Mikhail Zolotukhin

Feature Importance in Deep Learning

Progress report for eÄlytelli project

October 25, 2021

University of Jyväskylä

Faculty of Information Technology

Preface

Deep Learning is a sub-field of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks. The key advantage of deep learning models is their ability to perform automatic feature learning using raw data. Deep learning methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features. Automatically learning features at multiple levels of abstraction allow a system to learn complex functions mapping the input to the output directly from data, without depending completely on human-crafted features. Deep learning excels on problem domains where the inputs are analog which include images of pixel data, documents of text data or files of audio data. To process this information first layers of a deep neural network are utilized to learn spatio-temporal feature dependencies present in the data and find compact low-dimensional representations of high-dimensional data samples, whereas later layers are responsible for making a prediction based on the features learned in the former layers. Due to its scalability and feature learning abilities deep learning has found its application in almost every sector of industry ranging from object recognition for systems in autonomous vehicles to helping doctors detect and diagnose cancer. In the report, we are using data extracted during wood bleaching process and attempt to predict bleach ratio based on the given parameter values by learning temporal dependencies present in the data.

List of Figures

Contents

1 Background

1.1 Introduction

Deep neural networks consist of multiple layers of interconnected neurons, each building upon the previous layer to refine and optimize the prediction or categorization. This progression of computations through the network is called forward propagation. The input and output layers of a deep neural network are called visible layers whereas the rest layers are called hidden. The input layer is where the deep learning model ingests the data for processing, and the output layer is where the final prediction or classification is made. Another process called back-propagation uses algorithms, like gradient descent, to calculate errors in predictions and then adjusts the weights and biases of the function by moving backwards through the layers in an effort to train the model. Together, forward propagation and back-propagation allow a neural network to make predictions and correct for any errors accordingly. Over time, the algorithm becomes gradually more accurate. To address specific problems and datasets there are different types of neural networks having been developed some of which might be incredibly complex. For example, convolutional neural networks (CNNs) are used primarily in computer vision and image classification applications, they can detect features and patterns within an image, enabling tasks, like object detection or recognition [Krizhevsky, Sutskever, and Hinton 2012]. Recurrent neural networks (RNNs) are typically used in natural language and speech recognition applications as they leverage sequential or times series data [Graves 2014].

In deep learning, feature importance ranking (FIR) refers to a task that measures contributions of individual input features (variables) to the performance of a supervised learning model. FIR has become one of powerful tools in AI applications to facilitate understanding of decision-making by a learning system and discovery of key factors in a specific domain. Due to the existence of dependent and irrelevant features in high-dimensional data, feature selection is often employed to address the well-known curse of dimensionality challenge and to improve the generalization of a learning system, where a subset of optimal features is selected in terms of the predefined criteria to maximize the performance of a learning system

[Wojtas and Chen 2020]. Feature selection may be conducted at either population or instance level. The former methods would find out an optimal feature subset collectively for all the instances in a population, while the latter ones tend to uncover a subset of salient features specific to a single instance. In practice, FIR is always closely associated with feature selection by ranking the importance of those features in an optimal subset and can also be used as a proxy for feature selection [Guyon and Elisseeff 2003].

The main purpose of this report is to compare several methods for feature selection and feature importance evaluation. The rest of the report is organized as follows. In section 1.2, we briefly overview the dataset used in the experiments. Section 2.1 presents results when applying different deep learning models to the dataset, whereas model deployment details are highlighted in Section 2.2. Section 3.1 highlights two traditional correlation-based supervised feature selection methods. Feature importance calculation methods using deep learning are evaluated in Section 3.2 and 3.3. All the results are summarized in Section 4.1.

1.2 Data

The dataset used in our machine learning experiments contains information extracted during wood bleaching process. Bleaching of wood pulp is the chemical processing of wood pulp to lighten its color and whiten the pulp. The primary product of wood pulp is paper, for which whiteness is an important characteristic. Each sample in the dataset includes bleach ratio value and 82 parameters which affect this ratio to a greater or lesser extent. These parameters are divided into 5 categories according to the time moment they applied during the bleaching process. Both the feature and target values are floating number numerical variables. The task is to train a deep learning model to predict the bleach ratio based on the given parameter values and analyze effect of each of the parameters given on the bleach ratio value.

2 Deep learning models

2.1 Model evaluation

The simplest traditional neural network architecture consists of several fully-connected layers. In a fully-connected neural network layer, each neuron in a hidden or output layer is fully connected to all neurons of the previous layer with the output being calculated by applying the activation function to the weighted sum of the previous layer outputs. A deep learning model comprised of such layers is usually refereed as fast-forward neural network or multi-layered perceptron (MLP). MLPs have few trainable parameters and therefore learn fast compared to more complicated architectures, however they may suffer when dealing with spatio-temporal data such as images and time-series.

The pulp bleaching dataset we use in this report has some temporal dependencies between feature variables as they correspond to the actions taken during the bleaching process in different time moments. Therefore, bleach ratio prediction model theoretically should benefit from using convolutional or recurrent neural network layers. However, for the sake of demonstration, we first train neural network models which consist of only full-connected layers. As mentioned in the introduction, the dataset includes 82 features which means the input layer should have at maximum 82 neurons. Different features have different scales and some of them might be missing in some data samples. For this reason, we connect the input layer to a preprocessing layer which first standardizes all raw feature values scaling them into range $[0,1]$ and then masks not-a-number (NaN) values with -1. We then split the features preprocessed into 5 tensors according to their feature classes. Each of these tensors acts as an input to fully-connected layer of size 256 in order to learn each class features in a separate neural network stream. The resulting outputs are then stacked together and they can be fed to subsequent layers for further feature learning. The resulting network architecture is shown in Figure 1.

As mentioned in the introduction, the task is to predict bleach ratio value based on n feature classes available by the moment the prediction is made, where $n \in [1,5]$. Thus, the resulting output size may vary from 1×256 to 5×256. This output feature stack is then flattened and

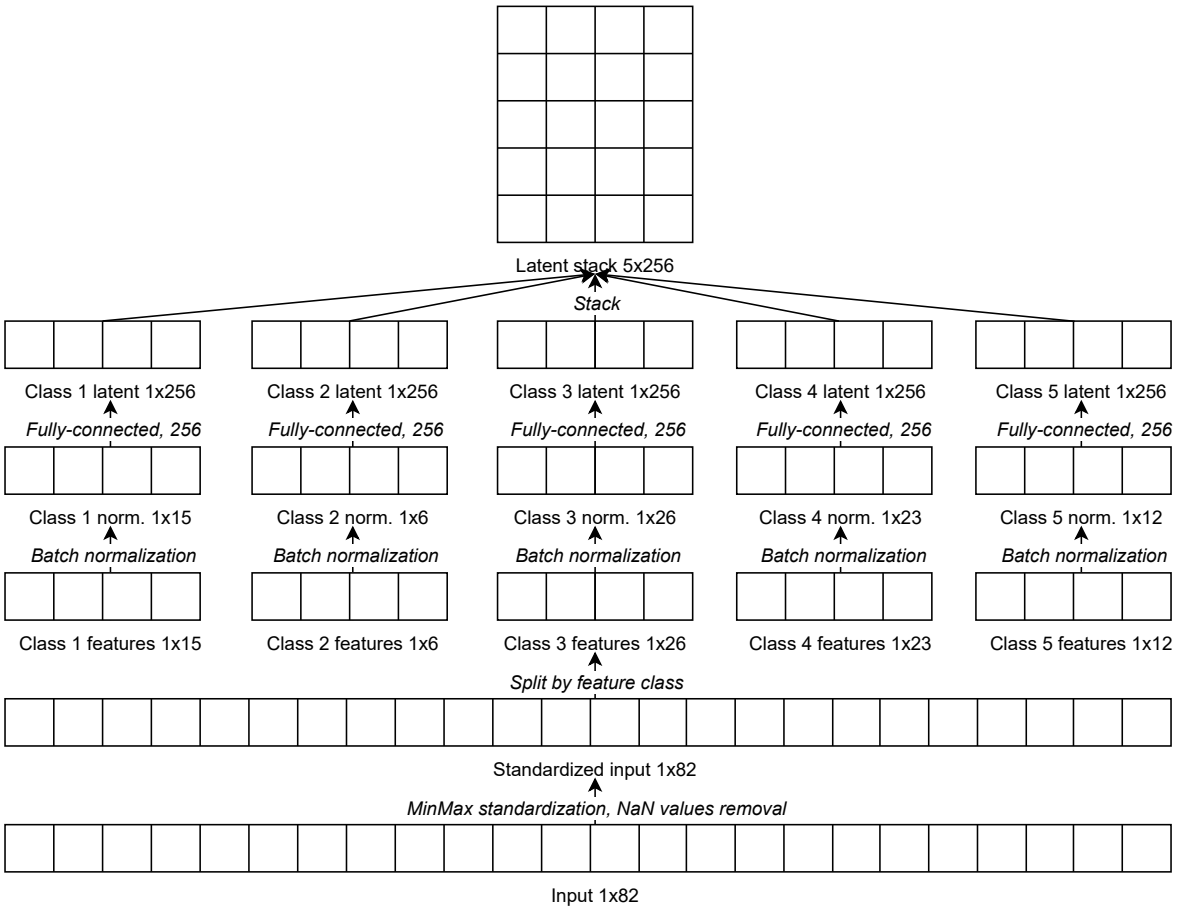


Figure 1. Input layers for neural network models used for bleach ratio prediction.

fed to a series of two fully-connected layers of size 2048. We also add dropout layers after each fully-connected one. Dropout is a technique for addressing the problem of overfitting. The key idea is to randomly drop neurons along with their connections from the network during training. This prevents the network from relying on a narrow range of certain features. Five resulting fully-connected neural networks, one for each feature class combination, are trained for 3 times, each with new train-validation-test split.

Temporal dependencies in the data can be extracted with the help of recurrent neural networks. In distinction to a fully-connected layer, a recurrent layer assumes that input data samples are time-series. To accommodate this fact, each recurrent layer has its own internal state the value of which is calculated based on the state value of the previous sample. The output of the recurrent layer is essentially an activation of the weighted sum of the previous layer outputs added to the weighted sum of the previous state values. During the learning pro-

cess, derivatives are backpropagated through time, all the way to the beginning or to a certain point. All the derivatives multiply the same weight matrix which may result in either infinite or vanishing update values. While gradient exploding can be fixed by straight-forward clipping [Pascanu, Mikolov, and Bengio 2013], dealing with gradient vanishing requires an intelligent control over the state via forget gates [Hochreiter and Schmidhuber 1997]. Long short-term memory neural networks (LSTMs) are explicitly designed to avoid the long-term dependency problem by remembering information for long periods of time. The key advantage of LSTMs is the gates which regulate the process of adding and removing information in the cell state.

A LSTM network should in theory strive in cases of long-term temporal dependencies between data features. However, in case of the pulp bleaching dataset we use for evaluation in this report, the number of time steps is limited by 5 at maximum when all feature classes participate in bleach ratio value prediction. To evaluate LSTM algorithm, we add two LSTM layers of size 640 on top of the input structure defined in the previous section. The output of the second LSTM layer is then fed to the last fully-connected layer of size 2048 as it has been done in case of the MLP model. We also test a bidirectional LSTM (BILSTM) which is an extension of traditional LSTM which consists of two LSTMs connected in the following way: the first LSTM learns the input sequence as-is and the second one learns a reversed copy of the input sequence. This can provide additional context to the network and result in faster and even fuller learning of the problem.

Convolutional neural networks (CNNs) are usually employed for automatic extraction of low-level features such as edges, color, gradient orientation in image related problems. The main building block of CNN is the convolutional layer which calculates an integral that expresses the amount of overlap of the layer's filter as it is shifted over the input data. Similarly to the previous case, the integral value is passed through an activation function to account for non-linearity in data. As a rule, multiple convolutions are performed on the input, each using a different filter. Resulting feature maps are then stuck together and become the final output of the convolution layer. CNNs can be employed to handle data of any dimension, since the result of the convolution operation is always a scalar. After a convolution operation, pooling can be performed to reduce the dimensionality of the output. The most common type

of pooling is max pooling which takes the max value in the pooling window. Contrary to the convolution operation, pooling has no parameters. It slides a window over its input, and simply takes the max value in the window. CNNs usually consist of several convolutional layers mixed with few pooling layers followed by standard fully-connected layers. Stacking multiple convolutional layers allows one to learn both basic features as well as higher level representations to recognize objects in different shapes and positions. It is worth mentioning that convolutional layers can also be employed to extract features from temporal data. For example, DeepMind's Q-network that teaches itself to play Atari games, stacks last four frames of the historical data to produce an input for the CNN [Mnih et al. 2013].

Time series data requires kernel of a convolutional layer sliding in only one dimension, therefore one-dimensional CNN networks can be employed for the bleach ratio prediction problem. To evaluate CNN algorithm, on top of the input structure defined in the previous section, we add two 1D convolutional layers of size 2×1280 followed by 1D convolutional layer of size $n \times 1024$, where $n \in [1,5]$ depending on the number of feature classes available when the prediction takes place. The output of the last convolutional layer is fed to the last fully-connected layer of size 2048 similarly to the previous cases. In order to further improve prediction accuracy, we attempt to combine convolutional and recurrent layers. For this reason we substitute the last convolutional layer in the baseline CNN model with LSTM of size 640. The output of this LSTM layer again acts as an input for the last fully-connected layer of size 2048.

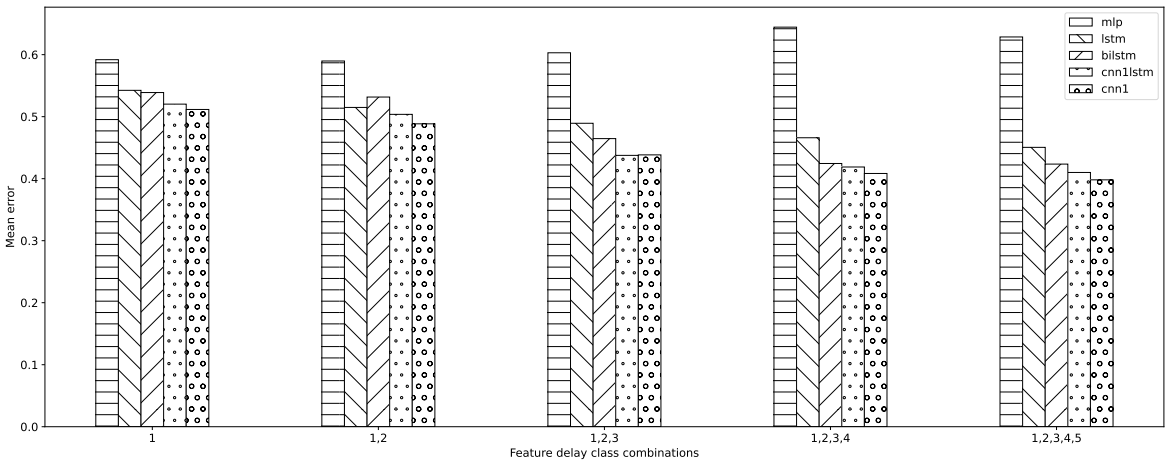


Figure 2. Mean prediction errors when using several supervised deep learning models.

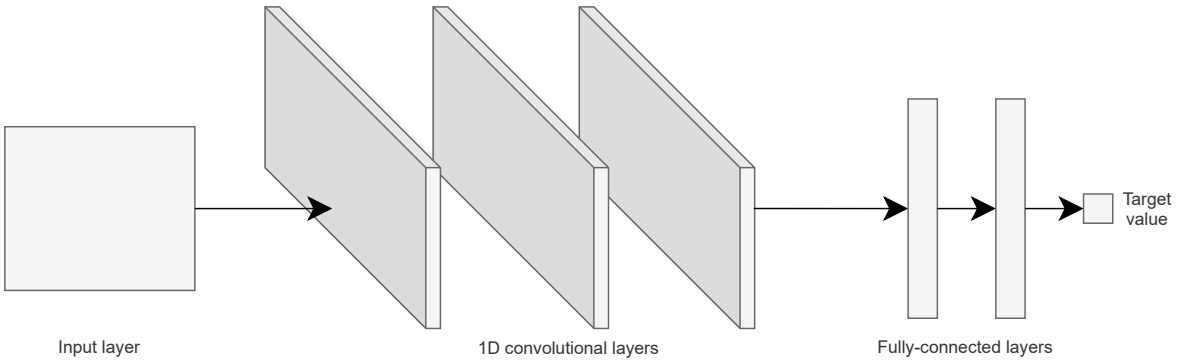


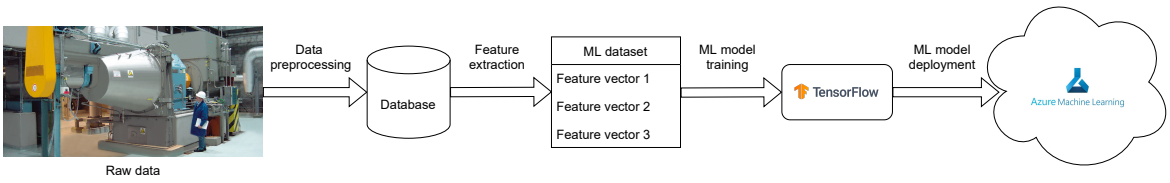
Figure 3. One-dimensional convolutional network used for bleach ratio prediction.

To test the models described above, we split our dataset into three parts: the first 40% are used for training, the second 20% - for validation in order to prevent overfitting, the last 40% are used for evaluating prediction error. We train a fully-connected neural network model which uses the combination of input layers described above. It is worth saying that Mean absolute prediction errors of the neural networks tested are shown in Figure 2. The results show that each new portion of feature values allows for more accurate bleach ratio prediction. As one can also notice, the one-dimensional CNN outperforms the rest of the models for all the feature class combinations used for prediction. It is also worth noticing that compared to LSTM-based models CNN is several times faster during both training and inference stage making it more attractive in our use-case. One dimensional CNN architecture can be found in Figure 3.

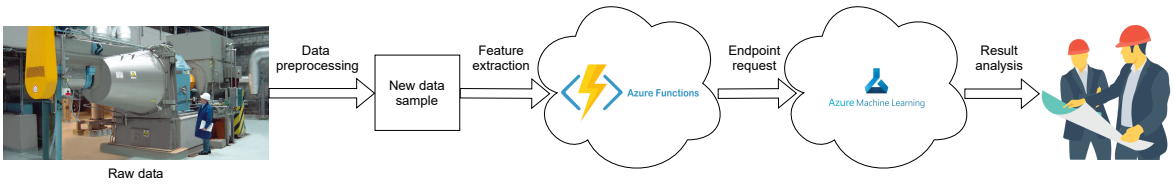
Finally, it is also worth mentioning that it is possible to train one neural network which predicts bleach ratio for any number of feature classes available by treating feature values which are unknown by the moment the prediction is made as NaN values, which are subsequently substituted with certain mask value, e.g. -1 as in our models. This would allow us to reduce the number of storage resources needed, however, according to the preliminary results, it comes with a small drop in accuracy. For this reason, we further continue training five different models, one for each delay class combination, since storage resources is not an issue in our use-case.

2.2 Model deployment

In this section, we briefly describe the process of deploying the models trained as a Microsoft Azure Machine Learning service. As a rule, a machine learning process includes two stages: training and inference (see Figure 4 for more details). During the former, a machine learning model is trained using data examples available, whereas at the latter the model trained is used to predict the result for a new input data sample. The models described in the previous section are first implemented in Tensorflow and trained locally on a GPU server using the bleach ratio prediction dataset. A code snippet for model compiling, training and saving using Tensorflow in Python are shown respectively in listings 2.1 and 2.2. In the listings, "features" is a list of feature names, "target" is the name of the target variable, "lr" is the model learning rate value, parameter "patience" controls early stopping to avoid overfitting in case the validation error starts growing.



(a) Training



(b) Inference

Figure 4. Machine learning using Microsoft Azure cloud services.

Listing 2.1. Compiling a deep learning model.

```
inputs = {
    colname: tf.keras.layers.Input(name=f'{col}', shape=(1,),
    dtype=tf.float32) for col in features
}
```

```
hidden = tf.keras.layers.Concatenate(axis=-1)(
    list(inputs.values())
)
hidden = ...  # define hidden layers of the model
outputs = tf.keras.layers.Dense(1, activation=)(hidden)
outputs = {target: outputs}
model = tf.keras.models.Model(inputs=inputs, outputs=outputs)
model.compile(
    loss=tf.keras.losses.MeanSquaredError(),
    optimizer=tf.keras.optimizers.Adam(learning_rate=lr)
)
```

Listing 2.2. Training and saving the deep learning model compiled.

```
model.fit(
    X_tr, Y_tr,  # training data
    validation_data=(X_val, Y_val,  # validation data
    epochs=epochs,  # number of training epochs
    batch_size=batch_size,  # batch size
    callbacks=[
        tf.keras.callbacks.EarlyStopping(
            monitor='val_loss',
            patience=patience,
            mode='min',
            restore_best_weights=True
        )
    ]
)
model.save('saved_model')
```

Once the models are trained and saved, we can upload the resulting binaries to Azure cloud.

For this purpose, login to your Azure account, set a subscription and list the workspaces you have access to. Alternatively, a new workspace can be created. The model trained earlier can be next uploaded to the cloud service as shown in Listing 2.3.

Listing 2.3. Registering the model.

```
$ az login
$ az account set -s <my subscription>
$ az ml workspace list --resource-group=<my resource group>
$ az ml workspace create --file workspace.yml --resource-group \
my-resource-group
$ az ml model register -n <model_name> -p <path to the models>
```

Next, an inference cluster is supposed to be created. For this purpose, in Azure portal, navigate to the workspace of the machine learning service. Click "Launch studio" and navigate to "Compute". Select "Inference clusters" and click "+New". Enter location information and select size of the virtual machine (VM) to be created to deploy the model trained. In the next menu, enter a name for your inference cluster and click "Create". Once the inference has been created, navigate to "Models" and click on the model name registered earlier. Next, click "Deploy" and select "Deploy to web service". Enter an endpoint name and description, select "Azure Kubernetes Service" as compute type and the inference cluster created in the previous step as compute name. In menu "Entry script file", select an inference script which is supposed to be used for evaluating new data samples (e.g. see Listing 2.4).

Listing 2.4. Entry script example.

```
import json
import tensorflow as tf

def init():
    global model
    model = tf.keras.models.load_model('saved_model')

def run(data):
    sample = json.loads(data)
```

```
    result = model.predict(sample)
    return result
```

In menu "Conda dependencies file", select a file with packages required for running the inference script. An example of such file is shown in Listing 2.5.

Listing 2.5. Conda dependencies.

```
name: test
channels:
    - defaults
dependencies:
    - python=3.8
    - numpy
    - tensorflow
    - pip
    - pip:
        - azureml-defaults
```

Finally, click "Deploy" to deploy the model as a web service. Once the deployment process has completed, navigate to "Endpoints" and find the endpoint just created. In case of success, the URL of the endpoint can be found in "REST endpoint.

During the inference stage, when a new data sample becomes available for the analysis, an Azure function can be used to automate the process. In particular, assuming this new sample is saved in some temporary database, a time-triggered function retrieves this sample from the database and sends to the model endpoint in the form of a HTTP request. The prediction result can be then saved in the same database or sent to the third party. See an example of such a function in Listing 2.6. In the listing, functions "get_data_from_db" and "put_data_into_db" are two auxiliary functions which are responsible for transferring data between the temporal database and the function. Those can be implemented for example using SQLAlchemy which is the database toolkit for Python. Listing 2.7 shows the commands to deploy such a function to Microsoft Azure cloud.

Listing 2.6. Azure function example.

```
import json, requests
import azure.functions as func

from utils import get_data_from_db, put_data_into_db

def main(mytimer: func.TimerRequest) -> None:
    endpoint_url = 'http://1.2.3.4/api/v1/service/test/score'
    row_id, data = get_data_from_db()
    result = requests.post(endpoint_url, json=data)
    jdata = result.json()
    prediction = jdata['target']
    put_data_into_db(row_id, prediction)
```

Listing 2.7. Azure function deployment.

```
finc init
func azure functionapp publish <FunctionAppName>
```

3 Feature importance

3.1 Feature correlation

Feature selection is the process of reducing the number of input variables when developing a predictive model. It is desirable to reduce the number of input variables to both reduce the computational cost of modeling and, in some cases, to improve the performance of the model. Statistical-based feature selection methods involve evaluating the relationship between each input variable and the target variable using statistics and selecting those input variables that have the strongest relationship with the target variable. These methods can be fast and effective, although the choice of statistical measures depends on the data type of both the input and output variables. Since the dataset under consideration contains only numerical variables, our task falls into the category of regression predictive modeling problems with numerical input variables. The most common techniques include Pearson (linear) correlation and Spearman's rank-based (nonlinear) correlation.

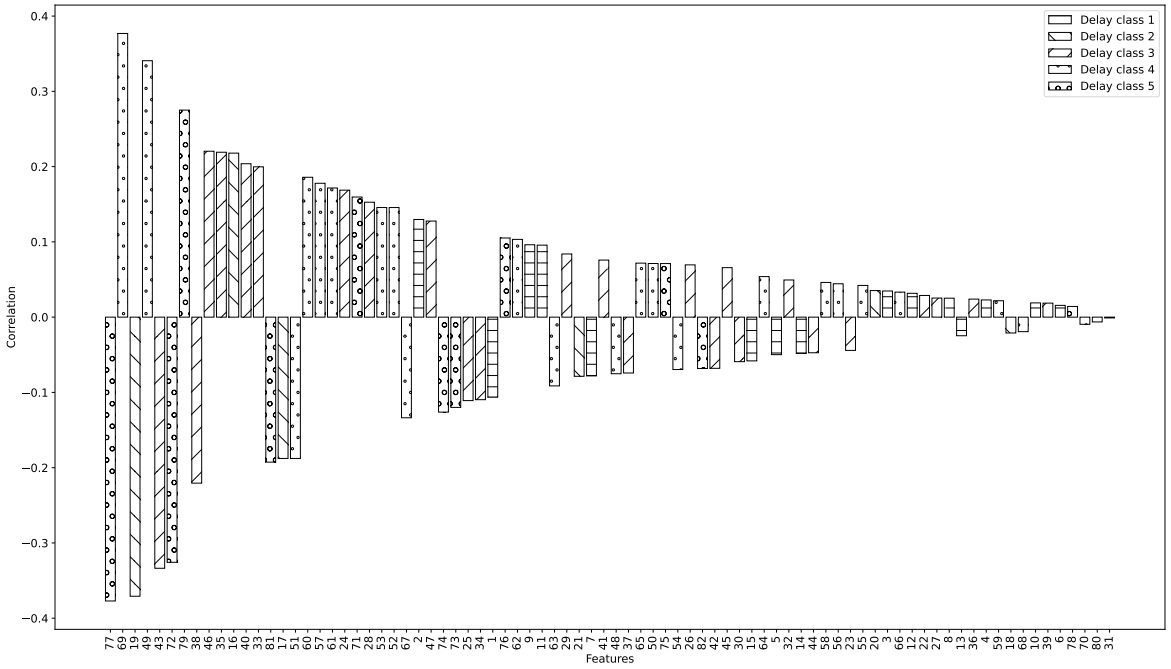
Pearson correlation is a measure of linear correlation between two sets of data. It is the ratio between the covariance of two variables and the product of their standard deviations. Pearson correlation can only reflect a linear correlation of variables, and ignores many other types of relationship or correlation. Given values of feature variable $x = \{x_1, x_2, \ldots, x_n\}$ and target variable $y = \{y_1, y_2, \ldots, y_n\}$ Pearson correlation ρ_{xy} is defined as:

$$\rho_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}, \text{ where } \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \text{ and } \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i. \tag{3.1}$$

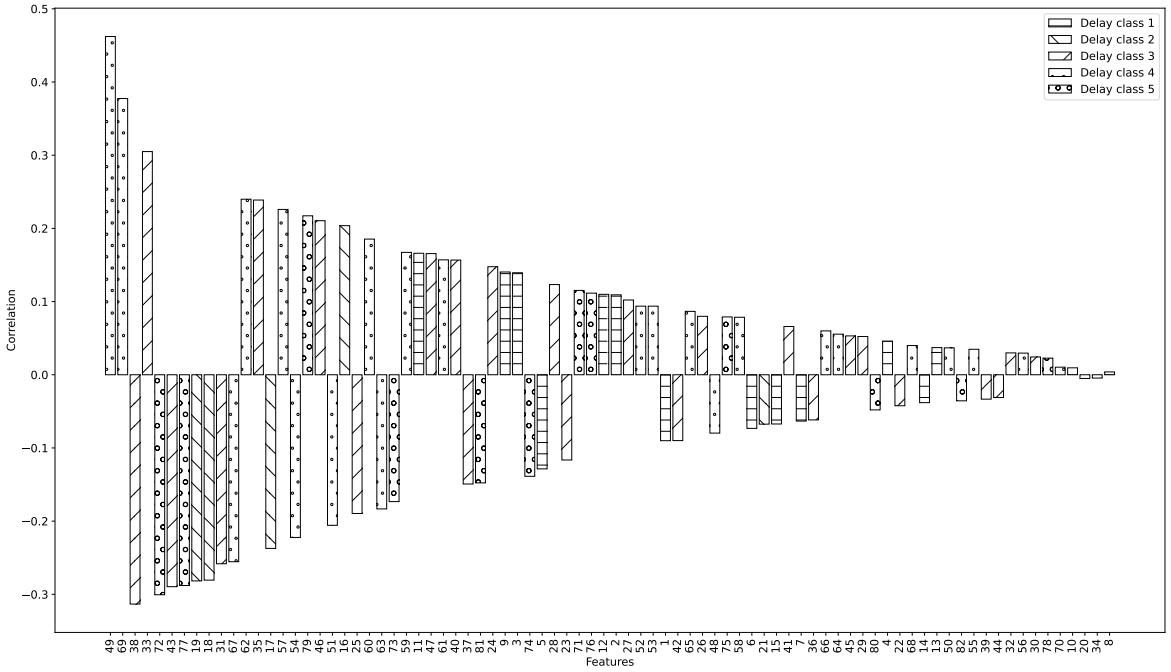
Spearman's rank correlation coefficient assesses how well the relationship between two variables can be described using a monotonic function. The Spearman correlation between two variables is equal to the Pearson correlation between the rank values of those two variables. While Pearson's correlation assesses linear relationships, Spearman's correlation assesses monotonic relationships. For variables x and y defined above Spearman correlation σ_{xy} can be calculated as follows:

$$\sigma_{xy} = 1 - \frac{6\sum_{i=1}^{n} d_i^2}{n(n^2-1)}, \tag{3.2}$$

where d_i is the difference between the two ranks of feature x_i and target y_i. Rank is simply



(a) Pearson correlation



(b) Spearman's rank

Figure 5. Correlation between feature and target values in the pulp bleaching dataset.

the position of this observation in the list of observations sorted in the ascending order, i.e. the minimal value of variable x (or y) would have rank 1 and the maximal one - rank n.

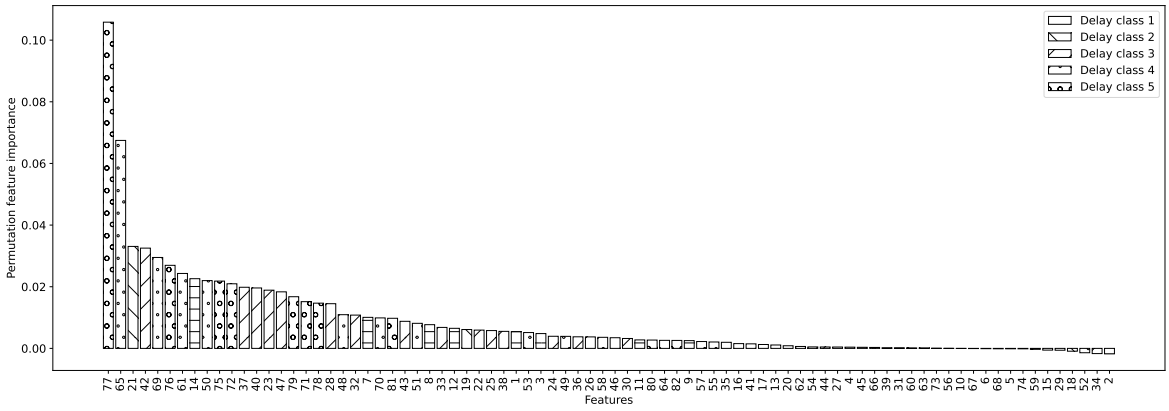
Pearson and Spearman correlation values between each feature and target variable are shown[1] in Figure 5. Correlation values are always between -1 and 1. We sort the results by its absolute values, since the bigger the absolute value the bigger the dependence of the target from the corresponding feature variable. As one can notice, according to these metrics, feature which are the most correlated with the target are feature 77 from the 5-th class and feature 47 from the 4-rd class. Pearson and Spearman correlation values for this variable are respectively -0.377 and 0.462.

3.2 Permutation importance

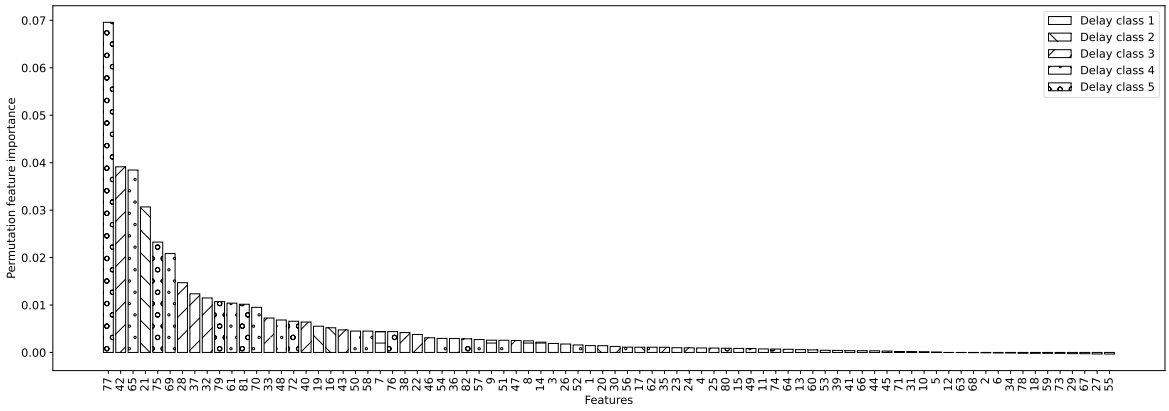
The second feature selection approach we test is permutation importance. Permutation feature importance measures the increase in the prediction error of a machine learning model after permuting the feature's values, which breaks the relationship between the feature and the target variable. The importance of a feature is measured by calculating the increase in the model's prediction error after permuting the feature. A feature is "important" if shuffling its values increases the model error, because in this case the model relies on the feature for the prediction. A feature is "unimportant" if shuffling its values leaves the model error unchanged, because in this case the model ignores the feature for the prediction [Fisher, Rudin, and Dominici 2019]. Permutation importance is easy to explain, implement, and use. Although calculation requires to make predictions multiple times depending on the number of permutations, this operation is not substantial compared to the model retraining. Also, permutation importance allows one to select features: if the score on the permuted dataset is higher then on the original one, it is a clear sign to remove the feature and retrain the model. For those reasons, permutation importance is wildly applied in many machine learning pipelines.

In order to employ this approach for feature selection, we train three the most accurate deep

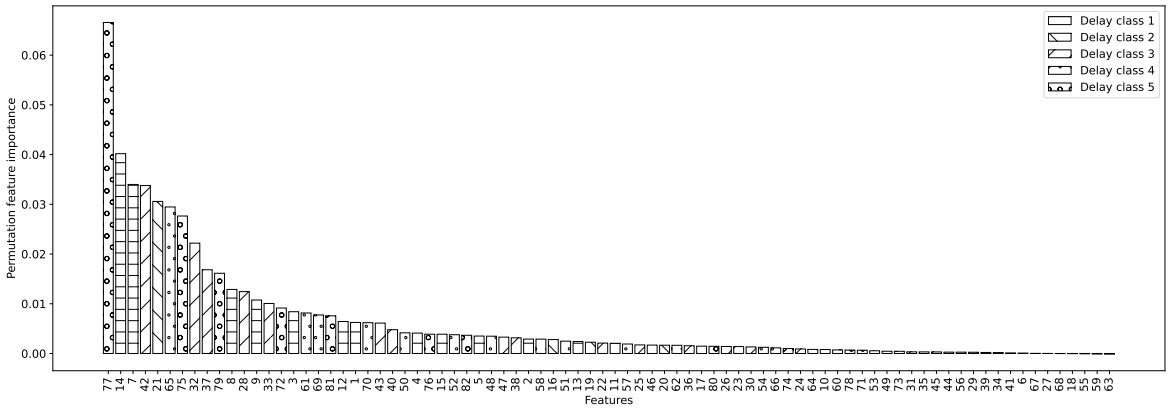
1. All figures are carried out in vector graphics format and therefore can be zoomed-in without loss of quality.



(a) CNN1



(b) CNN1+LSTM



(c) BILSTM network

Figure 6. Permutation feature importance in the pulp bleaching dataset calculated using three deep neural network models.

neural network models from the previous chapter to predict the target variable based on given feature values. The resulting prediction errors for one-dimensional convolutional, convolutional + LSTM and bidirectional LSTM models are respectively 0.398, 0.41 and 0.424 when predicting using all five feature classes. Permutation feature importance values calculated with these models are shown in Figure 6. In particular, each bar's height is simply equal to the difference between the prediction error based on the data when values of one of the input features are permuted and the original error value when non-permuted features are used. The bigger this value the more the neural network model relies on the corresponding feature to predict the target value. As one can notice, the feature which is the most important for each of the models tested is feature 76 from class 5. Permutation importance values of this variable for the models tested are respectively 0.106, 0.069 and 0.067.

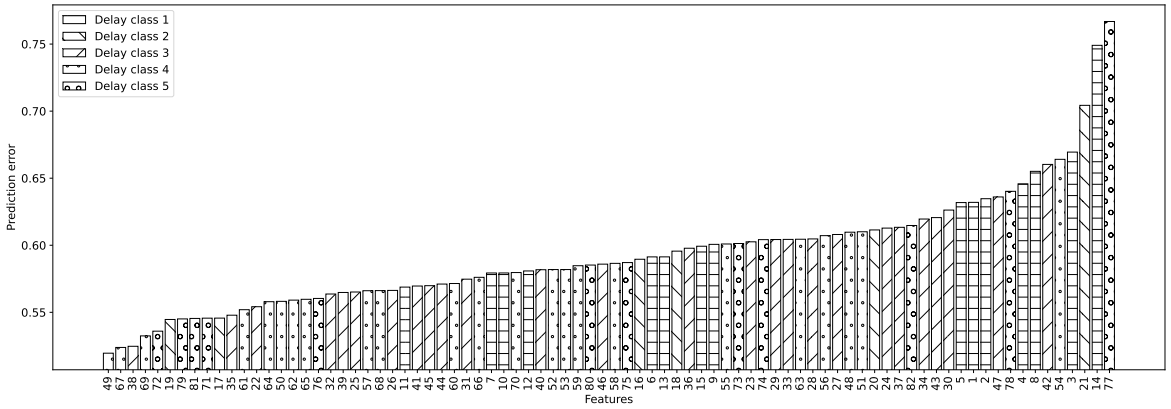
While being a very attractive choice for model interpretation, permutation importance has several problems, especially when working with correlated features. When calculating importance of a feature, the model is forced to make predictions for shuffled data points which have not been seen in the training data. To make such predictions, the model extrapolates to previously unseen regions which may lead to unexpected predictions. These points from new regions strongly affect the final error value and hence, permutation importance. To test this hypothesis, we can check how strong feature 77 is correlated with others. Maximal Pearson and Spearman correlation values for this feature are respectively 0.649 and 0.752 which are values of correlation with features 71 and 66. These values indicate quite high positive correlation, however they are not that high compared to correlations between other feature couples. Further investigation is required to answer the question whether this feature is important for the models trained or its high importance value is caused by simply falling into an unseen data region. Pearson and Spearman correlation values for each pair of features present in the pulp bleaching dataset can be found in Appendices of this report.

3.3 Prediction importance

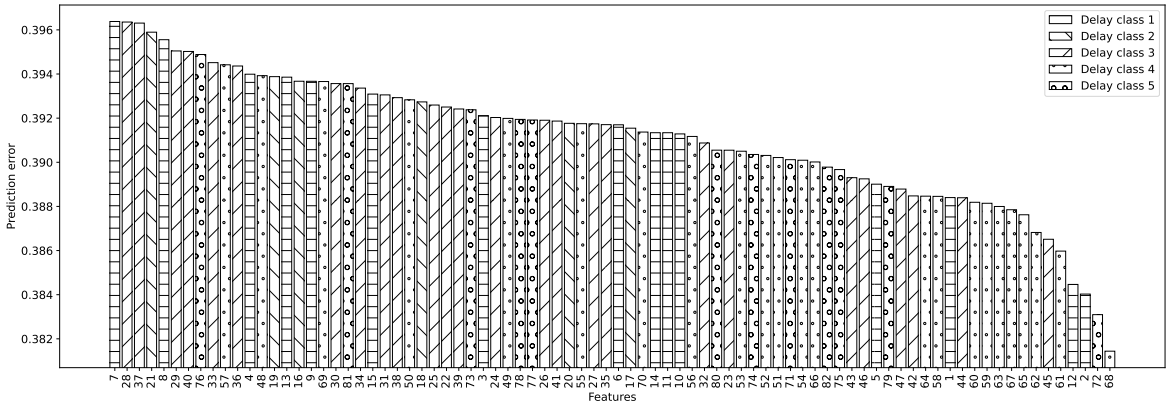
As mentioned in the previous section, high permutation importance value for a feature does not always indicate that the feature is important for the target value prediction, as it can be caused by the model used for evaluation not being able extrapolate properly on unseen data

points and regions. In order to solve this problem, we have to retrain the model for each new feature combination. The most straight-forward approach is to train a neural network model to predict the target value using only one of the input features. The lower the prediction error of such model the higher importance of the corresponding feature. On the other hand, if a feature is not important, it can easily by removed from the training data without significant impact on the prediction accuracy. Thus, the approach is to drop a feature, retrain the model, and then compare the resulting scores in order to find features dropping which increases the prediction error in a greater extent [Lei et al. 2017]. Similarly, instead of dropping the feature, one can permute feature values in the training data and retrain the model with the resulting data [Mentch and Hooker 2015]. This method differs from the one used in the previous section since it requires to retrain the model for each feature permutation. The obvious drawback is increasing in computational resources and time. However, when using this approach the prediction model does not require to extrapolate on unseen data regions.

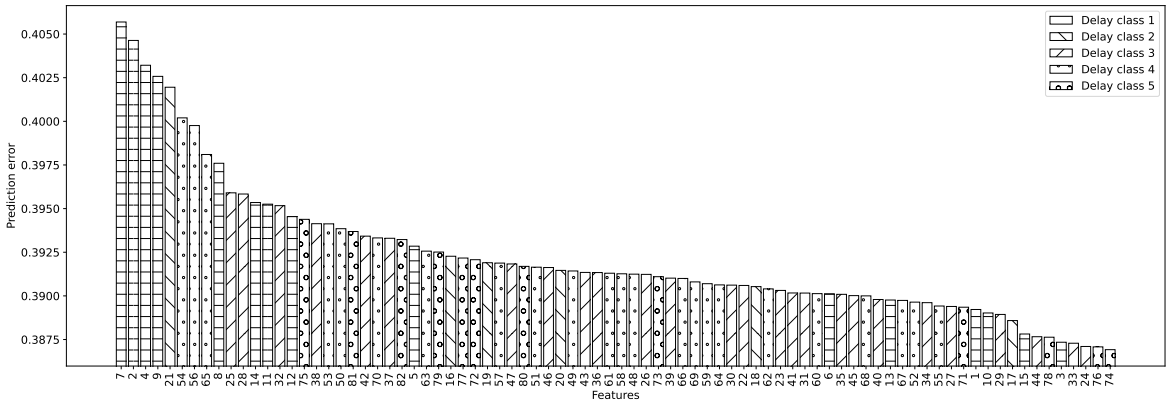
We examine three aforementioned methods using the pulp bleaching dataset. When only one feature is used to predict the bleach ratio, we use the MLP network from the previous chapter. For the other two cases, one-dimensional convolutional neural network is used. The experiment results can be found in Figure 7. As one can notice, when values of only one feature are used for bleach ratio prediction, the most important features are 49, 67, 38 and 69. These features are also among the most correlated with the target label as it can be seen from Figure 5. When using the second approach, i.e. retraining the model after dropping a variable, features from the first three delay classes seem to be the most important, in particular feature 7 from class 1. Similarly, according to the third approach, i.e. retraining the model after feature permutation, feature 7 has the highest score followed by three other features from the 1-st class, namely 2, 4 and 9. As one can notice from the figure the importance values obtained with the second and third approach do not deviate from each other significantly.



(a) One feature remains



(b) One feature is dropped



(c) One feature is permuted

Figure 7. Bleach ratio prediction importance with model retraining.

4 Conclusion

4.1 Summary and feature ranking

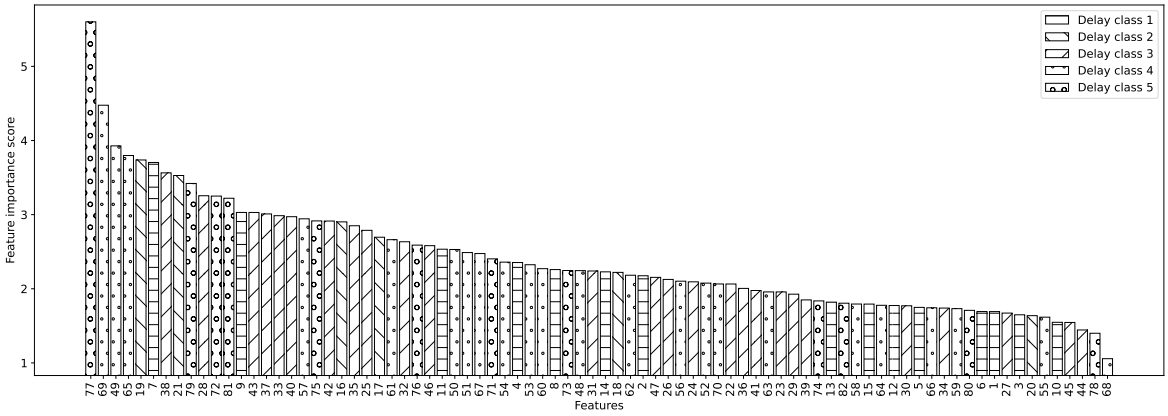


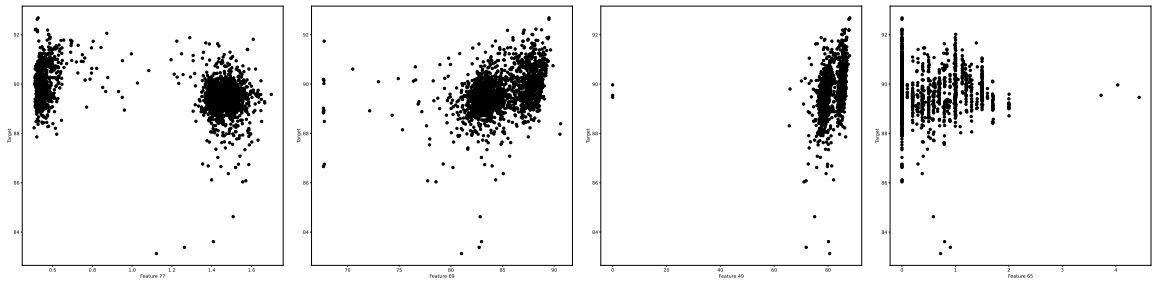
Figure 8. Features ranked using all the importance metrics described in this report.

In this report, we discussed several techniques which can be employed to evaluate feature importance for target variable prediction with deep neural network models and evaluated those using the pulp bleaching dataset. All the code used to train neural network models and plot importance scores can be found in [Zolotukhin 2021]. In order to summarize the results, we rank features present in this dataset using all the techniques described in the report. For this purpose, we first standardize correlation and feature importance coefficients calculated in the previous sections using min-max standardization and sum the resulting values in order to calculate the final score for each feature:

$$S_i = \sum_{k=1}^{N} \frac{s_{ki} - \min_i(s_{ki})}{\max_i(s_{ki}) - \min_i(s_{ki})}, \tag{4.1}$$

where S_i is total importance score for the i-th feature, N is the number of metrics used for feature importance calculation, s_{ki} is importance of the i-th feature according to the k-th metric, and n is the number of features in the dataset. It is worth mentioning that in the case when importance is evaluated by predicting the target variable with only one feature, score s_{ki} can be equal to the multiplicative inverse of the prediction error, since the lower the prediction error of such model the higher importance of the corresponding feature. For the rest of the cases, score s_{ki} is simply equal to the i-th feature importance value or its absolute

value in case of Pearson and Spearman correlations. The result can be found in Figure 8. As one can notice, the feature which stands out is feature 77 from class 5. The next three features are feature 69, feature 49 and feature 65, all of which are from class 4. These features are plotted in Figure 9.



(a) Feature 77 (b) Feature 69 (c) Feature 49 (d) Feature 65

Figure 9. Values of the most important features plotted against values of the target variable.

Bibliography

Fisher, Aaron, Cynthia Rudin, and Francesca Dominici. 2019. *All Models are Wrong, but Many are Useful: Learning a Variable's Importance by Studying an Entire Class of Prediction Models Simultaneously.* arXiv: 1801.01489 [stat.ME].

Graves, Alex. 2014. *Generating Sequences With Recurrent Neural Networks.* arXiv: 1308.0850 [cs.NE].

Guyon, Isabelle, and André Elisseeff. 2003. "An Introduction to Variable and Feature Selection". *J. Mach. Learn. Res.* 3, number null (): 1157–1182. ISSN: 1532-4435.

Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. "Long short-term memory". *Neural computation* 9 (8): 1735–1780.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton. 2012. "ImageNet Classification with Deep Convolutional Neural Networks". In *Advances in Neural Information Processing Systems,* edited by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, volume 25. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Lei, Jing, Max G'Sell, Alessandro Rinaldo, Ryan J. Tibshirani, and Larry Wasserman. 2017. *Distribution-Free Predictive Inference For Regression.* arXiv: 1604.04173 [stat.ME].

Mentch, Lucas, and Giles Hooker. 2015. *Quantifying Uncertainty in Random Forests via Confidence Intervals and Hypothesis Tests.* arXiv: 1404.6473 [stat.ML].

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. *Playing Atari with Deep Reinforcement Learning.* arXiv: 1312.5602 [cs.LG].

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. 2013. "On the difficulty of training recurrent neural networks". In *International conference on machine learning,* 1310–1318. PMLR.

Wojtas, Maksymilian, and Ke Chen. 2020. *Feature Importance Ranking for Deep Learning*. arXiv: 2010.08973 [cs.LG].

Zolotukhin, Mikhail. 2021. *Metsa Group*. https://github.com/mizolotu/metsa_group.

Appendices

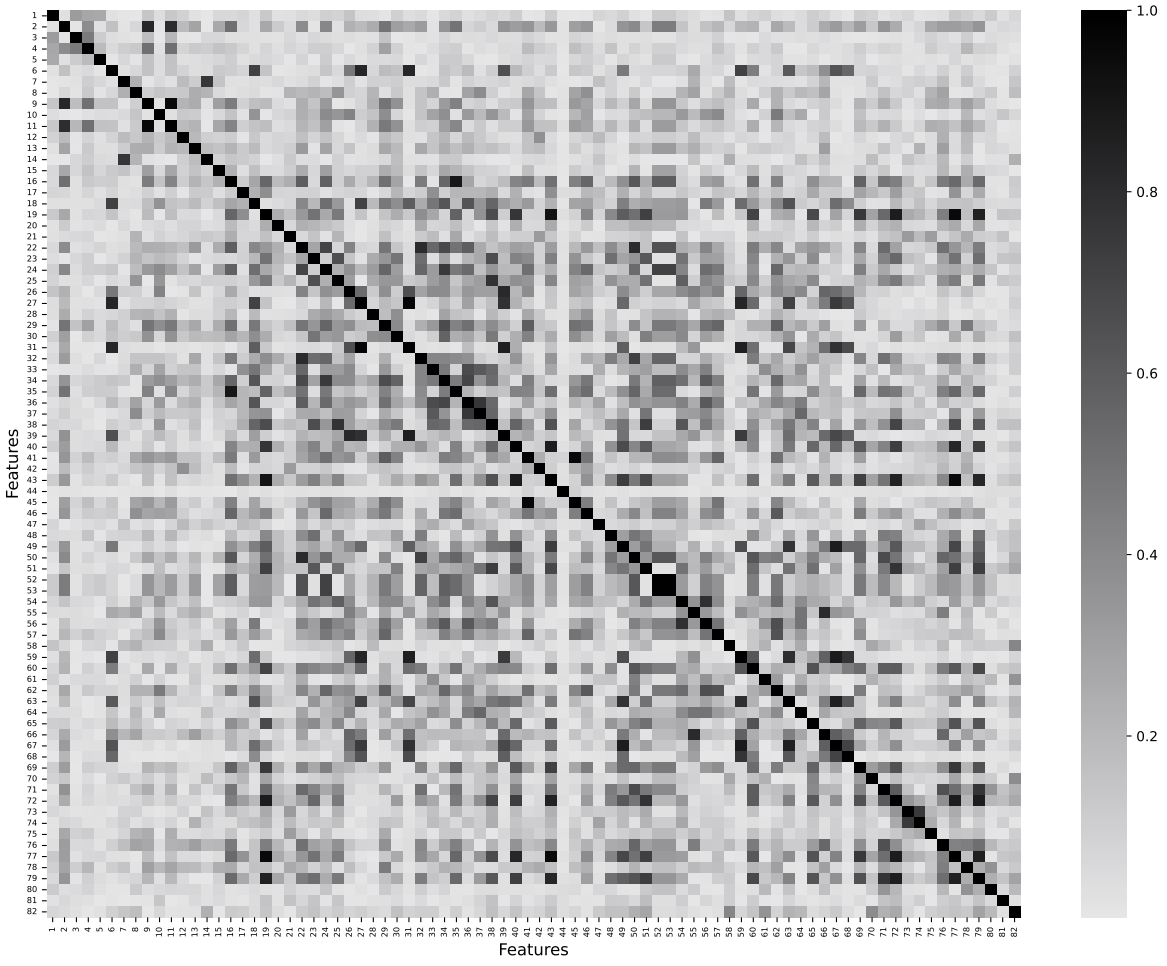


Figure 10. Pearson correlation for each pair of features.

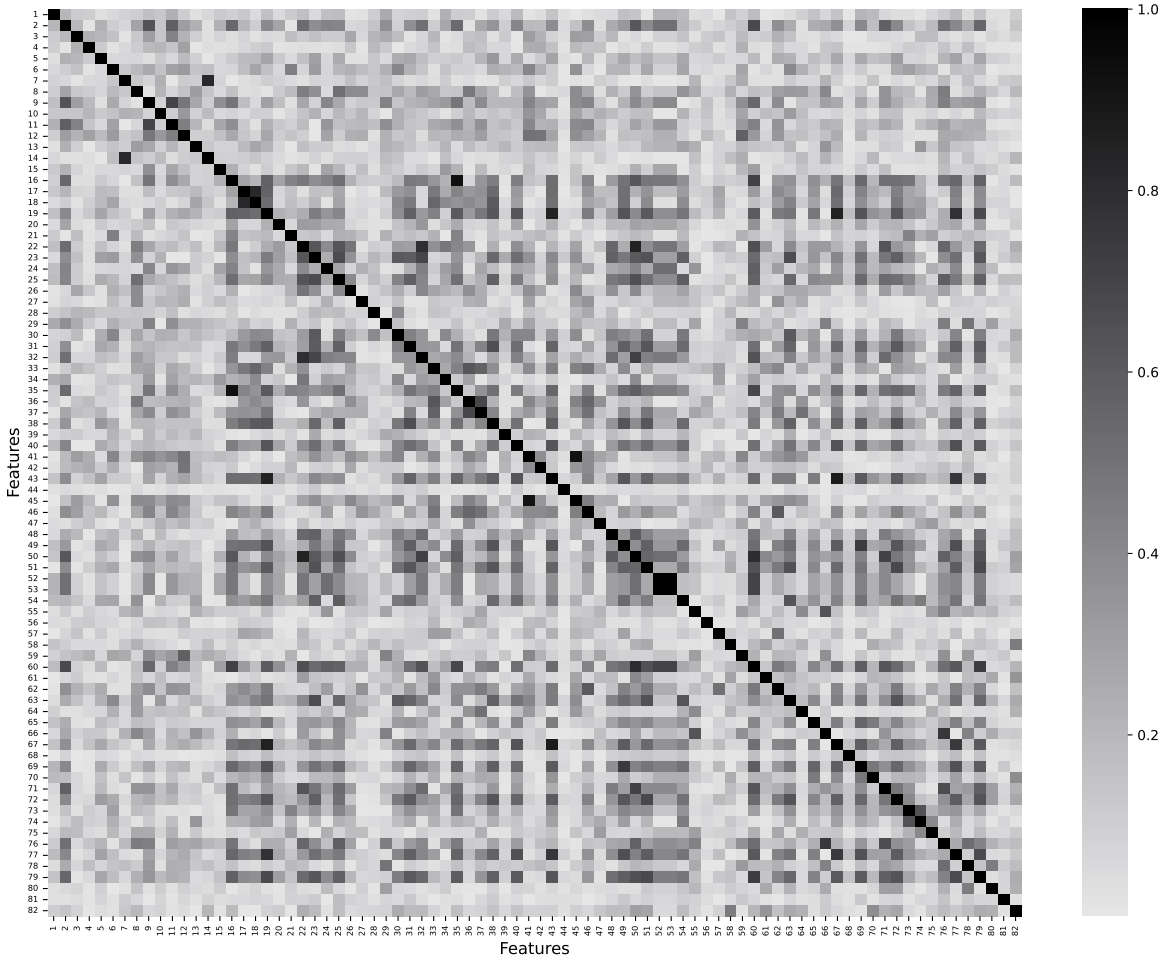


Figure 11. Spearman's rank coefficient correlation for each pair of features.