Separate Term Project

DATA11002 Introduction to Machine Learning (Autumn 2020)

Juuso Luhtala

03 September 2022

Contents

1 Introduction

In this report we evaluate four different classifiers belonging to the classifier families **neural networks**, **generalized linear models**, **random forests** and **support vector machines** using four different data sets containing a two-class classification problem. This report partially reproduces some of the analysis done in the paper *Do We Need Hundreds of Classifiers to Solve Real World Classification Problems?* (Fernández-Delgado et al., 2014a, 2014b). We will be referencing this paper throughout this report.

The data sets are processed with R and the classifiers are implemented using the model training interface from the R package `caret` (Kuhn, 2020a, 2020b). The selected models *avNNet*, *glmnet*, *parRF* and *svmPoly* belong to the classifier families **neural networks (NNET)**, **generalized linear models (GLM)**, **random forests (RF)** and **support vector machines (SVM)**, respectively. All of these models are available through the `train` interface/function of the `caret` package.

In Section 2, we review the data sets we selected to use in this report and discuss their characteristics and how the data was pre-processed. In Section 3, we introduce the classifiers we selected and discuss briefly the classifier families they belong to. In Section 4, we discuss the validation methodology used in evaluating the selected classifiers, go through the implementation details of the classifiers and visually inspect parameter tuning in model selection. In Section 5, we collect and analyze the results of our validation methods and compare them with the results of Fernández-Delgado et al. Finally, in Section 6, we discuss our conclusions from our research and from the research of Fernández-Delgado et al.

1.1 How to Reproduce This Report and to Execute Separately Distributed Code

This PDF report has been generated with R Markdown from the file `Separate_term_project_report.Rmd`. Three R files `learning_utilities.R`, `check_datasets.R` and `learn_models.R` (located in a directory called R), the file `references.bib` and the file `harvard-educational-review.csl` are distributed separately in a zip file. The file `learning_utilities.R` contains utility functions used to train the classifiers. The file `check_datasets.R` contains functions for performing various checks on the data sets. The R script `learn_models.R` trains all of the models using the `caret` package and finally, saves the resulting model objects in the file `RData/models.RData`. The files `references.bib` and `harvard-educational-review.csl` are included so that the R Markdown file can be successfully knitted and references will be correct.

The file `Separate_term_project_report.Rmd` can be knitted (or the containing code can be executed) using the RStudio development environment. For any execution of code to be successful, the file `RData/models.RData` must have been created/saved by the R script `learn_models.R` and the user must have a data directory containing the appropriate data files which are specified in the file `learn_models.R` and which are provided by Fernández-Delgado et al. Additionally, a file called `results.txt` provided by Fernández-Delgadoet al. containing their accuracy tables must be located in a directory called `Original_paper_results`.

2 Datasets Selected

Of the 121 data sets Delgado et al. specified (Fernández-Delgado et al., 2014b), we selected four data sets containing a two-class classification problem for this report. The selected data sets were **breast-cancer-wisc-prog**, **haberman-survival**, **parkinsons** and **spambase**. The dataset **breast-cancer-wisc-prog** has also the abbreviation **bc-wisc-prog** in the paper by Fernández-Delgado et al. and thus, we also use this abbreviation in this report.

The author of this report has worked with real world patient data performing various data analysis tasks and this topic will also feature in the author's Master's thesis. Therefore, the data sets **bc-wisc-prog**, **haberman-survival** and **parkinsons** were natural choices since they contain medical and survival data.

On the other hand, the fourth data set, **spambase**, gives a more balanced alternative with significantly more observations and features and a more balanced majority-minority class distribution.

The selection of these four data sets is quite evidently biased and can in no way approximate the diverse 121 data sets that Fernández-Delgado et al. used.

2.1 Pre-Processing and Missing Values

Fernández-Delgado et al. provided a set of pre-processed files (with the naming convention `datasetname_R.dat`) to be used with the classifiers implemented through `caret`. We checked that the data in these files was approximately centered i.e. had a mean which was approximately zero and it was also approximately scaled i.e. had a standard deviation which was approximately one. Here the wording approximately refers to the fact that the target values of zero and one were reached for every predictor in every selected data set within a tolerance level of 10^{-5}. Furthermore, the argument `preProcess= c("center", "scale")` is given to the `train` function when the models are trained through the `caret` interface in order to ensure that the the data is centered and scaled.

Fernández-Delgado et al. reported that all missing predictor values were replaced with the value 0. We checked that the data in files `breast-cancer-wisc-prog_R.dat`, `haberman-survival_R.dat`, `parkinsons_R.dat` and `spambase_R.dat` had no missing values even though some of the original data sets contained some missing values. Therefore, we conclude that Fernández-Delgado et al. had already replaced any missing values with zero in these pre-processed files.

Fernández-Delgado et al. converted nominal values into numeric values by the process of quantization. These are already included in the pre-processed files provided by them. However, it seems that our selected data sets contains no predictors which were originally nominal. Therefore, the note Fernández-Delgado et al. make of the impact of quantization to distance based classifiers should not be very relevant in this report.

In summary, our pre-processing matches Fernández-Delgado et al. since they did not perform any additional pre-processing. Next, we will briefly describe our selected data sets in more detail. The descriptions are based (sometimes verbatim) on the files `datasetname.names` provided by Fernández-Delgado et al.

2.2 breast-cancer-wisc-prog

The data set **breast-cancer-wisc-prog** (abbreviated bc-wisc-prog) presents the classification problem of determining whether breast cancer status will be recurrent or nonrecurrent based on 33 real valued features. The vast majority of the features (30) are computed from a digitized image of a fine needle aspirate of a breast mass. The remaining three features are time (recurrence time if outcome is recurrent and disease-free time if outcome is nonrecurrent), tumor size and the number of positive axillary lymph nodes observed at time of surgery (lymph node status).

This data set contains 198 observations in total with 33 features. The majority class represents 76.3 % of the observations (151 nonrecurrent outcomes). The original data contained four missing lymph node status cases.

2.3 haberman-survival

The data set **haberman-survival** contains cases from a study that was conducted between 1958 and 1970 on the survival of patients who had undergone surgery for breast cancer. The two possible outcomes are survival for 5 years or longer after surgery or death within 5 years of surgery.

The data set contains only 3 predictors: Patient's age at the time of surgery, patient's year of operation (scale year - 1900) and the number of positive axillary nodes detected. The majority class represents 73.5 % of the observations. The original data contained no missing values.

2.4 parkinsons

The data set **parkinsons** contains biomedical voice measurements from 31 people, 23 who have Parkinson's disease. Each feature is a particular voice measure and there are a total of 22 features. Each patient has around 6 measurements and the total number of observations is 195.

The majority class (patients with Parkinson*s) represents 75.4 % of the observations. There is no mention of missing values in the documentation of this data set.

2.5 spambase

The data set **spambase** is by far the largest data set that we use in this report. It contains 4601 observations and 57 mostly real valued features (the others have integer values). The task is to predict whether an email is spam (1) or not (0).

The majority class (email that are not spam) represents 60.6 % of the observations. Again, there is no mention of missing values.

3 Selected Classifier Families and Their Representatives

The author of this report selected the classifiers *avNNet*, *parRF* and *svmPoly* because they performed extremely well in a wide variety of performance metrics both in multi-class and two-class classification problems in the paper by Fernández-Delgado et al. Their respective classifier families were also generally the most successful ones in the aforementioned paper.

The author of this report has some previous (albeit modest) experience with generalized linear models and the R package `glmnet`. Therefore, *glmnet* was selected as a somewhat familiar model to compare against the other classifiers. Fernández-Delgado et al. point out that researchers tend to favor classifiers which they are familiar with even though better alternatives might be available. Thus, the selection of *glmnet* as the fourth classifier captures this theme and also allows the author of this report to compare a familiar classifier against the more exotic yet successful classifiers. Although *glmnet* had only mediocre performance in general classification tasks in the paper by Fernández-Delgado et al., in two-class classification problems it performed much better. This fact is another strong justification for selecting *glmnet* as the fourth classifier.

Fernández-Delgado et al. trained *avNNet*, *parRF* and *svmPoly* models through the `train` interface of the `caret` package denoting them as **avNNet_t**, **parRF_t** and **svmPoly_t**, respectively (the t refers to the `caret` package). However, they found some errors when trying to use *glmnet* through the caret interface and thus, they used the direct R implementation of *glmnet* denoting it as **glmnet_R**. However, we were able to use *glmnet* through `caret`. Therefore, we trained all of the models using `caret` and we will denote them simply as *avNNet*, *glmnet*, *parRF* and *svmPoly*.

We will next examine the the selected classifiers in more detail. In addition, a modest amount of background theory behind these classifiers is discussed.

3.1 Neural Networks (NNET): avNNet

The implementation *avNNet* belongs to the `caret` package and it aggregates several neural networks which are trained using several different random seeds. The tunable parameters are the number of hidden units (parameter **size**), the weight decay (parameter **decay**) and the logical argument **bag**.

The weight decay parameter acts as regularizer against overfitting (Hastie et al., 2009, p. 398). It is considered better to have more hidden units since too few hidden units cannot necessarily capture the nonlinearities in the data but having too many hidden units (and thus potentially overfitting) can be controlled with regularization (Hastie et al., 2009, p. 400). Surprisingly these theoretical notions are not supported by the

parameter tuning results on our selected four datasets as can be seen later in Section 4.3 since the resulting *avNNet* models tend to favor smaller number of hidden units.

Fernández-Delgado et al. specify the values size $\in \{1, 3, 5\}$, decay $\in \{0, 0.1, 0.0001\}$ and bag = FALSE. They also use the default value of training 5 different neural networks (the parameter `repeats` = 5 for the avNNet function from the `caret` package). The selection of bag = FALSE implies that bagging is not used when training the model.

3.2 Generalized Linear Models (GLM): glmnet

The model *glmnet* belongs to the `glmnet` package and it fits generalized linear models using penalized maximum likelihood (Friedman et al., 2010; Hastie et al., 2021). We will use logistic regression to solve the classification problem and we assume that the response variable takes binary values in $\mathcal{G} = \{1, 2\}$. In regularized logistic regression, the objective is to minimize the penalized negative binomial log-likelihood

$$\min_{(\beta_0,\beta)\in\mathbb{R}^{p+1}} -\left(\frac{1}{N}\sum_{i=1}^{N} y_i \cdot (\beta_0 + x_i^\intercal \beta) - \log(1 + \exp(\beta_0 + x_i^\intercal \beta))\right) + \lambda\left(\frac{1}{2}(1-\alpha)\|\beta\|_2^2 + \alpha\|\beta\|_1\right),$$

where $y_i = \mathbb{I}(g_i = 1)$ (Hastie et al., 2021, p. 16). The role of the penalty function (the latter term) is to avoid overfitting by penalizing more complex models since the terms $\|\beta\|_2^2$ and $\|\beta\|_1$ both grow when the components of β grow larger and also if more of the components of β have non-zero values.

The parameter α controls the **elastic net penalty function** and through this parameter both ridge ($\alpha = 0$) and lasso ($\alpha = 1$) regression and a mix of both can be fitted (Hastie et al., 2021, p. 2). Therefore, α is also known as the **mixing variable** (in `caret` the parameter alpha is known as the **mixing percentage**) which controls how much either the ridge or lasso penalty is weighted in the optimization problem. The parameter λ controls the weight of the entire penalty function (in `caret` the parameter lambda is known as the **regularization parameter**).

Fernández-Delgado et al. trained the *glmnet* model directly using the R package `glmnet`. Therefore, they do not specify any λ values to use for the parameter tuning. The quite likely reason for this is that when used directly, by default, the *glmnet* function trains up to 100 different generalized linear models (with different values of λ) selecting the lambda range automatically based on data. The `caret` package however requires both λ and α values

Since we decided to use *glmnet* through `caret`, we had to find suitable values for the tunable parameters α (parameter **alpha**) and λ (parameter **lambda**). We separately trained a *glmnet* model directly with the `glmnet` package for each selected data set and then inspected the λ sequences that these models generated. We picked an approximate minimum and maximum value for λ based on these sequences. We generated 100 values linearly spread out between $\log(\min(\lambda))$ and $\log(\max(\lambda))$ and converted the values back to original scale by raising them to the power of e. Our goal was to approximately mimic the way `glmnet` generates its λ sequences (Hastie et al., 2021, p. 6). For the values of alpha we selected $\alpha \in \{0, 0.2, 0.5, 0.8, 1\}$ since Fernández-Delgado et al. did not specify any values, although they specifically mention lasso and also vaguely refer to the elastic net.

We realize that *glmnet* might be the best tuned model out of the four models we implement because of the extremely large amount of different parameter configurations $100 \cdot 5 = 500$. This might violate the spirit of the paper by Fernández-Delgado et al., since their goal was not to find the absolutely best tuning parameters for each data sets by searching over a large amount of different parameter configurations. However, had we used the direct R implementation of *glmnet*, training a model for even a singular value of α would have resulted in *glmnet* trying to use up to 100 different values for lambda by default. Therefore, our approach seems justified in the end.

3.3 Random Forests (RF): parRF

The model *parRF* is parallel implementation of the random forest classifier from the `randomForest` package. The random forest classifier is described by the authors of the `randomForest` package in their article *Classification and Regression by randomForest* (Liaw & Wiener, 2002) as follows:

1) Draw n_{tree} bootstrap samples from the original data
2) For each of the bootstrap samples, grow an unpruned classification tree, with the following modification: At each node, randomly sample m_{try} of the predictors and choose the best split from among those variables
3) Predict new data by majority votes among the n_{tree} trees.

The idea of random forests is to train a large collection of **de-correlated** trees and then to to average them in regression problems or to pick the majority vote in classification problems (Hastie et al., 2009, pp. 587–588). The goal is to reduce variance by trying to grow trees which have small pairwise positive correlation. The random selection of predictors for splitting candidacy is how this is attempted. As noted in the above algorithm, we try to do this by tuning the parameter m_{try}.

The model *parRF* has only one tunable parameter, **mtry**, which corresponds to the aforementioned m_{try}. Fernández-Delgado et al. specified the values $m_{try} = 2, 4, 6, 8$ for parameter tuning. We also use these values when training our models unless the amount of predictors is less than m_{try}. This is the case for the data set **haberman-survival** which has only 3 predictors. Therefore, we followed the convention of Fernández-Delgado et al. and used $m_{try} = 2, 3$ for this particular data set. All the other three data sets we used contain more predictors than the the largest value of $m_{try} = 8$ so no further modification for the possible values of **mtry** were necessary.

3.4 Support Vector Machines (SVM): svmPoly

The implementation *svmPoly* belongs to the `kernlab` package (Karatzoglou et al., 2004) and it fits a SVM with a polynomial kernel function. The polynomial kernel is of the form

$$k(x, x') = (\text{scale} \cdot \langle x, x' \rangle + \text{offset})^{\text{degree}}.$$

The tunable parameters for *svmPoly* are the polynomial degree (parameter **degree**), the scale (parameter **scale**) and the cost (parameter **C**). Fernández-Delgado et al. specify the values scale = 0.001, 0.01, 0.1, degree = 1, 2, 3 and C = 0.25, 0.5, 1 for parameter tuning. The value offset is set to 1 which is the default value for polynomial kernel in the `kernlab` package (Karatzoglou et al., 2019).

4 Validation Methodology, Implementation Details and Parameter Tuning Visualizations

4.1 Validation Methodology

In their paper, Fernández-Delgado et al. describe how they developed and estimated their models in two phases. First, they performed the parameter tuning for each model using a simple 50-50 training and test set split. Second, they selected the best parameter values found in the first phase and then they performed 4-fold cross-validation to estimate prediction accuracy. The resulting 4-fold cross-validation produced four different accuracies which were averaged to get an estimate for the models prediction ability.

We try to replicate the methodology described by Fernández-Delgado et al. as closely as possible.

4.2 Implementation Details

For each data set, Fernández-Delgado et al. provided training and test set indices for the first phase in a file called `conxuntos.dat`. For the 4-fold cross-validation in the second phase, they provided the four training-validation index pairs in the file `conxuntos_kfold.dat`. All of these index collection contained the value 0, and we added 1 to each index in these indes sets since R starts to index from 1 and we assume that `caret` as an R package requires indices which start from the value 1. Additionally, some data sets produced errors when this procedure was not done. Finally, we verified that the index sets were correct for each of the data set (verified that the index sets equal $\{1, 2, \ldots, N\}$, where N is the number of observations (rows) in the data set).

We inserted these processed tuning and cross-validation index sets into the the `trainControl` function of `caret` by using the `index` and `indexOut` parameters, respectively. The `trainControl` function controls the computational details of the `train` function (Kuhn, 2020b). In the terminology of `caret`, `index` contains the training indices and `indexOut` contains the holdout (i.e. validation/test) indices for a selected data set. The value returned by `trainControl` was fed as a parameter for the `train` function of the `caret` package.

We programmed this process with R and tried to use the ready-made functionality of `caret` as much as possible. For each model the tunable parameters were specified in the parameter `tuneGrid`. As was mentioned before, we selected the pre-processing option of centering and scaling the data. We tried to have some influence on the reproducibility of results by setting the same seed before each model was trained on a particular data set (the same seed for each model-data combination). We did not specify any parallel computing backend. However, the lack of parallel processing should not influence the accuracies of the models, only speed of execution.

Finally, we illustrate the process described in the above paragraphs by showing (a slightly simplified version of) the R code for this two-phased `caret` model development:

```
caret_pipeline <- function(dataset, tuning_index, tuning_indexOut,
                           cv_index, cv_indexOut, model_name, modelGrid, seed) {

  set.seed(seed)
  tuneFit <- train(as.factor(clase) ~ ., data = dataset,
                   method = model_name,
                   preProcess= c("center", "scale"),
                   trControl = trainControl(returnResamp = "all",
                                            index = tuning_index,
                                            indexOut = tuning_indexOut),
                   tuneGrid = modelGrid)

  cvFit <- train(as.factor(clase) ~ ., data = dataset,
                 method = model_name,
                 preProcess= c("center", "scale"),
                 trControl = trainControl(returnResamp = "all",
                                          index = cv_index,
                                          indexOut = cv_indexOut),
                 tuneGrid = tuneFit$bestTune)

  return(list(tuneFit, cvFit))
}
```

4.3 Parameter Tuning Visualizations

In Figure 1, we visualize the parameter tuning process for each of the 16 model-data combinations. Each of the individual 16 plots corresponds to a unique model-data combination. The data sets form the rows in

order **bc-wisc-prog**, **haberman-survival**, **parkinsons** and **spambase**. The columns specify the different models in order *avNNet*, *glmnet*, *parRF* and *svmPoly*. From this plot one can visually compare the accuracies between each model for a particular data set or one can see how different parameters for the same model achieve sometimes drastically different accuracies.

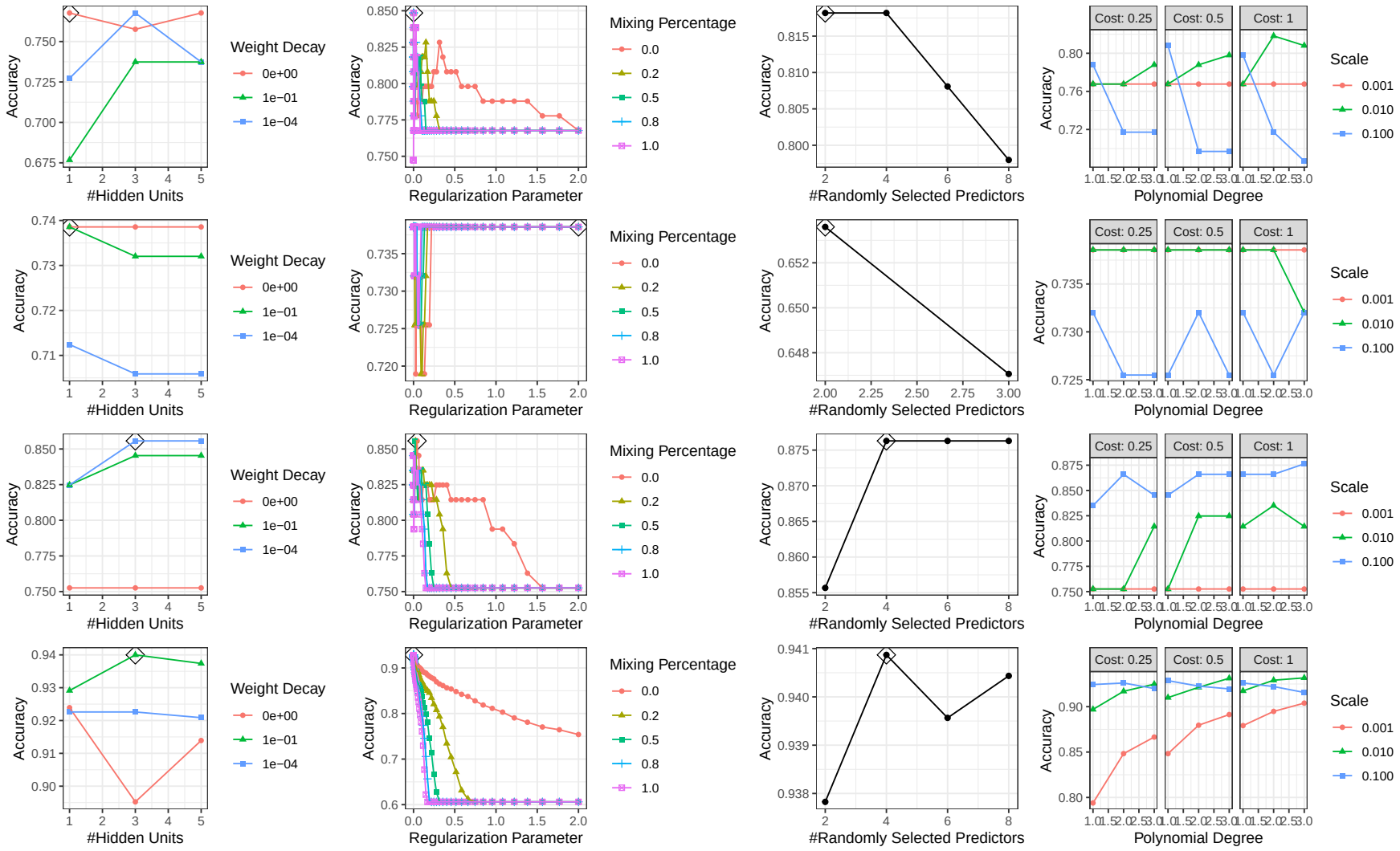

Figure 1: Accuracy and tuning parameter values for all classifier and data set combinations. The rows correspond to data sets (bc-wisc-prog, haberman-survival, parkinsons, spambase) and the columns correspond to models (avNNet, glmnet, parRF, svmPoly). In the first three columns the parameter combination giving the highest accuracy is highlighted with a diamond. The fourth column does not contain any highlighting since the existing functionality of the caret package could not place the diamond.

5 Results

5.1 Model Accuracies Estimated With 4-Fold Cross-Validation

After parameter tuning, the selected model accuracies were estimated with 4-fold cross-validation. In Table 1, we see the results of the 4-fold cross-validation for each classifier and for each dataset. The accuracy reported for each classifier-dataset combination is the average accuracy obtained in the four cross-validation trials. We see that for the most part there are no drastic differences in classifier performance for a particular dataset. The classifier *parRF* (belonging to the family random forests) performs much worse than the others on the data set **haberman-survival**. Similarly, the classifier *glmnet* (belonging to the family generalized linear models) performs notably worse than the others on the data set **parkinsons**. Apart from these two classifier-data combinations, other major under- or over-performers do not stand out.

Table 1: Average accuracies (in %) obtained by 4-fold cross-validation. The datasets correspond to rows and the columns correspond to the four selected classifiers.

	avNNet	glmnet	parRF	svmPoly
bc-wisc-prog	77.6	80.6	80.1	78.1
haberman-survival	76.3	73.7	66.4	73.7
parkinsons	88.8	83.7	89.8	88.8
spambase	94.1	92.4	95.1	93.3

5.2 Accuracy Table Obtained by Other Researchers

In Table 2, the accuracies reported by Fernández-Delgado et al. for the same classifiers and data sets are shown. Notice that there are some differences in the results, although for the most part they are only minor.

Table 2: Average accuracies (in %) obtained by Fernández-Delgado et al. using the same 4-fold cross-validation procedure. The datasets correspond to rows and the columns correspond to the classifiers used by Fernández-Delgado et al.

	avNNet_caret	glmnet_R	parRF_caret	svmPoly_caret
breast-cancer-wisc-prog	77.8	81.0	80.9	78.3
haberman-survival	76.1	74.0	67.7	74.2
parkinsons	90.3	85.2	89.2	91.3
spambase	94.3	92.1	95.1	93.5

5.3 Discussion of the Differences Between the Two Accuracy Tables

In Table 3, we have calculated the absolute difference between our accuracy results and those reported by Fernández-Delgado et al. We first rounded our accuracy percentages to have the same level of accuracy as was reported by Fernández-Delgado et al. (1 decimal place).

Table 3: Absolute difference (in percentage points) between the accuracies achieved in this report and the accuracies reported by Fernández-Delgado et al.

	avNNet	glmnet	parRF	svmPoly
bc-wisc-prog	0.2	0.4	0.8	0.2
haberman-survival	0.2	0.3	1.3	0.5
parkinsons	1.5	1.5	0.6	2.5
spambase	0.2	0.3	0.0	0.2

One possible way to explain the differences in accuracy results for the classifiers *avNNet* and *parRF* is that the randomness included in these classification procedures. We do not know what seeds, if any, Fernández-Delgado et al. used so this avenue is impossible to explore any further. For *glmnet* we do not know exactly what values of λ and α Fernández-Delgado et al. used, although they mention Lasso and elastic net in their specification. Thus, we have one possible explanation for the differences of *glmnet* classification results. Additionally, the differences are very small for most classifier-data combinations.

The difference of the performance of *svmPoly* on the **parkinsons** data set is much harder to explain. Our understanding of support vector machines and the package `kernlab` which provides the classifier *svmPoly* is very limited. However, it seems that there is no randomness involved in the training and validation of this classifier once the index sets have been explicitly specified.

5.3.1 The Results of svmPoly With the parkinsons Data Set

Fernández-Delgado et al. provided detailed results of their work. From these detailed results we find out that for the **parkinsons** data and the *svmPoly* classifier, they calculated the 4-fold cross-validation error using the parameter values C=1, degree=3 and scale=0.1. We found out that we obtained the same exact parameter values, however our accuracy estimates differ:

```
Accuracies$svmPoly[3] # Accuracy for parkinsons data set
```

```
## [1] 88.77551
```

```
# Model: svmPoly, data: parkinsons, 4-fold CV parameters
fitted_models[[15]][[2]]$results$C
```

```
## [1] 1
```

```
fitted_models[[15]][[2]]$results$degree
```

```
## [1] 3
```

```
fitted_models[[15]][[2]]$results$scale
```

```
## [1] 0.1
```

Could our procedure for adding the value 1 to every index have been the wrong approach? On the other hand it seems justified, since indexing in R starts from 1 and an R package should follow this convention.

Other possible explanations for the different results for the *svmPoly* classifier is that something has changed in the underlying software implementations (`kernlab`, `caret`, R version etc.).

5.4 Friedman Ranks and Average Accuracies

In Table 4, the classifiers are ranked within each data set according to their performance. The highest accuracy gives the lowest rank (1) and vice versa. Two different classifier pairs achieved the same exact

accuracy. They they were given the average of the ranks that would have been assigned if the accuracies were not exactly the same and the classifiers below them were ranked normally (their rank is decided simply by how many classifiers place above them).

Table 4: The classifier ranks for each dataset. The highest accuracy gives the lowest rank, the lowest accuracy gives the highest rank. When classifiers achieved the same exact accuracy, they were given the average of the ranks that would have been assigned if the accuracies were not exactly the same.

	avNNet	glmnet	parRF	svmPoly
bc-wisc-prog	4.0	1.0	2	3.0
haberman-survival	1.0	2.5	4	2.5
parkinsons	2.5	4.0	1	2.5
spambase	2.0	4.0	1	3.0

In Table 5, we see the four classifiers ordered according to their Friedman ranks. Our accuracy results and the ordering of the classifiers differ from those reported by Fernández-Delgado et al. in Table 9 of their paper (Fernández-Delgado et al., 2014a) but this is to be expected since we only used four data sets compared to the 121 data sets used by Fernández-Delgado et al.

Table 5: The classifiers ordered according to their Friedman rank. For each classifier the Friedman rank is the mean rank achieved in all four datasets. Similarly, for each classifier the reported accuracy is the mean accuracy achieved in all four datasets. The mean accuracy is rounded to one decimal place.

Rank	Classifier	Mean Accuracy (%)
2.000	parRF	82.9
2.375	avNNet	84.2
2.750	svmPoly	83.5
2.875	glmnet	82.6

The random forest classifier *parRF* performed very poorly in the **haberman-survival** data set compared to the other classifiers. Thus, its average accuracy is weighted down although, in the other three data sets it performed extremely well compared to the other classifiers. The neural network classifier *avNNet* performed relatively well in all data sets and thus it has the highest average accuracy. The support vector machine classifier *svmPoly* also had more stable performance through the four different data sets, and thus it also has a higher average accuracy than *parRF*. The generalized linear model classifier *glmnet* has the worst performance both in Friedman ranking and in average accuracy. However, the actual numerical values are not that far of from the other classifiers. Based on only four data sets, drawing any definitive conclusions for the superiority of a particular classifier is not advisable.

5.5 Probability of Achieving the Maximum Accuracy (PAMA)

In Table 6, the probability of achieving the maximum accuracy (PAMA) is shown. PAMA is calculated by counting how many times a classifier achieves rank 1 and then dividing the result with the number of data sets.

Table 6: The probability of achieving maximum accuracy (PAMA). PAMA is calculated by how many times a classifier achieves rank 1 and then dividing the result with the number of datasets.

No.	Classifier	PAMA (%)
1	parRF	50
2	avNNet	25
3	glmnet	25
4	svmPoly	0

5.6 Percentage of the Maximum Accuracy (PMA)

In Table 7, the "maximum accuracies" on any given data set are shown. This "maximum accuracy" only represents the highest accuracy obtained by any of our four chosen classifiers. Thus, there might be some other more accurate classifiers on this data or the existing classifiers could be trained/tuned better so that they could achieve a higher accuracy. Therefore, the "maximum accuracies" are our best guess for the actual maximum accuracy.

Table 7: The 'maximum accuracies' for each data set (in %). The 'maximum accuracy' is quoted since it only represents the best accuracy obtained by one of the chosen four classifiers.

	bc-wisc-prog	haberman-survival	parkinsons	spambase
Max_dataset_accuracy	80.6	76.3	89.8	95.1

In Table 8, we see percentage of maximum (data set) accuracy obtained by each classifier for each data set. Our previous observations for classifier performance, are now much easier to notice.

Table 8: The percentage of maximum (data set) accuracy obtained by each classifier. Here the average accuracies obtained with 4-fold cross-validation are divided by the 'maximum accuracy' for any particular data set.

	avNNet	glmnet	parRF	svmPoly
bc-wisc-prog	96.2	100.0	99.4	96.8
haberman-survival	100.0	96.6	87.1	96.6
parkinsons	98.9	93.2	100.0	98.9
spambase	98.9	97.2	100.0	98.2

In Table 9, we see the percentage of the maximum accuracy (PMA) where the percentage of maximum accuracy obtained by each classifier for each data set is averaged over all of the data sets. From Table 8 we could already see that *avNNet* and *svmPoly* had a steady level of good performance. This observation is confirmed by the average values. Although, *parRf* performs extremely well on three out four data sets, one particularly bad performance lowers its average so much that it places last on this PMA list (although it nearly has the same average accuracy as *glmnet*).

Table 9: The percentage of the maximum accuracy (PMA). The percentage of maximum accuracy obtained by each classifier for each data set is averaged over all the datasets. The PMA column shows this average value.

No.	Classifier	PMA (%)
1	avNNet	98.5
2	svmPoly	97.6
3	glmnet	96.7
4	parRF	96.6

6 Discussion of Results

While writing this report and implementing the classifiers, we realized how useful frameworks like `caret` really are. Without these type of frameworks, it would be very laborious to implement any machine learning models one is not intimately familiar with. Luckily packages like `caret` significantly reduce the amount of effort needed to implement unfamiliar machine learning models.

We learned that random forests, neural networks and support vector machines are very powerful machine learning families. Based on results of Fernández-Delgado et al. and the results we presented in this report, when one is tasked with solving a machine learning problem, these families should be among the considered solutions.

6.1 Usefulness and Limitations of the Approach of Fernández-Delgado et al.

The approach outlined by Fernández-Delgado et al. is generally useful since their methodology is general enough to give a fair idea of what classifiers might perform well in average. Thus, if a researcher does not have much prior information of how different classifiers perform on their data, they should base their selection on the analysis done by Fernández-Delgado et al.

In their paper Fernández-Delgado et al. state quite clearly many limitations for how they are selecting the best classifiers. For example, they discuss the problems with converting nominal values into the quantitative scale and they point out that 4-fold cross-validation is not powerful enough.

The amount of combinations for parameter tuning is generally very low in the paper by Fernández-Delgado et al., although this approach is understandable since they were not trying to find the most tuned models and they wanted to keep the computational cost down.

The issue of sensitivity and specifity is not discussed much by Fernández-Delgado et al., although they mention ROC curves when discussing binary classification problems. For some of the 121 data sets they used, simple accuracy is not a very good quality measure for a classifier, since in some cases false positives and false negatives in classification can have huge real world consequences.

6.2 Selecting a Classifier for Real World Problems

Fernández-Delgado et al. provide a very useful framework for comparing and selecting a classifier for real world problems. We could choose a large collection of classifiers and then test them on our real world data set in a manner similar to Fernández-Delgado et al. but introducing some improvements in the process.

In the first phase, the parameter tuning process could be made more exhaustive i.e. we could test a larger combination of different parameter values and then select the best parameter. In the second phase, per the suggestions made by Fernández-Delgado et al., we could use a more powerful cross-validation procedure (for

example 10-fold or leave-one-out cross-validation). Finally, the winner of this cross-validation procedure would be our selected classifier.

The selection and implementation of a diverse set of different classifiers is made easier by machine learning frameworks such as the `caret` package. However, the process we described in the previous paragraph would have a large computational cost. Therefore, parallel processing should be used as much as possible in order to reduce the computational cost.

References

Fernández-Delgado, M., Cernadas, E., Barro, S., & Amorim, D. (2014a). Do We Need Hundreds of Classifiers to Solve Real World Classification Problems? *Journal of Machine Learning Research*, *15*(90), 3133–3181. http://jmlr.org/papers/v15/delgado14a.html

Fernández-Delgado, M., Cernadas, E., Barro, S., & Amorim, D. (2014b). *Do We Need Hundreds of Classifiers to Solve Real World Classification Problems?* retrieved March 13, 2021. http://persoal.citius.usc.es/manuel.fernandez.delgado/papers/jmlr/

Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. *Journal of Statistical Software*, *33*(1), 1–22. http://www.jstatsoft.org/v33/i01/

Hastie, T., Qian, J., & Tay, K. (2021). *An Introduction to glmnet.* retrieved March 13, 2021. https://cran.r-project.org/web/packages/glmnet/vignettes/glmnet.pdf

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2nd ed.). Springer.

Karatzoglou, A., Smola, A., & Hornik, K. (2019). *Package "kernlab".* retrieved March 14, 2021. https://cran.r-project.org/web/packages/kernlab/kernlab.pdf

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. http://www.jstatsoft.org/v11/i09/

Kuhn, M. (2020a). *caret: Classification and Regression Training.* https://CRAN.R-project.org/package=caret

Kuhn, M. (2020b). *Package "caret".* retrieved March 13, 2021. https://cran.r-project.org/web/packages/caret/caret.pdf

Liaw, A., & Wiener, M. (2002). Classification and Regression by randomForest. *R News*, *2*(3), 18–22. https://CRAN.R-project.org/doc/Rnews/