_{\text{training}}$)

For $\mathcal{X}_{\text{validation}}$: create cp partitions of equal size ($p_{\text{validation}}$)

For every partition $p_{\text{training}}$:

  • Apply method:
      o Let
          ▪ lin = $\langle \vec{x}_i, \vec{x}_j \rangle$
          ▪ rbf = exp($-\frac{||\vec{x}_i - \vec{x}_j||^2}{2}$)
          ▪ pol = $\langle \vec{x}_i, \vec{x}_j \rangle^2$
      o If method==lin: select lin
        Else if method==pol: select pol

        Else if method==rbf: select rbf

        Else if method==ran: randomly select one of {pol, lin, rbf}

        Else if method==burn:

          ▪ If $p_{\text{training}}$ ==1: for every method in {pol, lin, rbf}
              • Compute kernel matrix $K_{\text{training}}$ on numeric features
              • Augment $K_{\text{training}}$ with raw features
              • Build classifier $C$ on $K_{\text{training}}$
              • Compute $K_{\text{validation}}$ ,populated by elements where, for the kernel function,
                observations $\vec{x}_i$ come from $p_{\text{validation}}$ (with same numeric features as in $p_{\text{training}}$ ==1)
                and $\vec{x}_j$ come from numeric features of $p_{\text{training}}$ ==1
              • Augment $K_{\text{validation}}$ with the raw features from $p_{\text{validation}}$
              • Deploy $C$: $\hat{Y}_{prob} = \mathrm{C}(K_{\text{validation}})$
              • Compute AUC
          ▪ Select one of {pol, lin, rbf} which yielded max AUC
  • Compute kernel matrix $K_{\text{training}}$ on numeric features
  • Augment $K_{\text{training}}$ with raw features
  • Build classifier $C$ on $K_{\text{training}}$
  • Compute $K_{\text{validation}}$ ,populated by elements where, for the kernel function, observations $\vec{x}_i$ come from $p_{\text{validation}}$
    (with same numeric features as in current $p_{\text{training}}$) and $\vec{x}_j$ come from numeric features of current $p_{\text{training}}$
  • Augment $K_{\text{validation}}$ with the raw features from $p_{\text{validation}}$
  • Deploy C: $\hat{Y}_{\text{prob}} = \mathrm{C}(K_{\text{validation}})$
```

```
Optimize classifier weights for weighted averaging of predicted probabilities with genetic algorithm on validation space

  • Choose initial population of sets of weights
  • Repeat 100 times
      o For each set of weights ω=(w_1,…w_l) compute weighted average for a given object $\vec{x}_i$ having a set of
        $\hat{Y}_{prob,i} = (\hat{Y}_{i,1}, \ldots \hat{Y}_{i,l})$ , with l=(1,…rp times cp) the number of classifiers (= number of weights):
            $\mu_i(\hat{Y}_{prob}) = \sum_{l=1}^{l} w_l(\hat{Y}_{prob,i,l})$
      o Evaluate fitness (AUC)
      o Select best-fit set of weights
      o Apply crossover operator
      o Apply mutation operator
  • Choose set of optimal weights (weights with best fit; highest AUC): ω=(w_1,…w_l) with l=rp times cp
```

Figure 2: Pseudo code for the estimation phase of Kernel Factory

The prediction phase (see Figure 3) starts by constructing the same column partitions and applying scaling as in the training set. Next, the features from the training set are used to compute the *K*'s. Finally, the weighted average of all predictions is computed using the weights obtained from the genetic algorithm.

```
Prediction

Input
  • Input space $\mathcal{X}_{new}$ =(x1,…,xc) where c is the number of features and let r be the number of objects
Apply order of rows and columns of $\mathcal{X}_{training}$ to $\mathcal{X}_{new}$
Scale $\mathcal{X}_{new}$ : for every feature : $\frac{x_c}{range(x_{c,training})}$
For $\mathcal{X}_{new}$: create cp partitions ($p_{new}$) of equal size
For every classifier C:
  • Compute $K_{new}$ on numeric features populated by elements where, for the kernel function, observations $\vec{x}_i$ come
    from $p_{new}$ (with same numeric features as in current $p_{training}$) and $\vec{x}_j$ come from numeric features of current
    $p_{training}$
  • Augment the kernel matrix with the raw features from $p_{new}$
  • Deploy C: $\hat{Y}_{prob} = C(K_{new})$
Compute weighted average using optimal weights: for a given object $\vec{x}_i$ having a set of $\hat{Y}_{prob,i} = (\hat{Y}_{i,1}, \ldots \hat{Y}_{i,l})$, with
l=(1,…rp times cp) the number of classifiers (= number of weights) , use optimal ω:
    $\mu_i(\hat{Y}_{prob}) = \sum_{l=1}^{l} w_l(\hat{Y}_{prob,i,l})$

$\mu_i$ is the confidence that object $\vec{x}_i$ has Y==1
```

Figure 3: Pseudo code for the prediction phase of Kernel Factory

4 Empirical Study

4.1 Data

Our experiments use 11 datasets from the UCI repository and 3 synthetic datasets. The first 11 datasets were selected because we have used them in past research, because the dependent variable is binary and because they contain continuous predictor variables (required for kernels). Moreover, most of them are used in Breiman's Random Forest paper (Breiman 2001). The 3 synthetic datasets are constructed using the R package mlbench (Leisch, & Dimitriadou 2010) and are chosen because KIRF improves on Random Forest (Fan, 2009) on these datasets. The underlying reason is that, in these specific datasets, the radial basis function (RBF) helps Random Forest to classify the Gaussian distribution present in these datasets. It is well documented that using kernel functions only has a positive effect on predictive performance if the right kernel function is used and even has a harmful effect if the wrong kernel function is used. Hence, KIRF should only be used with the right kernel and only if this improves on Random Forest. Therefore, it makes little sense to compare Random Forest and KIRF blindly (without testing different kernels) on different datasets and make generalizations about which algorithm is best (Fan, 2009). Random Forest and KIRF (with different kernels) should always be tested together and the best algorithm should be selected. Hence to give KIRF an honest chance in the comparison with the other algorithms, we use these synthetic datasets because we know they have Gaussian distributions and that a RBF kernel will add positively to the predictive performance.
In general, we expect that Kernel Factory will improve on KIRF because of the internal kernel selection procedure (in the case of method=burn or random). In either case, Kernel Factory will not require manually testing multiple kernels.
Table 2 gives a brief summary of the datasets. *N* is the number of observations, *p* is the number of predictor variables
Table 2: properties of the datasets used in the empirical study

Data	N	p, continuous	p, categorical
Heart (Cleveland)	303	5	8
Hepatitis	155	6	13
Ionosphere	351	32	1
Pima (Diabetes)	768	8	0
Credit	690	6	9
Sonar	208	59	0
Wdbc (Cancer)	569	30	0
HeartHun (Hungary)	294	5	7
GermanCredit	1000	7	13
AustralianCredit	690	6	8
HorseColic	368	9	13
Ringnorm	1000	10	0
Peak	1000	6	0
Circle	1000	20	0

The last three datasets are the synthetic ones. The criterion variable was initially continuous in the Peak dataset and is transformed to a binary variable by assigning a 1 if the value is greater than the mean and 0 otherwise.

4.2 Implementations of Algorithms

Random Forest requires two parameters to be set: the number of trees and the number of variables to try at each split. We follow Breiman's recommendation (Breiman 2001) to use a large number of trees (1000) and the square root of the total number of variables as the number of predictors.
For KIRF we used the same settings for Random Forest as above. For the Gaussian radial basis function we set the Gaussian kernel parameter σ equal to 1 because a decision tree is invariant to monotonic transformations of data (Fan 2009). For the polynomial kernel function we used a degree of 2, a scale of 1 and an offset of 0. The categorical predictor variables are not involved in the computation of kernels but are kept as extra attributes during tree construction.
In using Kernel Factory, two parameters to set are the number of row and column partitions. Because the algorithm is designed to overcome practical limitations of computational resources, we chose parameter values aimed at accommodating the server we used for our experiments. We used int($\log_4$(p+1)) for the number of row partitions and both int($\log_5$(p+1)) and 1 for the number of column partitions. From some preliminary testing we found that Kernel Factory is more sensitive to column partitioning than to row partitioning, hence the difference in the logarithms' bases. We also wanted to test with one column partition because column partitions are rather meant to introduce extra diversity in the ensemble members while it are the row partitions have a much bigger impact on the speed of the algorithm due to their direct impact on the size of the K's. Setting the number of column partitions to one and comparing it to int($\log_5$(p+1)) allows us to understand the impact of that process. The base algorithm was Random Forest, using the same settings as above. The settings for the kernels is also identical to the ones we used in KIRF. We employed a population size of 100, 200 iterations and a mutation chance of 0.01 for the genetic algorithm.
We have submitted an open-source R-software package of the algorithm (kernelFactory) to CRAN (Ballings, & Van den Poel 2012). Packages that are used by Kernel Factory, KIRF and Random Forest are kernlab (Karatzoglou, Smola, Hornik, & Zeileis, 2004), randomForest (Liaw & Wiener, 2002), genalg (Willighagen 2005), and ROCR (Sing, Sander, Beerenwinkel, & Lengauer 2009).

4.3 Model Performance Evaluation

To evaluate the performance of a model we use accuracy or percentage correctly classified (PCC) and, the area under the receiver operating characteristic curve (AUC or AUROC).
PCC is defined as follows:

$$PCC = \frac{TP+TN}{P+N} \qquad (5)$$

with TP: True Positives, TN: True Negatives, P: Positives (event), N: Negatives (non-event)
An important disadvantage of PCC is that it is sensitive to the chosen cut-off value of the posterior probabilities (Baecke, & Van den Poel 2012; Thorleuchter & Van den Poel 2012) that decides when an object is predicted to be of class zero or one. While accuracy is the performance of a model at only one cut-off value AUC is the performance of a model across all threshold values. Several authors (Provost, Fawcett, Kohavi 1998; Langley 2000; Coussement & Van den Poel 2008) argue AUC to be an objective criterion for classifier

performance. As such AUC is a more adequate performance measure than PCC (Baecke, & Van den Poel 2011).
More formally, the receiver operating characteristic (ROC) curve is obtained from plotting sensitivity and 1-specificity considering all possible cut-off values (Hanley, & McNeil, 1982). AUC ranges from .5, if the predictions are not better than random, to 1, if the model predicts the event perfectly (Baecke, & Van den Poel 2011). AUC is defined as follows:

$$AUC = \int_0^1 \frac{TP}{(TP+FN)}\, d\,\frac{FP}{(FP+TN)} = \int_0^1 \frac{TP}{P}\, d\,\frac{FP}{N} \qquad (6)$$

,with TP: True Positives, FN: False Negatives, FP: False Positives, TN: True Negatives, P: Positives (event), N: Negatives (non-event)
Reported performance metrics are all medians over five times two-fold cross validation (5x2cv) (Dietterich 1998; Alpaydin 1999). This procedure of cross validation randomly divides the sample in two equal parts and repeats this process five times. Each part is used both as a training and validation part. This results in ten performance metrics per model (Dietterich 1998). The same splits are used for all algorithms. We also report the inter quartile range as a measure of dispersion.
In order to determine whether models are significantly different in terms of AUC or PCC, we follow Demšar's recommendation (Demšar 2006) to use the nonparametric Friedman test with Nemenyi's post-hoc test (Nemenyi, 1963) for comparisons of the algorithms. In this context we report the average ranks per dataset of the algorithms. Algorithms are ranked, per fold separately, with the best algorithm receiving the rank 1, the second receiving the rank of 2, etc. It is important to note that this approach incorporates the relatedness of the folds (algorithm ranks are computed per fold and then the average rank is computed per dataset) and are not treated as independent (as is the case when computing the median). In order words, when ranks are computed the order of the folds is preserved and comparisons are made per fold, whereas when the median is computed the order of the folds is not preserved because they are sorted by predictive performance and the middle one is selected. Hence computing ranks in this fashion allows stricter comparison than computing the median.
We opted for testing on the folds per dataset, as opposed to testing the median across the datasets, because the datasets' predictive performances are not commensurate. Hence, controlling for the family-wise error, the probability of at least one false positive in any of the comparisons, is debatable because the costs of these false positives differs across datasets (also see Webb 2000): the datasets are not a family. Moreover, there are no tests available for multiple datasets that can consider the folds for each dataset (Demšar 2006). More concretely, although we will compare KIRF and Kernel Factory on all datasets it makes no sense to blindly compare Random Forest with KIRF and Kernel Factory on all datasets. As aforementioned, the choice of the kernel depends on the data and can be determined by cross-validation (Fan 2009). Hence, it is obvious that KIRF nor Kernel Factory will be used if they perform considerably worse than Random Forest due to the wrong kernel choice. In sum, because we want to make declarations per dataset, and not merely per algorithm, we opted to test on the folds.

4.4 Results

Table 3 and Table 5 show the median PCC and AUC respectively of the 10 cross-validation folds. In all tables and the rest of this text, KF stands for Kernel Factory, KIRF stands for Kernel-Induced Random Forest, RF stands for Random Forest and cp1 stands for column partitions equal to one. If cp1 is not specified along RF it means that int($\log_5$(p+1)) is used to determine the number of column partitions. Table 4 and 6 contain the interquartile ranges for PCC and AUC respectively. Table 7 (PCC) and Table 9 (AUC) report the average ranks (lower is better) and Table 8 (PCC) and Table 10 (AUC) report selected differences of the average rankings. As mentioned in the model performance evaluation section, taking the median of the folds does not respect that performances are related per fold, whereas taking the Friedman test does (Demšar 2006). This is also the reason why results can sometimes deviate to a limited extent. It has to be noted that using the rank as opposed to the median can be considered a stricter comparison of predictive performance.

Table 3: The median of the 10 folds for PCC

PCCMedian	KFrbf	KFrbfcp1	KIRFrbf	KFpol	KFpolcp1	KIRFpol	KFlin	KFlincp1	KIRFlin	KFran	KFrancp1	KFburn	KFburncp1	RF
Heart	0.7591	0.7228	0.6755	0.7285	0.7492	0.6457	0.7318	0.7185	0.6601	0.7252	0.7360	0.7426	0.7393	0.8212
Hepatitis	0.8129	0.8194	0.8141	0.8064	0.8259	0.8258	0.8129	0.8000	0.8193	0.8193	0.8129	0.7821	0.8129	0.8442
Ionosphere	0.9003	0.9060	0.9259	0.8807	0.9145	0.9371	0.8889	0.9088	0.9288	0.9059	0.9205	0.8892	0.9062	0.9345
Pima	0.7174	0.7396	0.7435	0.7135	0.7448	0.7357	0.7109	0.7370	0.7344	0.7292	0.7344	0.7292	0.7383	0.7565
Credit	0.7913	0.8174	0.7377	0.7957	0.8130	0.7000	0.8217	0.8014	0.7014	0.7870	0.7957	0.8058	0.7971	0.8725
Sonar	0.7452	0.7500	0.7933	0.7308	0.7548	0.7067	0.7308	0.7500	0.7115	0.7356	0.7404	0.7356	0.7500	0.7981
wdbc	0.9419	0.9472	0.9439	0.9455	0.9421	0.9437	0.9437	0.9473	0.9419	0.9438	0.9420	0.9561	0.9438	0.9542
HeartHun	0.7551	0.7823	0.7449	0.7619	0.7517	0.7041	0.7721	0.7517	0.7007	0.7483	0.7585	0.7279	0.7687	0.8095
GermanCredit	0.6900	0.7050	0.6700	0.4910	0.7040	0.6660	0.3050	0.7130	0.6640	0.3050	0.7040	0.3040	0.7050	0.7550
AustralianCredit	0.7855	0.7986	0.7391	0.7971	0.8000	0.6942	0.8000	0.7913	0.7014	0.7913	0.8087	0.8000	0.7942	0.8638
HorseColic	0.7147	0.7255	0.7201	0.7147	0.7092	0.7011	0.7255	0.7120	0.6984	0.6902	0.7337	0.7228	0.7201	0.7745
Ringnorm	0.8850	0.9290	0.9270	0.8720	0.8960	0.9020	0.8780	0.8950	0.9020	0.8840	0.8930	0.8800	0.9290	0.8910
Peak	0.9100	0.9840	0.9890	0.8930	0.9420	0.9400	0.8860	0.9340	0.9410	0.8920	0.9830	0.8960	0.9870	0.9270
Circle	0.8330	0.8810	0.8990	0.8330	0.8330	0.8330	0.8330	0.8320	0.8330	0.8330	0.8520	0.8330	0.8760	0.8350

Table 4: The interquartile range of the 10 folds for PCC

PCCIQR	KFrbf	KFrbfcp1	KIRFrbf	KFpol	KFpolcp1	KIRFpol	KFlin	KFlincp1	KIRFlin	KFran	KFrancp1	KFburn	KFburncp1	RF
Heart	0.0342	0.0498	0.0198	0.0517	0.0362	0.0521	0.0383	0.0809	0.0560	0.0553	0.0607	0.0756	0.0514	0.0243
Hepatitis	0.0399	0.0536	0.0617	0.0303	0.0611	0.0597	0.0512	0.0718	0.0486	0.0373	0.0304	0.0364	0.0503	0.0662

Ionosphere	0.0409	0.0357	0.0243	0.0406	0.0280	0.0255	0.0287	0.0280	0.0313	0.0595	0.0372	0.0455	0.0346	0.0270
Pima	0.0234	0.0286	0.0208	0.0273	0.0124	0.0286	0.0410	0.0280	0.0312	0.0202	0.0228	0.0267	0.0345	0.0234
Credit	0.0384	0.0428	0.0341	0.0123	0.0319	0.0609	0.0565	0.0232	0.0536	0.0355	0.0232	0.0507	0.0181	0.0232
Sonar	0.0529	0.0529	0.0745	0.0601	0.0457	0.0529	0.0409	0.0505	0.0721	0.0361	0.0264	0.0601	0.0745	0.0625
wdbc	0.0141	0.0204	0.0086	0.0149	0.0183	0.0168	0.0133	0.0132	0.0159	0.0158	0.0262	0.0132	0.0070	0.0096
HeartHun	0.0833	0.0561	0.0374	0.0833	0.0255	0.0680	0.0816	0.0731	0.0646	0.0476	0.0629	0.0697	0.0374	0.0272
GermanCredit	0.3995	0.0170	0.0250	0.3965	0.0210	0.0155	0.2890	0.0255	0.0200	0.0160	0.0125	0.0185	0.0220	0.0095
AustralianCredit	0.0159	0.0536	0.0536	0.0529	0.0138	0.0399	0.0601	0.0341	0.0312	0.0254	0.0500	0.0377	0.0486	0.0109
HorseColic	0.0204	0.0462	0.0312	0.0312	0.0380	0.0190	0.0177	0.0217	0.0217	0.0557	0.0435	0.0326	0.0353	0.0190
Ringnorm	0.0140	0.0070	0.0110	0.0130	0.0170	0.0065	0.0125	0.0135	0.0065	0.0115	0.0195	0.0120	0.0055	0.0280
Peak	0.0155	0.0090	0.0075	0.0150	0.0245	0.0140	0.0205	0.0180	0.0150	0.0190	0.0330	0.0195	0.0105	0.0235
Circle	0.0150	0.0270	0.0245	0.0150	0.0150	0.0150	0.0150	0.0140	0.0150	0.0150	0.0185	0.0150	0.0150	0.0150

Table 5: The median of the 10 folds for AUC

AUCMedian	KFrbf	KFrbfcp1	KIRFrbf	KFpol	KFpolcp1	KIRFpol	KFlin	KFlincp1	KIRFlin	KFran	KFrancp1	KFburn	KFburncp1	RF
Heart	0.8286	0.8081	0.7341	0.8267	0.8345	0.6805	0.7973	0.8167	0.6800	0.7947	0.8251	0.8407	0.8226	0.8993
Hepatitis	0.8276	0.8199	0.8188	0.7918	0.7864	0.7915	0.8006	0.8095	0.7978	0.7961	0.8227	0.7927	0.8213	0.8681
Ionosphere	0.9478	0.9528	0.9701	0.9583	0.9558	0.9746	0.9644	0.9589	0.9747	0.9559	0.9633	0.9583	0.9563	0.9756
Pima	0.7712	0.8046	0.8047	0.7795	0.8007	0.7978	0.7851	0.7952	0.8004	0.7855	0.7995	0.7919	0.7996	0.8188
Credit	0.8768	0.8863	0.8003	0.8714	0.8849	0.7639	0.8767	0.8852	0.7656	0.8760	0.8748	0.8709	0.8781	0.9301

Sonar	0.8609	0.8617	0.8865	0.8481	0.8519	0.8212	0.8456	0.8490	0.8219	0.8454	0.8514	0.8500	0.8586	0.9156
wdbc	0.9886	0.9888	0.9874	0.9872	0.9857	0.9844	0.9880	0.9872	0.9850	0.9853	0.9879	0.9864	0.9853	0.9904
HeartHun	0.8369	0.8276	0.7969	0.8349	0.8202	0.7305	0.8144	0.8285	0.7331	0.8178	0.8215	0.8127	0.8275	0.8894
GermanCredit	0.6737	0.6564	0.5976	0.6611	0.6701	0.5817	0.6535	0.6570	0.5838	0.6791	0.6610	0.6677	0.6801	0.7881
AustralianCredit	0.8768	0.8769	0.8074	0.8818	0.8752	0.7591	0.8730	0.8748	0.7568	0.8802	0.8867	0.8863	0.8756	0.9337
HorseColic	0.7889	0.7613	0.7621	0.7763	0.7562	0.7262	0.7763	0.7606	0.7247	0.7764	0.7800	0.7653	0.7669	0.8353
Ringnorm	0.9554	0.9794	0.9788	0.9505	0.9617	0.9653	0.9527	0.9627	0.9654	0.9581	0.9610	0.9565	0.9796	0.9555
Peak	0.9749	0.9995	0.9996	0.9620	0.9927	0.9940	0.9600	0.9927	0.9944	0.9675	0.9991	0.9676	0.9996	0.9868
Circle	0.8915	0.9864	0.9867	0.6212	0.4286	0.5245	0.6078	0.4245	0.5239	0.7283	0.7352	0.8789	0.9834	0.7748

Table 6: The interquartile range of the 10 folds for AUC

AUCIQR	KFrbf	KFrbfcp1	KIRFrbf	KFpol	KFpolcp1	KIRFpol	KFlin	KFlincp1	KIRFlin	KFran	KFrancp1	KFburn	KFburncp1	RF
Heart	0.0379	0.0639	0.0231	0.0365	0.0334	0.0734	0.0547	0.0737	0.0587	0.0614	0.0999	0.0944	0.0556	0.0187
Hepatitis	0.0532	0.0407	0.0897	0.0474	0.0762	0.1133	0.0860	0.0396	0.1210	0.1004	0.0908	0.1045	0.0521	0.0664
Ionosphere	0.0156	0.0261	0.0207	0.0236	0.0196	0.0191	0.0174	0.0213	0.0201	0.0095	0.0198	0.0058	0.0160	0.0113
Pima	0.0342	0.0358	0.0225	0.0524	0.0178	0.0207	0.0264	0.0340	0.0238	0.0191	0.0171	0.0195	0.0382	0.0155
Credit	0.0331	0.0385	0.0435	0.0437	0.0227	0.0616	0.0328	0.0272	0.0550	0.0462	0.0224	0.0376	0.0268	0.0221
Sonar	0.0442	0.0351	0.0314	0.0242	0.0578	0.0317	0.0231	0.0348	0.0300	0.0297	0.0255	0.0238	0.0413	0.0320
wdbc	0.0065	0.0073	0.0108	0.0031	0.0091	0.0070	0.0055	0.0060	0.0073	0.0053	0.0065	0.0085	0.0076	0.0052
HeartHun	0.0328	0.0257	0.0612	0.0318	0.0330	0.0753	0.0905	0.0633	0.0753	0.0700	0.0348	0.0600	0.0326	0.0362
GermanCredit	0.0466	0.0348	0.0302	0.0548	0.0189	0.0253	0.0471	0.0094	0.0256	0.0365	0.0249	0.0441	0.0398	0.0104

AustralianCredit	0.0346	0.0492	0.0367	0.0266	0.0093	0.0483	0.0291	0.0284	0.0464	0.0188	0.0377	0.0287	0.0318	0.0177
HorseColic	0.0325	0.0303	0.0313	0.0408	0.0381	0.0465	0.0189	0.0471	0.0483	0.0661	0.0610	0.0438	0.0401	0.0208
Ringnorm	0.0093	0.0029	0.0016	0.0093	0.0064	0.0047	0.0066	0.0045	0.0054	0.0049	0.0113	0.0032	0.0025	0.0064
Peak	0.0038	0.0005	0.0004	0.0077	0.0030	0.0026	0.0070	0.0029	0.0025	0.0041	0.0033	0.0068	0.0006	0.0060
Circle	0.0298	0.0080	0.0039	0.0986	0.0553	0.0576	0.0627	0.1275	0.0611	0.0844	0.5141	0.0403	0.0085	0.0499

Table 7: Average rankings of the folds (per dataset) for PCC

Average rankings (PCC)	KFrbf	KFrbfcp1	KIRFrbf	KFpol	KFpolcp1	KIRFpol	KFlin	KFlincp1	KIRFlin	KFran	KFrancp1	KFburn	KFburncp1	RF	Friedman chi² (13), p <
Heart	5.00	6.75	11.75	6.25	5.70	13.55	8.30	6.55	12.55	6.45	7.65	6.55	6.65	1.30	77.98, .001
Hepatitis	7.85	8.15	8.15	8.70	5.35	5.95	9.00	7.70	5.95	8.85	7.85	10.35	8.85	2.30	32.16, .001
Ionosphere	9.25	7.60	5.15	10.50	7.60	3.40	10.40	8.35	4.00	9.00	6.25	11.75	9.00	2.75	59.66, .001
Pima	11.05	5.55	6.25	11.10	5.70	5.65	10.05	7.15	6.90	9.20	6.40	9.40	7.25	3.35	40.25, .001
Credit	7.50	5.70	11.60	7.30	4.75	13.45	6.00	5.95	13.15	8.00	7.45	6.10	7.05	1.00	82.84, .001
Sonar	8.25	6.20	3.70	7.70	7.65	11.85	9.50	6.65	10.20	7.80	8.35	8.20	6.90	2.05	46.19, .001
wdbc	8.70	6.55	6.70	6.15	9.15	8.50	6.85	6.85	8.05	8.55	10.35	5.30	8.90	4.40	20.72, .005
HeartHun	7.15	6.15	8.20	8.00	7.25	11.70	8.10	6.50	11.70	7.80	6.60	8.10	5.35	2.40	41.33, .001
GermanCredit	8.75	5.40	9.05	9.05	4.50	9.75	10.80	4.50	10.05	11.65	4.70	11.00	4.80	1.00	80.07, .001
AustralianCredit	7.40	6.80	11.45	5.80	6.20	13.50	6.05	7.30	13.45	6.80	5.80	7.30	6.05	1.10	80.71, .001
HorseColic	8.20	7.05	6.35	7.75	8.15	10.20	7.10	7.25	10.85	9.60	5.65	8.10	7.50	1.25	39.58, .001

Ringnorm	9.70	2.00	2.15	11.90	7.25	5.60	11.55	7.65	6.25	10.45	7.20	11.25	2.05	10.00	93.88, .001
Peak	10.35	2.55	1.60	12.35	6.90	6.30	13.00	6.60	6.30	12.30	4.25	11.65	2.35	8.50	117.19, .001
Circle	9.20	2.75	1.25	9.65	9.65	9.65	9.65	10.05	9.65	9.15	6.20	9.15	2.55	6.45	100.72, .001

Table 8: Selected differences of the average rankings of the folds (per dataset) for PCC

Selected differences of average rankings	KFrbf- KFrbfcp1	KIRFrbf- KFrbfcp1	KIRFrbf- KFrbf	KFpol- KFpolcp1	KIRFpol- KFpolcp1	KIRFpol- KFpol	KFlin- KFlincp1	KIRFlin- KFlincp1	KIRFlin- KFlin	KFran- KFrancp1	KFburn- KFburncp1	KFran- KFburn	KFrancp1- KFburncp1	RF- KIRFrbf	RF- KFrbfcp1
Heart	-1.75	5	**6.75**	0.55	**7.85**	**7.3**	1.75	6	4.25	-1.2	-0.1	-0.1	1	**-10.45**	-5.45
Hepatitis	-0.3	0	0.3	3.35	0.6	-2.75	1.3	-1.75	-3.05	1	1.5	-1.5	-1	-5.85	-5.85
Ionosphere	1.65	-2.45	-4.1	2.9	-4.2	**-7.1**	2.05	-4.35	**-6.4**	2.75	2.75	-2.75	-2.75	-2.4	-4.85
Pima	5.5	0.7	-4.8	5.4	-0.05	-5.45	2.9	-0.25	-3.15	2.8	2.15	-0.2	-0.85	-2.9	-2.2
Credit	1.8	5.9	4.1	2.55	**8.7**	6.15	0.05	**7.2**	**7.15**	0.55	-0.95	1.9	0.4	**-10.6**	-4.7
Sonar	2.05	-2.5	-4.55	0.05	4.2	4.15	2.85	3.55	0.7	-0.55	1.3	-0.4	1.45	-1.65	-4.15
wdbc	2.15	0.15	-2	-3	-0.65	2.35	0	1.2	1.2	-1.8	-3.6	3.25	1.45	-2.3	-2.15
HeartHun	1	2.05	1.05	0.75	4.45	3.7	1.6	5.2	3.6	1.2	2.75	-0.3	1.25	-5.8	-3.75
GermanCredit	3.35	3.65	0.3	4.55	5.25	0.7	**6.3**	5.55	-0.75	**6.95**	6.2	0.65	-0.1	**-8.05**	-4.4
AustralianCredit	0.6	4.65	4.05	-0.4	**7.3**	**7.7**	-1.25	6.15	**7.4**	1	1.25	-0.5	-0.25	**-10.35**	-5.7
HorseColic	1.15	-0.7	-1.85	-0.4	2.05	2.45	-0.15	3.6	3.75	3.95	0.6	1.5	-1.85	-5.1	-5.8
Ringnorm	**7.7**	0.15	**-7.55**	4.65	-1.65	**-6.3**	3.9	-1.4	-5.3	3.25	**9.2**	-0.8	5.15	**7.85**	**8**
Peak	**7.8**	-0.95	**-8.75**	5.45	-0.6	-6.05	**6.4**	-0.3	**-6.7**	**8.05**	**9.3**	0.65	1.9	**6.9**	5.95
Circle	**6.45**	-1.5	**-7.95**	0	0	0	-0.4	-0.4	0	2.95	**6.6**	0	3.65	5.2	3.7

Table 9: Average rankings of the folds (per dataset) for AUC

Average rankings (AUC)	KFrbf	KFrbfcp1	KIRFrbf	KFpol	KFpolcp1	KIRFpol	KFlin	KFlincp1	KIRFlin	KFran	KFrancp1	KFburn	KFburncp1	RF	Friedman chi² (13), p <
Heart	6.10	7.10	12.00	5.40	5.90	13.10	8.30	6.0	13.50	7.40	6.80	6.20	6.2	1.0	83.27, .001
Hepatitis	4.90	6.80	7.60	8.40	9.70	9.85	7.20	7.9	9.35	7.20	6.70	10.50	6.5	2.4	33.63, .01
Ionosphere	10.45	10.25	3.40	10.30	9.50	3.20	6.95	8.9	3.00	9.90	7.65	9.20	10.1	2.2	73.99, .001
Pima	10.50	5.80	7.70	8.80	5.70	7.90	10.40	6.3	8.00	10.30	6.80	9.50	6.0	1.3	45.79, .001
Credit	6.80	5.60	12.00	8.20	4.40	13.60	5.90	5.4	13.40	7.60	6.70	7.20	7.2	1.0	89.44, .001
Sonar	6.70	6.20	3.70	7.55	8.75	12.40	8.30	7.5	12.40	9.30	7.70	6.70	6.5	1.3	63.09, .001
wdbc	7.35	4.30	6.70	7.20	7.75	10.60	7.00	7.8	9.45	10.15	8.70	6.60	7.4	4.0	26.49, 0.05
HeartHun	5.90	6.00	10.40	5.80	7.20	12.90	8.60	6.9	12.50	6.70	6.80	8.10	6.1	1.1	66.48, .001
GermanCredit	5.70	7.80	11.80	7.70	6.20	13.20	6.90	7.1	13.20	5.20	7.00	6.30	5.9	1.0	80.48, .001
AustralianCredit	6.80	6.50	12.20	6.50	6.80	13.30	7.30	7.6	13.50	6.10	5.30	5.30	6.8	1.0	85.22, .001
HorseColic	5.50	8.60	7.90	6.90	9.00	11.30	5.85	7.1	12.00	8.30	5.50	7.85	8.2	1.0	53.19, .001
Ringnorm	9.00	1.80	2.10	11.70	8.20	5.90	11.80	7.8	5.50	9.60	7.60	10.40	2.1	11.5	94.38, .001
Peak	10.40	2.55	1.75	12.70	7.30	5.80	13.20	6.9	5.70	11.80	3.80	11.90	2.3	8.9	121.52, .001
Circle	4.90	1.90	1.50	8.80	13.30	11.30	9.20	12.6	11.10	7.50	8.40	5.10	2.7	6.7	112, .001

Table 10: Selected differences of the average rankings of the folds (per dataset) for AUC

Selected differences of average rankings	KFrbf-KFrbfcp1	KIRFrbf-KFrbfcp1	KIRFrbf-KFrbf	KFpol-KFpolcp1	KIRFpol-KFpolcp1	KIRFpol-KFpol	KFlin-KFlincp1	KIRFlin-KFlincp1	KIRFlin-KFlin	KFran-KFrancp1	KFburn-KFburncp1	KFran-KFburn	KFrancp1-KFburncp1	RF-KIRFrbf	RF-KFrbfcp1

Heart	-1	4.9	5.9	-0.5	**7.2**	**7.7**	2.3	**7.5**	5.2	0.6	0	1.2	0.6	**-11**	-6.1
Hepatitis	-1.9	0.8	2.7	-1.3	0.15	1.45	-0.7	1.45	2.15	0.5	4	-3.3	0.2	-5.2	-4.4
Ionosphere	0.2	**-6.85**	**-7.05**	0.8	**-6.3**	**-7.1**	-1.95	-5.9	-3.95	2.25	-0.9	0.7	-2.45	**-1.2**	**-8.05**
Pima	4.7	1.9	-2.8	3.1	2.2	-0.9	4.1	1.7	-2.4	3.5	3.5	0.8	0.8	**-6.4**	-4.5
Credit	1.2	**6.4**	5.2	3.8	**9.2**	5.4	0.5	**8**	**7.5**	0.9	0	0.4	-0.5	-11	-4.6
Sonar	0.5	-2.5	-3	-1.2	3.65	4.85	0.8	4.9	4.1	1.6	0.2	2.6	1.2	-2.4	-4.9
wdbc	3.05	2.4	-0.65	-0.55	2.85	3.4	-0.8	1.65	2.45	1.45	-0.8	3.55	1.3	-2.7	-0.3
HeartHun	-0.1	4.4	4.5	-1.4	5.7	**7.1**	1.7	5.6	3.9	-0.1	2	-1.4	0.7	**-9.3**	-4.9
GermanCredit	-2.1	4	6.1	1.5	**7**	5.5	-0.2	6.1	**6.3**	-1.8	0.4	-1.1	1.1	**-10.8**	**-6.8**
AustralianCredit	0.3	5.7	5.4	-0.3	**6.5**	**6.8**	-0.3	5.9	6.2	0.8	-1.5	0.8	-1.5	**-11.2**	-5.5
HorseColic	-3.1	-0.7	2.4	-2.1	2.3	4.4	-1.25	4.9	6.15	2.8	-0.35	0.45	-2.7	**-6.9**	**-7.6**
Ringnorm	**7.2**	0.3	**-6.9**	3.5	-2.3	-5.8	4	-2.3	**-6.3**	2	**8.3**	-0.8	5.5	**9.4**	**9.7**
Peak	**7.85**	-0.8	**-8.65**	5.4	-1.5	**-6.9**	**6.3**	-1.2	**-7.5**	**8**	**9.6**	-0.1	1.5	**7.15**	**6.35**
Circle	3	-0.4	-3.4	-4.5	-2	2.5	-3.4	-1.5	1.9	-0.9	2.4	2.4	5.7	5.2	4.8

The performance of two classifiers differs significantly if their average ranks differ (see Table 8 and 10) by at least the critical difference of 6.2748 (which is the critical difference for a p-value of 0.05, 14 classifiers and 10 folds) (see Demšar 2006 for the formula). The significant differences in Table 8 and Table 10 are put in bold.
Comparing (1) KFrbf and KFrbfcp1, (2) KFpol and KFpolcp1, (3) KFlin and KFlincp1, (4) KFran in KFrancp1, and (5) KFburn and KFburncp1 in both Table 3 and Table 5 we observe that increasing the number of column partitions, as such introducing more diversity, does not have a consistent effect across the datasets. Even more so, on the datasets that we created with a Gaussian distribution (the bottom three datasets in Table 3), setting the number of column partitions to 1 has a beneficial effect for the KF's with a radial basis function (KFrbfcp1, KFrancp1, KFburncp1). Most of these benefits are also significant (see column KFrbf-KFrbfcp1, KFran-KFrancp1, KFburn-KFburncp1 in Table 8 and Table 10). Note that method *ran* will have a probability of one third to select RBF per partition and method *burn* will select the best performing kernel function on the first partition and use that kernel for all partitions. In this case, for the last three datasets, it was always the radial basis function that has been selected by KFburn(cp1). In Table 3, there are very low PCC's for KFpol, KFlin, KFran, and KFburn for the German Credit dataset. In Table 5 the same can be observed for KFpolcp1 and KFlincp1 for the Circle dataset. Moreover Table 4 and Table 6 also show a high inter quartile range (IQR) for these algorithms on these datasets. These are strong signs of overfitting and is likely to be caused by the high dimensionality of the K's.
Comparing (1) KFrbfcp1 and KIRFrbf, (2) KFpolcp1 and KIRFpol, and (3) KFlincp1 and KIRFlin shows that KF performs similarly or considerably higher, and rarely considerably lower. For example, in Table 3 and 5 on the Credit and Australian Credit datasets KF performs around .10 PCC and .10 AUC better than KIRF. This conclusion is confirmed when looking at the columns KIRFrbf-KFrbfcp1, KIRFpol-KFpolcp1, KIRFlin-KFlincp1 in Table 8 and Table 10. In Table 8 (PCC), all 4 significant differences are in favor of KF. In Table 10 (AUC), out of 9 significant differences, 7 are in favor of KF.
KF with method *ran* and *burn* are both designed to choose the kernel for the user. Hence a direct comparison is in order. Interestingly, from Table 3 and Table 5 we see that KFran(cp1) and KFburn(cp1) are very competitive. Hence, none of the differences are significant in Table 8 and 10 (see columns KFran-KFburn and KFrancp1-KFburncp1).
Finally, a comparison of RF with KF and KIRF fulfills our expectations in that RF is superior to the other two when the wrong kernel is selected, or a kernel is selected when none is needed. In contrast, when the right kernel is known (or determined in advance by cross-validation), KIRF and KF do perform considerably better (see columns KFrbfcp1, KIRFrbf. KFrancp1, KFburncp1, RF for the last three datasets in Table 3 and 5). On the 3 relevant datasets KIRFrbf and KFrbfcp1 perform better than RF (significantly in most cases) (see columns RF-KIRFrbf and RF-KFrbfcp1 in Table 8 and 10).

5 Conclusions

In this study we propose an ensemble method for kernel machines. The training data is randomly split into a number of mutually exclusive partitions defined by a row and column parameter. Each partition forms an input space and is transformed by a kernel function into a kernel matrix K. Subsequently, each K is used as training data for a base binary classifier (Random Forest). This results in a number of predictions equal to the number of partitions. A weighted average combines the predictions into one final prediction. To optimize the weights, a genetic algorithm is used.

This approach has the advantage of simultaneously promoting (1) diversity, (2) accuracy, and (3) computational speed. (1) Diversity is fostered because the individual *K*'s are based on a subset of features and observations, (2) accuracy is sought by optimizing the weights with the genetic algorithm, and (3) computational speed is obtained because the computation of each *K* can be parallelized.
Using five times two-fold cross validation we benchmark the classification performance of Kernel Factory against Random Forest and Kernel-Induced Random Forest (KIRF). Our findings are fourfold.
First, the number of column partitions matters. While partitioning the columns is primarily a way of introducing diversity in the ensemble, we found that one partition works better than many. We recommend using one partition and creating more in case of numerical problems when computing the *K*'s (which is often the case in datasets with many features).
Second, Kernel Factory is significantly better than Kernel-Induced Random Forest (KIRF) on several datasets (and performs rarely significantly worse than KIRF). Along with the superior speed of Kernel Factory on large datasets, we recommend it over KIRF.
Third, the two methods (random and burn-in) that automatically select the kernel function perform equally well and using them is a viable strategy when the right kernel function is unknown in advance.
Fourth and final, when using a kernel is appropriate, and the right kernel is specified, both Kernel Factory and Kernel-Induced Random Forest outperform Random Forest significantly.
The main practical implication of this study is that problems involving large datasets, that otherwise would be impossible to analyze using KIRF, can now be analyzed using Kernel Factory. Hence, Kernel Factory opens a doorway to increases in classification performance by kernel functions.
We have made available an open-source R-software package of the algorithm (kernelFactory) at CRAN (Ballings & Van den Poel 2012).

6 Future Research and Limitations

We have six avenues for future research, which at the same time can be considered the limitations of this study.
The first avenue is to try other base classifiers. In this study we use Random Forest as a base classifier because it is good at handling a large number of predictors, and because we wanted to make a direct comparison with KIRF. Other options would be logistic regression with variable selection techniques, or Support Vector Machines. Random Mulitnomial Logit (Prinzie & Van den Poel 2008) is an example of an ensemble method that might benefit from kernels.
The second direction for future research is to use other kernel functions (e.g., Üstün, Melssen, & Buydens 2006). Particularly interesting developments are kernels for categorical data (see Li & Racine 2007). As in KIRF, we exclude the categorical variables when computing the kernel matrix and add them afterwards. It might prove valuable to use kernel functions that can handle categorical data.
While parameter values of Kernel Factory are inspired by practical reasons (computational speed), a third direction is to optimize these parameters. More specifically, there is quite a large body of research investigating what the optimal values are for selection, mutation, and crossover in genetic algorithms (e.g, DeJong & Spears 1990). It may prove valuable to integrate this research and investigate whether it applies to weight optimization for classifier ensembles. Moreover, while in this study we determine the number of partitions by taking the log of the number of rows and columns, these parameters will probably benefit from

optimization too. Although we compare different settings (two values for number of column partitions), future research should study this more in depth.
A fourth avenue is to investigate larger datasets. The true value of Kernel Factory over KIRF lies in large data. Hence, future research should take a special interest in data with a high number of observations and predictors.
The fifth direction for future research is ensemble pruning. Currently all ensemble members are used and weighted in the prediction phase. In order to make Kernel Factory less memory demanding and less computationally expensive members with weight close to zero could be excluded from the scoring phase. Zhou, Wu & Tang (2002) demonstrate that this principle works quite well.
The sixth and final direction is to further explore the mechanism of Kernel Factory with a bias-variance decomposition of the classification error (e.g., Zhou, Wu & Tang 2002). This will provide insight to which factors Kernel Factory owes its strengths.

Acknowledgment

Both authors acknowledge the IAP research network grant No. P6/03 of the Belgian government (Belgian Science Policy).

References

Alpaydin, E. (1999). Combined 5 x 2cv F test for comparing supervised classification learning algorithms. *Neural Computation,* 11(8), 1885–1892.

Baecke, P., & Van den Poel, D. (2011). Data Augmentation by Predicting Spending Pleasure Using Commercially Available External Data. *Journal of Intelligent Information Systems*, 36(3), 367-383.

Baecke, P., & Van den Poel, D. (2012). Including spatial interdependence in customer acquisition models: a cross-category comparison. *Expert Systems with Applications,* 39(15), 12105-12113.

Ballings, M., & Van den Poel, D. (2012). kernelFactory: An ensemble of kernel machines. R package version 0.0.1.

Breiman L. (2001). Random Forests. *Machine Learning*, 45(1), 5 – 32.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). Classification and regression trees. Belmont, C A: Wadsworth.

Buckinx, W., & Van den Poel, D. (2005). Customer base analysis: Partial defection of behaviorally-loyal clients in a non-contractual FMCG retail setting. *European Journal of Operational Research*, 164(1), 252–268.

Coussement, K., & Van den Poel, D. (2008). Churn prediction in subscription services: An application of support vector machines while comparing two parameter-selection techniques. *Expert Systems with Applications*, 34(1), 313-327.

DeJong, K.A., & Spears, W.M. (1990). An Analysis of the Interacting Roles of Population Size and Crossover in Genetic Algorithms. Proc. First Workshop Parallel Problem Solving from Nature, Springer-Verlag, Berlin, 38-47.

Demšar, J. (2006). Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, 7, 1-30.

Dietterich, T.G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7), 1895–1924.

Duda, R.O., Hart, P.E., & Stork, D.G. (2001). Pattern Classification. Wiley, NY, Chapter 8.

Fan, G. (2009). Kernel-Induced Classification Tree and Random Forest. Technical Report, Dept. of Statistics and Actuarial Science, University of Waterloo.

Friedrich Leisch & Evgenia Dimitriadou (2010). mlbench: Machine Learning Benchmark Problems. R package version 2.1-1.

Hanley, J.A., & McNeil, B.J. (1982). The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143(1), 29–36.

Jäkel, F., Schölkopf, B., & Wichmann, F. A. (2007). A tutorial on kernel methods for categorization. *Journal of Mathematical Psychology*, 51(6), 343-358.

Karatzoglou, A., Smola, A., Hornik, K., Zeileis, A. (2004). kernlab - An S4 Package for Kernel Methods in R. *Journal of Statistical Software*, 11(9), 1-20.

Kim, Y., Street, W.N., & Menczer, F. (2002). Meta-Evolutionary Ensembles, Proceedings of 2002 International Joint Conference on Neural Networks, 3, 2791-2796.

Langley, P. (2000). Crafting papers on machine learning. In: Langley, P. (Ed.), *Proceedings of 17th International Conference on Machine Learning*. ICML-2000. Stanford University, Stanford.

Larivière, B., & Van den Poel, D. (2005). Predicting customer retention and profitability by using random forests and regression forests techniques. *Expert Systems with Applications*, 29(2), 472-484.

Li, Q., & Racine, J. (2007). Nonparametric Econometrics: Theory and Practice. Princeton University Press, Princeton.

Liaw, A., & Wiener,M. (2002). Classification and Regression by randomForest. *R News*, 2(3), 18-22.

Luo, T, Kramer, K, Goldgof, D.B., Hall, L.O., Samson, S., Remsen, A., & Hopkins, T. (2004). Recognizing plankton images from the shadow image particle profiling evaluation recorder. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, 34 (4), 1753–1762.

Merkle E. C., & Shaffer V. A. (2011). Binary recursive partitioning: Background, methods, and application to psychology. *British Journal of Mathematical & Statistical Psychology*, 64(1), 161-181.

Nemenyi, P. B. (1963). Distribution-free multiple comparisons. PhD thesis, Princeton University.

Park, J.I., Liu, L., Ye, X.P., Jeong, M.K., Jeong, Y.S. (2012). Improved prediction of biomass composition for switchgrass using reproducing kernel methods with wavelet compressed FT-NIR spectra. *Expert Systems with Applications*, 39(1), 1555-1564.

Prinzie, A., Van den Poel, D. (2008). Random forests for multiclass classification: Random MultiNomial Logit. *Expert Systems with Applications*, 34(3), 1721-1732.

Provost, F., Fawcett, T., & Kohavi, R. (1998). The case against accuracy estimation for com-paring induction algorithms. In: J., Shavlik (Ed.), *Proc. of 15th International Conference on Machine Learning*. ICML-1998. Morgan Kaufman, San Francisco, CA.

R Core Team (2012). R installation and Administration. Version 2.15.1, 30.

Schölkopf, B., & Smola A. J. (2002). Learning with Kernels, Support Vector Machines, Regularization, Optimization, and Beyond. Cambridge, MA: MIT Press.

Shawe-Taylor, J., & Cristianini, N. (2004). Kernel Methods for Pattern Analysis. Cambridge, UK, Cambridge University Press.

Sing, T., Sander, O., Beerenwinkel, N., & Lengauer, T.(2009). ROCR: Visualizing the performance of scoring classifiers. R package version 1.0-4.

Sylvester, J. , & Chawla, N. V. (2005). Evolutionary ensembles: Combining learning agents using genetic algorithms. In AAAI Workshop on Multi-agent Learning, pp. 46-51.

Thorleuchter, D.; Van den Poel, D. (2012) Predicting e-commerce company success by mining the text of its publicly-accessible website. *Expert Systems with Applications*, 39(17), 13026-13034.

Üstün, B., Melssen, W.J., & Buydens, L.M.C. (2006). Facilitating the application of Support Vector Regression by using a universal Pearson VII function based kernel. *Chemometrics and Intelligent Laboratory Systems*, 81(1), 29 – 40.

Webb, I. (2000). Multiboosting: A technique for combining boosting and wagging. *Machine Learning*, 40(2), 159–197.

Willighagen, E. (2005). genalg: R Based Genetic Algorithm. R package version 0.1.1.

Yao, X. ,& Liu, Y. (1998). Making Use of Population Information in Evolutionary Artificial Neural Networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 28(3), 417-425.

Zhou, Z.-H., Wu, J., & Tang, W. (2002). Ensembling Neural Networks: Many Could Be Better Than All. *Artificial Intelligence,* 137(1/2), 239-263.