machine learning & knowledge extraction

Article

FeaSel-Net: A Recursive Feature Selection Callback in Neural Networks

Felix Fischer [1,*], Alexander Birk [1], Peter Somers [2], Karsten Frenner [1], Cristina Tarín [2] and Alois Herkommer [1]

1 Institute of Applied Optics, University of Stuttgart, 70569 Stuttgart, Germany
2 Institute for System Dynamics, University of Stuttgart, 70563 Stuttgart, Germany
* Correspondence: felix.fischer@ito.uni-stuttgart.de

Abstract: Selecting only the relevant subsets from all gathered data has never been as challenging as it is in these times of big data and sensor fusion. Multiple complementary methods have emerged for the observation of similar phenomena; oftentimes, many of these techniques are superimposed in order to make the best possible decisions. A pathologist, for example, uses microscopic and spectroscopic techniques to discriminate between healthy and cancerous tissue. Especially in the field of spectroscopy in medicine, an immense number of frequencies are recorded and appropriately sized datasets are rarely acquired due to the time-intensive measurements and the lack of patients. In order to cope with the curse of dimensionality in machine learning, it is necessary to reduce the overhead from irrelevant or redundant features. In this article, we propose a feature selection callback algorithm (FeaSel-Net) that can be embedded in deep neural networks. It recursively prunes the input nodes after the optimizer in the neural network achieves satisfying results. We demonstrate the performance of the feature selection algorithm on different publicly available datasets and compare it to existing feature selection methods. Our algorithm combines the advantages of neural networks' nonlinear learning ability and the embedding of the feature selection algorithm into the actual classifier optimization.

Keywords: classification algorithms; dimensionality reduction; feature selection; linear discriminant analysis; machine learning; neural networks; principal component analysis

check for updates

Citation: Fischer, F.; Birk, A.; Somers, P.; Frenner, K.; Tarín, C.; Herkommer, A. FeaSel-Net: A Recursive Feature Selection Callback in Neural Networks. *Mach. Learn. Knowl. Extr.* **2022**, *4*, 968–993. https://doi.org/10.3390/make4040049

Academic Editors: Basabi Chakraborty and Saptarsi Goswami

Received: 12 October 2022
Accepted: 29 October 2022
Published: 31 October 2022

1. Introduction

Nowadays, the trend in many industries, such as the automotive industry and life sciences, is toward real-time data acquisition and multi-modal sensing, which ultimately produce a vast amount of data while always considering multiple features and many physical dimensions. It lies in the nature of large vectors of measured data that some of the observed features contribute less information than others for the understanding and modeling of real-world phenomena. Using many sensors yields an increasing production and service cost due to the demand for more components [1] and greater processing power, as well as RAM and data storage. In addition to the economic downsides, there is also a very significant statistical problem with unnecessarily many dimensions in most machine learning (ML) algorithms, which affects their performance in processing the measured data. With a poor sample-to-dimension ratio, outliers and noise in data get too much attention and the algorithms tend to overfit, regardless of if they are used for regression, clustering, or classification tasks [2]. In order to achieve reasonable generalizability in the training process of most ML algorithms, the number of samples needed grows exponentially when the number of dimensions—in the context of feature reduction, dimensions, attributes, and features are inter-changeable definitions—grows linearly [3]. This behavior is often referred to as the *curse of dimensionality* in the literature. To tackle this phenomenon, either the number of samples must be drastically increased, which yields even more data and longer acquisition times, or the dimensionality has to be reduced.

A conservative and common approach to coping with the problem of overabundant dimensions is the use of a feature extraction method, such as principal component analysis (PCA) [4,5] or linear discriminant analysis (LDA), during the pre-processing stage. Both techniques aim to describe the data in a much smaller subspace with only a few latent features that are created through a linear transformation of the original features. Another form of dimensionality reduction can be achieved by using graph-based algorithms, such as t-SNE [6] or the more recently published Uniform Manifold Approximation and Projection (UMAP) [7]. Variational autoencoders (VAEs) are generative models that can also be used for feature extraction purposes [8,9]. Unlike PCA and LDA, both VAEs and graph-based algorithms are nonlinear dimensionality reduction techniques. However, all previously mentioned methods are mostly employed for the compression of the entire original dataset to obtain less computationally intensive ML models without losing a significant amount of information. A problem with the reduction is that the ensuing extracted features cannot be easily interpreted and are certainly not measurable, since direct reference to the original features is lost. Despite the fact that they do not necessarily need to be interpretable for subsequent supervised or unsupervised learning algorithms, it is still important for scientists to have physically meaningful features that are measurable.

Feature selection, contrary to keeping the information of the whole dataset, only focuses on relevant subsets. Those subsets are chosen such that only the most informative ones are preserved. Moreover, the original features remain unchanged before being fed into the subsequent algorithm. The big advantage of feature selection methods is that after the evaluation has been made, it is easy to deduce how many and which observations are necessary for the desired estimation, i.e., they provide information about how sparse the measurement can be while the algorithm still leads to acceptable results.

According to [10], the available methods can be organized into three different categories: *filtering*, *wrapping*, and *embedded* methods. Filter techniques calculate the relevance score for each dimension and low-scoring features are removed. All filter methods work independently from the following algorithm; they can be seen as a pre-processing operation and can be used for the dimensionality reduction of subsequent deep-learning-based classifications [11]. On the other hand, wrappers communicate with the ML algorithm and perform differently for each algorithm and depending on the hyper-parameters used. Wrappers (oftentimes randomly) pre-select features that a subsequent classifier trains on and evaluate the classifier's performance with these specific features. The search for the best-suited features can be executed either exhaustively (e.g., k-fold cross-validation [2]) or heuristically (e.g., sequential feature selection). Unfortunately, the first two types of feature selection (FS) are not integrable into the actual learning algorithm. This ability is provided by embedded methods, such as decision-tree-based algorithms [12,13], or by applying ℓ_1 or ℓ_2 regularizations on ML models that shrink uninformative parameters to almost, but not exactly, zero [14–16]. In these types of selectors, the FS algorithm and the classifier converge to the features of the highest importance. Other approaches focus solely on the feature space represented by the input layer of a neural network [17–19], and a strict binarization is provided. All methods use relaxed versions of the ℓ_0 regularization. However, these regularizations highly depend on different hyper-parameters and the initialization of the weights in neural networks, which have an impact not only on the selected features, but also the size of the feature subset.

In this paper, we propose FeaSel-Net (FeaSel), a new recursive feature selection algorithm, which can easily be embedded into any neural network. The algorithm itself is a network pruning algorithm that—unlike dropout [20]—only prunes definite nodes at the input layer and permanently excludes their contribution to the optimization. Similarly to Guyon's recursive feature extraction (RFE) [21], we rank the importance of each feature for the decision making in classification tasks and recursively prune nodes in the input layer of our neural network until a desired number of features is obtained. The bias from the initialization of weights is bypassed when delaying the pruning process to a later epoch, where the classifier already performs well.

To prove and describe its functionality, FeaSel-Net was applied on the *Wine Classification* dataset [22]. The results are compared to those of existing FS methods based on PCA and LDA, the tree-based eXtreme Gradient Boosting (XGBoost) algorithm [13], and an approach using stochastic gates (STG) [17], which outperformed the other ℓ_0 regularizers. The performance of another unbiased classifier fed with only the distilled features was investigated. For deeper feature importance and dependency estimation, we define a new weighted Jaccard matrix.

2. Comparison Methods—Linear Transformations Using PCA and LDA

This chapter provides an overview on the comparison methods used for the feature selection derived from the PCA approaches described in [23,24]. They are additionally extended by LDA, since PCA itself is a purely unsupervised clustering method, and the class information is not considered during the transformation process. Both analyses are *filter methods* based on linear transformation and are commonly used in bioinformatics [25–27], where the feature importance is often measured by the transformations' loadings. Their FS capabilities will be compared to the proposed algorithm in Section 4.4.

2.1. Principal Component Analysis

Originally introduced by Karl Pearson [4], the principal component analysis (PCA) is a popular technique in multivariate statistics for processing complex datasets. The analysis reduces high-dimensional data by linearly transforming the initial data into a latent variable space, where the newly created variables are inherently uncorrelated and orthogonal to each other. This space is defined by the q first-ordered principal components (PCs), which are uncorrelated (i.e., orthogonal) axes that obtain the highest variance when data are projected onto them. The original dataset $\mathbf{X} \in \mathbb{R}^{p\times n}$ consists of p features and n samples or observations, whereas the reduced score matrix $\mathbf{Y} \in \mathbb{R}^{q\times n}$ is obtained after the transformation. In order to find the weights for the linear transformation, the eigenvalue problem

$$\Sigma\mathbf{V} = \lambda\mathbf{I}\mathbf{V} \tag{1}$$

has to be solved for the covariance matrix $\Sigma \in \mathbb{R}^{p\times p}$ of the dataset, where $\lambda \in \mathbb{R}^{1\times p}$ represents the eigenvalues and $\mathbf{V} \in \mathbb{R}^{p\times p}$ is the corresponding eigenvector matrix. The magnitude of the eigenvalues directly implies the explained variance by each PC. Since the majority of the information is stored in the components that maintain the most variance, the corresponding eigenvalues and vectors are sorted in decreasing order. For the purpose of feature reduction, only the q first eigenvectors are chosen. The values of the eigenvectors are scaled by the standard deviation based on each PC's eigenvalue in order to relativize the impact of each vector on the transformation. The resulting matrix is conventionally called the loading matrix

$$\mathbf{A}_{ij} = \mathbf{V}_j\sqrt{\lambda_j} \qquad \text{where } \mathbf{A} \in \mathbb{R}^{p\times q} \tag{2}$$

which contains a loading vector $\mathbf{A}_j$ for each considered component j and is tantamount to the desired weights for the linear transformation. Multiplying the transposed loading matrix with the original dataset yields the $q \times n$-sized score matrix

$$\mathbf{Y} = \mathbf{A}^{\mathsf{T}}\mathbf{X} \tag{3}$$

which describes the linearly transformed dataset in the new q-dimensional latent variable space. By mean-centering the data

$${}_{c}x_{ij} = x_{ij} - \bar{x}_i \qquad \text{with} \qquad \bar{x}_i = \frac{1}{n}\sum_{j=0}^{n} x_{ij} \tag{4}$$

along the sample axis beforehand, the covariance matrix is computed using a simple matrix product scaled by the reciprocal of the degrees of freedom:

$$\Sigma = \mathrm{Cov}({}_{\mathrm{c}}\mathbf{X}) = \frac{1}{n-1}{}_{\mathrm{c}}\mathbf{X}^{\mathsf{T}}{}_{\mathrm{c}}\mathbf{X}. \tag{5}$$

In many cases in which the features inherently show different variances (e.g., metrical data with various orders of magnitude or in the simultaneous processing of metrical and categorical data), the data must also be standardized, which yields

$${}_{\mathrm{s}}x_{ij} = \frac{{}_{\mathrm{c}}x_{ij}}{s_i} \qquad \text{with} \qquad s_i = \sqrt{\frac{1}{n-1}\sum_{j=0}^{n}\left(x_{ij} - \bar{x}_i\right)^2}. \tag{6}$$

The standardization occurs along the same axis as that specified in Equation (4) and is done by dividing the mean-centered data by the empirical standard deviation $\mathbf{s}_i$.

The magnitudes of the loadings inside $\mathbf{A}$ are correlations between original and latent variables and are often considered as a suitable metric for the information content $\mathcal{I}$ within a dataset [23]. Since the first few PCs are the most informative components, this is also the subspace in which the biggest loadings occur. Thus, another evaluation is carried out through the application of the ℓ_1 norm for each loading vector

$$\mathcal{I}_i = ||\mathbf{a}_i||_1 = \sum_{j=0}^{q}|a_{ij}| \tag{7}$$

where only the q most important components are considered [24].

2.2. Linear Discriminant Analysis

An linear discriminant analysis (LDA) is a linear transformation method that aims to maximize the distance between the means while it minimizes the variances within one class. Consequently, the class information for each sample has to be considered. This is done by calculating the features' means and variances for each individual class. Therefore, the algorithm belongs to the group of supervised classification algorithms. Whilst PCA solves the eigenvalue problem for the covariance matrix, LDA makes use of two different scatter matrices: the scatter-within $\mathbf{S}_{\mathrm{w}}$ and scatter-between $\mathbf{S}_{\mathrm{b}}$ matrices, which are defined as follows:

$$\mathbf{S}_{\mathrm{w}} = \sum_{i=1}^{n_{\mathrm{c}}}{}_{\mathrm{c}}\mathbf{X}_i^{\mathsf{T}}{}_{\mathrm{c}}\mathbf{X}_i \tag{8}$$

$$\mathbf{S}_{\mathrm{b}} = \sum_{i=1}^{n_{\mathrm{c}}} n_i \cdot ({}_{\mathrm{c}}\bar{\mathrm{x}}_i - {}_{\mathrm{c}}\bar{\mathrm{x}})({}_{\mathrm{c}}\bar{\mathrm{x}}_i - {}_{\mathrm{c}}\bar{\mathrm{x}})^{\mathsf{T}}. \tag{9}$$

In order to get the scatter-within matrix, the dataset is split into n_{c} classes, and their respective scatter-within matrices are calculated. Afterwards, these n_{c} scatter-within matrices are summed up. There are similarities between the covariance matrix from Equation (5) and each commensurately sized scatter-within matrix, whereas the latter does not include the scaling factor. The scatter-between matrix, on the other hand, is a metric for the distance between the classes' means $\bar{\mathbf{x}}_i$ and the overall mean $\bar{\mathbf{x}}$, which should be maximized after the transformation. The number of samples n_i in Equation (9) is used as a weighting factor. A slightly different eigenvalue problem

$$\mathbf{S}_{\mathrm{w}}^{-1}\mathbf{S}_{\mathrm{b}}\mathbf{V} = \lambda\mathbf{I}\mathbf{V} \tag{10}$$

from that with the PCA is then solved for the matrix product of the inverse scatter-within and scatter-between matrices. This ensures a combination of both objectives: minimizing the scattering within each class and maximizing the means' difference.

3. Methodology—Recursive Pruning of Inputs in Neural Networks

The backbone of the proposed FeaSel-Net algorithm is the pruning of irrelevant input nodes to counteract the curse of dimensionality and simplify classification tasks. This is done by extracting the main contributing features for certain decisions. Hereby, the focus is on two major aspects in order to surpass the performance of the state-of-the-art feature selection methods:

(a) using a nonlinear evaluation method and
(b) embedding the FS algorithm into the classifier.

The crucial issue in such embedded approaches lies in the communication of the FS algorithm and the classifier. Inspired by the RFE from [21] and the recent approaches of [28], we make use of recursive and sequential pruning of feature nodes in the input layer. This recursivity is indicated by the loop structure in the process diagram of Figure 1.

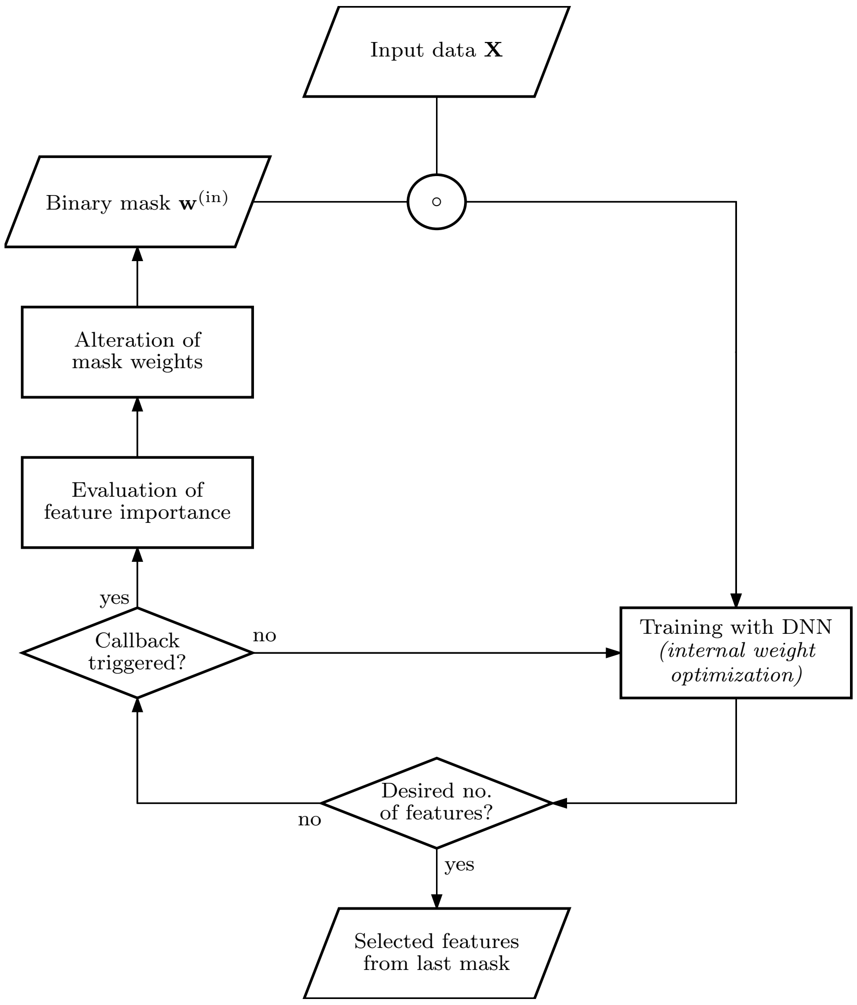

Input data $\mathbf{X}$
Binary mask $\mathbf{w}^{(\text{in})}$
$\circ$
Alteration of mask weights
Evaluation of feature importance
yes
Callback triggered?
no
Training with DNN *(internal weight optimization)*
Desired no. of features?
no
yes
Selected features from last mask

Figure 1. Process diagram of the FS callback. The $\circ$-operator describes the Hadamard product.

At the start of the algorithm, the complete dataset $\mathbf{X} \in \mathbb{R}^{p \times n}$ with p features and n samples is considered and transferred to the classifier input unmanipulated, i.e., the binary mask $\mathbf{w}_{\text{in}} \in \mathbb{R}^{p \times 1} = \mathbf{1}$. We use a deep neural network (DNN) as a classifier due to its inherent nonlinear properties. Its training process is executed within the lower loop in the process diagram. When the performance of the classifier is satisfyingly reliable and the callback is triggered, the algorithm exits the lower loop and enters the upper evaluation part. In this part, the importance of each feature is evaluated and a distinct proportion of the most informative features is selected. All of the others are pruned and neglected in ensuing optimization loops. This is done by altering the weights of the binary mask $\mathbf{w}^{(\text{in})}$.

Subsequently, the DNN has to adapt to the increased difficulty of using sparser information. Everything is recursively repeated until either the desired number of features has been obtained or the classification accuracy drops beneath a given threshold and is unable to recover despite ongoing optimization.

The output of the algorithm is the last mask evaluated during the training, and a classifier is already pre-trained for the masked input. The outline and implementation of the proposed algorithm are split into the following two components: the *classifier* and the *feature selection algorithm*.

3.1. Classification with Deep Neural Networks

To prove the functionality in a general sense, we implemented a standard DNN consisting of one input l_{in}, one output l_{out}, and multiple hidden layers l_i. For the standalone classifier, only fully connected (FC, i.e., dense) layers were applied. In the forward pass of FC-type architectures, the output state vector

$$\mathbf{x}^{(l)} = \phi\left(\mathbf{W}\mathbf{x}^{(l-1)} + \mathbf{b}\right) = \phi(\mathbf{u}) \tag{11}$$

is calculated by multiplying the previous layer's state vector $\mathbf{x}^{(l-1)}$ with a randomly initiated weight matrix $\mathbf{W}$ and then adding a bias vector $\mathbf{b}$. This bias vector is implemented to enable even more flexibility by shifting the input data. The weights and biases are trainable parameters. Afterwards, an activation function ϕ is applied on the resulting $\mathbf{u}$ vector. As previously mentioned, this activation function is what makes the neural network a nonlinear classifier and provides advantages compared with linear transformations. An application of any arbitrary function is possible. The only restriction is the piece-wise differentiability of the function such that the back-propagation [29] algorithm is exercisable. We made use of the rectified linear unit (ReLU) function

$$\phi(\mathbf{u}) = \max(\mathbf{0}, \mathbf{u}), \tag{12}$$

which is a commonly used activation function that has been shown to deliver good results in fully connected architectures. Other typically used functions are tanh or sigmoid functions. The sigmoid function together with ReLU can be seen in Figure 2b,c. The linear function in Figure 2a represents a pseudo-activation within PCAs or LDAs after the transformation induced by the loadings in Equation (3) instead of the weight matrix $\mathbf{W}$.

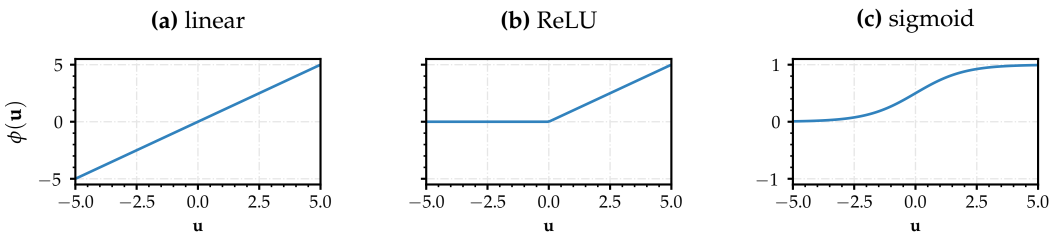


Figure 2. Different activation functions with a linear function (**a**) for comparison with the LDA. Nonlinearity is induced by ReLU (**b**) and sigmoid (**c**) functions as examples.

In our proposed models, we use a feed-forward structure in which the number of nodes $n^{(l)}$ in each layer decreases as we go deeper into the neural network. Since FeaSel-Net is embedded into classifiers, the number of output nodes $n^{(\text{out})}$ has to correspond to the number of classes n_c. Unlike in the intermediate layers, the activation function used to compute the class prediction in the output layer l_{out} is the softmax function

$$\mathbf{y} = \phi(\mathbf{u}) = \frac{e^{\mathbf{u}}}{\sum_{i=1}^{n_c} e^{\mathbf{u}}} \tag{13}$$

which causes the output vector $\mathbf{y} \in \mathbb{R}^{n_c}$ to resemble probabilities. The output with the highest probability represents the predicted class. To train the network, we create the ground-truth target vector $\hat{\mathbf{y}}$ via one-hot encoding and use the sparse categorical cross-entropy (CE) loss function

$$H(\mathbf{X}, \mathbf{y}) = -\sum_{\mathbf{x} \in \mathbf{X}} \hat{\mathbf{y}}(\mathbf{x}) \cdot \log \mathbf{y}(\mathbf{x}). \tag{14}$$

This loss is minimized by using the Adam optimizer [30].

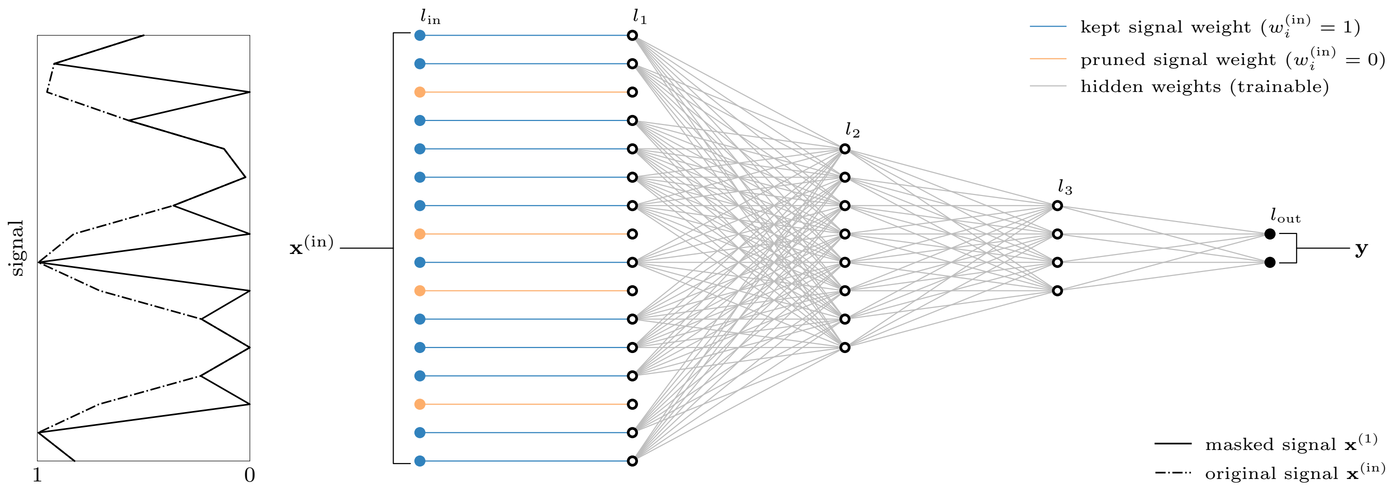
Achieving an embedded feature selection algorithm is a rather challenging task, since it is not possible to alter the network architecture during the training process. Once the network is instantiated, its numbers of inputs, parameters, and layers are fixed; however, it is necessary to manipulate these to prune the input layer. To do so, we implemented a new embedded feature selection algorithm in the existing Keras and TensorFlow framework.

3.2. Feature Selection Callback

The communication behavior of the FS algorithm and the neural network is induced by implementing an appropriate and specifically constructed callback within the model. Usually, callbacks are used to log evaluation metrics, such as loss and accuracy values, or to preclude early stopping. In general, they provide the possibility to interact with the deep learning algorithm and adjust several parameters during the training at different entry points, such as at the end of an epoch or batch. The callback developed in this paper has the ability to assess the importance of input nodes and to prune nodes that are irrelevant by manipulating the weights of an upstream mask layer. The individual steps of the callback are explained in more detail below.

3.2.1. Implementation of the Callback Using Binarized Masking Layers

Section 3.1 described a standard fully connected multi-layer architecture for classification tasks, which we slightly adapted to attain the ability to mask the original signal according to Figure 1. This adapted architecture is shown in Figure 3.



signal
1
0
$\mathbf{x}^{(\text{in})}$
l_{in}
l_1
l_2
l_3
l_{out}
$\mathbf{y}$
kept signal weight ($w_i^{(\text{in})} = 1$)
pruned signal weight ($w_i^{(\text{in})} = 0$)
hidden weights (trainable)
masked signal $\mathbf{x}^{(1)}$
original signal $\mathbf{x}^{(\text{in})}$

Figure 3. The *LinearPass* layer (l_{in}) precedes a conventional FC architecture with several hidden layers (l_{1-3}) and induces masking ability. Straight connections between the first two layers indicate the weight vector $\mathbf{w}$ of this layer. The sinusoidal signal in this example is incomplete, since some of the features have already been pruned.

We introduce a new and simplistic but effective layer that is able to constrain the input signal with a binarized weight vector $\mathbf{w}^{(\mathrm{in})} \in [0,1]p \times 1$. In our implementation, the layer type is called the *LinearPass* layer. Its output

$$\mathbf{x}^{(1)} = \mathbf{w}^{(\mathrm{in})} \circ \mathbf{x}^{(\mathrm{in})} \tag{15}$$

is the masked input for the actual neural network.

We deliberately do not want any parameters to be trained and initially set all weights $\mathbf{w}^{(\mathrm{in})} = \mathbf{1}$ to obtain an unmasked and unmanipulated signal. The connections in Figure 3 will be set to zero if the corresponding feature is not found to be important. This happens whenever the callback is triggered. Manipulations of the bias vectors are not provided.

3.2.2. Callback Triggers

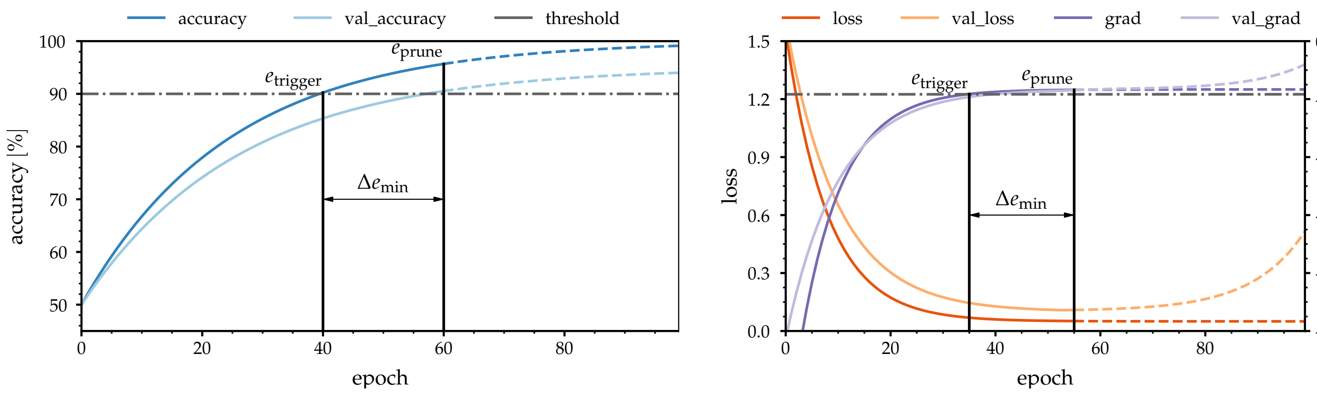
Standard callbacks are triggered every training epoch, and they log the loss and accuracy values for the training dataset ${}_{\mathrm{tr}}\mathbf{X}$ and validation dataset ${}_{\mathrm{v}}\mathbf{X}$. We utilize these recurring logged values to assess the performance of our model at each epoch e and query two trigger conditions:

(a) Threshold criterion:
The loss gradient or accuracy values surpass a pre-defined threshold τ.

(b) Consistency criterion:
The threshold is surpassed for a minimal number of consecutive epochs Δe_{min}.

When the logged values reach the threshold τ, an internal counter starts. Only when the threshold criterion is continuously satisfied for Δe_{min} are the features pruned according to their importance. The evaluation itself is described in Sections 3.2.3 and 3.2.4.

The trigger process for the accuracy-based feature selection is shown in Figure 4a. At epoch $e = 40$, when the accuracy threshold value of $\tau_{\mathrm{a}} = 90\,\%$ is surpassed for the first time, the pruning process starts with a delay of 20 epochs at $e = 60$. Assuming that the accuracy value decreases and falls below the threshold again, the count for the consistency criterion is reset. Figure 4b analogously illustrates the loss-based trigger behavior.



(**a**) accuracy-based (**b**) loss-based

Figure 4. Implemented triggers for the feature selection callback applied in a fictional optimization process. The dashed lines indicate the hypothetical trend of each accuracy value after the pruning; (**a**) visualization of the accuracy-based case and (**b**) the loss-based case.

Since different datasets and metrics rarely provide similar quantitative results, the algorithm utilizes the gradient of the current loss values. Here, the pruning process is triggered when the decline of the loss becomes stagnant. The low loss gradient threshold of $\tau_{\mathrm{g}} = 0.005$ yields a pruning precisely at the moment of training stagnation. Thereby, we suspect that pruning at the moment of optimization stagnation will prevent over-fitting of the data. Hence, a potential increase in the validation loss, as indicated by the dashed line, is avoided.

3.2.3. Creating an Evaluation Subset

At first, an evaluation dataset ${}_{\mathrm{e}}\mathbf{X}$ has to be generated, and it is appropriated for an isolated view of each feature node. To do so, we make use of the leave-one-out cross-validation (LOOCV) [2], an extreme variation of the k-fold cross-validation where $k = 1$. The typical usage of the cross-validation is the k-time alternation of training and validation samples and the choice of the best-performing composition. Figure 5a shows this type of validation exemplarily for a small fictional dataset. Since we are interested in the importance of features rather than samples, we implement the LOOCV in combination with the disposal of one feature alternating at each iteration step; see Figure 5b.

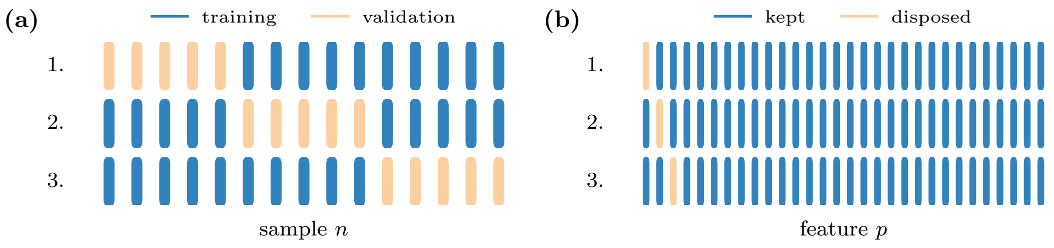


Figure 5. The first three iterations in different cross-validation types applied on a 15 × 30 dataset: (**a**) k-fold cross-validation with $k = 3$ and (**b**) LOOCV with $k = 1$.

Mathematically, this mask-like behavior can be expressed using a bit-wise inverted identity matrix:

$$\mathbf{M} = \mathbf{J} - \mathbf{I} \tag{16}$$

where $\mathbf{I}$ is an identity matrix and $\mathbf{J}$ is a matrix of ones with the size $p \times p$. We can now apply a mapping function $g : \mathbb{R}^{p\times n} \mapsto \mathbb{R}^{p\times n\cdot p}$ to generate a p-time replication of the same training data ${}_{\mathrm{tr}}\mathbf{X}$, where exactly one feature is masked in each replication. The resulting evaluation data,

$${}_{\mathrm{e}}\mathbf{X} = \begin{bmatrix} {}_{\mathrm{tr}}x_{00}m_{00} & {}_{\mathrm{tr}}x_{01}m_{01} & \cdots & {}_{\mathrm{tr}}x_{0p}m_{0p} \\ {}_{\mathrm{tr}}x_{10}m_{00} & {}_{\mathrm{tr}}x_{11}m_{01} & \cdots & {}_{\mathrm{tr}}x_{1p}m_{0p} \\ \vdots & \vdots & \ddots & \vdots \\ {}_{\mathrm{tr}}x_{n0}m_{00} & {}_{\mathrm{tr}}x_{n1}m_{01} & \cdots & {}_{\mathrm{tr}}x_{np}m_{0p} \\ \vdots & \vdots & & \vdots \\ {}_{\mathrm{tr}}x_{00}m_{p0} & {}_{\mathrm{tr}}x_{01}m_{p1} & \cdots & {}_{\mathrm{tr}}x_{0p}m_{pp} \\ {}_{\mathrm{tr}}x_{10}m_{p0} & {}_{\mathrm{tr}}x_{11}m_{p1} & \cdots & {}_{\mathrm{tr}}x_{1p}m_{pp} \\ \vdots & \vdots & \ddots & \vdots \\ {}_{\mathrm{tr}}x_{n0}m_{p0} & {}_{\mathrm{tr}}x_{n1}m_{p1} & \cdots & {}_{\mathrm{tr}}x_{np}m_{pp} \end{bmatrix}, \tag{17}$$

are then tested, and the impact of the masked features is evaluated with respect to the test loss. In case a vast number of samples and features leads to an enormous amount of data, we provide a possibility to use only a subset of the training data as evaluation data, which would have an equal size for all classes. To further accelerate the evaluation process, already masked features are not regarded in the mapping of Equation (16), and each of its new rows

$$\mathbf{m}_i := \mathbf{m}_i \cdot w_i^{(\mathrm{in})} \tag{18}$$

is deleted if $w_i^{(\mathrm{in})}$ has already been set to zero. To prevent buffer overflows in huge datasets, especially at the beginning of the algorithm when $\mathbf{M}$ is still complete, the mapped data are split into batches ${}_{\mathrm{e}}\mathbf{X}_f$, where each batch represents one feature f.

3.2.4. Evaluation of Feature Importance

Within the callback, the training process is paused and the significance of the masked features is evaluated. Therefore, the same loss function as during the training process

(Equation (14)) is also used for the evaluation, but in contrast to the training, the losses are not averaged over the complete epoch or its batches, but rather over each feature masking the dataset. This is done because the interest lies in the deterioration of discriminability due to these missing features. Alternatively, one can look at this behavior as dividing the set into one batch per feature, where the losses of each batch are averaged. These considerations yield another new metric based on Equation (14), which we call feature omission impact (FOI):

$$\mathcal{I}_f\left({}_{\mathrm{e}}\mathbf{X}_f, \mathbf{y}\right) = H_f\left({}_{\mathrm{e}}\mathbf{X}_f, \mathbf{y}\right). \tag{19}$$

At this point, it has to be clarified that previously pruned features are not evaluated by not mapping them in Equation (17), because they inherently cannot provide any information to the classifier.

3.2.5. Feature Pruning

Since the negative influence of masked input nodes on the classification performance is evaluated, the view for interpreting the resulting entropy values has to be changed. While the entropy should normally be minimized to achieve unambiguous predictions, we now keep the features f that resulted in the highest FOI and the biggest differences in the results of unmasked prediction. The weights

$$\mathbf{w}_i^{(\mathrm{in})}[\mathrm{argsort}(\boldsymbol{\mathcal{I}})] = 0 \qquad \forall\, 0 < i \leq n_{\mathrm{p}} \in \mathbb{R} \tag{20}$$

of the masking *LinearPass* layer are manipulated by sorting the features according to their information richness from the lowest to the highest and setting the least informative features' weights to zero. The pruning number n_{p} defines how many entries are pruned. Some datasets have many features, and pruning one feature after another is tremendously time-consuming. Thus, the algorithm offers two possibilities for setting the pruning number. It is either set once for the linear pruning method or it is constantly re-calculated for the exponential decrease in information depending on the pruning rate π. After the pruning, the optimization process is resumed with the masked input ${}_{\mathrm{e}}\mathbf{X}(\mathbf{w}^{(\mathrm{in})})$. As soon as the consistency and threshold criteria are met again, the recursive binary mask is obtained using the adapted evaluation set ${}_{\mathrm{e}}\mathbf{X}(\mathbf{w}^{(\mathrm{in})})$ as the input for the next pruning step in Equation (19).

The sorting algorithm for the the indices i in $\mathbf{w}^{(\mathrm{in})}$ is defined by

$$\mathrm{argsort}(\mathbf{x})_i := |\{j \in \{1, \ldots, n\} | (x_j < x_i)\}|, \tag{21}$$

and it sorts the features from the lowest to the highest $\mathcal{I}_f$.

Eventually, the feature selection process is finished when one of the following criteria is met:

(a) Success criterion:
The number of leftover features n_{e} reaches the desired number of features q.

(b) Non-convergence criterion:
The threshold is not reached within a given number of epochs $\Delta e_{\max}$.

4. Results

We apply the proposed method on the *Wine Classification* dataset provided by [22] to demonstrate the performance of the FeaSel algorithm and compare it with the linear methods described in Section 2, as well as the XGBoost algorithm and the nonlinear stochastic gates (STG) method. This multivariate dataset is broadly used in ML research and covers LDA and PCA investigations [31,32], as well as fully connected neural network approaches [33]. Despite being multivariate, it is still small enough to gain an overview on what happens during the algorithm's execution. The dataset consists of $n_{\mathrm{s}} = 178$ samples divided into $n_{\mathrm{c}} = 3$ classes of almost equal set size. The original number of features is $p = 13$. Since the feature values are partially of greatly different magnitudes (e.g., *alcohol* compared to *proline*), a standardization of the values for each feature according to

Equation (6) is necessary. Additionally, we assure the correct dimensionality for a fully connected neural network and use a training–testing split of 80%, with which we obtain the dimensionalities given in Table 1. These subsets are used for either of our FS methods. The evaluation dataset dimensionality is a multiplication of the training dataset with the trace of the mask matrix $\mathrm{tr}(\mathbf{M})$. A consistent dataset for all methods is obtained by using the same random seed for all training–testing splits.

Table 1. Dataset size, dimensionality, and number of samples per class.

Dataset (i)	Data $_{i}\mathbf{X}$	Targets $_{i}\mathbf{y}$	Samples per Class
Training (tr)	[142, 13]	[142, 3]	[45–57–40]
Validation (v)	[36, 13]	[36, 3]	[14–14–8]
Evaluation (e)	$\mathrm{tr}(\mathbf{M}) \cdot {}_{\mathrm{tr}}\mathbf{X}$	$\mathrm{tr}(\mathbf{M}) \cdot {}_{\mathrm{tr}}\mathbf{y}$	
Test (te)	[178, 13]	[178, 3]	[59–71–48]

4.1. Feature Selection with PCA

The PCA loadings for the training subset are calculated according to the steps described in Section 2.1. Figure 6 shows the scree plot for the first eight components of PCA for the *Wine Classification* dataset. Since a distinct elbow—an elbow is a subjective criterion for the decision on whether to include components in the transformation and is characterized by a strong kink in the scree plot [34]—was not observed in the plot, the average explained variance with 7.7% was chosen as the threshold. Components with a variance lower than this threshold were not regarded. Thus, we narrowed down the transformation to the first three components, which made up 66.0% of our data's information.

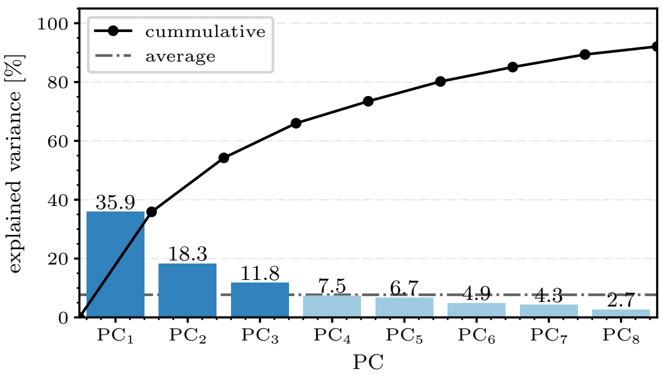
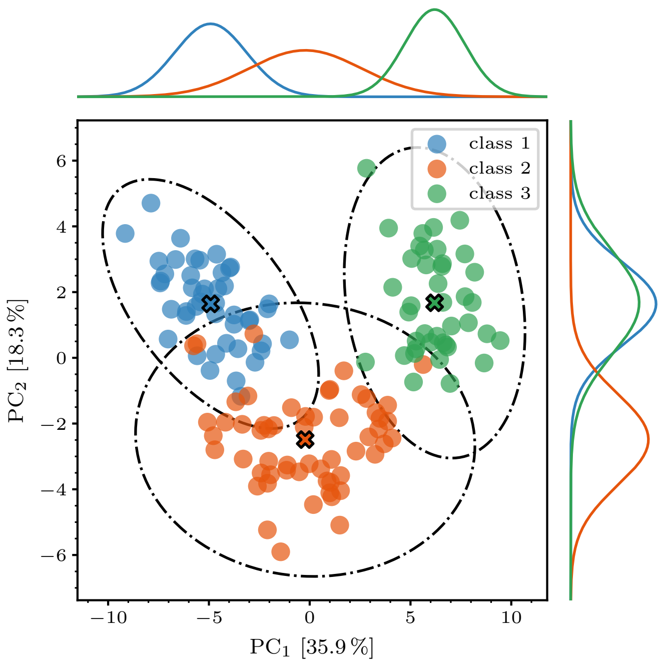
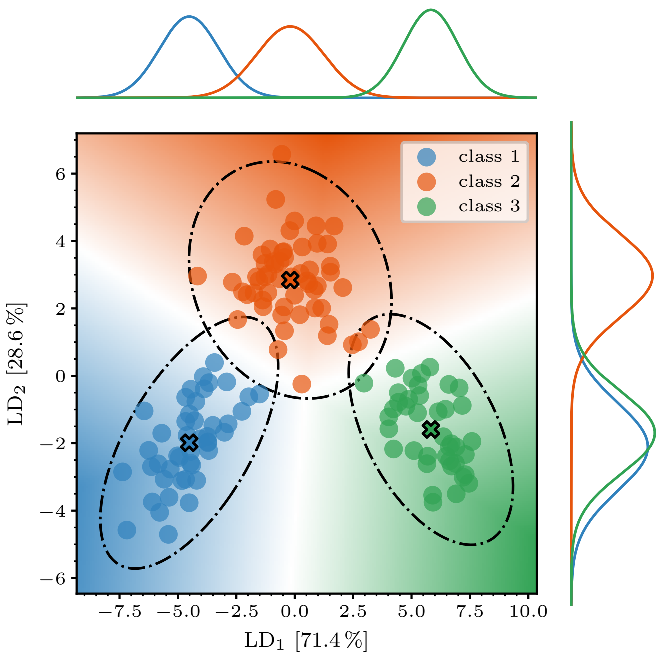


Figure 6. Scree plot from the PCA of the *Wine Classification* dataset. The average explained variance is surpassed after considering the first three components.

The resulting score in Figure 7a looks promising in terms of separating the classes when considering the latent space defined by the first two components only. Although there are a few intersections and overlaps, we can clearly observe a class separation. This capability breaks down when considering the third component in combination with one of those mentioned before. Hence, we confine ourselves to the interpretation of the loadings in the first two components. Figure 8a shows the contribution of each feature to the specific PC. The evaluation and, thus, the sorting of the feature importance is done according to [24], where the feature contribution is calculated by applying Equation (7) on the q most important eigenvectors $\mathbf{V}$ along the feature axis. Since we want to include the components' explained variance in our consideration, we use the scaled loadings $\mathbf{A}$ instead. For the examined dataset, we set $q = 2$ due to the previous considerations. The features are distilled by choosing the first three features in Figure 8a (*proline*, *hue*, and *color intensity*) as the most prominent, but there is only a modest difference from their successors.

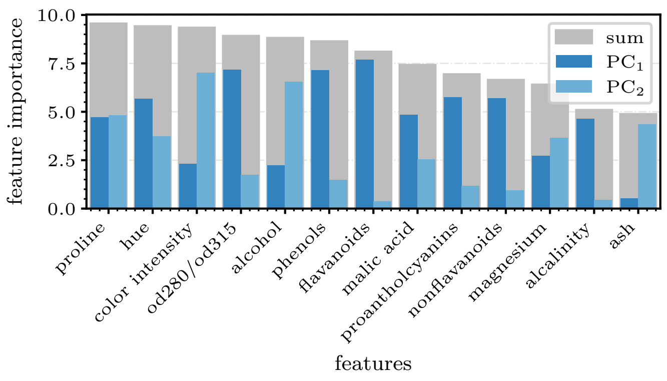


(**a**) PCA score plot

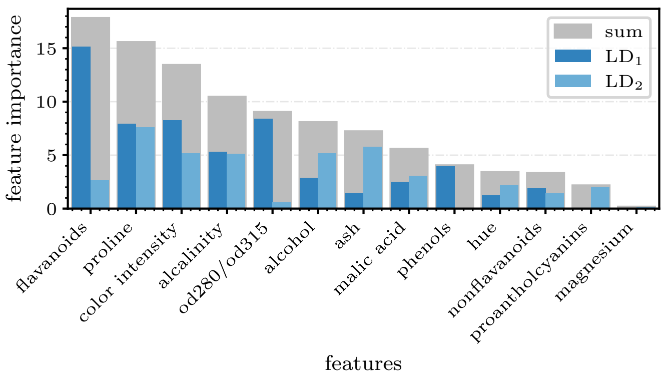


(**b**) LDA score plot

Figure 7. PCA (**a**) and LDA (**b**) score plots for the first two components of the *Wine Classification* dataset. The dashed ellipses are 3σ confidence intervals around the classes' scattering centers and are projected onto each axis at the right and top side of the plot. The colored background in (**b**) represents the probabilities of a sample belonging to a specific class.



(**a**) PCA contribution plot



(**b**) LDA contribution plot

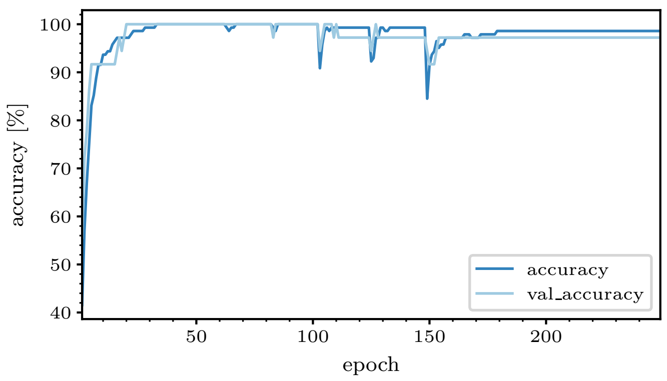
Figure 8. Contribution plots for the first two components of the *Wine Classification* dataset. The bar heights for the PCs (**a**) and linear discriminants (LDs) (**b**) do not represent the contribution in percent, but the proportion of the sum. In a descending manner, the most important features are displayed on the left side of the plots.

4.2. Feature Selection with LDA

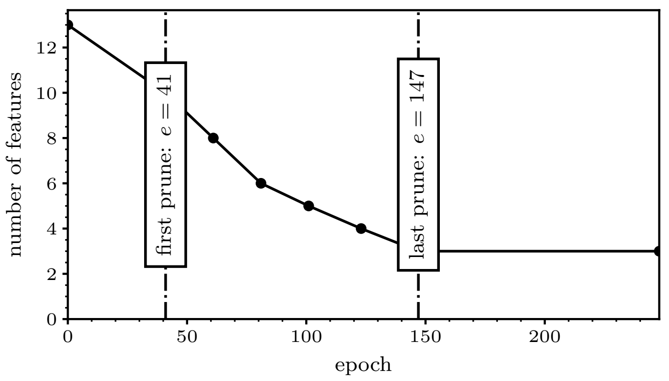
The same research group from [24] also showed the applicability of this feature evaluation method with LDA [35]. We have already discussed the additional class information that this transformation type offers and can clearly see an improvement in the separation in Figure 7b. It is even possible to completely separate the clusters for the training set. When looking at the decision boundaries in this score plot, there is no training sample point that has been misclassified. Furthermore, the projected normal distribution of the orange *class* 2 cluster clearly shows the influence of the LDA objective—the transformation wants to minimize the variances within the classes. Although the distribution in the score plot of Figure 7a and, especially, the projection of PC_1 is flat and wide, it is almost circular and narrower in the LDA score plot; see Figure 7b. Because of this distinctiveness and the fact that PC_1 and PC_2 together explain 100% of the data, we are allowed to investigate only these components again. Just like before, the most important features are extracted from the contribution plot in Figure 8b, where a concise difference in the results can be seen. With the help of LDA, it is found that *flavanoids* seem to be much more relevant than *hue*, which had the third lowest contribution, unlike in the PCA, where it achieved the second highest. Nevertheless, there is a consensus on the importance of *proline* and *color intensity*. Another conspicuity is the noticeably increased variance in $\sigma_{\text{LDA}} = 5.18\%$ in the contribution compared to that for PCA ($\sigma_{\text{PCA}} = 1.54\%$). Altogether, these findings lead to an unambiguous tendency in which LDA will perform better in terms of feature selection.

4.3. Feature Selection Using Feasel-Net

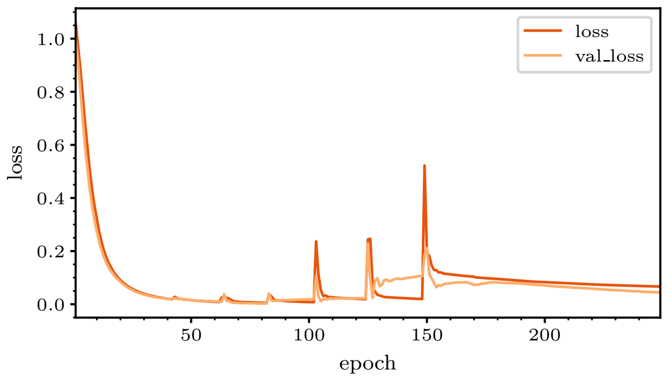
When using the same data with the newly proposed FS algorithm, we can clearly observe the recursiveness throughout the model's optimization process, as shown in Figure 9a,c. After 21 epochs, the accuracy value surpasses the threshold for the first time, and $\Delta e = 20$ epochs later, we can observe the first pruning, where *magnesium*, *phenols*, and *nonflavanoids* have been deleted. A drop in the classification accuracy cannot be perceived at either $e = 41$ or 20 epochs later, which is when the second pruning happens, with *malic acid* and *proantholcyanins* being deleted. The FOIs at these first two pruning steps are shown in Figure 10a,b. Since the training is continued with quasi pre-trained weights after each pruning, the model recovers quickly. The most prominent drop is observed at the last pruning epoch at $e = 147$, which is self-evident due to the further decreasing amount of information. When reaching $q = 3$ parameters, the algorithm stops the training process after $\Delta e_{\text{max}} = 100$ more epochs to optimize the discrimination one last time with the selected features. Outstandingly high values of 99.1% were achieved for the training accuracy and 97.6% for the validation accuracy, even though the input nodes were reduced to these three aforementioned features and the information was compressed to $\kappa = 23.1\%$. The exact number of features at each pruning epoch can be retrieved from Figure 9b. By the end of the FeaSel-Net application, the three selected features were *flavanoids*, *proline*, and *alcohol*. Figure 9d shows the pruning history in terms of how often a feature was masked throughout the FS process. Features such as *magnesium* were considered to be the least important and were masked since the very first pruning step, whereas *hue* showed the darkest color and was, hence, the most important apart from the chosen features. This coincides with the findings from the PCA and LDA methods. The confusion matrix in Figure 11a shows perfect sensitivity for *class* 1 and *class* 3 and is satisfactory for the other class. Altogether, an average sensitivity of SEN = 98.6% was achieved. In terms of classification accuracy and specificity, the results were of an even higher magnitude, with ACC = 98.8% and SPE = 99.2%.



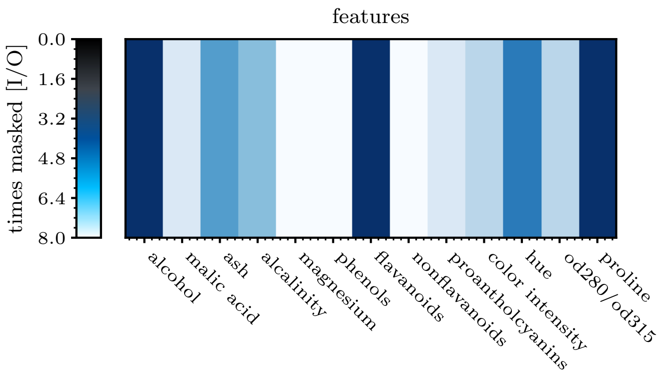
(**a**) accuracy



(**b**) pruning

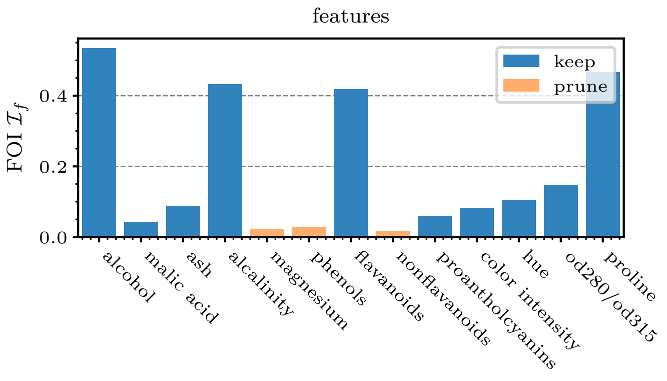


(**c**) loss

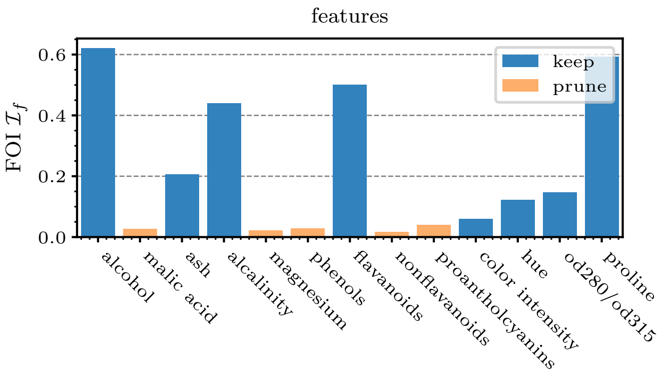


(**d**) masking

Figure 9. Feature selection callback histories of a single FeaSel-Net run on the *Wine Classification* dataset. The monitored accuracy (**a**) and the corresponding loss (**c**) are shown on the left. A characteristic sawtooth pattern can be observed. The threshold for the pruning is set to $\tau_{acc} = 0.98$ and the desired number of features is $q = 3$. An exponential pruning with a rate of $\pi = 0.2$ for the decrease in the number of features is chosen. An overview on the decreasing input information (i.e., pruning of features); (**b**) and the development of the masking layer (**d**) is given on the right. Chosen features can be retrieved from the dark blue bars in the masking history. Darker bars represent features that are pruned late in the recursive process and, thus, tend to hold more information.



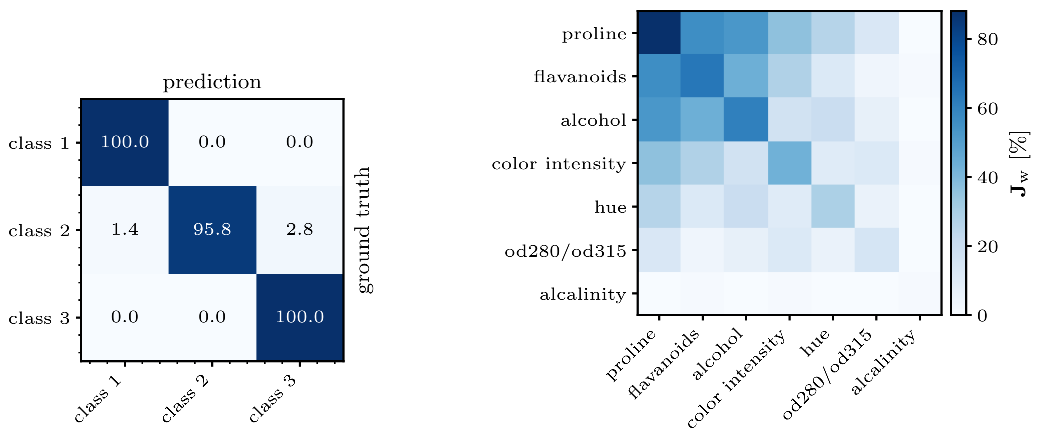
(**a**) first prune



(**b**) second prune

Figure 10. An exemplary feature evaluation based on the min–max scaled-average cross-entropy at different stages in the callback. The bars in (**a**) show the feature omission impact (FOI) in the first pruning stage, whereas the second prune is depicted in (**b**). Orange bars represent features that are or have already been pruned at each specific step.

Since the weights of fully connected layers in neural networks are randomly initialized, volatility in the extracted features has to be expected, e.g., by using the uniformly distributed values according to Glorot [36]. Therefore, 100 executions or runs n_r of FeaSel applications are statistically evaluated to prove the consistency and the extent of this fluctuation. In order to assess the probability of finding the most relevant features and to analyze the inter-dependencies among them, a weighted Jaccard matrix $\mathbf{J}_\mathrm{w}$ is introduced in Appendix A.1. The diagonal in Figure 11b can be interpreted as a normalized histogram for the selected features. Furthermore, the matrix directly implies that the features retrieved from the elaborated execution discussed before (*proline*, *flavanoids*, and *alcohol*) were also chosen the most often, with percentages of 87.7–60.8% of all runs. It additionally shows that these three features were commonly picked together, and in the case that some other feature was picked, it was most likely selected together with the *prolines*, which emphasized the importance of this attribute. *Alkalinity* was the last feature to be displayed, since every other feature was not chosen at all. Hence, these can be neglected in good conscience. The other features that were determined by linear transformations (*color intensity* and *hue*) occurred in the fourth and fifth positions, respectively.



(**a**) confusion matrix (**b**) weighted Jaccard matrix

Figure 11. Different evaluations of the effectiveness of FeaSel-Net. A confusion matrix (**a**) for a test set $_\mathrm{te}\mathbf{X}$ consisting of training and validation samples with only three features shows how good the classification result of a single selection is, whereas the weighted Jaccard matrix (**b**) shows the inter-dependencies of the features in 100 FS runs.

4.4. Comparison of Different Feature Selection Methods

All three previously analyzed methods yielded slightly different sets of features that were distilled after the selection. To validate the performance of each selection method, a classification using only the masked dataset was executed. We also compared it to the features chosen with the tree-based XGBoost algorithm [13] and a recent approach with the so-called stochastic gates (STG), where the FS was already embedded as a regularizer into a deep-learning-based classifier. The regularizing hyperparameter λ had an immense effect on the number of leftover features. It was set to 0.3 such that an average number of 2.9 selected features was achieved. Further information on this method can be found in [17]. Since the STGs was embedded into neural networks, there was much randomness. Similarly to the FeaSel-Net approach, we applied it several times and calculated its weighted Jaccard matrix. The certainty of selections using STGs was found to be lower than FeaSel-Net's, which is shown in Appendix A.3. Additionally, a random FS with *alcohol*, *alkalinity*, and *color intensity* is included in Table 2 to demonstrate the importance of choosing features accurately. Even though only one (*alkalinity*) out of these three randomly selected features was not found to be important by the investigated FS methods, it still performed clearly worse.

Table 2. Selected features from all analyzed FS methods.

Method	Feature 1	Feature 2	Feature 3
PCA	proline	hue	color intensity
LDA	flavanoids	proline	color intensity
XGBoost	flavanoids	proline	color intensity
STG	od280/od315	proline	flavanoids
random	alcohol	alkalinity	color intensity
FeaSel-Net	proline	flavanoids	alcohol

In order to compare the effectiveness of the feature selection methods, another fully connected neural network was implemented that accepted only three input values (i.e., the masked signal). We purposefully chose only three features to ensure variations among each selected feature set on the one hand, as well as, on the other, to use as little information as possible in order to clearly show differences between each FS method. It output a probability vector corresponding to the three classes; see Table A3. A fair comparison of the discriminability was ensured by using the exact same model with the same hyper-parameters for all methods. The only variation was given by the input signal, which was chosen according to Table 2. The model was, parameter-wise, distinctly smaller than the classifier network used in Section 4.3. It was trained 25 times, and the average and standard deviation for accuracy (ACC), sensitivity (SEN), and specificity (SPE) were calculated. Outliers from PCA, LDA, and XGBoost optimizations were removed in the evaluation. Figure 12 shows the performance parameters for each retrieved feature mask. The overall accuracy of the classification using the features from FeaSel-Net amounted to ACC $= 98.50 \pm 0.43\%$, which was an increase of 0.6 % compared to the second-best result (LDA and XGBoost) and even 1.8% higher than the PCA benchmark, whereas SPE $= 98.91 \pm 0.44\%$ also improved at the same level. When looking at SEN $= 98.0 \pm 1.04\%$, an increase of 1.1% compared to the LDA and 3.2% compared to PCA method was achieved. Using the features that were retrieved by using the STG, the results were only slightly better than the PCA-derived selection. On average, the superior FeaSel-Net approach had a 6.8% higher accuracy, a distinctly improved sensitivity with 11.5%, and a 5.0% better specificity than that obtained when using randomly chosen features.

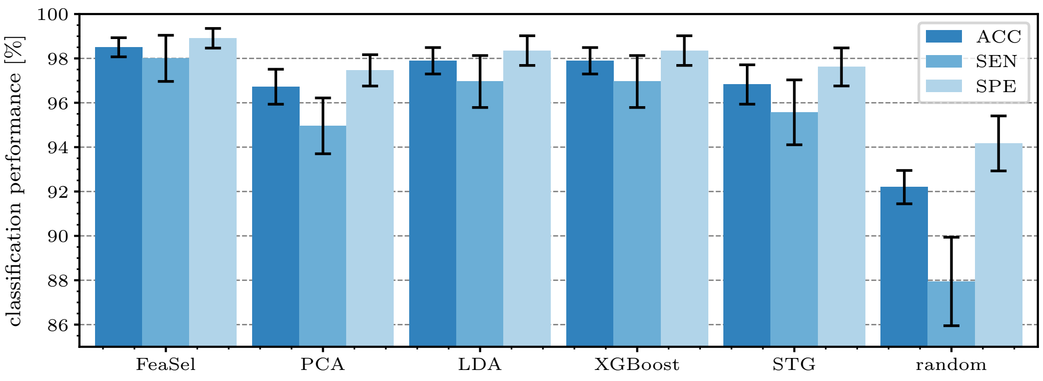


Figure 12. Discriminability in classifications with a subset of the original features according to the findings in Table 2. Each subset has the size $p = 3$. The classifier and parameters are identical for all methods.

4.5. Generalizability of Masked Data

FeaSel-Net was specifically developed to use less input information and still provide reasonable classification results without dramatic drops in the performance. To investigate the improvements due to the preceding feature selection, we compared the results of training histories from 25 unmasked and masked optimizations of the *Wine Classification* dataset over 250 epochs, as shown in Figure 13. The network used for the masked data was the same reduced classifier as in Section 4.4, and for the unmasked set, an architecture with a similar number of trainable parameters and network depth was used to provide the same optimization potential; see Table A3. Two outlier runs were removed from the unmasked and one from the masked dataset. FeaSel-Net's average accuracy after 250 epochs of training with only three input features accounted for $\text{ACC}_{\text{Wine}} = 98.1\%$ compared to 99.4% with the unmasked set. On the other hand, its validation accuracy's variance averaged over all epochs $e > 50$ amounted to $\pm 1.12\%$ and was more than eight times smaller than the unmasked averaged variance of $\pm 8.91\%$. An average validation accuracy of more than 95.0% was achieved in epoch $e = 24$ already, and the maximum value accounted for 97.8 %. The unmasked dataset unsurprisingly achieved better overall classification results with 99.2% instead, but it had to be trained 40 epochs longer until 95.0% was reached. We could not identify an over-fitting during training in either dataset, since a small number of parameters and, thus, a modest complexity were used for the model. When comparing these results, it can be seen that a significantly faster convergence was obtained, and the variance was significantly smaller when masking the data according to the FeaSel-Net findings beforehand. Both observations are indicators for a better generalizability. The smaller variance in the masked optimizations proves a steady and reliable finding of the global minimum.

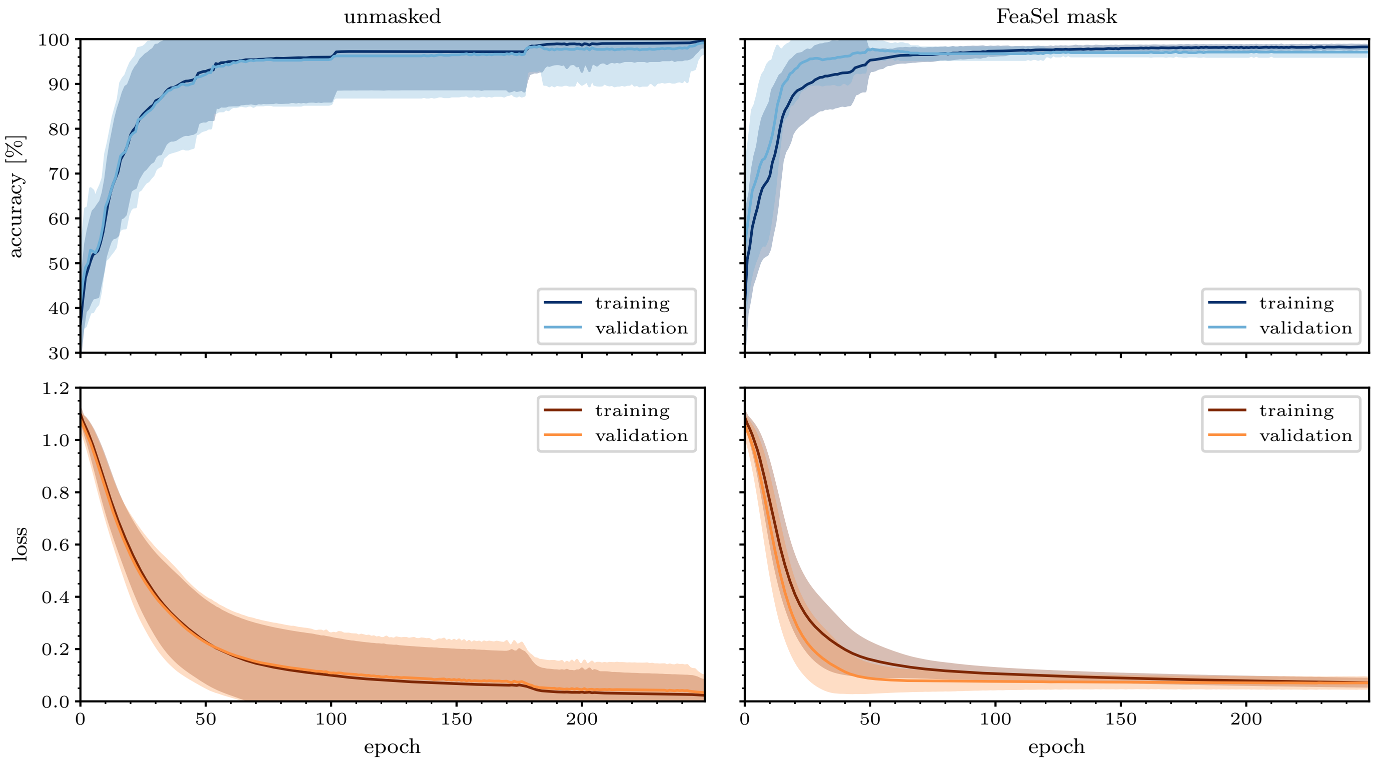


Figure 13. Comparison between the unmasked and masked *Wine Classification* dataset. The mask consists of the three most important features according to FeaSel-Net (*proline*, *flavanoids*, and *alcohol*). Darker lines indicate averages and colored, transparent areas represent the standard deviation of 25 optimization processes.

4.6. Investigation of Different Datasets

In this section, an analysis of other datasets from [22] that confirm the potential of FeaSel-Net as an alternative for the feature selection of 1D data is undertaken . With the (feature-wise) very small *Iris* [37], medium-sized *Mice Protein Expression* (*MPE*) [38], and extremely large *Arcene* datasets [39], we want to tackle data attributed with different levels of complexity. The parameters specified for each FeaSel-Net run in the following are based on the settings for the Wine Classification set; see Table A2. Changes in the parameters will be mentioned explicitly.

4.6.1. Iris

The *Iris* dataset consists of samples with only four attributes and is separated into three classes. According to Fisher's findings in [37], the attributes of *petal length* and *petal width* contribute the most information to a discrimination between the three classes. In his work, he originally introduced an earlier version of the LDA and found that the petal attributes' coefficients for the transformation were roughly 1.5–4.5 times higher and, thus, more important than the sepal attributes. With the FeaSel-Net method, we found the exact same attributes to be more relevant. The *petal length* was chosen in 79.4% and *petal width* in 72.5% of all runs; see Figure A1a. Because of the minute number of attributes, a desired number of features of $q = 2$ was specified, with which a compression ratio of $\kappa = 50\%$ was achieved.

4.6.2. Mouse Protein Expression

The authors of [38] originally investigated protein levels in mice exposed to contextual fear conditioning. They used self organizing maps (SOM), an unsupervised clustering technique, to identify biologically important differences in medication-induced protein levels of healthy mice and mice with trisomy 21. Feature selection with FeaSel-Net achieved even better results on this medium-sized dataset with $p = 77$ features. In 92% of all cases, the protein *APP* was chosen to be the most important feature followed by *pCAMKII* with 86%. A distinct cluster of the first six proteins can be noticed in Figure A1b. All proteins found in this Jaccard matrix were also found to be important discriminants for generating the SOMs. Since the number of samples was 5–10× larger compared to the numbers in the other datasets, a batch size of 64 instead of 16 was specified. With $q = 3$ set for the algorithm, a compression ratio of $\kappa = 3.9\,\%$ was obtained, and it resulted in a classification accuracy of approximately $\mathrm{ACC}_{\mathrm{MPE}} = 85.4\%$.

4.6.3. Arcene

This particularly large dataset with $p = 10\,\mathrm{k}$ features was part of the NIPS 2003 feature selection challenge. With the application of FeaSel-Net on this dataset, we wanted to focus on the computability of such large datasets. When looking at the results, less distinctive feature importance distributions were obtained in the Jaccard matrix of Figure A1c. While the matrices of the *Iris* and *MPE* sets looked structured and some attributes were considerably more important than others, selection in the *Arcene* dataset yielded sparser and more chaotic results. However, the feature at position 9274 still seemed to be very important, since it was chosen more than every fourth time in a highly compressing setting with $q = 10$ features left and a compression rate of $\kappa = 0.1\%$. In 35 executions, this was more than 250 times as often as that when choosing the features completely randomly. It was also seemingly dominant because it was the only attribute that was chosen in combination with every other attribute depicted in the Jaccard matrix at least once. On average, the classification with the reduced input yielded a $\mathrm{ACC}_{\mathrm{Arcene}} = 94.3\%$ classification accuracy. An overview of all datasets is given in Table 3.

Table 3. Overview of the compression of the input using FeaSel-Net and the resulting classification performance after 250 epochs of training with only three input features.

Dataset $_{\text{te}}X$	**Features p**	**Features q**	**Compress. κ**	**ACC [%]**	**FS ACC [%]**
Iris	4	2	50.0	98.6 ± 0.1	92.7 ± 9.0
Wine Classification	13	3	23.1	99.5 ± 1.7	98.1 ± 0.5
MPE	77	3	3.9	96.4 ± 9.6	87.1 ± 3.8
Arcene	10,000	10	0.1	96.8 ± 0.5	94.3 ± 0.2

4.7. Computation Time

Finally, the computation time of the proposed FeaSel-Net was examined for all of the previously introduced datasets. For this benchmark test, we used cuDNN, a GPU-accelerated library that is usable within the TensorFlow environment, and Accelerated Linear Algebra (XLA) compilation was enabled. The hardware consisted of an RTX 2070 SUPER GPU and an AMD Ryzen 5 2600X CPU. As described in Section 3.2.3, nodes that were pruned prior to the current pruning epoch were not evaluated. This linearly accelerated the evaluation process, as can be seen in Figure 14. In the logarithmically scaled plot, a linear increase in the time with the number of evaluated nodes can be perceived until n_e reached 1000. The kink at this position was due to an acceleration of the evaluation, where the evaluation set was divided into a maximum number of $n_\text{b} = 1000$ batches. For the *Arcene* dataset, the batch size became $n_\text{b} = 10$, i.e., ten adjacent features were evaluated at the same time. This yielded an almost constant duration of approximately 40 s for the evaluation of datasets with $n_\text{e} > 1000$. The zig-zag behavior was induced by ensuring natural numbers as batch sizes and, consequently, varying numbers of batches. When reaching, e.g., $n_\text{e} = 6400$, seven adjacent features were considered simultaneously; therefore, $n_\text{b} = 915$ batches were generated. Looking at smaller numbers of evaluated features (detailed view; $n_\text{e} < 50$), there was a noticeable jump of nearly half a second between the *Arcene* dataset and the *MPE* dataset, which occurred because of the larger neural network architectures during the FS process.

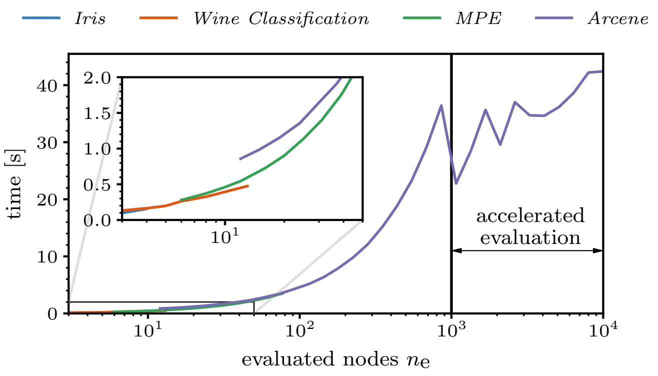


Figure 14. Computation time t for the feature omission impact at different stages in the FeaSel-Net algorithm. The x-axis shows the number of evaluated nodes n_e.

The overall computation time of FeaSel-Net amounted to 16.85 ± 2.04 s when averaged over 25 executions. Compared to the 22.42 ± 0.17 s of the STG method, a slight improvement of the computation time was obtained. This was mainly due to the early stopping criteria implemented in FeaSel-Net. All linear methods need less than a second for their computation. Compared to linear methods, feature selection with neural network approaches generally takes a lot of time. In particular, when the amount of data increases, the difference will be very noticeable. Thus, there is a trade-off between computation time and finding the best features.

5. Conclusions

In this work, we introduced FeaSel-Net, a feature selection algorithm that can easily be embedded in any fully connected neural network classifier. With its novel concept of recursively pruning the information in the input layer of the network, it forces the classifier to constantly adapt to the repeated omission of information such that the discrimination ability of the classifier remains at a supreme level. The evaluation of the feature omission impact is done by applying the leave-one-out cross-validation along the feature axis and assessing the impact of the missing feature on the classification result.

By comparing the outcome of FeaSel-Net when applied on the popular *Wine Classification* dataset with those of traditionally used linear approaches (PCA, LDA, and XGBoost), it was proven that the inherent nonlinear transformations in FeaSel-Net were beneficial. Another comparison with the current stochastic gates method showed that regularizer-based feature selection strongly depends on initialization and that recursive pruning methods such as FeaSel-Net select with a higher certainty. A classification executed in connection with each analyzed feature selection method with a dataset reduced to the specific features led to the best results when using FeaSel-Net's findings. In a different experiment in which the training process of unmasked data was juxtaposed with one with pre-selected features, it was confirmed that the algorithm could be a remedy against the curse of dimensionality. The application of FeaSel-Net to three more datasets of extremely contrasting sizes—from only four and up to 10k features—underlined that it covered several different use cases.

Applications of FeaSel-Net in other domains, such as physics, automotive development, or even the financial sector, are possible. However, the motivation for developing this algorithm originated from the field of spectroscopy and the urge to find new potential biomarkers that recent statistical approaches cannot reveal. With a sparser input and selective regions of interest in the spectral domain, spectrometer scans can be executed faster and measurement systems can be specifically engineered for the specific tasks. In fact, an article in which FeaSel-Net was applied on the Raman spectra of tumorous bladder tissue has already been published [40].

Even though the application of the algorithm is restricted to 1D datasets so far, it also has the potential to be used to prune uninformative filters—for example, in image processing. In order to analyze this potential, FeaSel-Net needs to be implemented in convolutional neural networks in future works.

Author Contributions: Conceptualization, K.F. and A.H.; methodology, F.F.; software, F.F. and A.B.; validation, F.F., A.B. and P.S.; data curation, F.F.; writing—original draft preparation, F.F. and A.B.; writing—review and editing, P.S., C.T. and A.H.; visualization, F.F.; supervision, A.H.; project administration, K.F.; funding acquisition, C.T. and A.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by DFG—Deutsche Forschungs Gemeinschaft, grant number RTG 2543/1.

Data Availability Statement: The package developed in this paper was implemented in Python and is available for download at PyPI (https://pypi.org/project/FeaSel-Net/) accessed on 11 October 2022. All visualizing data are contained within the manuscript or supplementary material. All other data and scripts are available on reasonable request from the corresponding author.

Acknowledgments: The work described in this paper was conducted in the framework of the Graduate School 2543/1 "Intraoperative Multisensory Tissue Differentiation in Oncology" (project A2 and B3) funded by the German Research Foundation (DFG—Deutsche Forschungsgemeinschaft).

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Appendix A. Statistical Analysis

Appendix A.1. Weighted Jaccard Matrices

This section provides an overview of the derivation of the weighted Jaccard matrices used to show inter-dependencies between the features. Corresponding to the Jaccard matrix of the *Wine classification* dataset, further matrices are shown in Figure A1.

The entries of these matrices are normalized Jaccard coefficients [41],

$$\mathbf{J}(i,j) = \frac{|R_i \cap R_j|}{|R_i \cup R_j|} \tag{A1}$$

where each set

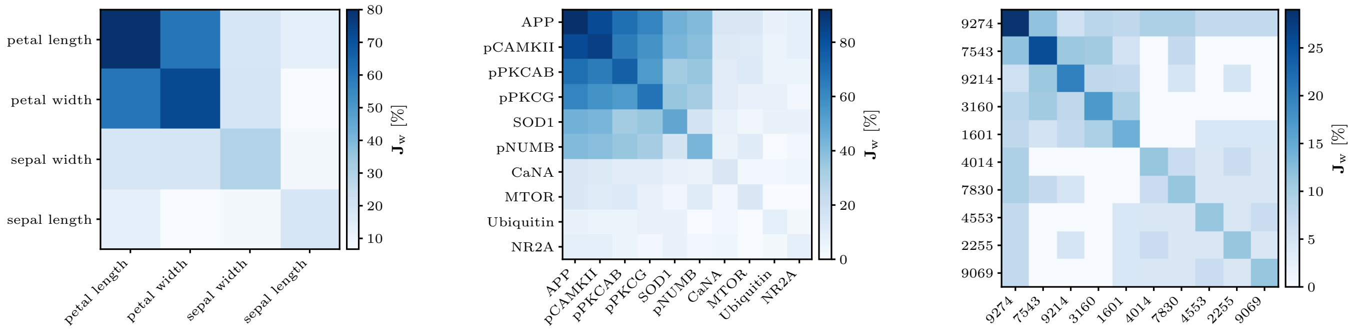
$$R = \{r \mid f \in m_r \text{ and } 0 < r \leq n_\mathrm{r}\} \tag{A2}$$

is a set of runs r in which a specific feature f has been found to be among the most important features in the output mask m_r. Its cardinality describes how often each feature is chosen, and it also stores the index r, the run in which it is chosen. This is done for all n_r runs. Subsequently, the weighting is done by summarizing all binary masks $\mathbf{w}^{(\mathrm{in})}$ in a sorted weight vector

$$\mathbf{s} = \operatorname{argsort}\left(\frac{1}{n_\mathrm{r} \cdot q} \sum_{i=0}^{n_r} \mathbf{w}^{(\mathrm{in})}\right) \tag{A3}$$

for the matrix, which is sorted by applying the argsort function defined in Equation (21). A weighted identity permutation matrix $\mathbf{P}$ is calculated by simply multiplying the sorting vector with an identity matrix $\mathbf{I} \in \mathbb{R}^{p \times p}$. A symmetrical 2D sorting and scaling operation yields the following weighted Jaccard matrix:

$$\mathbf{J}_\mathrm{w} = \sqrt{\mathbf{P}\mathbf{J}\mathbf{P}^\top}. \tag{A4}$$



(**a**) *Iris* (**b**) *Mice Protein Expression* (**c**) *Arcene*

Figure A1. Weighted Jaccard matrices for all analyzed datasets. FeaSel-Net is applied 100 times on the *Iris* (**a**) and *MPE* (**b**) datasets and 35 times on the *Arcene* (**c**) dataset. Both *MPE* and *Iris* show features that are chosen very consistently (i.e., more than 80% of the FS runs). In the case of the *MPE* dataset, this means an increase of almost 1,500% compared to random guessing. For the *Arcene* set, it is even an increase of approximately 28,500%.

Appendix A.2. Statistical Tests

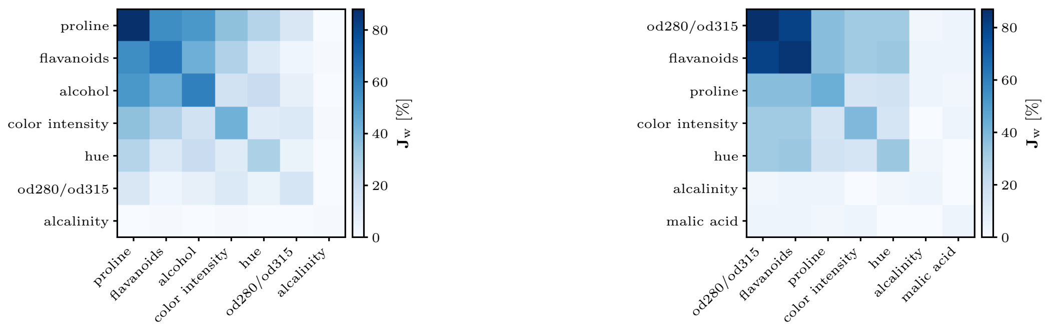
To prove that the FS algorithm has a statistical significance when selecting the subsets, a χ^2-test is applied on the data subsets coming from multiple runs. The H_0 hypothesis is a uniform selection of the features. Higher values of χ^2 imply a more significant result. All p-values are exactly or roughly zero and underline the deterministic output of FeaSel-Net.

Table A1. Statistical χ^2 test applied on the feature sets selected by FeaSel-Net.

Dataset	χ^2-Value	p-Value
Wine Classification	497.01	0
Iris	56.75	2.9×10^{-12}
MPE	4281.7	0
Arcene	23,307.14	0

Appendix A.3. Comparison of Certainty with Stochastic Gates

With the stochastic gates method, another neural network approach to feature selection is used. To assess the certainty of this FS method, we generated another Jaccard plot (as shown in Figure A2) by analyzing 50 applications of the STG method with $\lambda = 0.3$.



(**a**) FeaSel-Net (**b**) STG

Figure A2. Comparison of Jaccard matrices from the *Wine Classification* dataset using FeaSel-Net (**a**) and stochastic gates (**b**).

At first glance, the STG matrix seems to be more deterministic, since *od280/od315* and *flavanoids* are often chosen together (more than 80%), but the third feature is evidently less strictly chosen. The features *proline*, *color intensity*, and *hue* can all be chosen with the same certainty. FeaSel-Net, on the other hand, provides the highest certainty with its most important feature (*proline*). It also shows a distinct cluster of exactly three features that are chosen together, which is also the desired number of features. When applying the χ^2 test for the features retrieved by using the STG method, a statistical value of $\chi^2 = 259.79$ is obtained, which is only half the value from the application of FeaSel-Net.

Appendix B. Information on the Neural Networks

Appendix B.1. Available Parameters for the Callback Instantiation

After outlining the FS algorithm in Section 3, we would like to provide an overview on all of the available parameters for building, triggering, and evaluating within the callback in Table A2.

Table A2. Specified values for the FeaSel-Net building parameters.

Parameter	**Symbol**	**Value**
Build		
compression rate	κ	None
leave-one-out cross-validation	–	True
number of features	q	3
number of features to prune per step	n_p	None
number of samples	n_s	None
pruning rate	π	0.2
pruning type	–	exponential
Trigger		
decay	δ	0.0
maximal number of epochs	Δe_max	100
minimal number of epochs	Δe_min	20
threshold (accuracy-based)	τ_a	0.98
threshold (loss-based)	τ_g	None
type	–	accuracy
Evaluation		
accelerate	–	False
decision metric	–	average
metric	–	cross-entropy
normalization	–	min-max
rationalize	–	False
remove outliers	–	True
scale	–	False

Some of these parameters only work in combination with certain types, e.g., the desired number of features q and the compression ratio in which the number of features is the dominant parameter. However, we do not explain each parameter in depth here, since several parameters have already been explained in Section 3 and others are described in Section 4. Further information can be retrieved from the documentation in the FeaSel-Net code.

Appendix B.2. Model Architectures Used as Classifiers

Detailed information about the model architecture used for the *Wine Classification* dataset is given in Table A3. The table shows the architecture for the FeaSel-Net, unmasked, and reduced classifiers. The ReLU activation according to Equation (12) was selected for all layers with numerical indices. In the output layer, the softmax function from Equation (13) was applied, whereas the *LinearPass* (LP) layer was linearly activated. The optimization of the neural networks in all displayed architectures was done by using the *Adam* optimizer with a learning rate of $\eta = 0.005$.

Table A3. Different model architectures used for the *Wine Classification* dataset. The table shows the number of nodes in each layer and the number of parameters.

Model Type	l_{in}	l_{LP}	l_1	l_2	l_3	l_{out}	**Params**
FeaSel-Net	13	13	13	8	5	3	370
Reduced	3	-	6	12	6	3	207
Unmasked	13	-	9	4	3	3	193

Appendix B.3. Comparison of the Classification Performance

The classification performances were further split up into the three classes shown in Table A4. Comparing the results of each method, it can be seen that for all performance parameters, FeaSel-Net outperformed the others in two out of the three classes.

Table A4. Discriminability with masks retrieved from the different FS methods in percentages. Bold values indicate the best performance.

Method	**Class 1**	**Class 2**	**Class 3**	**Overall**
ACC (accuracy):				
FeaSel-Net	$\mathbf{99.19 \pm 0.36}$	$\mathbf{97.75 \pm 0.56}$	98.56 ± 0.37	$\mathbf{98.50 \pm 0.43}$
PCA	97.02 ± 0.85	95.97 ± 0.84	97.17 ± 0.67	96.72 ± 0.79
LDA	97.80 ± 0.54	96.86 ± 0.86	$\mathbf{99.02 \pm 0.38}$	97.89 ± 0.59
XGBoost	97.80 ± 0.54	96.86 ± 0.86	$\mathbf{99.02 \pm 0.38}$	97.89 ± 0.59
STG	97.93 ± 0.97	95.24 ± 1.06	97.30 ± 0.63	96.82 ± 0.89
random	91.27 ± 0.64	93.42 ± 0.80	91.9 ± 0.81	92.20 ± 0.75
SEN (sensitivity):				
FeaSel-Net	$\mathbf{99.32 \pm 1.09}$	95.44 ± 0.84	$\mathbf{99.25 \pm 1.18}$	$\mathbf{98.00 \pm 1.04}$
PCA	93.93 ± 0.99	$\mathbf{96.53 \pm 1.44}$	94.53 ± 1.35	94.96 ± 1.26
LDA	96.54 ± 1.77	96.24 ± 0.90	98.10 ± 0.85	96.96 ± 1.17
XGBoost	96.54 ± 1.77	96.24 ± 0.90	98.10 ± 0.85	96.96 ± 1.17
STG	97.29 ± 1.55	92.17 ± 1.68	97.25 ± 1.16	95.57 ± 1.46
random	85.73 ± 1.99	91.73 ± 1.04	86.37 ± 2.94	87.94 ± 2.00
SPE (specificity):				
FeaSel-Net	$\mathbf{99.13 \pm 0.17}$	$\mathbf{99.29 \pm 0.77}$	98.31 ± 0.38	$\mathbf{98.91 \pm 0.44}$
PCA	98.56 ± 0.87	95.68 ± 0.60	98.14 ± 0.64	97.46 ± 0.71
LDA	98.42 ± 0.51	97.27 ± 1.20	$\mathbf{99.36 \pm 0.29}$	98.35 ± 0.67
XGBoost	98.42 ± 0.51	97.27 ± 1.20	$\mathbf{99.36 \pm 0.29}$	98.35 ± 0.67
STG	98.25 ± 0.94	97.27 ± 1.04	97.32 ± 0.59	97.61 ± 0.86
random	94.01 ± 0.94	94.55 ± 1.4	93.94 ± 1.34	94.17 ± 1.24

Interestingly, the performances in *class* 2 were worse than those in the other classes, which was an indicator for an imbalance between the sample numbers. With 71 samples, *class* 2 was overrepresented in the dataset; contrary to the belief that over-representation yields better results in specific classes [42,43], it actually generalized worse in all studied methods. Since the imbalance equally existed in the biased training set and the unbiased test set, it can only be explained by the inherent properties of *class* 2, which were more demanding for a correct classification, or that the selected features exhibited less potential for an unambiguous prediction. Nonetheless, on average, throughout all classes, we obtained solid improvements in all evaluated parameters with the proposed method.

References

1. Koshmak, G.; Loutfi, A.; Linden, M.; Tamura, T. Challenges and issues in multisensor fusion approach for fall detection: Review paper. *J. Sens.* **2016**, *2016*, 6931789. [CrossRef]
2. Lever, J.; Krzywinski, M.; Altman, N. Model selection and overfitting. *Nat. Methods* **2016**, *13*, 703–704. [CrossRef]
3. Verleysen, M.; François, D. The curse of dimensionality in data mining and time series prediction. In *Computational Intelligence and Bioinspired Systems*; Cabestany, J., Prieto, A., Sandoval, F., Eds.; IWANN 2005. Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2005; Volume 3512, pp. 758–770. [CrossRef]
4. Pearson, K. LIII. On lines and planes of closest fit to systems of points in space. *Lond. Edinb. Dublin Philos.* **1901**, *2*, 559–572. [CrossRef]
5. Hotelling, H. Analysis of a complex of statistical variables into principal components. *J. Educ. Psychol.* **1933**, *24*, 471–441. [CrossRef]
6. van der Maaten, L.; Hinton, G. Visualizing data using t-SNE. *J. Mach. Learn. Res.* **2008**, *9*, 2579–2605.
7. McInnes, L.; Healy, J.; Melville, J. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv* **2020**, arXiv.1802.03426.
8. Masci, J.; Meier, U.; Cireşan, D.; Schmidhuber, J. (Eds.) Stacked convolutional auto-encoders for hierarchical feature extraction. In *International Conference on Artificial Neural Networks*; Springer: Berlin/Heidelberg, Germany, 2011. [CrossRef]
9. Zabalza, J.; Ren, J.; Zheng, J.; Zhao, H.; Qing, C.; Yang, Z.; Du, P.; Marshall, S. Novel segmented stacked autoencoder for effective dimensionality reduction and feature extraction in hyperspectral imaging. *Neurocomputing* **2016**, *185*, 1–10. [CrossRef]
10. Saeys, Y.; Inza, I.; Larrañaga, P. A review of feature selection techniques in bioinformatics. *Bioinformatics* **2007**, *23*, 2507–2517. [CrossRef]
11. Majid Mehmood, R.; Du, R.; Lee, H.J. Optimal Feature Selection and Deep Learning Ensembles Method for Emotion Recognition From Human Brain EEG Sensors. *IEEE Access* **2017**, *5*, 14797–14806. [CrossRef]
12. Breiman, L. Random forests. *Mach. Learn.* **2001**, *45*, 5–32. [CrossRef]
13. Chen, T.; Guestrin, C. XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; Association for Computing Machinery: New York, NY, USA, 2016; pp. 785–794. [CrossRef]
14. Zhang, D.; Zou, L.; Zhou, X.; He, F. Integrating Feature Selection and Feature Extraction Methods With Deep Learning to Predict Clinical Outcome of Breast Cancer. *IEEE Access* **2018**, *6*, 28936–28944. [CrossRef]
15. Figueroa Barraza, J.; López Droguett, E.; Martins, M.R. Towards Interpretable Deep Learning: A Feature Selection Framework for Prognostics and Health Management Using Deep Neural Networks. *Sensors* **2021**, *21*, 5888. [CrossRef] [PubMed]
16. Liu, Z.; Yu, Y.; Sun, Z. A hidden feature selection method based on l2,0-norm regularization for training single-hidden-layer Neural Networks. In Proceedings of the 2019 IEEE Symposium Series on Computational Intelligence (SSCI), Xiamen, China, 6–9 December 2019; pp. 1810-1817. [CrossRef]
17. Yamada, Y.; Lindenbaum, O.; Negahban, S.; Kluger, Y. Feature Selection using Stochastic Gates. In Proceedings of the 37th International Conference on Machine Learning, Virtual, 13–18 July 2020; pp. 10648–10659.
18. Chang, C.H.; Rampasek, L.; Goldenberg, A. Dropout feature ranking for deep learning models. *arXiv* **2017**, arXiv.1712.08645.
19. Louizos, C.; Welling, M.; Kingma, D.P. Learning sparse neural networks through l0 regularization. *arXiv* **2017**, arXiv.1712.01312.
20. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
21. Guyon, I.; Weston, J.; Barnhill, S.; Vapnik, V. Gene selection for cancer classification using support vector machines. *Mach. Learn.* **2002**, *46*, 389–422. [CrossRef]
22. Dua, D.; Graff, C. UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences 2019. Available online: http://archive.ics.uci.edu/ml (accessed on 11 October 2022).
23. Malhi, A.; Gao, R.X. PCA-based feature selection scheme for machine defect classification. *IEEE Trans. Instrum. Meas.* **2004**, *53*, 1517–1525. [CrossRef]
24. Song, F.; Guo, Z.; Mei, D. (Eds.) Feature selection using principal component analysis. In Proceedings of the 2010 International Conference on System Science, Engineering Design and Manufacturing Informatization, Yichang, China, 12–14 November 2010; Volume 1. [CrossRef]
25. Hopes, K.; Cauchi, M.; Walton, C.; MacQueen, H.; Wassif, W.; Turner, C. A novel method for the analysis of clinical biomarkers to investigate the effect of diet on health in a rat model. *Analyst* **2015**, *140*, 3028–3038. [CrossRef]
26. Han, H. Nonnegative principal component analysis for mass spectral serum profiles and biomarker discovery. *BMC Bioinform.* **2010**, *11*, S1. [CrossRef]
27. Tarpley, L.; Duran, A.L.; Kebrom, T.H.; Sumner, L.W. Biomarker metabolites capturing the metabolite variance present in a rice plant developmental period. *BMC Plant Biol.* **2005**, *5*, 8. [CrossRef]
28. Champion, K.; Lusch, B.; Kutz, J.N.; Brunton, S.L. Data-driven discovery of coordinates and governing equations. *Proc. Natl. Acad. Sci. USA* **2019**, *116*, 22445–22451. [CrossRef] [PubMed]
29. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533–536. [CrossRef]
30. Kingma, D.P.; Ba, J.L. Adam: A method for stochastic optimization. *arXiv* **2017**, arXiv.1412.6980.

31. Barth, J.; Katumullage, D.; Yang, C.; Cao, J. Classification of wines using principal component analysis. *J. Wine Econ.* **2021**, *16*, 56–67. [CrossRef]
32. Fu, J.; Huang, C.; Xing, J.; Zheng, J. Pattern classification using an olfactory model with PCA feature selection in electronic noses: Study and application. *Sensors* **2012**, *12*, 2818–2830. [CrossRef]
33. Kumar, S.; Kraeva, Y.; Kraleva, R.; Zymbler, M. A deep neural network approach to predict the wine taste preferences. In *Intelligent Computing in Engineering. Advances in Intelligent Systems and Computing*; Solanki, V., Hoang, M., Lu, Z., Pattnaik, P., Eds.; Springer: Singapore, 2020; Volume 1125, pp. 1165–1173. [CrossRef]
34. Saâdaoui, F.; Bertrand, P.R.; Boudet, G.; Rouffiac, K.; Dutheil, F.; Chamoux, A. A dimensionally reduced clustering methodology for heterogeneous occupational medicine data mining. *IEEE Trans. Nanobiosci.* **2015**, *14*, 707–715. [CrossRef]
35. Song, F.; Mei, D.; Li, H. (Eds.) Feature selection based on linear discriminant analysis. In Proceedings of the 2010 International Conference on Intelligent System Design and Engineering Application, Changsha, China, 13–14 October 2010; Volume 1. [CrossRef]
36. Glorot, X.; Bengio, Y. (Eds.) Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, Sardinia, Italy, 13–15 May 2010; Volume 9.
37. Fisher, R.A. The use of multiple measurements in taxonomix problems. *Ann. Eugen.* **1936**, *7*, 179–188. [CrossRef]
38. Higuera, C.; Gardiner, K.J.; Cios, K.J. Self-organizing feature maps identify proteins critical to learning in a mouse model of down syndrome. *PLoS ONE* **2015**, *10*, e0129126. [CrossRef]
39. Guyon, I.; Gunn, S.; Ben-Hur, A.; Dror, G. (Eds.) Result analysis of the NIPS 2003 feature selection challenge. In *Advances in Neural Information Processing Systems 17 (NIPS 2004)*; MIT Press: Cambridge, MA, USA, 2004; Volume 17.
40. Becker, L.; Fischer, F.; Fleck, J.L.; Harland, N.; Herkommer, A.; Stenzl, A.; Aicher, W.K.; Schenke-Layland, K.; Marzi, J. Data-Driven Identification of Biomarkers for In Situ Monitoring of Drug Treatment in Bladder Cancer Organoids. *Int. J. Mol. Sci.* **2022**, *23*. [CrossRef]
41. Levandowsky, M.; Winter, D. Distance between sets. *Nature* **1971**, *234*, 34–35. [CrossRef]
42. Thai-Nghe, Nguyen, Gantner, Zeno.; Schmidt-Thieme, L. (Eds.) Cost-sensitive learning methods for imbalanced data. In Proceedings of the 2010 International Joint Conference on Neural Networks, Barcelona, Spain, 18–23 July 2010. [CrossRef]
43. Yan, Y.; Chen, M.; Shyu, M.L.; Chen, S.C. (Eds.) Deep learning for imbalanced multimedia data classification. In Proceedings of the 2015 IEEE International Symposium on Multimedia (ISM), Miami, FL, USA, 14–16 December 2015. [CrossRef]