Master Thesis

for the acquisition of the academic degree

Master of Science (M.Sc.)

in the subject of Statistics

Faculty of Maths, Computer Science and Statistics

Department: Statistics

Knowledge distillation

—

Compressing arbitrary learners into a neural net

submitted by
Jakob Bodensteiner

Supervisors:	Prof. Dr. Bernd Bischl
	M. Sc. Florian Pfisterer
Date:	03.04.2020

Declaration of authorship

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references. I am aware that the thesis in digital form can be examined for the use of unauthorized aid and in order to determine whether the thesis as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future theses submitted. Further rights of reproduction and usage, however, are not granted here. This paper was not previously presented to another examination board and has not been published.

Munich, 03.04.2020

. . . . . .

Jakob Bodensteiner

Contents

Chapter 1

Introduction

Knowledge distillation for binary classification was first introduced by Caruana et al. (2006). The goal of this technique is to 'compress' large ensemble models into small and fast neural nets with negligible loss in predictive performance. This means that for specific data, the knowledge of an ensemble learner gets transferred into a neural net. The intention behind is, to get the same predictive behavior, while receiving one kind of learner, for any problem. There was little scientific follow up. The next significant research paper was published 9 years later by Hinton et al. (2015). It introduced a method to make use of soft labels, to improve knowledge distillation even for multiclass classification problems and mobilized more research on this topic. Since then, more papers on model compression have been publicized. These papers concentrate on reducing the size of large neural nets into small ones. Examples, therefore, are Sau and Balasubramanian (2016) or Mirzadeh et al. (2019).

When just focusing on the compression of large neural nets into smaller ones, one misses another point of view on knowledge distillation. There are ongoing discussions on how and if neural nets are applicable to tabular data.

Knowledge distillation can be part of the answer to this question. The goal of knowledge distillation is, to reach the same predictive behavior as the original learner. In this thesis, it is shown, that a xgboost model on a tabular data set can be distilled successfully, performing almost as good as xgboost itself. Additionally, this method could be very helpful when thinking of deployment. If it is possible to transform any learner into a neural net, training and data transformation can be done in any language with any learner, while the deployed version for prediction is always a neural net.

Throughout this thesis, knowledge distillation for binary classification is investigated. The focus especially lies on transforming learners like SVM, random forest and xgboost on tabular data into neural nets.

1.1 Knowledge distillation

Before discussing a few aspects surrounding this topic, a basic understanding of knowledge distillation is conveyed.

The basic components required are a data set with a target variable and two machine learning algorithms. One of these algorithms is the so called 'student'. The other one is the 'teacher'. The teacher is an arbitrary machine learning algorithm, already fitted on the data. The student is a neural net.

The goal is, to distill this teacher's predictive behavior into the student net.

Therefore, many pseudo observations are generated from the original data set. These pseudo observations are then labeled by the teacher. This generated data set, consisting of labeled pseudo observations, bears the knowledge of the teacher about the original data set applied to pseudo observations. By fitting the student net on this pseudo data set, it learns the behavior of the teacher algorithm. Figure 1.1 shows this process.

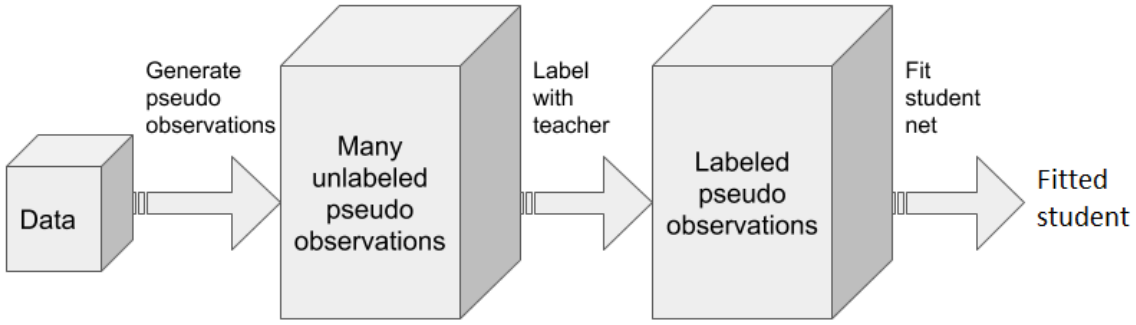

Figure 1.1: Knowledge distillation - The process of transferring the knowledge from a teacher algorithm into a student net

1.2 Notation

To keep the definitions clear, a notation holding for the whole thesis is provided.

1. $D = (X, y)$ is an arbitrary tabular data set. W.l.o.g. it consists of N observations (x_i, y_i) with $i \in \{1, ..., N\}$. Each observation consists of J features and one target variable. While $y_i \in \{0, 1\}$ represents this binary target variable, x_i is the vector of feature values for all $i \in \{1, ..., N\}$. Further $x_{i,j}$ is the j-th feature value of the i-th observation for all $j \in \{1, ..., J\}$. So, X is the feature matrix, and y the binary target vector. The features in X can be numeric or categorical or a mix of both.

2. $\hat{f}_{teacher}$ is an arbitrary machine learning algorithm, already fitted on D.

3. $D^* = (X^*, y^*)$ is the data set representing the pseudo observations. X^* is the unlabeled pseudo feature matrix. y^* is the label of the pseudo observations, generated by predictions of $\hat{f}_{teacher}$ on X^*: $y^* = \hat{f}_{teacher}(X^*)$.

4. $f_{student}$ is the (not yet fitted) student neural net. $\hat{f}_{student}$ is the result of fitting $f_{student}$ on the pseudo observations D^*.

1.3 Research questions

Knowledge distillation is a broad field. The research for this paper is focused on transferring knowledge of a teacher on tabular data with binary classification into a neural net. Two questions are stated, to give a guideline for this thesis. The experimental setup in section 4 is designed to answer these questions.

1.3.1 Can knowledge distillation outperform a neural net fitted directly on the data?

This question is crucial for the significance of this paper. If fitting a regular neural net on the data performs as good as fitting one with a teacher in the background, there would be no point in knowledge distillation.

1.3.2 Is it possible to copy an arbitrary learner and how good can the approximation become?

This is the question, also (Caruana et al., 2006) aimed to answer. In general, if a process tries to copy another process, it is always rated on its performance compared to the original version. In this case, the student neural net is trying to copy the behavior of the teacher algorithm.

The first approach is, to compare the performance by both $\hat{f}_{teacher}$ and $\hat{f}_{student}$ on test data. The closer the values of the performance measures are, the better the student imitates the teacher. For some use cases, this can be seen as a good enough measure for the 'goodness' of copy. But hereby only the mean over the performance measure is considered. Although this is the crucial measure to decide for or against a trustful machine learning algorithm, one important issue is not considered. Two algorithms can have the same average performance on the test data while predicting differently on single observations.

1.4 Research contributions

To answer the first research questions, in section 4.1.2.1 $f_{student}$ is compared with its teacher and a directly fitted net in a benchmark experiment on 19 tabular data sets with binary classification. These data sets show more diversity compared to the eight tasks investigated in Caruana et al. (2006). The observations per task range from less than 1,000 to more than 45,000. Only 'real' tabular data sets are used, in contrast to some letter classification in Caruana et al. (2006). Unlike in any other paper in section 4.1.2.2, observation-wise predictions between students and teachers are compared, to answer the second research question. Furthermore some modifications to the data generating method MUNGE by Caruana et al. (2006) are implemented to improve the technique. For the student net, different shapes and activation functions are tried as well. As $\hat{f}_{teacher}$, not just the ensemble methods random forest and xgboost but also SVM and in section 5, machine learning pipelines are used and evaluated.

Chapter 2

Related Work

2.1 Knowledge distillation

This thesis is mainly based on two papers, Caruana et al. (2006) and Hinton et al. (2015).

Caruana et al. (2006) introduced the idea of compressing ensemble learners into neural nets on binary data sets. An effective method to generate unlabeled pseudo observations called 'MUNGE' is introduced (section 3.1). It is shown that neural nets using this method as a generator of pseudo observations would outperform those nets, using other methods like sampling randomly from the hypercube surrounding the original data points. The general capability of neural nets, to compress ensemble learners is demonstrated by comparing the performance of teacher ensembles and student nets on eight data sets. Among these data sets, there is also image classification, which today would be tackled with deep learning anyway.

Hinton et al. (2015) expanded knowledge distillation to multiclass classification problems. Additionally a term called 'dark knowledge' was used to improve knowledge distillation. Hereby the pseudo observations are not labeled by the hard class prediction of the teacher, like in Caruana et al. (2006), but by the probabilities for the respective classes. This method is also used in this thesis and therefore introduced in section 3.2.2. Targeting multiclass classification problems, this work is more focused on knowledge distillation for typical deep learning problems, like speech and letter recognition. There is also no comparison over tasks, it rather shows different application possibilities.

2.2 Model compression

The term 'model compression' sometimes is used as a synonym for knowledge distillation (Caruana et al., 2006). But in most cases, it means something else. Model compression in the context of neural nets is mainly about making large neural nets smaller and faster. In computer vision, neural nets are supposed to be small and fast, to improve power efficiency and performance on mobile devices for example. There are different approaches to compress feed-forward networks, including knowledge distillation, quantization and pruning (Cheng et al., 2017).

Knowledge distillation is basically done the same way as shown in figure 1.1. Just the teacher, in this case, is a neural net as well and the student needs to be a smaller net than the teacher net is. Mirzadeh et al. (2019), for example, states that distillation from a large into a small net can be improved, by transferring the knowledge not directly into the target size. Instead one after another, the knowledge is distilled into a just slightly smaller network until the target size is reached or the performance decrease is too high.

Pruning describes the process, where successively connections between units or even whole units are dropped regarding their importance for classification. After deleting these connections, the net is fitted and evaluated, to monitor the predictive performance after modification. This process is continued, till a target number of connections remains or till the performance drops under a defined value (Yeom et al., 2019).

Quantization is a method especially to reduce the memory for neural nets. Hereby weights are clustered together, to require less memory (Cheng et al., 2017). For example Chen et al. (2015) creates hash buckets, to store and share weights.

These methods also can be applied together. Ashok et al. (2018) uses pruning and distillation, while Polino et al. (2018) presented a possibility to combine quantization and knowledge distillation.

As stated above, these methods aiming to make large nets smaller. These large nets are typically applied to text or visual recognition problems and no tabular data. So, this is not exactly the goal of this thesis, since there is another topic, this thesis attempts to contribute to.

2.3 Deep learning on tabular data

There are various discussions on the web and in scientific literature, whether deep learning is useful for tabular/structured data or not.

The rather conservative doctrine on this topic is 'If your data set doesn't consist of images, speech or language and you have limited data, then neural nets for most part are useless (or not that great).' (Student, 2018). Another qoute came up in a discussion on kaggle.com, stating 'XGBoost is usually extremely good for tabular problems, and deep learning the best for unstructured data problems.' (kaggle-user anokas, 2017). kaggle.com is one of the most popular machine learning platforms world wide. Companies or research institutes can host data science challenges for price money. Thus, opinions published by advanced kaggle users are influencing the community of data scientists. A quote here states that 'XGBoost is the leading model for working with standard tabular data' (kaggle-user DanB, 2018). So, the picture becomes clear: For many data scientists, deep learning is not the first choice when it comes to tabular data.

Taking a look at another platform called openML.org only confirms this general opinion. openML is a platform, similar to kaggle, but with another intention. It is made to conduct experiments on many different tasks. 'A task consists of a data set, together with a machine learning task to perform, such as classification or clustering and an evaluation method' (Vanschoren et al., 2013). On each task, the performance of learners and their hyper parameter

sets is visible. Considering the tasks on openML and their best performing algorithms, it is hard to find a neural net being the best performing algorithm for tabular data or even within a close range to the top performers. For tabular data, support vector machine (SVM), random forest and xgboost are the dominant algorithms. This might be the case, because neural nets are hardly tried on openML tasks. Either way it indicates, how infamous neural nets are for this purpose.

On the other hand, there are approaches to find deep learning structures specialized for tabular data. A paper on Self-Normalizing Neural Networks has been published by Klambauer et al. (2017). There, regular neural nets make use of an activation function called 'selu' (Scaled Exponential Linear Unit). On a benchmark comparison it is shown that these selu nets can outperform e.g. random forest or SVM on tabular data. Another approach to make neural nets more suitable for tabular data with categorical features are embeddings. Guo and Berkhahn (2016) show the general usage of embeddings for neural nets. It is a method well known from word2vec (Mikolov et al., 2013), where categorical features - like words - are transformed into vectors, to make the intrinsic information available for the neural net. An alternative attempt is to make use of the advantages of the dominant tree ensemble methods, by 'transforming' part of the tree structures into neural nets. Examples therefore are Arik and Pfister (2019), Yang et al. (2018), Popov et al. (2019) or Tanno et al. (2018). All these approaches use different tree-like differentiable structures as core of the neural net. These nets are directly fitted on the data, without a teacher. The goal behind this is to overcome one of the biggest issues neural nets are having with tabular data. I.e. that 'conventional DNNs based on stacked convolutional layers or multi-layer perceptrons (MLPs) are vastly overparametrized – the lack of appropriate inductive bias often causes them to fail to find optimal solutions for tabular decision manifolds' (Arik and Pfister, 2019). This inductive bias is for example the choice of machine learner by the data scientist. If structural changes in the neural net force it to behave more like a tree based ensemble, this can be seen as this inductive bias as well.

One problem with these approaches is, that they are hardly validated on benchmarks, and at best sporadically compared to each other. They are evaluated on other data sets and compared to different other algorithms. Nonetheless they share the same intention, to find a deep learning algorithm, performing well on tabular data.

Besides from copying arbitrary learners, knowledge distillation can be seen as this inductive bias, too. The teacher, a pre trained model, bearing knowledge of the underlying data, 'biases' the student net to learn the 'right' connections.

Chapter 3

Method

The process of knowledge distillation is a regular machine learning algorithm. A learner with tunable hyperparameters receives a data set with a target variable as input. The method described in this thesis heavily relies on the method introduced by Caruana et al. (2006).

The following pseudo code defines the algorithm called 'knowledge distillation' for this thesis.

Algorithm 1: Knowledge distillation

Input : $D = (X, y)$
Hyperparams : $\lambda_{student} = (\hat{f}_{teacher}, \lambda_{datagen}, \lambda_{net})$
Result : Fitted student net $\hat{f}_{student}$

1 generate unlabeled pseudo observations X^* from X via $\lambda_{datagen}$
2 label X^* with teacher: $y^* = \hat{f}_{teacher}(X^*)$ and combine it to $D^* = (X^*, y^*)$
3 fit $f_{student}$ (defined by λ_{net}) on D^*
4 return fitted student neural net $\hat{f}_{student}$

The hyperparameters in algorithm 1 are split into three parts. $\lambda_{datagen}$ are the parameters, defining the generation of pseudo observations. $\hat{f}_{teacher}$ is the teacher, which gets its knowledge distilled. λ_{net} are the parameters, defining the student neural net's design.

$\hat{f}_{teacher}$ is the most important hyperparameter since it indicates the label, the student net considers as truth. It is an arbitrary learner fitted on D, needing no further explanation. The other parameters, defining how exactly the three main steps (1 - 3) from algorithm 1 are processed, are explained in the following subchapters.

3.1 MUNGE

A major point of knowledge distillation is the artificial generation of pseudo observations. Since the goal is to copy any learner on tabular data sets with unknown distributions, it is hardly possible to use parametric methods to sample data. Instead, a special nonparametric method, called 'MUNGE' introduced by Caruana et al. (2006), is used. ''To munge data' literally means 'To modify data in a way that cannot be described succinctly" (Caruana et al.,

2006). There are other methods available to generate pseudo observations. The first best guess would be to draw random combinations from the hypercube represented by all observations. But this method generates a lot of completely unrealistic points and is therefore not as good as MUNGE (Caruana et al., 2006). Another possible method being considered is Synthetic Minority Over-sampling Technique, also called SMOTE (Bowyer et al., 2011). As the name states, it was developed to generate more observations from the minority class to balance the number of observations per class. It linearly combines feature values from nearest neighbors within the same target class to generate new pseudo observations.

Throughout this paper, MUNGE with slight changes is used to generate pseudo observations. First, the original method from Caruana et al. (2006) is shown. Afterwords modifications are introduced.

The MUNGE algorithm has a few parameters to set, defining how exactly pseudo observations are generated. They are among the hyperparameters of the knowledge distillation algorithm. These parameters are:

1. $K \in \mathbb{N}$ - The size multiplier
2. $swap\_prob \in [0, 1]$ - The swapping probability
3. $var\_param \in \mathbb{R}_0^+$ - The variance parameter

Algorithm 2 shows the pseudo code for MUNGE. In step 2 the index of the nearest neighbor of each observation is determined. For continuous features, the euclidean distance is used and for categorical features the hamming distance. Hereby the nearest neighbor of the i-th observation, gets the index $inn \in \{1, ..., N\}$. In consequence it holds: x_{inn} is the nearest neighbor of x_i.

In steps 9 to 15 the swapping of feature values is conducted. If the j-th feature value of observation i ($x'_{i,\ j}$) is categorical, it is swapped with the j-th feature value of the nearest neighbor ($x'_{inn,\ j}$). And if it is numeric, it is also swapped, since the new value of for $x'_{i,\ j}$ is drawn from a normal distribution centered around the feature value of its nearest neighbor. The other way around, $x'_{inn,\ j}$ is sampled around $x'_{i,\ j}$. So in expectation, these two feature values are getting swapped as well. Since this swapping happens with probability $swap\_prob$, this parameter is called swapping probability.

Swapping always influences both observations. inn is the index of the nearest neighbor of the i-th observation in the original feature set X. During each of the K loops, X' is looped through. So, the neighbor of x'_i could have been modified through a swap with another feature having this observation as its nearest neighbor. This means the nearest neighbor could have been modified through a swapping process in advance. Thus, it does not necessarily have exactly the same feature values each time X' is looped through since one observation can be the closest to multiple others.

Through swapping and not necessarily using the exact same nearest neighbor each time, many new and slightly different data points are generated. But they are likely in a reasonable area around the observations in X. The variance of these pseudo observations around the original points can be regulated by $swap\_prob$ and var_param. High $swap\_prob$ and low var_param yield to a broad variance of new data points and the other way around. Through

resetting X' in each of the K loops to X, X' is always beginning the loop as the original data set X. This way, new points are prevented from drifting too far away by swapping over and over again.

Algorithm 2: MUNGE

Input : $D = (X, y)$
Parameters : K, *swap_prob*, *var_param*
Result : Unlabeled pseudo observations X^*

for $i \in \{1, ..., N\}$ **do**
  determine index of nearest neighbor of x_i as $inn \in \{1, ..., N\}$
end
set $X^* := \emptyset$
loop K **times**
  set $X' := X$ *(copy unlabeled original data)*
  for $i \in \{1, ..., N\}$ **do**
    for $j \in \{1, ..., J\}$ **do**
      with probability *swap_prob* **do**
        if $x'_{i,\,j}$ *is categorical* **then**
          set $x'_{i,\,j} = x'_{inn,\,j}$ and $x'_{inn,\,j} = x'_{i,\,j}$
        else
          set $dist := |x'_{i,\,j} - x'_{inn,\,j}|$
          sample $x'_{i,\,j}$ from $\mathcal{N}(x'_{inn,\,j}, \frac{dist}{var\_param})$
          sample $x'_{inn,\,j}$ from $\mathcal{N}(x'_{i,\,j}, \frac{dist}{var\_param})$
        end
      end
    end
  end
  set $X^* = X^* \cup X'$
end
return X^*

In this thesis, only MUNGE and its modifications are used to generate pseudo observations. It shows way better performance for knowledge distillation than sampling randomly from the hypercube surrounding the data (Caruana et al., 2006) and also than SMOTE. The reason why lies in the constellation of the problem.

To distill the knowledge of $\hat{f}_{teacher}$ into $\hat{f}_{student}$, the intention behind generating many pseudo observations is to cover the relevant area of the feature space. Together with the teacher's labels, $f_{student}$ is fitted on this pseudo data. If the coverage of the relevant feature space is high and the student can approximate the teacher's predictions well, the behavior on new data points is supposed to be similar for $\hat{f}_{teacher}$ and $\hat{f}_{student}$. The data, $\hat{f}_{teacher}$ is fitted on is possibly high dimensional. Good coverage of high-dimensional spaces is hard. Due to the 'curse of dimensionality' (Chen, 2009), exponentially many observations would be required. This is the reason why sampling randomly does not lead to a good knowledge distillation. But it is assumed that the data lies on a low dimensional manifold since there are machine learning algorithms (used as teachers), that can predict well on this data. And MUNGE is a

non-parametric sampling technique aiming to approximate this manifold.

3.2 MUNGE modifications

In the implementation used for this paper, there are some points slightly different from the original method. The first difference is in step 2 of algorithm 2. In the original MUNGE, the numeric values are scaled between 0 and 1 to determine the nearest neighbor, too. In the implementation used for this work, they are scaled to the standard normal distribution $\mathcal{N}(0\ ,1)$. Since the reason for scaling is, to make all features comparable for the nearest neighbor calculation, it does not matter in which exact range it is done. Secondly, the size of the final munged data set X^* is not a multiple of N. Due to better comparability among data sets, it is a specific number, called *munge_size*. The algorithm iterates over the copies of X until the overall count of pseudo observations reaches *munge_size*.

Apart from these two points, MUNGE is implemented as it was defined by Caruana et al. (2006) and then extended by the modifications, introduced in the following sections. The intention behind these modifications is to possibly improve the MUNGE method, to get better-performing students. The goal is either to get a closer approximation of $\hat{f}_{teacher}$ or to even outperform the teacher, regarding its generalization error.

3.2.1 MUNGE with a portion of original data

The variance of the pseudo observations sampled by MUNGE is regulated by the two hyper-parameters *swap_prob* and *var_param*. When these parameters are set to provoke a large sampling variance (big *swap_prob* and small *var_param*), the danger of 'losing the focus' on the relevant area of the feature space rises. Then really unrealistic observations can be generated. These are observations where the teacher might predict strangly and the student tries to fit these points. Therefore it loses performance on the relevant observations. To avoid this, the hyperparameter $orig\_portion \in [0, 1]$ is defined. *orig_portion* is the portion of real data points (observations of D) sampled via bootstrapping from D in the returned 'munged' data set. An $orig\_portion = 0.1$ means, that in the artificially created data set 10% of the observations are real observations except for their label which is set by the teacher algorithm. Keeping the label of the original observation did not lead to performance improvement, so original observations are relabeled by $\hat{f}_{teacher}$.

3.2.2 Dark Knowledge

In Caruana et al. (2006), the pseudo observations are hard labeled. Meaning $y^* \in \{0, 1\}$ for all $i \in \{1, ..., target\_size\}$. Hinton et al. (2015) introduced a term called 'dark knowledge'. Most learners do not just predict classes by the class itself, but by assigning probabilities for each class. Especially for multi-class classification, these probabilities bear a lot of additional knowledge, compared to just one returned label (Hinton et al., 2015). But also for binary classification, it is a difference, if the student net tries to learn a positive label or just a positive probability of 75%. These probabilities are called soft labels. When $\hat{f}_{teacher}$ labels the pseudo

observations with these soft labels, knowledge distillation makes use of this dark knowledge, since it is additional information, which would stay unseen by hard labels only. This results in a boolean hyperparameter: *dark_knowledge*

3.2.3 MUNGE with linearly combined labels

In most data constellations, observations lying closely together are of the same class. A possibility to get even better than $\hat{f}_{teacher}$ could lie in linearly combining the soft teacher labels for pseudo observations with the original label of the nearest neighbors of this pseudo observation.

Algorithm 3 defines this linear combination with nearest neighbors.

Algorithm 3: Linear combination with nearest neighbors from original data

Input : $D^* = (X^*, y^*)$ and X
Parameters : α, k
Result : D^* with linearly combined labels

set *new target vector as* $y^{*new} = y^*$
for *each pseudo observation* $(x_i^*, y_i^*) \in (X^*, y^*)$ **do**
  find k nearest neighbors of x_i^* in original data X
  if *all knearest neighbors of* x_i^* *have the same label* y_{same} **then**
    set $y_i^{*\ new} = \alpha * y_i^* \ + \ (1 - \alpha) * y_{same}$
  end
end
return *relabeled data set* (X^*, y^{*new})

To put it in a nutshell, this linear combination aims to combine the knowledge of the teacher with a potential correction for the original k nearest neighbor's label. This method is especially applicable in this way, since only binary target variables are considered and $y^* \in (0, 1)$ is a soft label, respectively the probability for the positive class. $y_{same} \in \{0, 1\}$ on the other hand represents the original hard label, transferred into numeric value 1 (or 0) for the positive (or negative) class. The boolean hyperparameter *lin_comb_nn* decides if this combination is applied to the labeled pseudo observations.

This approach stands in conflict with the idea of knowledge distillation since $f_{student}$ does not learn from the teacher only but from combined labels. The goal is, to outperform the teacher. And it suits perfectly within the data generating method MUNGE.

3.2.4 MUNGE on the fly

The last modification of MUNGE is closely connected to the training method of neural nets. When training a neural net, one goes over and over through the training data until the maximum number of epochs is reached, or the improvement in performance stagnates. The data set is divided into partitions of batches for each epoch. In each epoch, the weights of the neural net are updated after each batch via backpropagation. These batches can be generated on the fly during training. Instead of creating one large MUNGE data set in the beginning,

the modification here is, to generate a 'munged' data set for each batch from scratch. These data sets of pseudo observations would be way smaller than the MUNGE data set, which is normally generated. The boolean hyperparameter, deciding if pseudo observations are generated for each batch from scratch or once, in the beginning, is defined as $munge\_otf$.

This method brings one big advantage. The memory requirements decrease heavily since no huge data set has to be generated. In addition, no max number of epochs would need to be defined. The training could just stop when there is no improvement anymore.

The set of these hyper parameters results in:
$\lambda_{datagen} = (munge\_size, swap\_prob, var\_param, orig\_portion, dark\_knowledge, lin\_comb\_nn, munge\_otf)$

3.3 Student net

While in Caruana et al. (2006) the method MUNGE is explained well, the properties of the student nets are hardly documented. There is no description, how many layers or which activation functions are used. It is also not stated which training loss the net minimizes or anything else about the hyperparameters of the net. It is just stated that the student nets consist of 4 to 256 hidden units. For these numbers of units the performance is compared on eight data sets. In Hinton et al. (2015), a little more information is given. 'relu' is used as activation function and by far more hidden units. In one experiment a net with 2 hidden layers and 1,200 units per layer is trained. But there is no experiment conducted, comparing the same students on different tasks.

In section 4, student nets are compared on 19 tabular data sets. Therefore the nets are constructed from a broad range of hyperparameters.

1. Shape of neural net - net_shape
   The shape of the neural net can be set to "conic" or "rectangular". If the shape is conic, each hidden layer consists of half the units, its forerunner does consist of. A rectangular shape, on the other hand, means, that from the first to the last hidden layer, they all share the same number of units.

2. First layer units - fl_units
   This parameter is a number (4, 8, 16, 32,...), defining the number of units in the first hidden layer. In consequence, it also defines the number of units in the other hidden layers, depending on the shape selected for the net.

3. Loss
   A typical loss for binary classification is the binary cross-entropy. The issue, in this case, is, that the neural net $f_{\{student\}}$ does not get hard labels as input, but the positive probabilities $y^* = \hat{f}_{teacher}$, predicted by the teacher. So it is actually not trained like a regular binary classification neural net. Instead, it is a regression on the probabilities. Therefore the mean squared error to the pseudo labels is chosen as loss to be minimized.

Apart from these parameters, tunable parameters for the student net are learning rate, weight decay, the number of hidden layers, the activation function, batch normalization and the dropout rate as well.

All together, they result in $\lambda_{net} = (net\_shape,\ fl\_units,\ act\_function,\ learning\_rate,\ dropout\_rate,\ weight\_decay,\ optimizer,\ batch\_norm)$

Chapter 4

Experiments

4.1 Benchmark comparison

In this section, a benchmark comparison between the teacher, its student, and a regular neural net is carried out on different data sets. Since, the teacher algorithms are already tuned and well-performing predictors on these tasks, they are used as the benchmark. This experiment should clarify, whether knowledge distillation can outperform a directly fitted neural net on different tabular data sets. It is also designed to show if knowledge distillation can be done on a broad range of binary classification problems and how close the students can approximate the predictive behavior of $\hat{f}_{teacher}$.

4.1.1 Setup

4.1.1.1 Tasks and teachers

The evaluation is done on 19 binary classification tasks from openML. They were selected according to the following criteria.

They all ...

1. are labeled with the OpenML-CC18 tag.
2. have xgboost, SVM or random forest learner - from mlr (Bischl et al., 2016) - among the best-performing algorithms.
3. consist of categorical, numeric or mixed features in tabular form.

In openML, classification tasks gathered under the OpenML-CC18 tag are tasks, that are 'carefully curated from the many thousands available on OpenML' (Bischl et al., 2017a). 'It includes data sets frequently used in benchmarks published over the last few years' (Bischl et al., 2017a). Since here it is the goal to set up a benchmark comparison, only data sets with this tag are selected. The implementation of knowledge distillation for this paper is written in R, mainly with the packages mlr (Bischl et al., 2016) and Keras (Chollet et al., 2017). Thus,

Table 4.1: Task and teacher selection - The task names are abbreviations (full names can be found in table 7.1). The number of features is counted without the target variable. For the 'SVM' teachers, the term in braces is their kernel type. The last column is the rank of the teacher for this task in the openML leader board from December 2019.

Task	Observations	Features	Feature types	Teacher	oML rank
ilpd	583	10	mixed	xgboost	4
qsar	1055	41	mixed	SVM(radial)	5
ozone	2534	72	numeric	xgboost	1
phoneme	5404	5	numeric	random forest	3
wdbc	569	30	numeric	SVM(linear)	3
banknote	1372	4	numeric	SVM(radial)	5
blood	748	4	numeric	random forest	1
spam	4601	57	numeric	xgboost	8
tic	958	9	categorical	SVM(radial)	1
diabetes	768	8	numeric	random forest	1
kc2	522	21	numeric	xgboost	2
pc1	1109	21	numeric	random forest	1
phishing	11055	30	categorical	random forest	9
wilt	4839	5	numeric	SVM(radial)	2
monks2	601	6	categorical	xgboost	2
elect	45312	8	mixed	xgboost	2
bankmark	45211	16	mixed	xgboost	1
credit	1000	20	mixed	SVM(radial)	20
chess	3196	36	categorical	SVM(polynomial)	1

only mlr algorithms are used from openML as $\hat{f}_{teacher}$. In this benchmark comparison, three well-known algorithms for tabular data are used as a teacher: xgboost, SVM and random forest. To get the best possible teacher algorithm for a task, only those tasks with one of these algorithms among the top-performing ones are selected.

Table 4.1 shows the selection of tasks with the corresponding teacher for this benchmark comparison. The range of observations goes from 569 up to 45,312. The number of features ranges from 4 to 72. Seven times SVM serves as $\hat{f}_{teacher}$ with different kernels, five times random forest and seven times xgboost. So, it represents a variety of data sets and teachers. The last column shows the rank of the teachers among all learners applied to the corresponding task on openML. 11 are ranked first or second. The others are all among the ten best learners. Only the SVM on 'credit', is ranked 20th. But it lies not even 0.02 behind the best performing algorithm, regarding the accuracy. Among these tasks, only 'credit' and 'chess' were used for pre-tests to develop the implementation and to gain basic knowledge about the hyperparameters.

4.1.1.2 Measures

On these tasks, teacher, student net and directly fitted neural net are compared, regarding performance measures. These measures are:

1. mmce - the mean misclassification error
2. logloss - also known as binary cross-entropy

4.1.1.3 Resampling and tuning

In openML, pre-defined cross-validation (CV) folds exist for each task. The same folds are used in this case for evaluation. Since the teacher's hyperparameters are set already, there is no tuning necessary. The student and the directly fitted net need tuning. Therefore nested resampling with model-based optimization is used. Model-based optimization is a tuning method, specially developed for algorithms with long training time. The exact process is explained in section 7.1. For this comparison, the implementation in mlrCPO (Bischl et al., 2017b) is used with the following parameters: Initially 40 points from the tuning space are sampled by Maximin Latin Hypercube Sampling (Carnell, 2019). As the surrogate model a random forest is used with jackknife-after-bootstrap variance estimation (Efron, 1992). The infill criterion is lower confidence bound. With this criterion one new hyperparameter set at a time is chosen and evaluated before the surrogate model is fitted again. The evaluation method is 3-fold cross-validation with logloss as the performance measure. All in all, 100 different hyperparameter sets are evaluated using 3-fold CV, to determine and finally evaluate the best model on each of the 10 outer CV folds.

4.1.1.4 Hyperparameters

The student net and the directly fitted net share part of their hyperparameters (λ_{net}). The tuning ranges for this experiment are shown in the left part of table 4.2. Most of them are well known, except for the activation functions. The choice here is between 'relu', 'selu' and 'hard_sigmoid'. relu is the classic activation function, also used by Hinton et al. (2015). selu was introduced by Klambauer et al. (2017). These 'self normalizing nets' (with selu activation function) are supposed to compete with tree-based ensemble learners, like xgboost on tabular data. On the one hand, it is interesting to see, how directly fitted nets perform with selu activation function. On the other hand, it might even improve student nets, so this parameter is available for both, the directly fitted net and the student. It is implemented as suggested with the initialization kernel 'Lecun Normal' and the dropout method 'Alpha Dropout' (Chollet et al., 2017). Additionally for selu no batch normalization is used.

$$hard\_sigmoid(x) = \begin{cases} 0 & x < -2.5 \\ x * 0.2 + 0.5 & -2.5 \leq x \leq 2.5 \\ 1 & 2.5 < x \end{cases} \tag{4.1}$$

Table 4.2: Tuning ranges for neural net and data generating hyper parameters

Neural net		Data generation	
Hyper parameter	Tuning range	Hyper parameter	Tuning range
net_shape	{conic, rectangular}	munge_size	100,000
act_function	{relu, selu, hard_sigmoid}	swap_prob	[0.01, 0.5]
fl_units	{16, 32, 64, 128, 256}	var_param	[0.5, 5]
hidden_layers	{2, 4, 8, 16}	orig_portion	[0.001, 0.5]
learning_rate	[0.000001, 0.1]	lin_comb_nn	{TRUE, FALSE}
dropout_rate	[0, 0.5]	dark_knowledge	TRUE
weight_decay	[0.0001, 0.01]	munge_otf	FALSE
optimizer	adam		
batch_norm	{TRUE, FALSE}		

The activation function hard_sigmoid (equation (4.1)) shares properties with a step function. Since the splits in tree-based learners can be represented by step functions, it might be helpful to distill knowledge from random forest or xgboost.

The hyperparameters exclusively used by $f_{student}$ are in the right columns of table 4.2. The first and the last two parameters are especially noticeable since there is no range defined. *munge_size* is set to 100,000 pseudo observations for all data sets and teachers. In the pre-tests, it seemed like a reasonable trade-off between performance and learning time. Additionally, it is used in the original paper (Caruana et al., 2006). In section 4.2.2, a deeper investigation on this hyperparameter is done, supporting this decision. The next fix parameter is *dark_knowledge*. It is set to true since only this method has a chance to match probabilities on the observation level. In consequence, the training loss for the student net is the mean squared error, while for the directly fitted net it is binary cross-entropy. The last fix parameter is $munge\_otf$. It did not improve the performance in pre-experiments but was slower, so there was no further investigation in this direction. If *lin_comb_nn* is set to true, the labels of pseudo observations are combined half and half with the label from the nearest neighbor's label in the original data set only if the three nearest neighbors from the original data set have the same label. This means, parameters α and k in algorithm 3 are set to 0.5 and 3. For the other parameters, neither was a hint on defaults in the literature, nor they could be narrowed more in pre-tests, so they are up to tuning in the benchmark experiment.

4.1.2 Results

In the following sections, the results of this benchmark comparison are shown. The teacher, its student and a neural net directly fitted on the data are compared. While on the next pages only mmce is analyzed, the results for logloss can be found in table 7.3. In section 4.2.5 recommendations for hyperparameter settings are provided.

4.1.2.1 Benchmark comparison

Figure 4.1 shows the mean misclassification error across all test sets of the 10-fold cross-validation. For each task, one plot shows first the regular net's mmce, then the student's and on the right the teacher's error. The means are connected by a line, to visualize the difference between these three algorithms. For all 19 tasks, the student net outperforms the directly fitted neural net. In most cases the teacher algorithm is the best performing one, except for the tasks 'wdbc', 'phishing' and 'monks2'. Here, the student is slightly better, than the teacher. Looking at these plots, one has to be careful, since the scaling is not uniform. For example, the tasks 'ilpd' and 'monks2' look the same at first glance. While in the first case, the student is just 3% better in accuracy than the directly fitted net, it is over 30% better for the 'monks2' task.

Table 4.3 shows corresponding numbers, including the mean misclassification errors presented by figure 4.1 in columns 'mmce NN', 'mmce student' and 'mmce teacher'. The other columns are differences of these errors. 'NN - student' is the average difference in mmce from the directly fitted net to student. In consequence, positive values mean a better performance by the student. 'Student - teacher' is the difference between student to teacher performance. Here, negative values show better performance by the student, compared to the teacher. The table is ordered by 'NN - student'. In just one case (task 'wdbc'), the directly fitted neural net gets closer than 1% within the accuracy of the student net. For the majority of the tasks (12), the difference in mmce is between 1% and 3%. Tasks 'ilpd', 'qsar', 'blood', 'tic' and 'phoneme' show a higher difference from 4% up to more than 10%. The largest difference appears for task 'monks2'. Here the student outperforms the directly fitted net by a huge margin of 34.28% in accuracy. Even the teacher is slightly worse than the student in this case.

These results indicate, that knowledge distillation driven neural nets are indeed able to outperform the same neural nets directly fitted on tabular data. In some cases, of this benchmark comparison, the difference between the student and the regular neural net is almost negligible, but for some tasks, the gain in performance by learning from a teacher is significant. Of course, the prerequisite is a well-tuned $\hat{f}_{teacher}$.

4.1.2.2 Student vs teacher

The original goal of knowledge distillation is not to outperform directly fitted neural nets. It is to get a copy of the teacher algorithm, regarding predictive behavior. So now, the second research question is answered and the 'quality' of knowledge distillation in this benchmark comparison is investigated.

Additionally to the mean misclassification error across all folds, figure 4.2 shows an estimation of the distributions of the mmce over the folds. Like in figure 4.1, the black points in the middle of the violin plot are the overall mmce. This time the scale is just different, since outliers of the cross-validation are shown as well. For tasks 'wilt', 'ilpd', 'qsar', 'ozone', 'wdbc' and 'blood', the violin plots show a pretty similar distribution for student and teacher algorithm. In the case of 'credit', 'phoneme' and 'spam', the plots look similar, just shifted a bit upwards. A completely different distribution is shown just by the task 'elect'. Regarding column 'Student - teacher' in table 4.3, the difference in mmce from teacher to student is almost 10%. In this case, knowledge distillation outperformed the directly fitted net, but it is far away from a 'copy' of the teacher algorithm. Except for the task 'credit', where the difference is 1.1%, on all other 17 tasks, the student is beneath 1% within the mmce of the teacher algorithm. In three cases the mmce is even slightly better than the teacher's mmce. And for 9 tasks, the difference is below a half percent (see table 7.2, a reorder of table 4.3 by 'Student - teacher'). This indicates, that knowledge distillation was applied successfully to most of these teachers.

As stated in section 1.3.2, a similar mean misclassification error is not necessarily the result of the same predictive behavior on single observations. To get the observation-wise comparison, the predictions on the test splits from the CV are used. Both, teacher and student predict the positive probability for each observation. For these predictions, the absolute and the squared error from teacher to student and the correlation are calculated.

Figure 4.3 shows these values for each task. The first two plots show the average squared (mse) and absolute error (mae). The plot on the right shows the Pearson correlation coefficient between teacher's and student's predictions. The tasks are ordered according to the mse. The mean squared error is indeed very low. Leaving out task 'elect', it is below 0.02 for all data sets, and below 0.01 for all others, except 'phoneme'. One has to bear in mind, that this is the measure, $\hat{f}_{student}$ has been trained on. While fitting, the student net aims to minimize the mse between its prediction for positive probability and the teacher's soft positive label for the pseudo observations.

The mean absolute error is also within a low distance to zero. 14 of the tasks show a mean absolute error below 0.05, four between 0.05 and 0.075, while only 'elect' is beyond 0.15.

The Pearson correlation coefficient is close to 1, while here the most extreme outlier is 'elect', too. In most cases the coefficient is greater than 0.95. Surprisingly, there are three tasks, 'ozone', 'pc1' and 'ilpd', all having fair results in mse and mae, but their correlation coefficient lies below 0.90. This seems a bit low, especially for ozone, which is ranked 8th, regarding the mse.

Overall, except for one problematic data set 'elect', knowledge distillation seems to be processed successfully, also on a single observation level. The results of this benchmark comparison indicate, that there are good reasons to use knowledge distillation for tabular data with binary classification. It not just outperformed directly fitted neural nets on all tasks, it also gets pretty close to the performance and behavior of the teacher. Whether it is close enough naturally depends strongly on the application.

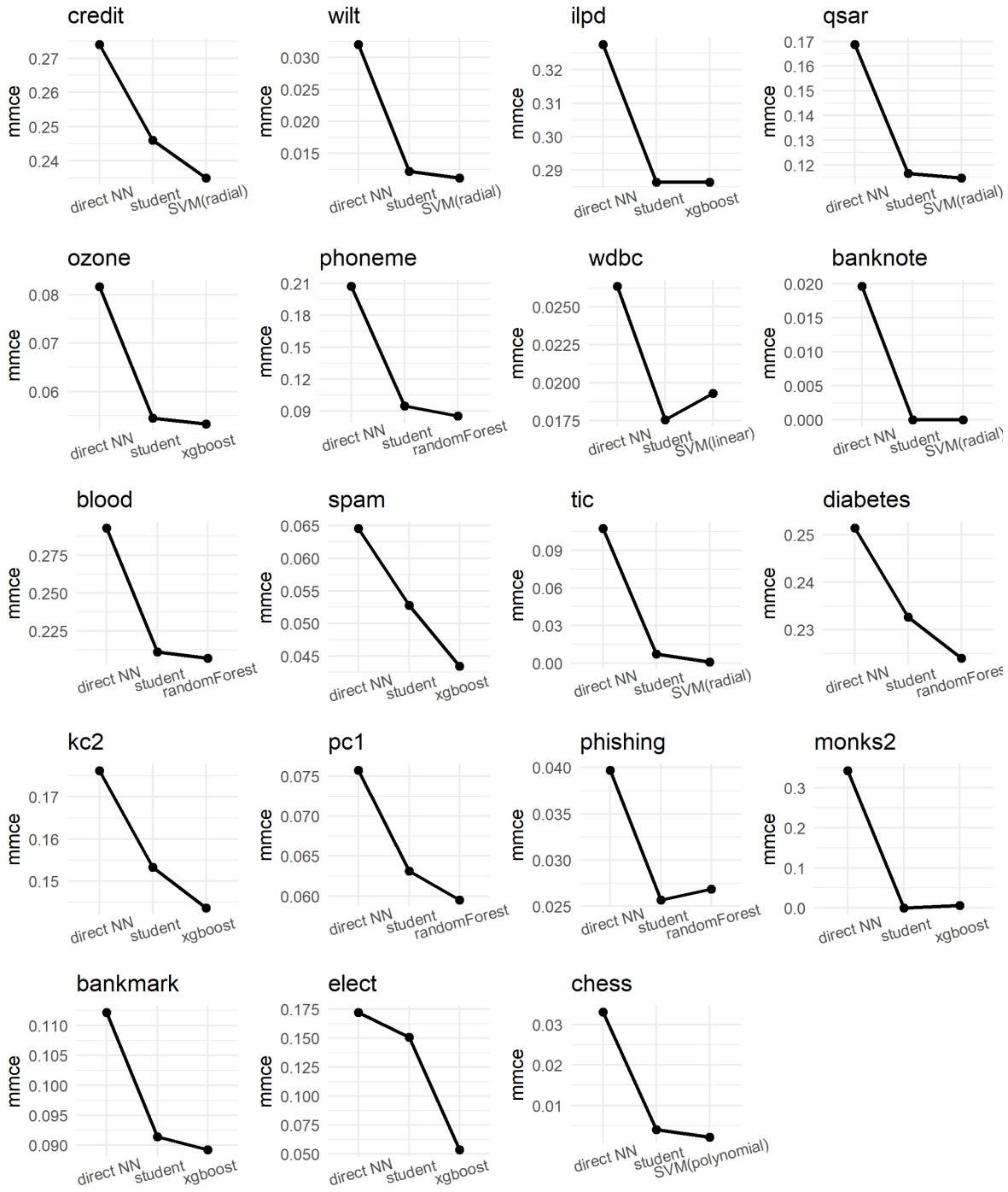

Figure 4.1: Overall mmce comparison - Comparison of directly fitted neural net's, student's and teacher's overall mean misclassification error on 19 tasks. The short name of the task is in the upper left corner of each plot.

Table 4.3: Overall mmce comparison with differences - The first three columns are the directly fitted neural net's, student's and teacher's overall mean misclassification error. The last two columns show the difference in mmce for direct NN to student and student to teacher.

Task	mmce NN	mmce student	mmce teacher	NN - student	Student - teacher
wdbc	0.0263	0.0175	0.0193	0.0088	-0.0018
spam	0.0645	0.0528	0.0435	0.0117	0.0093
pc1	0.0757	0.0631	0.0595	0.0126	0.0036
phishing	0.0397	0.0257	0.0269	0.0140	-0.0012
diabetes	0.2514	0.2327	0.2240	0.0187	0.0086
banknote	0.0196	0.0000	0.0000	0.0196	0.0000
wilt	0.0320	0.0122	0.0112	0.0198	0.0010
bankmark	0.1121	0.0915	0.0892	0.0207	0.0022
elect	0.1721	0.1508	0.0536	0.0213	0.0972
kc2	0.1762	0.1533	0.1436	0.0230	0.0097
ozone	0.0816	0.0545	0.0533	0.0272	0.0012
credit	0.2740	0.2460	0.2350	0.0280	0.0110
chess	0.0332	0.0041	0.0022	0.0291	0.0019
ilpd	0.3275	0.2864	0.2864	0.0411	0.0000
qsar	0.1688	0.1166	0.1147	0.0522	0.0019
blood	0.2928	0.2112	0.2072	0.0815	0.0040
tic	0.1074	0.0073	0.0010	0.1001	0.0062
phoneme	0.2073	0.0946	0.0851	0.1127	0.0094
monks2	0.3428	0.0000	0.0066	0.3428	-0.0066

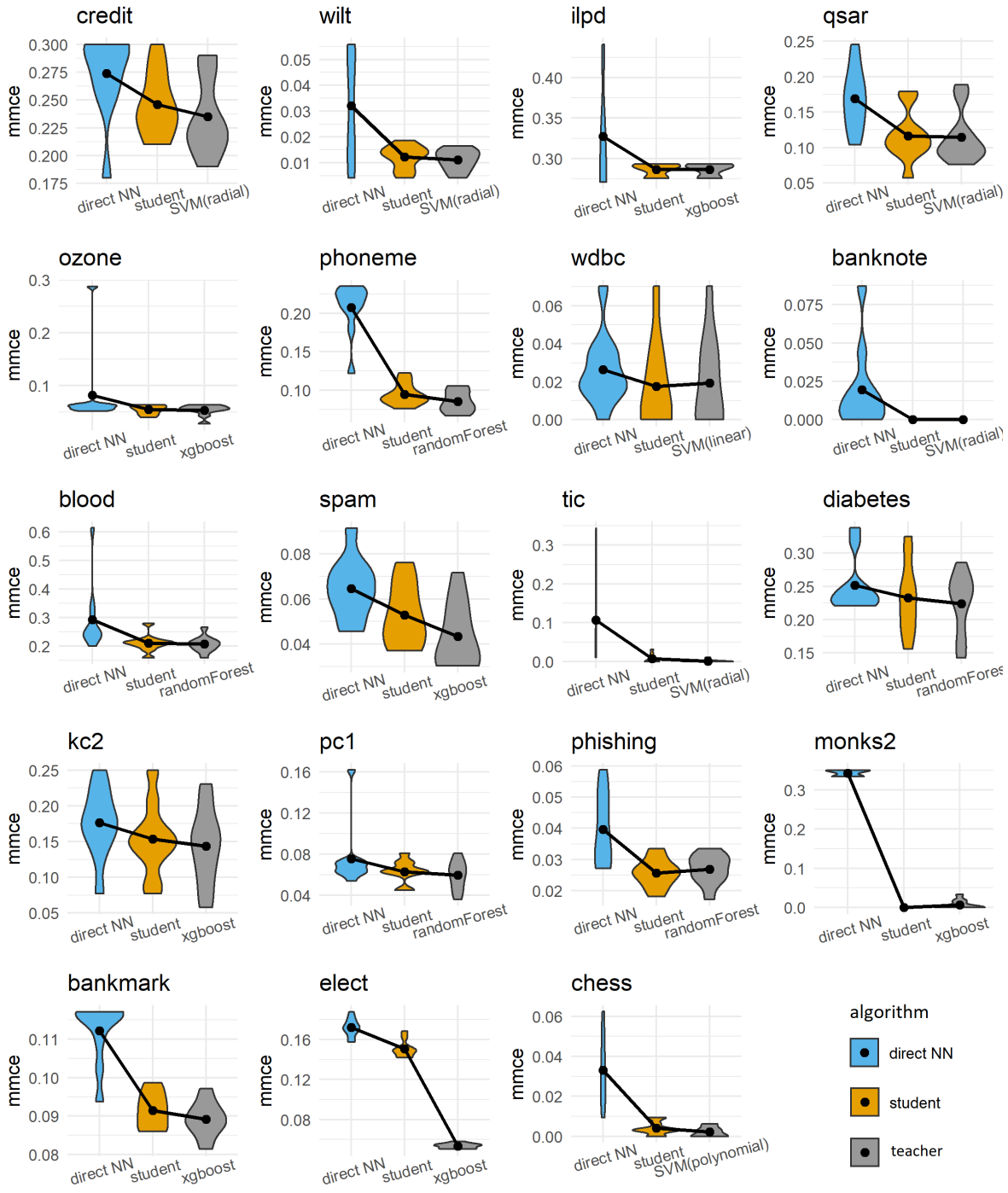

Figure 4.2: mmce distribution across CV folds - Directly fitted neural net (blue), student (orange) and teacher (grey) are evaluated with mean misclassification error on a 10-fold CV. The plots are so-called truncated violin plots. They are (truncated) estimations of the distribution of the fold-wise mmce.

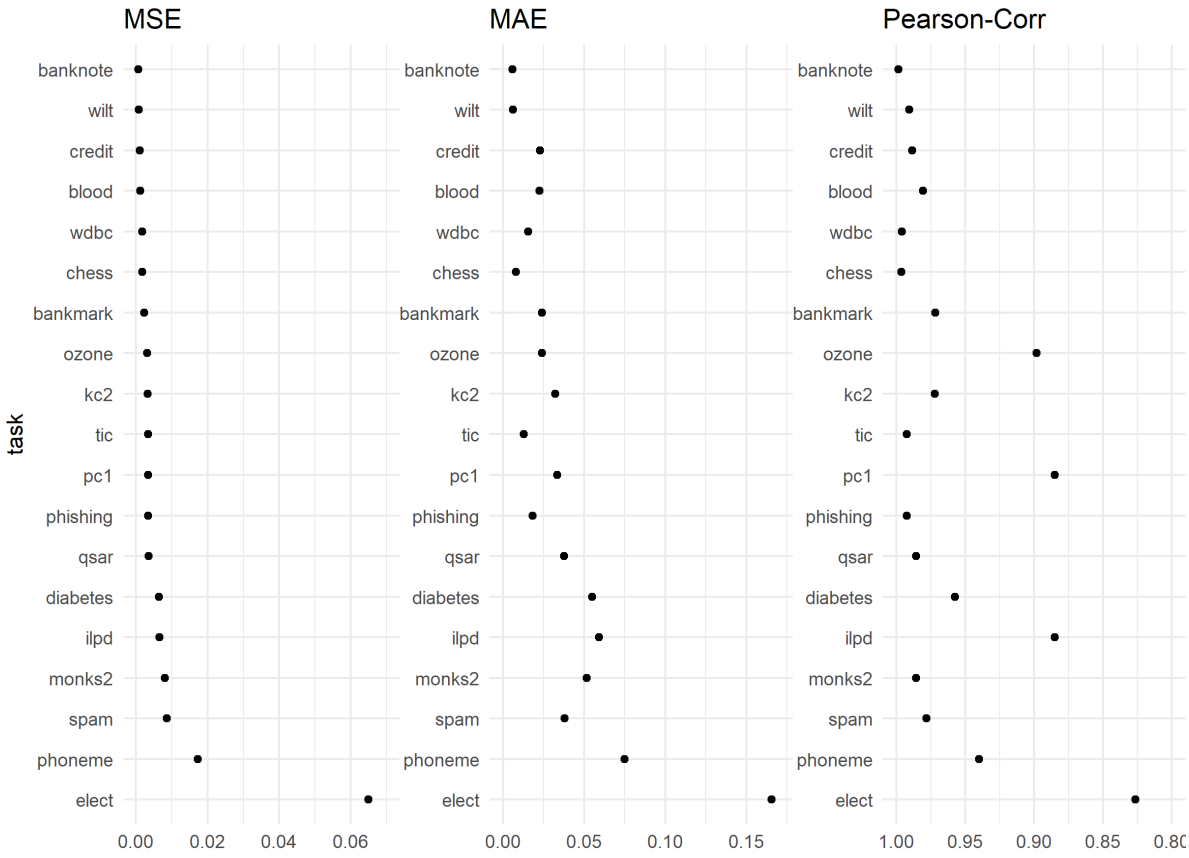

Figure 4.3: Errors and correlation between student and teacher - The teacher's and student's predictions (positive probabilities) for each task are compared. The two left plots show the mean squared error and the mean absolute error between teacher's and student's soft labels. The plot on the right shows the Pearson correlation coefficient for these positive probabilities of student and teacher. The tasks are ordered, according to the MSE.

4.2 Ablation Analysis

The following sections examine aspects of knowledge distillation that are not explicitly part of the research questions. Partly the results from the benchmark comparison are used, partly some additional calculations are made.

4.2.1 Teachers and activation functions

The first question to be answered is divided into two parts. First, are different teachers easier to copy than others among xgboost, random forest and SVM? Second, are there activation functions that suit better to copy different teacher algorithms?

To answer the first question, results from the benchmark comparison are considered. For each task, figure 4.4 shows the distribution of the differences between student and teacher predictions (observation-wise), calculated on the test splits of the CV. This difference is the error between student and teacher. To keep the plots comparable, they all show the area $[-0.1, 0.1]$. So in most cases, some outliers are cut off. For the sake of completeness, figure 7.1 in the appendix shows the whole span for all tasks. The plots are ordered and colored, regarding their teacher. There are again violin plots to estimate the distribution and additionally regular boxplots in the middle of them. The outliers are represented by black dots.

Starting with SVM as $\hat{f}_{teacher}$, 5 out of 7 tasks show an extremely low error with whiskers below 0.0125. In just two cases, the distribution has a higher variance. For the random forest, only the task 'phishing' has whiskers so close to 0. The other 4 data sets, especially 'diabetes', show a relatively high variance, where the outliers start between $|0.05|$ and $|0.1|$. Same holds for 'elect', 'kc2', 'ilpd' and 'monks2' with xgboost as teacher. Just 3 out of 7 tasks show a low variance for xgboost and none of them is as low as the ones for tasks with an SVM as the teacher.

It seems like SVM is the easiest teacher to get its knowledge distilled into a neural net. The two ensemble methods random forest and xgboost are a bit harder to copy. This assumption gets supported by figure 4.5. It shows basically the same like figure 4.3. Just in this case the errors and the correlation coefficient are grouped by the teacher classes. The values for mse, mae and Pearson correlation are similarly distributed for xgboost and random forest. SVM on the other hand shows constantly low values for the errors and a correlation coefficient, which is above 0.97 for all seven tasks.

These results point to the assumption, that in general an SVM is easier to copy with knowledge distillation, than ensembles like random forest or xgboost.

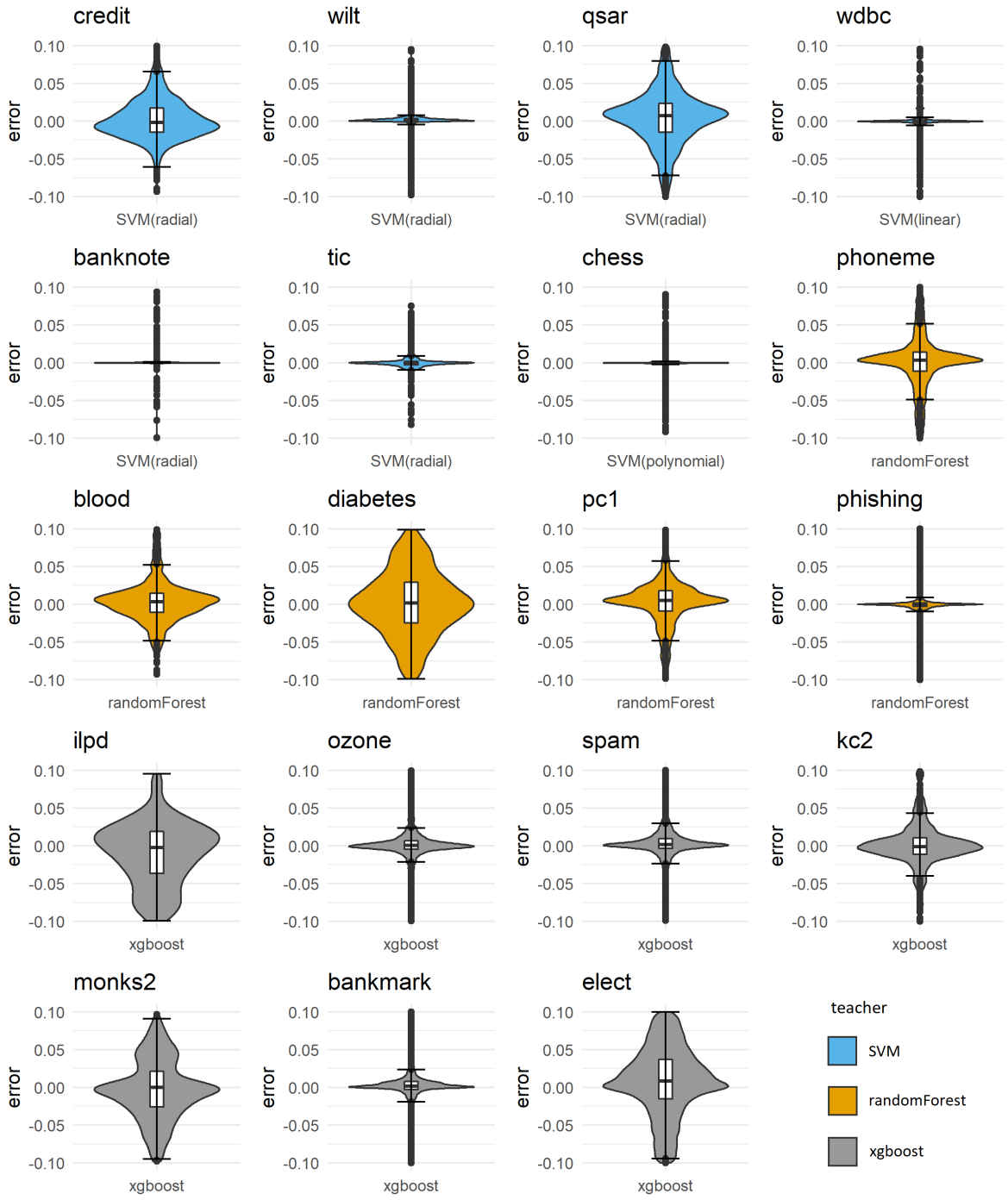

Figure 4.4: Distribution of error from student to teacher - The distribution of the observation-wise difference in positive prediction between teacher and student is ordered by the type of teacher. First SVMs (blue), second random forest (orange) and third xgboost (grey). The scale is [-0.1, 0.1], meaning some outliers are cut off.

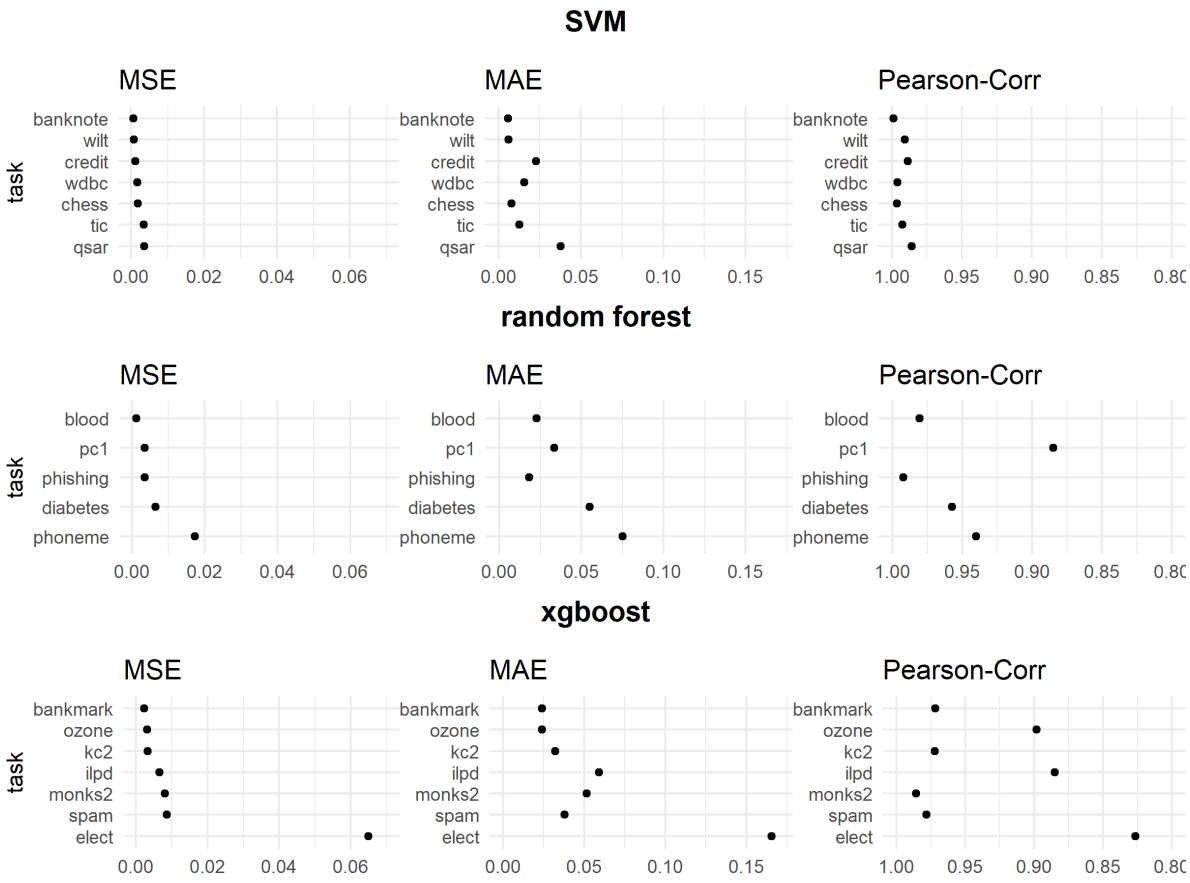

Figure 4.5: Errors and correlation by teacher type - The teacher's and student's predictions (positive probabilities) for each task are compared. They are ordered by the teacher types, SVM, random forest and xgboost. The plots on the left show the mean squared error and the mean absolute error between teacher's and student's soft labels. The plot on the right shows the Pearson correlation coefficient for these positive probabilities of student and teacher. The tasks are ordered, according to the MSE.

The second question in this context is which activation function is preferably chosen for which learner?

In the benchmark comparison, for each cross-validation fold, the hyperparameters were tuned. For the activation function, three parameters were available for the student and the directly fitted net: 'selu', 'relu' and 'hard_sigmoid'.

Figure 4.6 shows the tuning results for this parameter in each fold. For each task, the student's and the directly fitted net's choices are visualized. Additionally, for each teacher algorithm, the distribution of activation functions selected by the corresponding student can be seen, as well.

The first interesting point is the activation function hard_sigmoid. It does not seem to be of any use, neither for the student nor for the directly fitted net. Especially, it was not used

once, to distill the knowledge out of a random forest. So, in this context, it does not help, to copy a tree-based learner. Considering the plot in the left lower corner, there is no indication for an activation function in favor of one special teacher. In most cases relu is chosen. For random forest only relu was selected, for SVM it is definitely the favorite. For xgboost it is relu as well, while selu has a bit more appearances here compared to SVM. In general, relu is selected more often by the student, than selu. Only for the tasks 'ilpd' and 'monks2', relu has not been selected once. On seven of the other tasks, only relu was the result of tuning. Contrary to that, for the directly fitted net, selu seems to be the best choice. There is just one real outlier, the task 'monks2', where exclusively relu was used. For the other tasks, the directly fitted net used mainly selu as activation function.

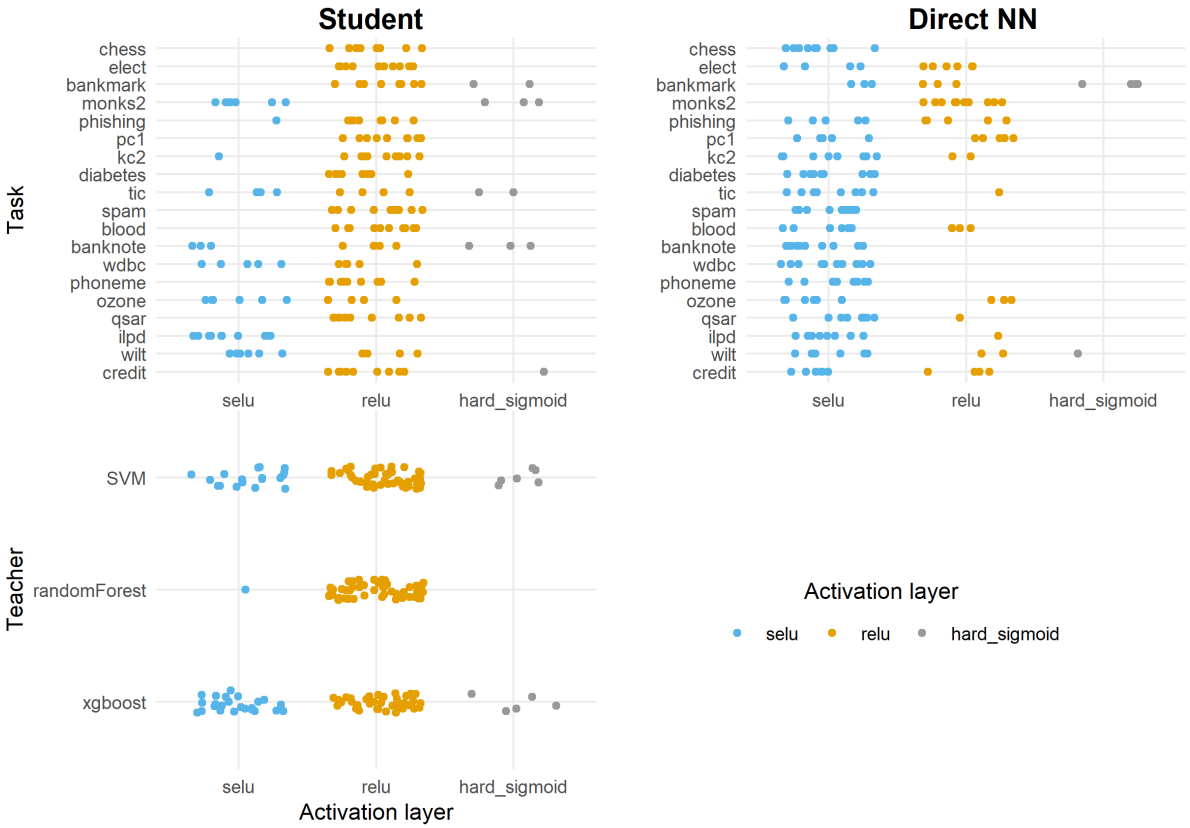

Figure 4.6: Activation functions selected by tuning - The results of MBO tuning in the benchmark experiment, regarding hyperparameter *activation_function*. For each task, the activation functions used for prediction on test data are plotted against the tasks for student and directly fitted net and once against the teacher type to learn from.

In summary, there is no clear preference for an activation function indicated by the type of teacher. In most of the cases, relu is the result of tuning, so it could be used as the default value for this parameter of the student net. But selu might be worth a try anyway.

Apart from that figure 4.6 shows, how strong the influence of knowledge distillation is on the

choice of hyperparameters, like *activation_function*. With the same parameters available, tuning resulted in a strong and contrary tendency, once towards relu and once towards selu.

4.2.2 MUNGE size

A parameter with a strong influence on knowledge distillation is *munge_size*. The goal of MUNGE is to cover the part of the hypercube in the feature space where potentially realistic observations are located. Thereafter, the $\hat{f}_{teacher}$ labels them and the student is fitted on these observations. So when new (real) observations need to be labeled by $\hat{f}_{student}$, it should be able to label them almost as the teacher would do, since the student has been trained on many pseudo observations, which are very similar to any new observation.

Derived from this intention behind MUNGE, the theory is formulated, that the larger the parameter *munge_size* is, the closer the student approximates the teacher's behavior.

When examining this theory, some problems arise. In the benchmark comparison, *munge_size* was set to 100,000. Choosing this number, had different reasons. First, it performed well in the pre-testing phase. Secondly, it was suggested by Caruana et al. (2006) (for no explained reason). Third, it is a trade-off between size and duration of fitting $f_{student}$ and, as a consequence, tuning its hyperparameters. To draw a picture of the influence of *munge_size*, the tuning results for the other hyperparameters from the benchmark comparison were combined with different munge sizes and evaluated on the same CV folds. The different evaluated numbers of generated pseudo observations are 10,000, 50,000, 100,000, 500,000 and 1 million.

Table 4.4 shows the results of this comparison, regarding *munge_size*. The measure is mmce. The first column presents the overall misclassification for $f_{student}$ with *munge_size* = 10,000, evaluated on each task. The next four columns show the improvement in mmce to the next higher evaluated *munge_size*. For example, column '100K - 500K', represents the difference in mmce from a student with 100,000 pseudo observations to one with 500,000. Positive values mean improvement and negative ones degradation in mmce. This way, the development from 10,000 to 1 million pseudo observations can be seen in one line. The last column is the evaluation of $\hat{f}_{teacher}$, and the column before is the mmce for $f_{student}$ trained on 1 million pseudo observations.

A systematic improvement in accuracy from 10,000 to 1 million observations is shown by tasks 'bankmark', 'elect', 'ilpd', 'kc2' and 'phoneme'. For the other tasks, there is at least one step with a negative value, meaning degradation of performance. The largest differences are obtained from 10,000 to 50,000. Here an improvement between 5% and 10% in accuracy is normal. Additionally, there is just one degradation for task 'wilt'. In the next step from 50 to 100 thousand observations, a large step is only on this task 'wilt', with a 5% improvement. Apart from that, there are six low degradations and the rest are slight improvements. Almost the same holds for the next step from 100 to 500 thousand. There are four cases with worsening, one of them relatively high with 3.4% for task 'monks2', and the rest are positive values, below 1% except for 'diabetes' and 'elect', with improvements between 1% and 2%. The last step is split in half. 9 tasks show a degradation, and 10 an improvement. The values in both directions are all 1% or lower.

Table 4.4: Influence of MUNGE size - Comparison of performance for different sizes. The mmce for different numbers of pseudo observations generated by MUNGE is compared. The first column shows the value for 10,000. The following column is the difference to the next higher number of pseudo observations, while positive values mean an improvement. In the last two columns the mean misclassification errors for the student with MUNGE size set to 1 million and the teacher's results are shown.

Task	10K	10K - 50K	50K - 100K	100K - 500K	500K - 1M	1M	Teacher
bankmark	0.1037	0.0119	0.0004	0.0006	0.0002	0.0907	0.0892
banknote	0.1095	0.1095	0.0000	-0.0022	0.0007	0.0015	0.0000
blood	0.2632	0.0560	-0.0040	0.0054	0.0013	0.2045	0.2072
chess	0.0147	0.0113	-0.0006	0.0013	-0.0006	0.0034	0.0022
credit	0.2760	0.0300	0.0000	0.0100	-0.0100	0.2460	0.2350
diabetes	0.2514	0.0288	-0.0101	0.0129	-0.0072	0.2269	0.2197
elect	0.2371	0.0735	0.0128	0.0168	0.0035	0.1305	0.0536
ilpd	0.3549	0.0668	0.0017	0.0000	0.0017	0.2847	0.2864
kc2	0.2206	0.0635	0.0038	0.0019	0.0019	0.1495	0.1436
monks2	0.1028	0.1028	0.0000	-0.0344	-0.0017	0.0361	0.0066
ozone	0.0619	0.0079	-0.0004	0.0028	-0.0004	0.0521	0.0533
pc1	0.1424	0.0676	0.0117	0.0045	-0.0027	0.0613	0.0595
phishing	0.0829	0.0569	0.0004	-0.0011	0.0014	0.0254	0.0269
phoneme	0.1696	0.0668	0.0083	0.0072	0.0011	0.0862	0.0851
qsar	0.1770	0.0632	-0.0028	0.0066	-0.0048	0.1147	0.1147
spam	0.0578	0.0063	-0.0013	0.0028	0.0015	0.0485	0.0435
tic	0.0438	0.0354	0.0011	0.0042	-0.0021	0.0052	0.0010
wdbc	0.0263	0.0070	0.0018	-0.0018	-0.0018	0.0211	0.0193
wilt	0.0528	-0.0121	0.0527	0.0000	0.0004	0.0118	0.0112

All in all, the theory, that larger *munge_size* results in a better approximation of the student does not hold in general. There are only 5 of 19 tasks, where this behavior can be seen. For all other tasks, there are some step backs in mmce when increasing the number of pseudo observations. But it is reasonable, to generate at least 50 or 100 thousand pseudo observations since in this area significant improvements are recognized, compared to just 10,000 observations. In some cases, it might even be worth a try to go beyond that, especially, when knowledge distillation is not successful like it is the case for the task 'elect'.

4.2.3 Linear combinations with nearest neighbors

In section 3.2.3, a modification to MUNGE was introduced, combining teacher labels with nearest neighbor labels from the original data set. The goal is, that misclassifications of the teacher, can be smoothed out through the nearest neighbors and $\hat{f}_{student}$ outperforms $\hat{f}_{teacher}$.

The question is: Does this method of linear combinations of teacher labels with nearest neighbors improve the student nets beyond the teacher's performance?

The first answer to this can be found in the benchmark comparison. Since this method (hyperparameter *lin_comb_nn*) was up to tuning, the percentage of times it was the result of MBO, is meaningful. This parameter was selected in just 24% of all folds from the 19 tasks. This is an indication that it is not as helpful as expected.

To investigate the influence of this hyperparameter more deeply, the tuning results from the benchmark comparison are used for another experiment. All other hyperparameters stay as the result from tuning, just *lin_comb_nn* is set once to TRUE and once to FALSE. Thereafter, on each fold the difference between mmce for the student without linear combination minus the student with linear combination is calculated and compared to each other.

Figure 4.7 shows the result of this comparison. The distribution of the fold-wise difference in test mmce of students without minus students with linear combination is shown. In consequence, positive values indicate, that the linear combination with nearest neighbors yields a lower mean misclassification error.

The overall distribution is roughly centered at zero, rather a little bit shifted to the left. This is consistent with most of the tasks. For many of them, the distribution of this difference is centered at 0. For tasks 'diabetes' and 'credit', the median is negative around 0.02, meaning degradation of 2% in mmce.

This analysis indicates, that this hyperparameter is of no use for knowledge distillation. It might even make the copy of $\hat{f}_{teacher}$ worse in contrary to the goal to improve it beyond.

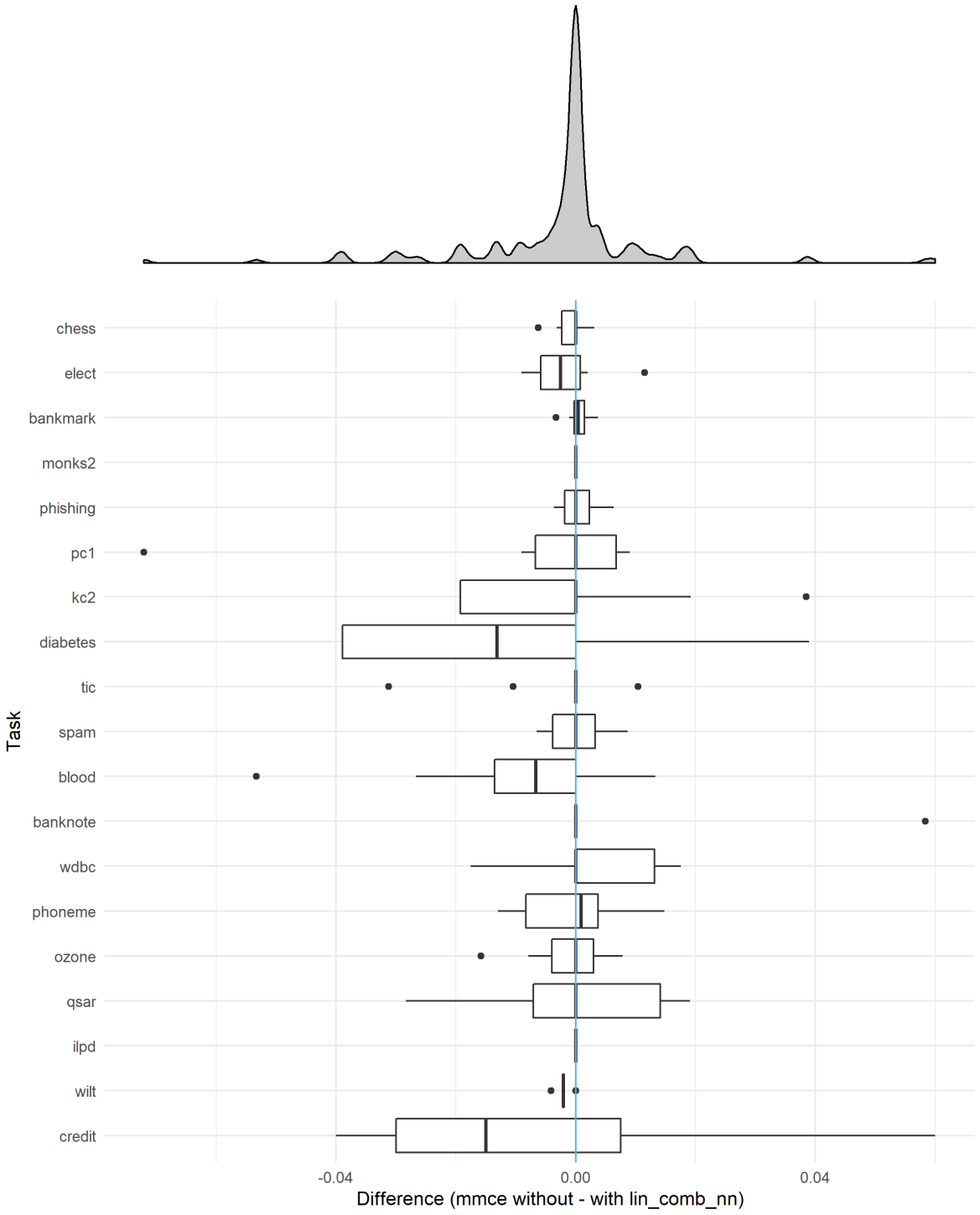

Figure 4.7: Influence of *lin_comb_nn* - The lower plot shows the distribution per task of the difference in test mmce of students without minus students with linear combination. The density plot on top shows the overall distribution of this difference over all tasks.

4.2.4 Approximation error

The last point investigated in this section is the meaning of the training error of $\hat{f}_{student}$. The concrete question is: What is possible to foresee, regarding the predictive behavior of the student on new data, only knowing the training error on pseudo observations.

Regular machine learning algorithms, like the directly fitted net in the benchmark comparison, are trained on exactly the same structured data, as they predict on thereafter. So, the training error can be set in direct context with the generalization error, for example the mmce on test data.

The student net is trained on pseudo observations with soft labels, coming from $\hat{f}_{teacher}$. The training loss in this case is the mean squared error between the teacher's label and the prediction of the student net. But when it comes to predicting on new data, the goal is not anymore to imitate the teacher, but just to regularly label data with probabilities for the positive class. So, for this special learner, train and test data is not homogenous.

The attempt to somehow transfer the loss to the teacher's labels on training data to a general evaluation measure for binary classification like mmce on test data would not make sense at all. Each teacher on each data set has a different level of accuracy for instance. And a low training loss, could only imply, that the student is close to the teacher. If the teacher himself has low accuracy, the student has a low accuracy as well. So, low training loss can lead to high or low accuracy on test data, regarding the overall generalization error.

The only 'generalization' that can be interpreted, is the approximation of $\hat{f}_{teacher}$. Especially, the squared differences of positive probabilities on test observations. When the student is fitted, it aims to reduce this difference to the teacher's soft labels. And thus, the generalization error, in this case, is low, if the training mse is close to the test mse.

Figure 4.8 shows the difference between the training mse (validation loss of student net) and the test mse on each fold, evaluated on the results of the benchmark comparison. All student nets are fitted with hyperparameter *lin_comb_nn* set to FALSE. This way, $\hat{f}_{student}$ learned from $\hat{f}_{teacher}$ only. The blue lines mark the values -0.01 and 0.01. Since, the test mse is subtracted from training mse, positive values mean a lower difference on test data than on training data.

For the majority of tasks, the median of the difference is pretty close to zero. Also for most cases all observations are within the blue bars. Only for task 'elect' the median is shifted to the right. It is also conspicuous, that all differences on folds with absolute values greater than 0.01 are positive.

This shows that the training error is an indication for the 'generalization' error, meaning that low validation mse results in a close approximation of the teacher. There are too few positive outliers to state, that the test error tends to be better than the training error. But it seems very unlikely, that the training error is too optimistic.

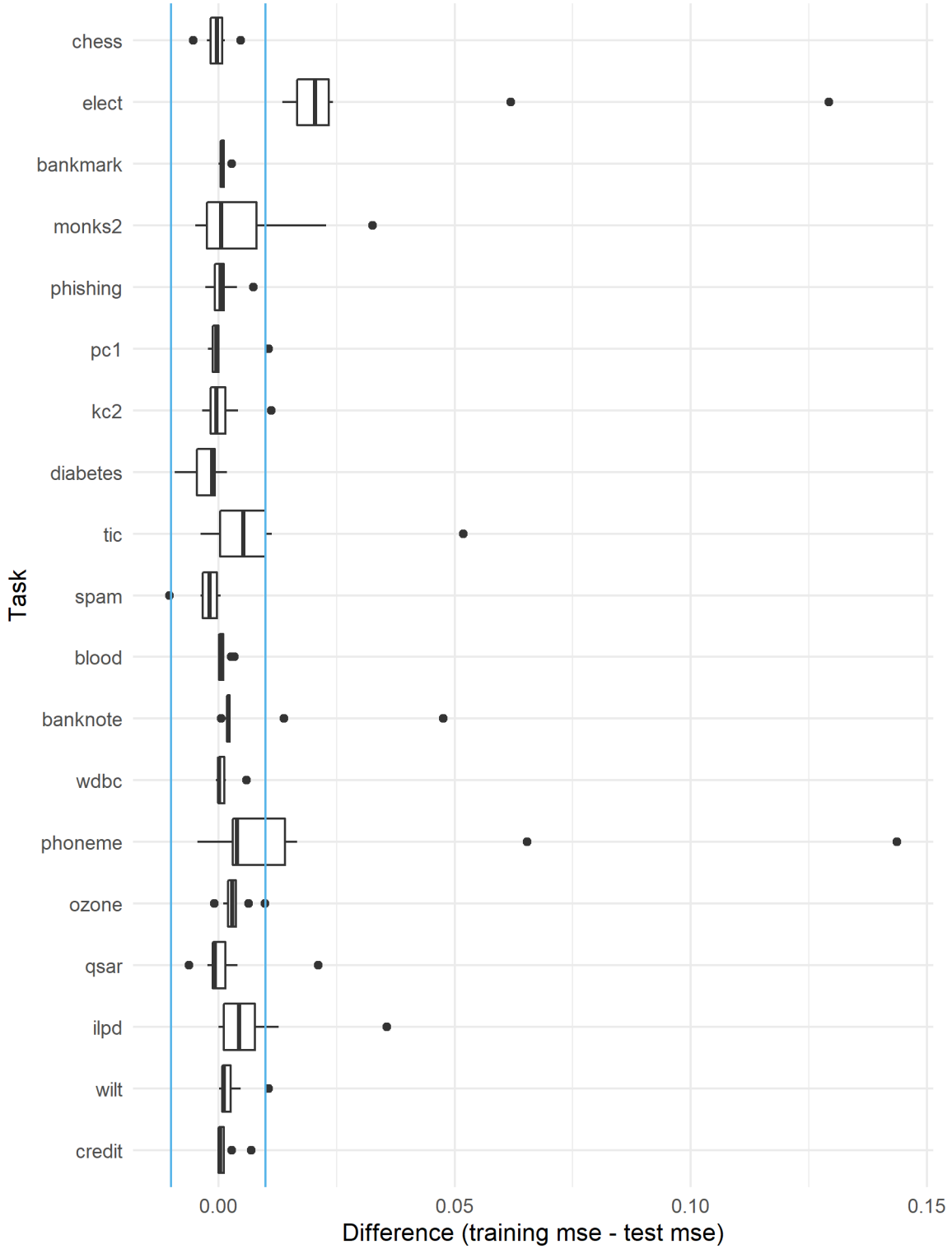

Figure 4.8: Difference of train and test mse - Distribution of the fold-wise difference between student's training mse and the mse between the student's and teacher's labels on test data.

Still one has to be careful, with the interpretation. Some issues have to be kept in mind, when using the student's validation loss to predict its behavior on new data. First, *munge_size* was set to 100,000 in this case. If it is too small, the training error might be low as well, but the test error can be way higher. Second, the training error is the mse between the prediction of the student and the pseudo labels. Let the teacher have an accuracy of around 75%, for example. Then it is most likely that it has many observations labeled with low confidence, roughly around 0.5 probability for the positive class. If the student is pretty close to the teacher, regarding these probabilities, just small differences can make a decision for the respectively other class and the predictive behavior is not that close anymore for a measure like mmce. Third, when methods like the linear combination with nearest neighbors (section 3.2.3), modify the pseudo labels, it gets very hard to make any assumption about the predictive behavior of $\hat{f}_{student}$.

4.2.5 Default values

To finish this experimental section, a suggestion for default values for the hyperparameters is given now. On the basis of the results from the benchmark experiment and the ablation analysis, the hyperparameter constellation shown in table 4.5 is a fair first try for knowledge distillation on any tabular data set with a fitted learner. Among these parameters, *learning_rate* is the one that most likely needs tuning, when knowledge distillation is applied to new data.

Table 4.5: Suggested default hyperparameters for knowledge distillation

Neural net		Data generation	
Hyper parameter	Tuning range	Hyper parameter	Tuning range
net_shape	rectangular	munge_size	100,000
act_function	relu	swap_prob	0.1
fl_units	256	var_param	1
hidden_layers	4	orig_portion	0
learning_rate	0.0001	lin_comb_nn	FALSE
dropout_rate	0	dark_knowledge	TRUE
weight_decay	0	munge_otf	FALSE
optimizer	adam		
batch_norm	TRUE		

Chapter 5

Application: Compressing an ML pipeline

Throughout this thesis, it has been shown, how and that knowledge distillation can be done successfully for a broad range of tabular data sets and learners. It goes well with SVM, random forest and xgboost. In many real use cases, before the actual learner is applied to the data, preprocessing methods are conducted on the data set.

So, the question arises, if it is possible to not just distill the knowledge of a learner but to compress a whole machine learning pipeline?

Here only the possibility is shown, so it is just a small experiment. Randomly, two of the task and teacher combinations from the benchmark comparison are selected ('qsar' and 'kc2' with teachers SVM and xgboost). To both learners the preprocesses scaling and principal component analysis (PCA) are added, which are applied before the learner is fitted on the data. These fitted pipelines of scaling, PCA and then SVM/xgboost are used as $\hat{f}_{teacher}$. The hyper parameters for the student algorithms are only tuned for the $learning\_rate \in \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$. The other parameters are set to default, suggest in section 4.2.5.

Table 5.1 shows the results. The student for 'qsar' performs slightly worse than the teacher does, while for task 'kc2' and the xgboost pipeline, the student is slightly better.

This is just one example of a pipeline, but it shows that knowledge distillation can be done successfully for such a machine learning pipeline as well.

Table 5.1: Pipeline compression - Results evaluated on 10-fold CV

Task	Algorithm	mmce
qsar	svm.pca.scale	0.1174858
qsar	student	0.1203069
kc2	xgboost.pca.scale	0.1723640
kc2	student	0.1648560

Chapter 6

Conclusion and outlook

On a benchmark experiment with 19 tasks, it was shown, that neural nets trained via knowledge distillation for binary classification can outperform nets of the same structure directly fitted on tabular data. The method performed well across differently structured data sets with less than 1,000 observations or with more than 45,000. It is possible to copy random forests, SVMs, xgboost models and even learning pipelines with scaling and PCA to preprocess the data. Additionally, the observation-wise investigation on predictions pointed out, that the approximation of the teacher algorithm is close to the soft label predictions of the teacher in most cases. Without success, some modifications to the method MUNGE were tried to improve the student's performance. Neither the linear combination with nearest neighbors turned into an improvement, nor the use of some original portion in the pseudo observations. But the use of dark knowledge was successful, especially for the approximation of the teacher's predictive behavior on single observations. Some new activation functions were tried with 'selu' and 'hard_sigmoid' but were outperformed by 'relu' for most cases.

This method might be useful in different fields. The original goal by Caruana et al. (2006), to compress ensemble learners into a neural net for deployment, could be easily done by transforming the fitted net into a binary file and use it in any other programming language. Another benefit of distilling any learner into a neural net is the obtained input gradients (Hechtlinger, 2016). By copying different learners into neural nets, it is possible to interpret their prediction by one uniform method, applicable to neural nets. Further, it could be worth it, to try this method on regression tasks. Most likely, the numeric target variable would need to be scaled, to be predictable for the student net. Then it should be no problem, since, for this thesis, the net was trained via minimizing the mse to the positive probabilities, as well.

To put it in a nutshell, there is a lot of potential in knowledge distillation, not just for binary classification.

Chapter 7

Appendix

7.1 Model-based optimization - mlrMBO

Model-based optimization is a method for 'expensive black-box optimization by approximating the given objective function through a surrogate regression model' (Bischl et al., 2017b). Since student nets, in this case, are time-consuming to fit and have many hyperparameters to be tuned over, MBO was chosen to find the best parameter set.

Tuning hyperparameters is the optimization of a machine learning algorithm f regarding its hyperparameter space Λ on a data set $D = (X, y)$. To be able to optimize an algorithm, an evaluation function has to be defined. Let $eval\_method(\lambda, f, D)$ be this evaluation function, which - w.l.o.g - has to be minimized.

Then, the optimal result of tuning is the solution to equation (7.1).

$$\lambda^* = \underset{\lambda \in \Lambda}{\operatorname{argmin}}\ eval\_method(\lambda, f, D) \tag{7.1}$$

The pseudo code for MBO's attempt to solve equation (7.1) is algorithm 4.

Algorithm 4: Model-based optimization

Input : $D = (X, y)$, f, Λ
Parameters : n, m, *budget*, f_{surr}, *eval_method*, *init_method*, $infill\_crit$
Result : Best hyperparameter set λ^*

```
sample n points from Λ via init_method resulting in Λ_sample
label each λ ∈ Λ_sample via eval_method resulting in (y, Λ_sample)
while budget is not exceeded do
    fit f_surr on (y, Λ_sample) to obtain f̂_surr
    propose m new points in Λ via f̂_surr and infill_crit
    evaluate new points and add the result tuples to (y, Λ_sample)
end
return λ* as the λ ∈ Λ_sample with the smallest corresponding y value
```

The parameters for MBO are:

1. n - The number of initially sampled points from Λ.
2. *init_method* - An algorithm determining how the n initial points are sampled from Λ. One standard method is 'Maximin Latin Hypercube Sample', which 'attempts to optimize the sample by maximizing the minimum distance between design points' (lhs, 2019). Sampling randomly is also an alternative.
3. *eval_method* - A method evaluating hyperparameter sets from Λ. Usually f with hyperparameter set λ is fitted and evaluated via cross-validation on D. So, $eval\_method(\lambda, f, D)$ and in consequence the labels of Λ_{sample} are the average loss over the CV folds.
4. f_{surr} - The surrogate model, approximating the evaluations of f for different hyperparameter sets. This model is a regression model, fitted on (y, Λ_{sample}) over and over again when new points are added to Λ_{sample}. The goal is, to find minima of $eval\_method(\lambda, f, D)$, regarding λ, by estimating its behavior with the known points (y, Λ_{sample}).
5. $infill\_crit$ - The infill criterion. It is a function to estimate how promising new points $\lambda \in \Lambda$ are. It aims to achieve a good trade-off between 'exploitation and exploration' (Bischl et al., 2017b). Therefore, the estimated expected value and the estimated variance of λ are set in context, targeting low expected value (goal is to minimize $eval\_method(\lambda, f, D)$) and high variance, since there is potential to improve. An example for an infill criteria is the 'lower confidence bound' (equation (7.1)). α regulates the balance. The estimation of $\hat{E}$ and $\hat{SD}$ are obtained by the surrogate model. The expected value is the prediction and the variance in a random forest can be estimated by the 'jackknife-after-bootstrap' method (Efron, 1992), for example.
6. m - The number of proposed new points from Λ per loop.
7. *budget* - The budget for the MBO process. This can be basically any condition, for example total time for tuning or a number of total evaluations. When this limit is reached, the algorithm stops and returns the best hyperparameter set found.

$$LCB(\lambda, \alpha) = \hat{E}(\lambda) - \alpha \hat{SD}(\lambda) \tag{7.2}$$

7.2 Additional tables and plots

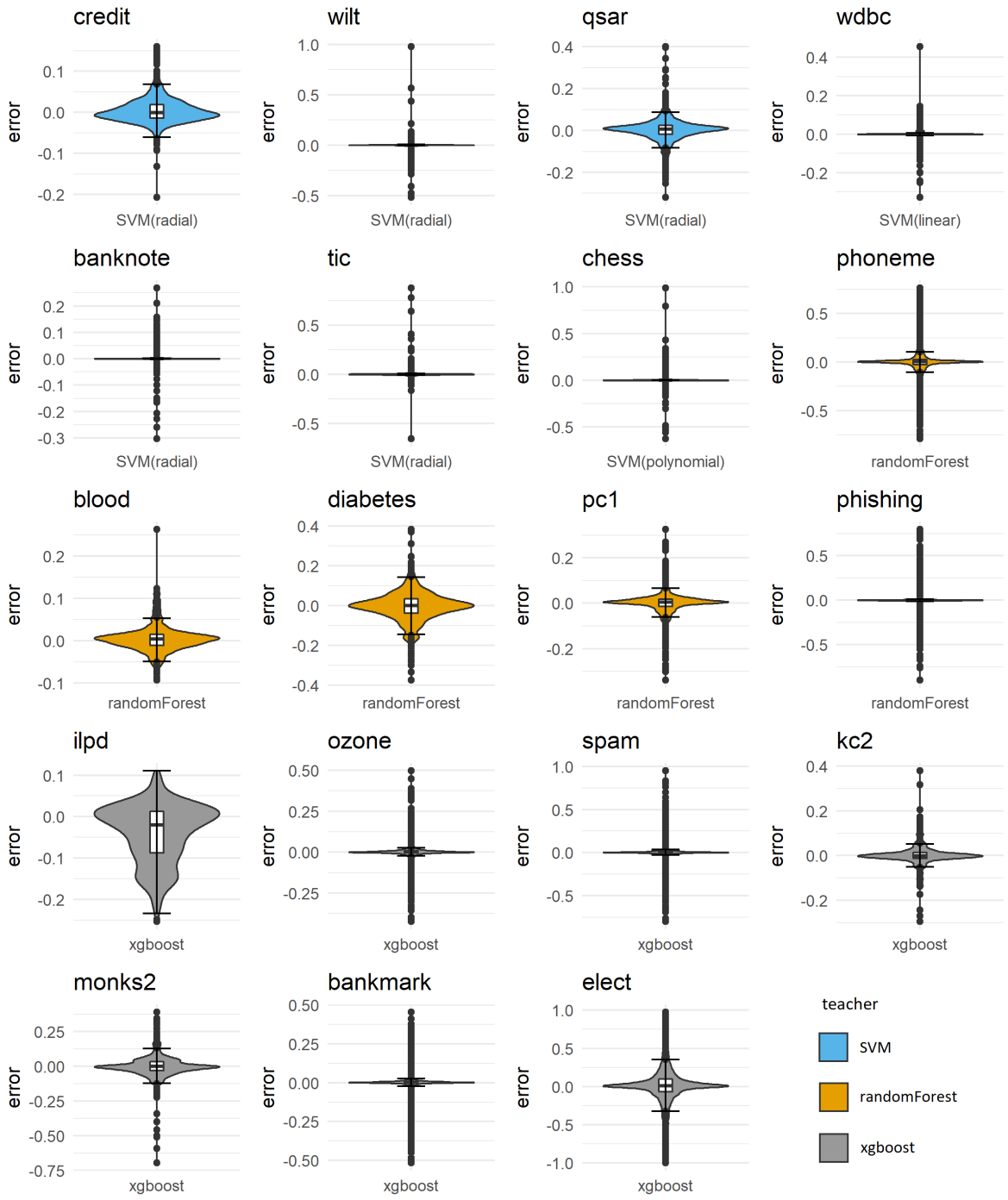

Figure 7.1: Distribution of error from student to teacher - The distribution of observation-wise difference in positive prediction between teacher and student is grouped by the teacher type.

Table 7.1: Task names and openML run IDs - The full names for the tasks and the run IDs for the teachers in the benchmark comparison.

Task	Full Name	Teacher's oML run ID
ilpd	ilpd	5620384
qsar	qsar-biodeg	4256262
ozone	ozone-level-8hr	5722374
phoneme	phoneme	8833050
wdbc	wdbc	3850987
banknote	banknote-authentication	2063917
blood	blood-transfusion-service-center	6613421
spam	spambse	6180012
tic	tic-tac-toe	3373008
diabetes	diabetes	5545055
kc2	kc2	4234286
pc1	pc1	3495761
phishing	PhishingWebsites	8110032
wilt	wilt	4201453
monks2	monks-problems-2	7781944
elect	electricity	7946315
bankmark	bank-marketing	6578472
credit	credit-g	6384382
chess	kr-vs-kp	4870477

Table 7.2: Overall mmce comparison with differences - The first three columns are the directly fitted neural net's, student's and teacher's overall mean misclassification error. The last two columns show the difference in mmce for direct NN to student and student to teacher. This table is ordered, regarding the last column.

Task	mmce NN	mmce student	mmce teacher	NN - student	Student - teacher
monks2	0.3428	0.0000	0.0066	0.3428	-0.0066
wdbc	0.0263	0.0175	0.0193	0.0088	-0.0018
phishing	0.0397	0.0257	0.0269	0.0140	-0.0012
banknote	0.0196	0.0000	0.0000	0.0196	0.0000
ilpd	0.3275	0.2864	0.2864	0.0411	0.0000
wilt	0.0320	0.0122	0.0112	0.0198	0.0010
ozone	0.0816	0.0545	0.0533	0.0272	0.0012
qsar	0.1688	0.1166	0.1147	0.0522	0.0019
chess	0.0332	0.0041	0.0022	0.0291	0.0019
bankmark	0.1121	0.0915	0.0892	0.0207	0.0022
pc1	0.0757	0.0631	0.0595	0.0126	0.0036
blood	0.2928	0.2112	0.2072	0.0815	0.0040
tic	0.1074	0.0073	0.0010	0.1001	0.0062
diabetes	0.2514	0.2327	0.2240	0.0187	0.0086
spam	0.0645	0.0528	0.0435	0.0117	0.0093
phoneme	0.2073	0.0946	0.0851	0.1127	0.0094
kc2	0.1762	0.1533	0.1436	0.0230	0.0097
credit	0.2740	0.2460	0.2350	0.0280	0.0110
elect	0.1721	0.1508	0.0536	0.0213	0.0972

Table 7.3: Overall logloss comparison with differences - The first three columns are the directly fitted neural net's, student's and teacher's overall logloss. The last two columns show the difference in logloss for direct NN to student and student to teacher.

Task	logl NN	logl student	logl teacher	NN - student	Student - teacher
wdbc	0.0825	0.0765	0.0812	0.0060	-0.0046
diabetes	0.5271	0.5039	0.4800	0.0233	0.0238
phishing	0.0927	0.0690	0.0754	0.0237	-0.0065
ilpd	0.5698	0.5423	0.5834	0.0275	-0.0411
elect	0.3797	0.3479	0.1379	0.0317	0.2100
spam	0.1830	0.1458	0.1271	0.0371	0.0187
pc1	0.2438	0.1942	0.2210	0.0496	-0.0268
wilt	0.0908	0.0355	0.0345	0.0553	0.0011
credit	0.5674	0.4950	0.4893	0.0724	0.0057
ozone	0.2171	0.1440	0.1427	0.0731	0.0013
chess	0.0947	0.0149	0.0138	0.0798	0.0011
banknote	0.0877	0.0003	0.0064	0.0874	-0.0061
bankmark	0.3026	0.2019	0.1944	0.1007	0.0075
kc2	0.4910	0.3678	0.3647	0.1232	0.0032
blood	0.6584	0.5237	0.4771	0.1346	0.0467
qsar	0.4746	0.3002	0.3040	0.1744	-0.0038
phoneme	0.4376	0.2519	0.2273	0.1857	0.0245
tic	0.2536	0.0256	0.0065	0.2280	0.0191
monks2	0.5984	0.0001	0.0569	0.5984	-0.0569

Bibliography

Arik, S. and Pfister, T. (2019). Tabnet: Attentive interpretable tabular learning. *ArXiv*, abs/1908.07442.

Ashok, A., Rhinehart, N., Beainy, F., and Kitani, K. M. (2018). N2n learning: Network to network compression via policy gradient reinforcement learning. In *International Conference on Learning Representations*.

Bischl, B., Casalicchio, G., Feurer, M., Hutter, F., Lang, M., Mantovani, R. G., van Rijn, J. N., and Vanschoren, J. (2017a). Openml benchmarking suites.

Bischl, B., Lang, M., Kotthoff, L., Schiffner, J., Richter, J., Studerus, E., Casalicchio, G., and Jones, Z. M. (2016). mlr: Machine learning in r. *Journal of Machine Learning Research*, 17(170):1–5.

Bischl, B., Richter, J., Bossek, J., Horn, D., Thomas, J., and Lang, M. (2017b). mlrmbo: A modular framework for model-based optimization of expensive black-box functions.

Bowyer, K. W., Chawla, N. V., Hall, L. O., and Kegelmeyer, W. P. (2011). SMOTE: synthetic minority over-sampling technique. *CoRR*, abs/1106.1813.

Carnell, R. (2019). Basic latin hypercube samples and designs with package lhs.

Caruana, R., Bucila, C., and Niculescu-Mizil, A. (2006). *Model Compression*.

Chen, L. (2009). *Curse of Dimensionality*, pages 545–546. Springer US, Boston, MA.

Chen, W., Wilson, J. T., Tyree, S., Weinberger, K. Q., and Chen, Y. (2015). Compressing neural networks with the hashing trick. *CoRR*, abs/1504.04788.

Cheng, Y., Wang, D., Zhou, P., and Zhang, T. (2017). A survey of model compression and acceleration for deep neural networks. *CoRR*, abs/1710.09282.

Chollet, F., Allaire, J., et al. (2017). R interface to keras. https://github.com/rstudio/keras.

Efron, B. (1992). Jackknife-after-bootstrap standard errors and influence functions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 54(1):83–127.

Guo, C. and Berkhahn, F. (2016). Entity embeddings of categorical variables. *CoRR*, abs/1604.06737.

Hechtlinger, Y. (2016). Interpretation of prediction models using the input gradient. *ArXiv*, abs/1611.07634.

Hinton, G. E., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531.

kaggle-user anokas (2017). kaggle.com - ranking of kaggle algorithms by competitions won.

kaggle-user DanB (2018). kaggle.com - what is xgboost.

Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. (2017). Self-normalizing neural networks. *CoRR*, abs/1706.02515.

lhs, T. (2019). maximinlhs: Maximin latin hypercube sample.

Mikolov, T., Chen, K., Corrado, G. S., and Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Mirzadeh, S., Farajtabar, M., Li, A., and Ghasemzadeh, H. (2019). Improved knowledge distillation via teacher assistant: Bridging the gap between student and teacher. *CoRR*, abs/1902.03393.

Polino, A., Pascanu, R., and Alistarh, D. (2018). Model compression via distillation and quantization. In *International Conference on Learning Representations*.

Popov, S., Morozov, S., and Babenko, A. (2019). Neural oblivious decision ensembles for deep learning on tabular data. *ArXiv*, abs/1909.06312.

Sau, B. B. and Balasubramanian, V. N. (2016). Deep model compression: Distilling knowledge from noisy teachers. *CoRR*, abs/1610.09650.

Student, I. E. C. S. (2018). quora.com - what machine learning approaches have won most kaggle competitions?

Tanno, R., Arulkumaran, K., Alexander, D. C., Criminisi, A., and Nori, A. V. (2018). Adaptive neural trees. *CoRR*, abs/1807.06699.

Vanschoren, J., van Rijn, J. N., Bischl, B., and Torgo, L. (2013). Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60.

Yang, Y., Morillo, I. G., and Hospedales, T. M. (2018). Deep neural decision trees. *CoRR*, abs/1806.06988.

Yeom, S.-K., Seegerer, P., Lapuschkin, S., Wiedemann, S., Müller, K.-R., and Samek, W. (2019). Pruning by explaining: A novel criterion for deep neural network pruning. *ArXiv*, abs/1912.08881.