CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Insight Into Driver Behavior and Usage of ADAS Functions Using Machine Learning

Master's thesis in Data Science and AI

Margarita Antonia Psychountaki & Rajath Rajendra Pai

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

Insight Into Driver Behavior and Usage of ADAS Functions Using Machine Learning

Margarita Antonia Psychountaki
Rajath Rajendra Pai

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Insight into driver behavior and usage of ADAS functions using machine learning

MARGARITA ANTONIA PSYCHOUNTAKI
RAJATH RAJENDRA PAI

Supervisor: Ashkan Panahi, Data Science and AI, Computer Science and Engineering
Advisor: Samin Dehghani, Rached Dardouri, Volvo GTT
Examiner: Ashkan Panahi, Data Science and AI, Computer Science and Engineering

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2024

Insight into driver behavior and usage of ADAS functions using machine learning

MARGARITA ANTONIA PSYCHOUNTAKI
RAJATH RAJENDRA PAI
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Advanced Driver Assistance Systems (ADAS) is a technology that prevents road accidents and improves driving safety. Front-Short Range Assist is a newly released function that alerts drivers of any vulnerable road users in the front area of the truck when it is moving at low speed.
This thesis aims to analyze and predict the deactivation of this function by the drivers, using recorded vehicle data from trucks and external weather and road type data. Several models were utilized to answer these questions: CNN, LSTM, a hybrid CNN-LSTM, and a Bi-LSTM autoencoder. The hybrid model achieved the highest performance. However, all models yielded poor predictive power, making it unclear whether the given dataset has a discernible pattern to predict and understand the reasons behind the deactivation.

Keywords: Data science, Machine learning, LSTM, CNN, ADAS, Deep learning.

Acknowledgements

We would like to express our deepest gratitude to our academic thesis supervisor, Ashkan Panahi, for his continuous guidance throughout this work. We also extend our sincere thanks to our company advisors Samin Dehghani and Rached Dardouri, as well as to Therese Gardell, Anton Jansson, and the entire team at Volvo for their invaluable help and support during the thesis.

Margarita Antonia Psychountaki and Rajath Rajendra Pai, Gothenburg, August 2024

Contents

List of Figures

List of Tables

1
Introduction

According to the World Health Organization (WHO), approximately 1.3 million individuals lose their lives annually due to road traffic accidents, while between 20 to 50 million sustain non-fatal injuries as a result of them [1]. These statistics highlight the significance of prioritizing road safety in the automotive industry.

There are two main categories of safety in the automotive industry: passive and active. Traditionally, the automotive industry has focused on passive devices and systems. These features are part of the vehicle and aim to protect the passengers in the case of an accident, such as seat belts, airbags, etc. In recent years and with the advance of technology, the automotive industry has shifted its focus to active safety, known as Advanced Driver-Assistance Systems (ADAS). In this approach, the developed systems aim to prevent the collisions.

ADAS uses automated technology, such as sensors and cameras, to detect obstacles or driver errors and prevent potential accidents. There are a plethora of functions utilized by ADAS. This work focuses on *Front Short Range Assist* (FSRA), a function responsible for the detection of Vulnerable Road Users (VRU) in the near range of the front of the truck. It has been noticed that drivers tend to deactivate this function. In this master thesis, we attempt to discover the motivation behind such behavior by exploring the relationship between the output of the truck's sensors and FSRA deactivation. Our goal is to discover the motivation behind such behavior by exploring the relationship between the truck's recorded signals and FSRA deactivation.

1.1 Front Short Range Assist [2]

FSRA is a system that alerts drivers to VRU, such as pedestrians and cyclists, in the near range of the front of the truck when the truck is moving at low speed. The function utilizes the output of a camera and multiple radars positioned in the front and side sections of the truck as input (Figure 1.2) and interacts with the driver by issuing warnings or alerts when a VRU is detected within the information and/or warning zone in front of the truck. In Figure 1.1, two typical scenarios, are displayed.

When the truck is standing still or moving slowly, the system monitors both zones, shown in Figure1.2. If a VRU enters information zone (1), the system informs the driver with a red alert light on the windscreen. If a VRU is detected in the warning

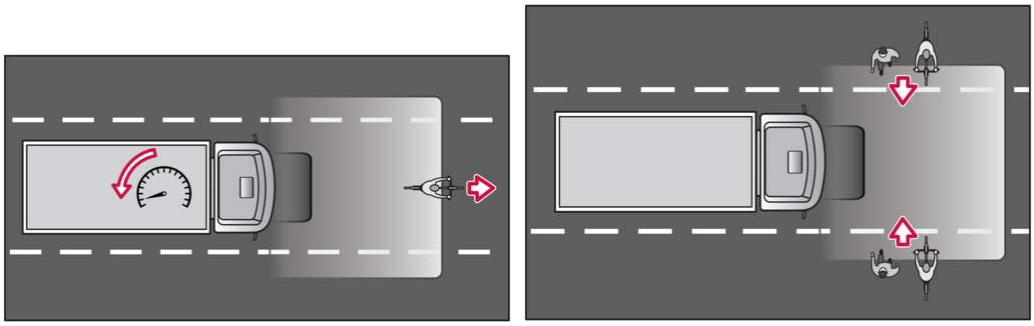


Figure 1.1: Two typical scenarios of FSRA activation: Bicycles in front of the truck, Crossing pedestrians or bicycles.

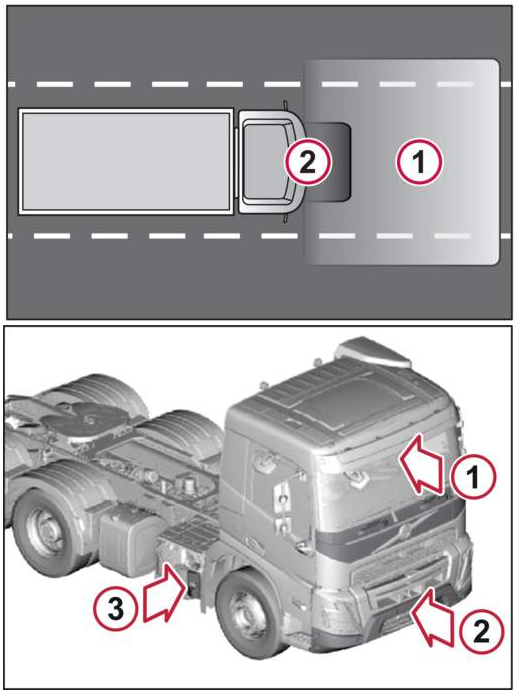


Figure 1.2: Left image: 1. Information zone 2. Warning zone Right image: Location of sensors. The truck has sensors on both sides.

zone (2), the driver is notified by a red warning light on the windscreen and a warning sound. The driver can turn off the system by pressing the corresponding switch. The driver has to press and hold the switch to deactivate the system.

1.2 Project Aim

As mentioned above, the goal of this thesis is to uncover the patterns of the driver's decisions to deactivate the FSRA function. Specifically, using multivariate time series data from truck and external weather and road-type datasets, we aim to develop a predictive model that can anticipate this behavior.

To the best of our knowledge, no previous study has attempted to answer this

question, since FSRA is a newly released feature. Despite the absence of prior research that addresses our specific question, there are well-studied techniques that can be applied to our problem. The problem at hand can be framed as a Multivariate Time Series Classification (MTSC) problem. Various algorithms have been proposed to address this challenge, and in our project, we will concentrate on four deep-learning approaches. As a baseline model, we utilize logistic regression, where we use as an input manually extracted features from the multivariate time series dataset. We implement three types of classifiers that employ a two-step pipeline. The first step comprises an encoder part followed by a fully connected neural network in the second step. We experiment with three kinds of encoders: a Convolutional-based (CNN) encoder, a Recurrent Neural Network (RNN) based encoder, and a concatenation of the outputs of a CNN-based and an RNN-based encoder. For the CNN-based encoder, we experimented with classical and dilated convolutional layers, and for the RNN-based encoder we utilized Long-Short Term memory (LSTM), Gated Recurrent Unit (GRU), and Bidirectional LSTM.

A primary problem we should deal with is highly different numbers of samples in different classes, an issue known as imbalanced datasets. We explore four different approaches to tackle this challenge: class weighting, oversampling the minority class, downsampling the majority class, and ensemble learning combined with undersampling. Additionally, we try to enhance the performance of our classifiers using pretrained autoencoder weights. In this setup, we ignore the class labels and train an autoencoder of which the encoder part corresponds to the encoder part of our classifier. We initialize or freeze the weights of the encoder part in our models to leverage the knowledge learned during pretraining.

Finally, we employ a bidirectional-LSTM-based autoencoder trained to reconstruct the majority class's signals. In this approach, we assume that in the minority class instances (deactivation by the driver) the driver will exhibit a distinct behavior and/or the environmental conditions will differ compared to cases where there is no deactivation (majority class). Thus, these samples can be regarded as anomalous cases where the model struggles to accurately recreate the original signal, leading to a high reconstruction error.

The primary goal of this thesis is to explore the relationship between the recorded signals of the trucks' sensors and FSRA deactivation. We leverage deep learning models to investigate whether there is a relationship that can help us predict this behavior. Our primary questions are:

1. Can any of the models discover patterns in our data that can distinguish the two classes?

2. Which algorithm demonstrates the best fit for our dataset?

3. If we can distinguish the two classes, which features are more dominant?

1.3 Related work

MTSC has gained popularity in various domains such as clinical diagnosis, stock price prediction, fault detection in manufacturing processes, prediction of hazardous driving situations, etc. Deep Neural Network approaches have successfully been applied in this class of problems. In [3], authors introduced a deep learning framework known as Multi-Channel Deep Convolutional Neural Networks (MC-DCNN). This model learns features from individual univariate time series in each channel and subsequently consolidates information from all channels to create a feature representation at the final layer. The learned representation is then employed as an input to a multilayer perceptron (MLP) for classification. Given the sequential nature of the data, Recurrent Neural Network (RNN) architectures are also proposed for this task. In [4], the authors proposed a long short-term memory fully convolutional network (LSTM-FCN) for the uni-variate time classification problem and explored the usage of attention mechanism with an attention long short-term memory fully convolutional network (ALSTM-FCN). The architecture of the network consists of two blocks: a fully connected convolutional and an LSTM block. The output of the two blocks is concatenated and passed onto a softmax classification layer. In the case of ALSTM-FCN, the LSTM block is replaced with an Attention LSTM. The proposed models managed to outperform the previous state-of-art architectures at least in 43 out of 85 UCR time series datasets. In [5], the authors extend the previous work in MTSC. The two models outperform in the 28 out of 35 datasets used. In [6], the authors combine Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) architectures. Specifically, 1D-CNN layers are utilized for feature extraction, and these features are subsequently passed to the LSTM layers. Similar approaches have been utilized to predict hazardous driving cases using similar data from truck sensors.
Given the imbalance of the dataset, another approach that could be utilized is the detection of rare events. In [7] the authors proposed the training of autoencoder on majority class samples and identifying positive samples during prediction, expecting that positive samples will yield a high reconstruction error.

1.4 Limitations

This project has two primary limitations. Firstly, the dataset comprises data gathered in the European region and the US, meaning that the distribution of the data is aligned with traffic conditions and regulations in these specific areas. Thus, the findings may not apply to different regions due to different driving cultures and behaviors. Secondly, we aim to infer the behavior of the driver using data collected by the sensors. It is important to note that while these signals are good indicators of the driver behavior, we do not observe the driver's behavior.

1.5 Risk analysis and ethical considerations

As we mentioned in Section 1.4, the collected data cannot be used to conclude all regions worldwide. The results and conclusions are applicable and specific only to the European region from where we collected the data.
There are no ethical considerations. The data used for this project has been anonymized, preventing the identification of drivers. Additionally, the drivers are aware of the purposes of the data collection, and they have provided their consent. Finally, the data is used by us only for the specific purpose described in this thesis.

1.6 Notation

In this work, scalars are denoted by a small letter, for example, x, vectors are denoted by a small bold letter, for example, $\mathbf{x}$ and matrices are denoted by a capital bold letter, for example, $\mathbf{X}$. In our thesis, $||\mathbf{X}||_F$ denotes the Frobenius norm of matrix $\mathbf{X}$.

We define a univariate time series as a vector $\mathbf{t} = [t_0, t_1, \ldots, t_{n-1}]^T$, which corresponds to a sequence of consecutive data points recorded at uniform intervals over time with n indicating the length of the time series.

A multivariate time series is a set of univariate time series sampled at the same time. We define it as a nxm matrix, $\mathbf{T}$ where n represents the length of the sequences and m is the number of the different univariate time series.

1.7 Outline

The thesis is organized as follows:

- In Chapter 2, we provide an overview of the basic concepts of deep machine learning and the main layers used in our architectures.
- In Chapter 3, we provide an overview of the dataset and the preprocessing steps.
- In Chapter 4, we focus on the implemented methods.
- In Chapter 5, we present the results of models across different settings.
- In Chapter 6, we discuss the obtained results, provide some insights, and propose future work.

2
Theory

2.1 Deep Learning

Deep Learning is a class of machine learning based on artificial neural networks organized into multiple layers and aims to learn representations of data. It has been proven extremely successful in solving complex problems across various domains, such as natural language processing, image, and signal processing. These methods can be applied to supervised, unsupervised, and semi-supervised tasks. In this work, we will use them for the first two.

2.1.1 Supervised Learning

Supervised Learning is a part of machine learning where the algorithm learns from labeled data. The data consists of input-output pairs. The goal of supervised learning is to learn the mapping between the input variables to output variables.
In supervised learning, the data will have an input vector $\mathbf{x}$ and will have a target output $\mathbf{y}$. For a new input $\mathbf{x}$ the algorithm learns to predict the output $\mathbf{y}$ based on the patterns that it has learned from the training data.

2.1.2 Unsupervised Learning

Unsupervised Learning is a type of machine learning where the algorithm will learn patterns from unlabeled data without any supervision. The primary goal is to find the hidden patterns within the data without any explicit guidance or instruction.

2.2 1D Convolutional layer

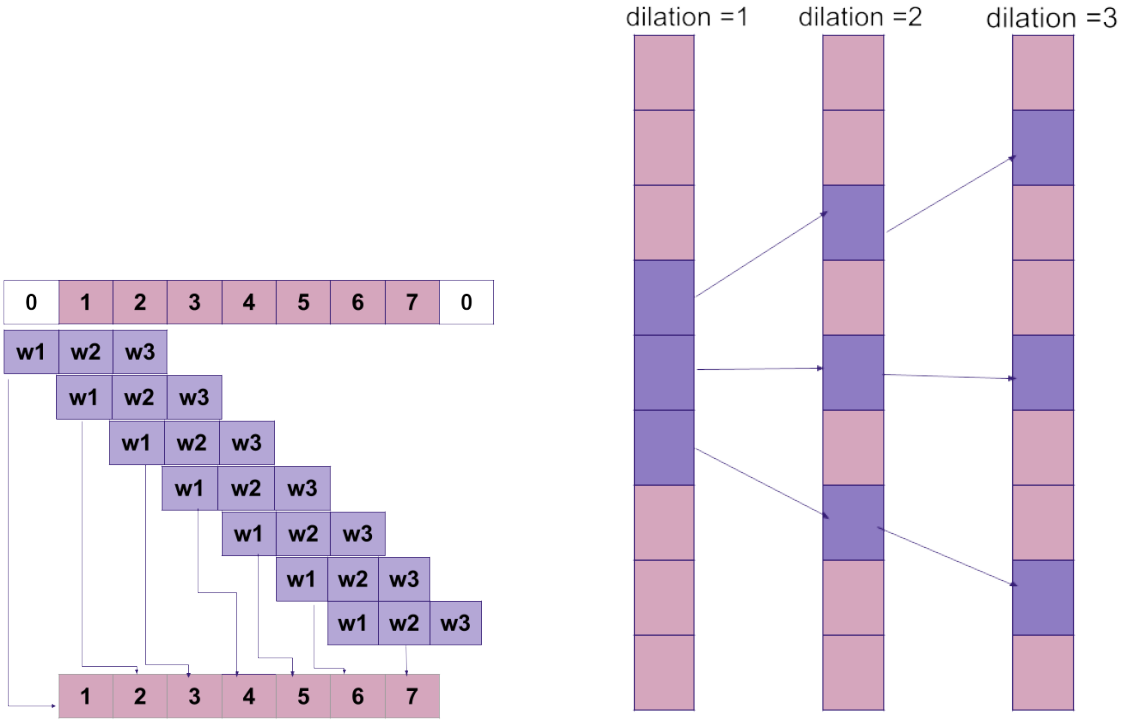
2.2.1 Standard

A 1D convolutional layer applies a convolutional operation to one-dimensional data. A kernel (or convolutional filter) slides across the input sequence. At each step of the process, the dot product of the weights of the kernel with the corresponding part of the input is calculated, producing a single point of the output sequence. This process is illustrated in Figure 2.1. The kernel uses shared weights for all parts of the input. Thus, it captures time shift-invariant features, namely, features that are the same across all the time series.

The size of the kernel is a hyperparameter, and the kernel itself is a set of learnable parameters that the model adjusts during the training. Another hyperparameter of the convolutional layer is the stride which indicates the number of time units the kernel moves at each step.

2.2.2 Dilated CNN

The difference between a standard and a dilated convolutional layer is the receptive field of the kernel, achieved by skipping timestamps in the input sequence. In this way, it is able to capture longer-time dependencies without increasing the number of parameters. In this case, there is an extra hyperparameter, dilation which indicates the number of the timestamps that are skipped. We provide an illustration in Figure 2.1.



We illustrate how the convolution operation is applied in an input sequence, where the kernel has a size of 3. Source [8].

We illustrate which parts of the input sequence a kernel with a size of 3 uses to produce one output point for different dilation parameters. Source [9].

Figure 2.1: Standard convolution and dilated convolution operations.

2.3 1D Max Pooling layer

1D Max pooling layer reduces the size of the input sequence while retaining the most dominant features. The 1D Max Pooling layer gets as input the output of a 1D convolution layer and divides it into consecutive no-overlapping sections. From

each section, it keeps the larger value. In this way, it compresses the information of each section in a single value. The size of the sections/pooling window is defined by our architecture and it is a hyperparameter that can be tuned. We illustrate an example in Figure 2.2.

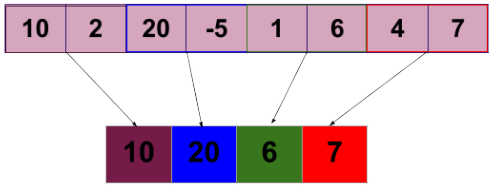


Figure 2.2: 1D max pooling layer with pooling window 2.

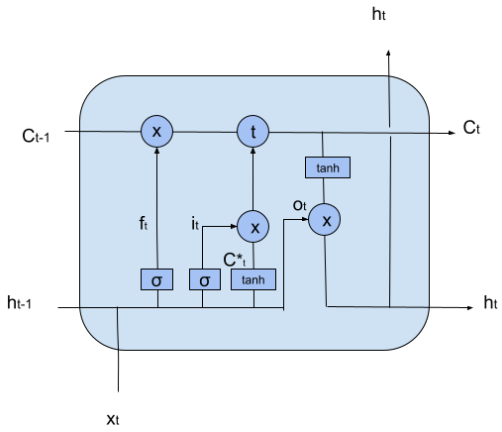
2.4 Recurrent Neural Network

Recurrent Neural Networks (RNNs) are models specifically designed to deal with sequential data. Standard NN architectures consider all inputs and outputs independent of each other, meaning they do not have memory. RNNs introduced hidden layers that can retain information from previous inputs. However, RNNs suffer from the exploding or vanishing gradient problem, where the gradient either becomes very high or very small, and models are not able to learn anymore [10]. Additionally, RNNs do not work well for long sequences. Two variations of RNNs, Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), have been proposed to address these problems.

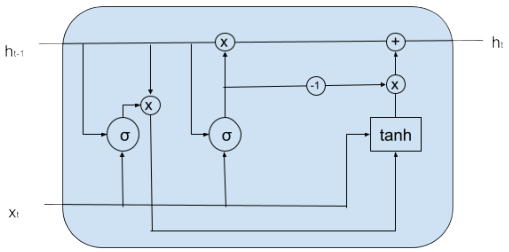
2.4.1 LSTM

The key component of the LSTM architecture [11] is the cell state. The LSTM uses gates to add or remove information from it. A gate is a structure that consists of a sigmoid layer and an element-wise multiplication component that combines the output of the gate and the input. The gates control the flow of information through them.
The LSTM architecture makes use of three gates.

1. Forget gate: the first step is to decide what information of the previous state needs to be forgotten/removed from the cell state C_{t-1}. The forget gate takes as input the current input and the hidden state from the previous timestamp and passes them through a sigmoid layer. The output is a number f_t between 0 and 1 which will be multiplied by the cell state of the previous timestamp C_{t-1} to determine how much of the C_{t-1} will be retained. If f_t equals 1, no information of C_{t-1} is forgotten; if it is 0 it implies complete forgetting.
2. Input gate: The next step is to decide what new information will be included in the cell state. This process is composed of two parts. The first part is input gate i_t which quantifies the importance of the new information, by passing the current input x_t and the previous hidden state h_{t-1} through a sigmoid layer. Then, by passing the current input x_t and the previous hidden state



We illustrate the architecture of LSTM. Source [12].



We illustrate the architecture of GRU. Source [13].

Figure 2.3: Comparison of LSTM and GRU architectures.

h_{t-1} through a *tanh* layer, which decides which part of the new information is needed. The output of the *tanh* layer is multiplied with the importance weights at the output of the input gate. Adding these new values with the outcome of the element-wise multiplication of f_t with C_{t-1} generates the new cell state C_t.

3. Output gate: After the calculation of the new cell state, what information will be passed to the output has to be determined. The first part is the output gate o_t which decides which part of the current input x_t and the previous hidden state h_{t-1} will be passed to the output by passing them through a sigmoid layer. Then, an element-wise multiplication is performed between the output of the output gate and the output of a *tanh* layer which takes as input the new cell state C_t.

2.4.2 GRU

Similar to LSTM, Gated Recurrent Unit (GRU) [14] also uses gates to control the flow of information, but it has a simpler architecture. As illustrated in Figure 2.3, the GRU does not use cell state only the hidden states. It utilizes two gates: reset gate (regulating short-term memory) and update gate (balancing short and long-term dependencies). Both gates take as input the current input x_t and the previous hidden state h_{t-1} and pass them through a sigmoid layer. The only difference is the use of different weight metrics.

To produce the next hidden state, two steps are needed. First, a candidate hidden state h^* is calculated. It is the output of a *tanh* layer which takes as input the summation of the current input x_t and the output of the element-wise multiplication of the output of reset gate with previous hidden h_{t-1}. Using the reset gate it is decided how important the previous hidden state is.

Then, the next hidden state is calculated as the summation of the output of the element-wise multiplication of the output of update gate with previous hidden h_{t-1} with the output of the element-wise multiplication of the output of (1 - update gate) with candidate hidden h^*. Based on this formula, if the update gate is close to 0, the influence of the previous hidden state will almost vanish and the next hidden state will rely solely on the candidate's hidden state. On the other hand, if the update gate is close to 1, the influence of the candidate's hidden state will almost vanish and the next hidden state will rely solely on the previous hidden state.

2.5 Bi-LSTM

Unlike traditional RNNs, bidirectional LSTM processes sequential data in two directions: one forward and one backward [15]. Specifically, it contains two LSTM layers. In one layer, the input sequence is fed as it is (forward pass), and in the other layer, the reversed input is fed (backward pass). Once the two passes are complete, the hidden states from both LSTM layers are combined at each time step.

2.6 Fully Connected Layer

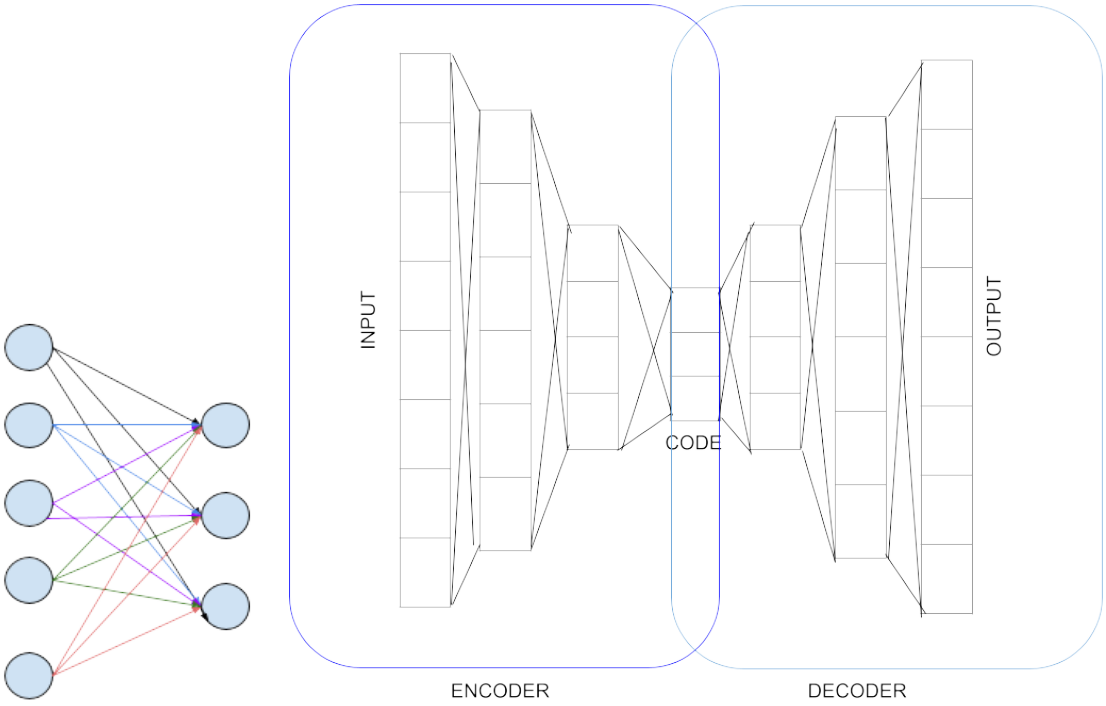
A fully connected layer connects each input neuron to each output neuron, linking every neuron in the input layer to each neuron in the output layer. As it is depicted in Figure 2.4. When multiple fully connected layers are stacked sequentially, a fully connected network is formed.

2.7 Autoencoder

An autoencoder is an unsupervised neural network that efficiently learns a representation of a dataset, typically in a lower dimension. It consists of three key components: encoder, code, and decoder (Figure 2.4). The encoder part compresses the input into a lower-dimensional latent space representation. This representation, called code, is fed to the decoder part. The decoder reconstructs the code to the original input. It is a lossy procedure. If the dimensionality of the code is lower than the original input, the autoencoder is called undercomplete; otherwise, it is called overcomplete. Typical uses of autoencoders include feature extraction, data denoising, anomaly detection, and generative tasks [16].

2.8 Cross-validation

Cross-validation is a method that provides a better estimate of the model's performance on unseen data. There are several reasons for performing cross-validation, such as hyper-parameter tuning and achieving a more robust evaluation of the model. The main idea of cross-validation is simple: it involves dividing the dataset into several folds or subsets. Then, the model is evaluated on one of the folds while the rest are used for training. The process is repeated multiple times, each time using



(a) Illustration of a fully connected layer.

(b) General architecture of an Autoencoder.

Figure 2.4: Illustrations of different architectures.

a different validation set. The results from each validation step are averaged to produce a more reliable estimate of the model's performance on unseen data.

There are several types of cross-validation. In this thesis, K-fold Stratified cross-validation and Hold-out cross-validation were used.

In K-fold stratified cross-validation, the dataset is divided into K folds such that in each fold the distribution of the classes is the same as the entire dataset. Then, k-1 folds are used as training and the remaining fold is used as validation. The validation of the model is repeated k times. Each time a new validation set is used. The results from each validation step are averaged to provide a final performance estimate.

In hold-out cross-validation, the dataset is randomly split into three sets: train, validation, and test set. The training set is used to train the model, validate to tune the parameters, and test to evaluate the model on unseen data.

2.9 Hyper-parameter tuning

The hyper-parameters of a machine learning model are essential for the performance of the model. The hyperparameters of the model are not learnable during the training like model parameters, but they are an integral part of the architecture. There are several methods to determine a good set of hyperparameters, such as grid search cross-validation and randomized search cross-validation. In this thesis, grid search cross-validation was used.

In grid search cross-validation, a hyperparameter search space is defined by the combination of all available values. Then, the grid search fits and evaluates the model using cross-validation for each combination. The combination that yields the best performance is selected.

2.9.1 Feature Importance

Feature importance corresponds to a score for each feature, indicating how much each feature contributes to the predictive power of the model. There are two categories of techniques to determine the feature importance: model-dependent and model-agnostic. In the first case, the score depends on the model. For example, in the case of Logistic Regression, the magnitude of the coefficients of the model corresponds to the feature importance score. In the second approach, the techniques can be applied to any model.

In this thesis, we use permutation feature importance [17]. The idea behind of the permutation feature importance is quite simple. The trained model is first used to compute the evaluation metrics. Then, in the initial test set, one feature at a time is randomly shuffled and then evaluation metrics are calculated for the modified test set. The impact on the evaluation metric indicates how important each feature is for the model. We define the score for each feature as:

$$FIS = Metric_{\text{shuffled feature dataset}} - Metric_{\text{initial dataset}}.$$

2.10 Evaluation

2.10.1 Supervised Learning

For the evaluation of supervised learning techniques, the most widely used metrics are accuracy, precision, recall, F1-score, ROC-AUC, and PR-AUC. However, the accuracy score can be misleading, especially in the case of unbalanced datasets. For instance, let's assume a binary classification problem where 90% of the samples correspond to class 0. If the model assigns all observations to class 0, the accuracy will be 90% (if the test set contains 90% samples from class 0). Additionally, ROC-AUC can be misleading for highly imbalanced datasets because it uses true negatives for calculating the false positive rate (FPR) (defined below). Even if there are many false positives, it yields a very low FPR, providing an overly optimistic picture of

the model's performance. Therefore, we place greater emphasis on precision, recall, F1-score, and PR-AUC to obtain a more accurate evaluation and a deeper understanding of the algorithms' performance.

We are dealing with a binary classification problem, so there are four possible outcomes:

- True Positive (TP): The model correctly predicts the label of a positive sample.
- False Positive (FP): The model classifies a negative sample as positive.
- True Negative (TN): The model correctly predicts the label of a negative sample.
- False Negative (FN): The model classifies a positive sample as negative.

The six metrics are defined as follows:

- Accuracy = $\frac{TP+TN}{TP+FP+FN+TN}$, t shows the percentage of correct predictions out of all predictions.
- Recall = $\frac{TP}{TP+FN}$, It measures the proportion of correct predictions for positive samples out of true positive samples. It indicates how many of the total true positives the model managed to classify correctly.
- Precision = $\frac{TP}{TP+FP}$, It measures the proportion of correct predictions for positive samples out of all samples classified as positive by the model. It indicates how much we can trust a positively classified sample.
- F1-score = $\frac{2xRecallxPrecision}{Recall+Presicion}$, it corresponds to the harmonic mean of precision and recall.
- ROC-AUC (Receiver Operator Characteristic curve -Area Under the Curve): The ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) across different threshold values. TPR is calculated as $\frac{TP}{TP+FN}$ and FPR as $\frac{FP}{TN+FP}$. The AUC measures the overall ability of the binary classifier to distinguish between classes summarizing the ROC curve.
- PR-AUC (Precision-Recall curve - Area Under the Curve): The PR curve plots precision against recall across different threshold values. The AUC measures the overall ability of the binary classifier to distinguish between classes by summarizing the precision-recall curve. It focuses on the model's ability to identify positive instances and provides an evaluation of its performance at various thresholds.

2.10.2 Unsupervised Learning

In this thesis, we utilize an autoencoder architecture as an unsupervised technique. The autoencoder is trained to minimize the mean reconstruction error. Therefore, the performance of the autoencoder is evaluated by the Average Relative Reconstruction Error (ARRE), which is defined as:

$$ARRE = \frac{1}{Number\ of\ samples} \sum_{i \in samples} \frac{||Reconstructed\ sample - Original\ Signal||_F^2}{||Original\ sample||_F^2}.$$

ARRE is an indicator of how different the initial sample and the reconstructed one are. A low ARRE indicates that the autoencoder effectively compresses the input while retaining the overall information.

2.11 Overfitting

A common problem in machine learning is overfitting. Overfitting occurs when the model performs well on the training set, but poorly in unseen data. In this case, the model cannot be generalized, which is the goal of ML applications. Essentially, it has memorized training samples instead of learning the underlying patterns. Underling causes of overfitting could include insufficient data, a model that is too complex for the given task, and data containing errors and fluctuations that the model misinterprets as patterns. Ways to mitigate overfitting include regularization, obtaining more data, early stopping, dropout, and other techniques.

3
Data

The dataset comprises two types of information: details about the vehicle's state and logs collected from three key sensors - namely, the forward-looking camera, the forward-looking radar, and the side radars. The data originates from field test trucks, in which both customers and Volvo employees adhered to their daily routines, creating a non-testing environment. Two types of recordings are available, and they are explained as follows:

Continuous Log Files

In this category, logs spanning the entire duration of a trip are recorded. However, it is important to note that only a subset of available signals is captured. The number of rows in these files varies depending on the duration the truck remained operational. The columns number between 114 and 131, depending on the truck's architecture.

Trigger Event Files

Trigger event files are specifically designed to capture data in response to significant occurrences. They record the 30 seconds before and after a trigger event, promoting the recording of all possible signals.

This diverse dataset provides a comprehensive view of the vehicle's behavior, capturing both continuous logs over the entire trip and detailed information surrounding specific trigger events. However, for this work, we make use only of Continuous Log files. As we describe above, a trigger event file is generated based on the activation of specific ADAS functions. However, if a function is deactivated by the driver the trigger files for this function cannot be created. Since we study the deactivation of FSRA this type of data is not useful for our study.

3.1 Data exploration

As mentioned, each continuous log file contains several signals related to the functions, the key point sensors, and the state of the vehicle. Among these signals, there is one related to the state of the FSRA function. We call this signal FSRA Status. FSRA Status can take seven discrete values, Table 3.1.

Value	Interpretation
Deactivated By Driver	The FSRA function has been purchased for this truck and it is deactivated by the driver.
Enable and Inactive	The FSRA function has been purchased for this truck, but due to specific conditions such as speed and gear position, the function remains inactive.
Enable and Active	The FSRA function has been purchased for this truck and it is active.
Information Alert	The FSRA function has been purchased for this truck and the driver gets an information zone alert.
Warning Alert	The FSRA function has been purchased for this truck and the driver gets a warning zone alert.
Error	The FSRA function has been purchased for this truck, but some of the sensors do not operate properly.
Not available	The FSRA function has not been purchased for this truck.

Table 3.1: FSRA Status

Our first step is to find the deactivations by the drivers in those files. We define a deactivation as a consecutive sequence of *Deactivated By Driver* values of the FSRA Status until interrupted by a different value. Thereby, a file can have none, one, or more deactivations. We found 5218 deactivations among all files. Deactivations were found in only 3755 out of the 37,697 files. In Table 3.2, we present how many deactivations each file exhibits, given that it has at least one.

Number of deactivations:	1	2	3	4	5	6	7	8	9
Number of files:	3184	341	112	46	28	16	13	3	3
Number of deactivations:	10	11	12	13	18	21	75	284	-
Number of files:	2	1	1	1	1	1	1	1	-

Table 3.2: Number of deactivations per file, given that there is at least one deactivation

Additionally, we examine the start timestamp of each deactivation. We know, that if there are deactivations within the first 3 seconds of the file, they cannot be done by the driver, as it requires a long push to switch off the button. As we can observe from Figure 3.5, approximately 25 % of positive samples are invalid for our study case.
So, we removed these samples from our dataset. Also, it is possible for a file to contain multiple deactivations. We examine the time interval between consecutive deactivations. In Figure 3.5, we can see that 1 % of the next deactivations cannot be done by the driver, as the time from the end timestamp of the previous deactivation to the start timestamp of the next deactivation is less than 3 seconds. We also removed these samples.

Another interesting statistic is the duration of the deactivation. We provide the results in Figure 3.1. As we can see, more than 50% of the deactivations last less than 5 min. Additionally, it is notable that the 20% of the deactivations last for 3 seconds.

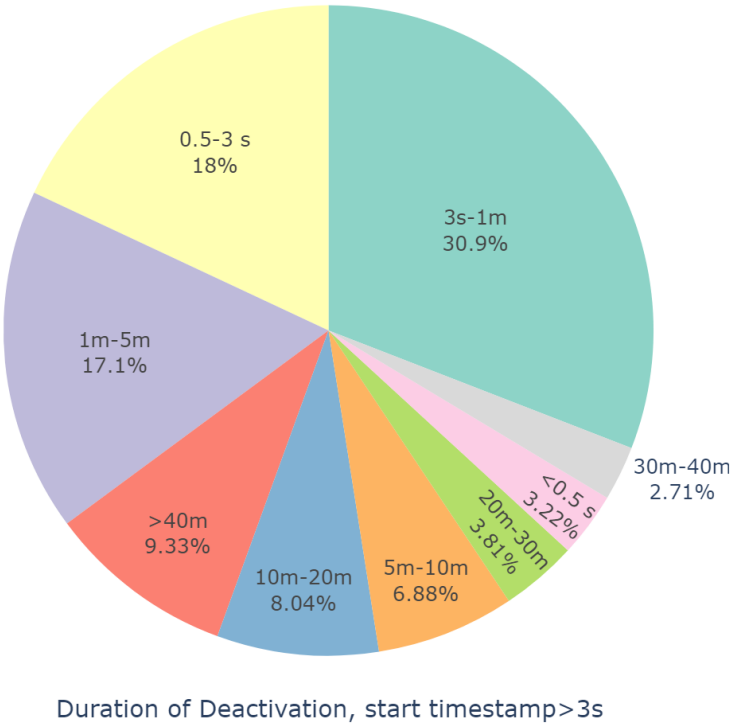


Figure 3.1: Duration of deactivation start timestamp greater than 3s

We are also interested in observing the status of FSRA just before the deactivation and immediately after the reactivation. We present the results in Figure 3.2. We observe that in the majority of the cases, the drivers choose to deactivate the FSRA when it would not have been activated anyway since the status of FSRA is Enable and Inactive. Only in 16.9% of cases, do the drivers deactivate the function when the status of FSRA is Enable and Active. Additionally, it is worth noting that the drivers never deactivate the function at the moment they receive a warning alert and only in 0.183% of cases when they receive an information warning. Also, we observe that in 0.209% of the cases, the previous status of FSRA was not available, which is not possible since the functionality has not been purchased.

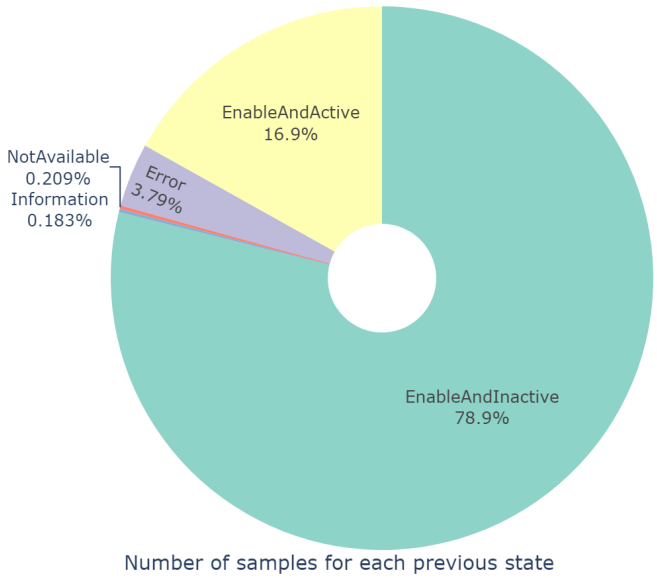


Figure 3.2: State of FSRA before deactivation

As we can observe in Figure 3.3, drivers deactivate the FSRA until the end of the ride only in 24.6 % of cases. Interestingly, the drivers reactivate it in 11.2% when it is Enable and Active. Lastly, we observe the value not Available in 3.47% of cases as the next state after deactivation, which is not possible since the functionality has not been purchased.

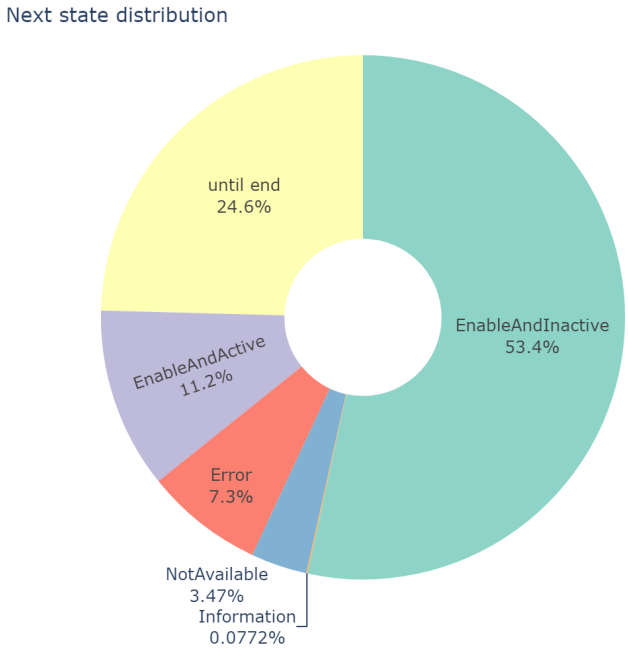


Figure 3.3: State of FSRA before reactivation

Next, we examine the speed of the truck 3 seconds before the deactivation and 3 seconds after the reactivation. In the majority of the cases before and after the vehicle is stationary. It is notable that in 10.5% of the cases, the drivers deactivate FSRA at high speed, as depicted in Figure 3.4.

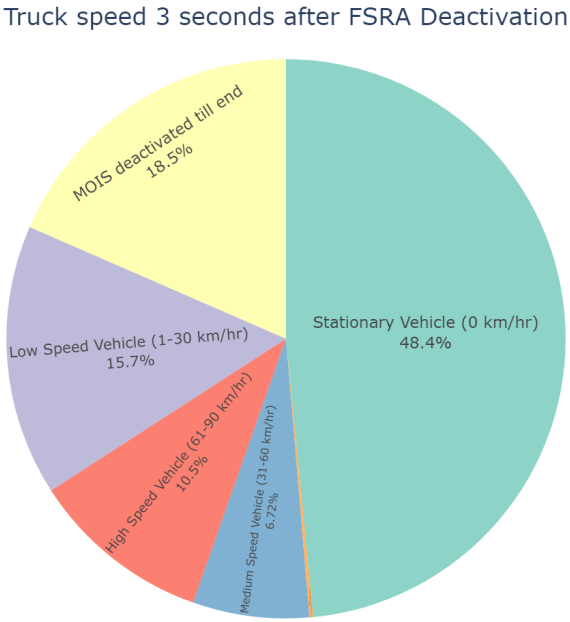
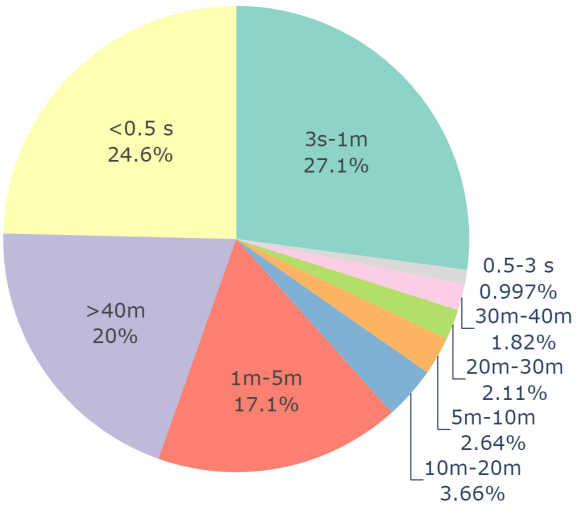


Figure 3.4: Truck speed 3 seconds after FSRA deactivation

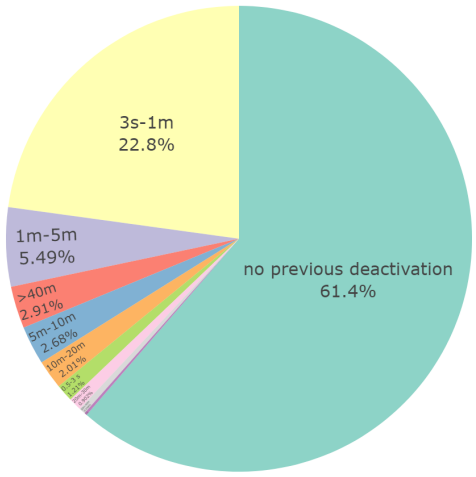
Lastly, we are interested in investigating if other ADAS functions are also deactivated 2 minutes before and after the deactivation of FSRA. We present the results in Table 3.3.

Number of deactivation	0	1	2	3	4
LKS	1070	210	12	3	0
AEB	1074	201	19	1	0
FDA	1266	25	3	0	1

Table 3.3: Number of deactivations 2 minutes before and after the deactivation of FSRA. LKS, AEB, and FDA are abbreviations of other ADAS functions as explained below.



a) Deactivation start timestamp



b) Time gap between current and previous deactivation

Figure 3.5: a) Start timestamp for deactivations. b) Time from the end timestamp of the previous deactivation to the start timestamp of the next deactivation

3.2 Dataset and Preprocessing

3.2.1 Features

The first step is to determine which subset of the available features is related to our problem. We decided to use the following: Speed, Acceleration Pedal Position, Brake Pedal Position, Location, Yaw Rate, the status of FSRA, and the status of other ADAS functions. The status of these additional ADAS functions can be a good indicator of the behavior of the drivers and provide additional insight into potentially dangerous driving environments. Specifically, we chose the ones that in-

teract with the driver either through sounds, lighting, or more actively and they are also available in all files that have FSRA deactivations. Among these, we chose the following four ADAS functions: Lane Keeping Assist (LKS), Automatic Emergency Braking (AEB), Speed Alert (SA), and Forward Distance Alert (FDA). LKS warns the drivers when the truck deviates from its lane and is within an active area. AEB aims to prevent collisions by automatically applying the brakes when necessary. Initially, there is a pre-warning signal(a red LED light), followed by flashing lights and an alarm if the driver does not respond. Then, the emergency brake is engaged, if there is still no reaction by the driver. SA warns the drivers when they exceed the speed limit. Finally, the FDA helps drivers maintain a safe distance from the vehicle ahead by warning them with a red light when they are too close to the other vehicle.

We observe these signals in a defined window, as we explain below. However, there is a history in the trip before that moment t_{start}. Thus, we created an extra feature for each alert, warning, and deactivation status of these ADAS functions, which indicates how many of them the driver received before the moment t_{start}. Additionally, we constructed an extra feature corresponding to the total duration of the trip for each file.

We also use Open-meteo, a free API for weather data. Using longitude, latitude, date, and time, we can retrieve historical weather data. Specifically, we include rainfall, snowfall, cloud cover, wind, humidity, temperature, and whether it is day or night in our dataset.

Additionally, we use the Overpass API to extract information about the road type. Using the latitude and longitude we retrieve the road type. Among the 32 variants identified, we've sorted them into 5 groups based on their primary purpose and usage: Major Roads, Secondary Roads, Local Roads, Special Purpose Roads, and Other/Unknown

Major Roads: These road types typically represent major highways, freeways, and expressways designed for high-speed traffic, connecting major cities or regions. They often have controlled access and multiple lanes, prioritizing long-distance travel.

Secondary Roads: These roads mainly connect major roads and smaller roads, helping in intercity or intracity travel. They usually have lower speed limits compared to major roads and have fewer lanes.

Local Roads: These roads mainly serve the local communities, helping access the residential areas, and business. They have lower speed limits. They include narrower streets, sidewalks, and intersections.

Special Purpose Roads: These roads are made for specific purposes like pedestrian pathways, cycling routes, race tracks, and emergency access points. They might not be used by standard motor vehicles. They are made with a particular user in mind.

Other/Unknown: These roads either contain insufficient information for classification or are miscellaneous categories like temporary constructions, developments, and decommissioned roads.

In Figure 3.6 and in Figure 3.7 we see the number of samples that each road type has.

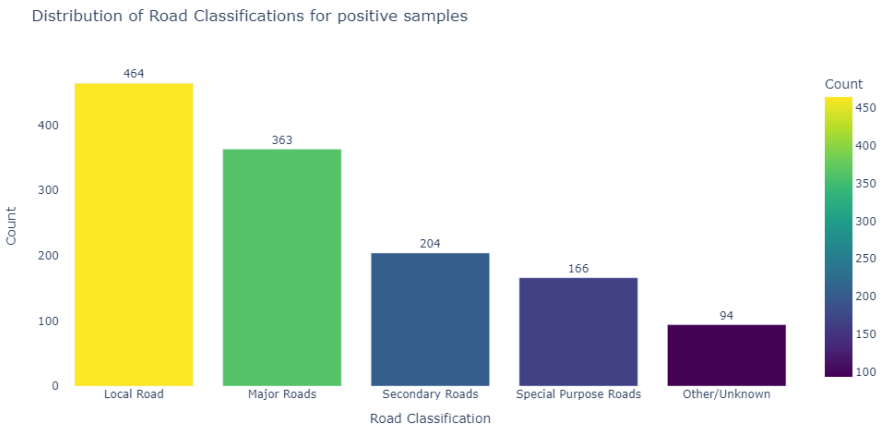


Figure 3.6: Distribution of road types for positive samples.

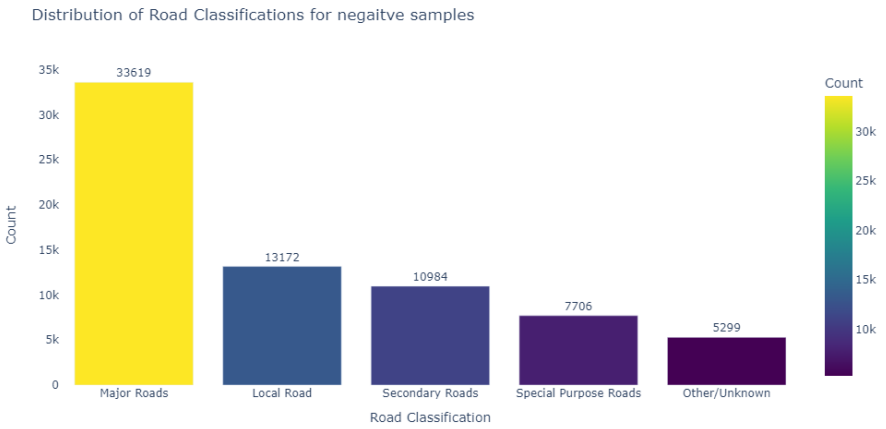


Figure 3.7: Distribution of road types for negative samples.

3.2.2 Time window

We need to determine the time window for our dataset. Specifically, we must decide on the duration preceding the deactivation event for the positive samples. We decided to include a 2-minute interval of the trip data before each deactivation, and thus, we defined the size of the window to be equal to 2 minutes. While having a longer time window could also be reasonable, we have shown previously that most

Name	Type
Acceleration Pedal position	Numerical - Time Series
Brake Pedal position	Numerical - Time Series
Speed	Numerical - Time Series
Yaw Rate	Numerical - Time Series
Duration of trip	Numerical - scalar
Status of AEB	Categorical (8 categories) - Time Series
Status of FDA	Categorical (7 categories) - Time Series
Status of LKS	Categorical (5 categories) - Time Series
Status of FSRA	Categorical (7 categories) - Time Series
Status of SA	Categorical (5 categories) - Time Series
# Information alerts of FSRA	Numerical - Scalar
# Warning alerts of FSRA	Numerical - Scalar
# Alerts of LKS	Numerical - Scalar
# Deactivations of LKS	Numerical - Scalar
# Pre-mitigation braking of AEB	Numerical - Scalar
# Mitigation braking of AEB	Numerical - Scalar
# Fully mitigation braking of AEB	Numerical -Scalars
# Deactivations of AEB	Numerical - Scalar
# Information alerts of FDA	Numerical - Scalar
# Warning alerts of FDA	Numerical - Scalar
# Deactivations of FDA	Numerical - Scalar
# Warning alerts of SA - Level 1	Numerical - Scalar
# Warning alerts of SA - Level 2	Numerical - Scalar
Location	Categorical (7 categories) - Scalar
Road Type	Categorical (5 categories) - Scalar
Day or night	Binary - Scalar
Rainfall	Numerical - Scalar
Snowfall	Numerical - Scalar
Cloud cover	Numerical - Scalar
Dew	Numerical - Scalar
Wind	Numerical - Scalar
Humidity	Numerical - Scalar
Temperature	Numerical - Scalar

Table 3.4: Features

deactivations occur within the first 5 minutes of each file. So, using a 2-minute window interval allows us to capture important driver behaviour in a concise data frame.
Different files may have sampling intervals. Specifically, some files have values for each feature recorded every 0.01 seconds, while others have values recorded every 1-second interval. For all samples to have the same length, we downsampled the data, retaining values recorded at every second. So, each time series has a length of 120 timestamps.

3.2.3 Positive and Negatives samples

- **Positive**: We consider deactivations valid if they last more than 0.5 seconds, and if a 2-minute time window is available before the deactivation occurs, dur-

ing which the status of FSRA differs from 'deactivated by the driver'. Using these criteria, we have a total of 1,295 positive samples.

- **Negative**: For the negative samples, we used the files that did not contain FSRA deactivations. From these files, we excluded those that had a duration of less than 2 minutes and those that did not have all chosen features available. From the remaining files, we constructed the negative samples. Specifically, from each file, we extracted two-minute samples with a 10-minute gap between each. This process was repeated for all eligible files. This resulted in a total of 83,959 negative samples.

3.2.4 Preprocessing

A subset of our features consists of time series, while the rest are scalars. The first step is to convert the scalar features into time series by repeating them. Additionally, some of the features are numerical while others are categorical. We applied one-hot encoding to non-binary categorical features. In one-hot encoding, we replace each categorical feature with k new features, where k corresponds to the number of the categories of that feature. For each data point, we assign a value of 1 to the feature which corresponds to the category of the data point, and 0 to all others. We present an example in the Table 3.5. Additionally, for each feature f, for each sample i and timestamp t, we applied z-score standardization:

$$x_{std,f,i,t} = \frac{x_{f,i,t} - x_{mean,f}}{\sigma_f}.$$

Apart from transforming the data to have the same variance, someone can rescale them in order for them to take values into the same range of values. For the last approach, we applied min-max normalization. For each numerical feature, we calculate its maximum and minimum values in the training set only. So, for each feature f, for each sample i and timestamp t, we apply min-max normalization formula:

$$x_{norm,f,i,t} = \frac{x_{f,i,t} - x_{min,f}}{x_{max,f} - x_{min,f}}.$$

id	color		id	color red	color blue	color green
1	blue		1	0	1	0
2	red	- - >	2	1	0	0
3	green		3	0	0	1

Table 3.5: Applying one-hot encoding

These transformations of data usually improve the model's convergence.

3.3 Anomalies

3.3.1 Variability in number of columns

Some files exhibit inconsistency in the number of columns, with certain files containing more columns than others. This non-uniformity across files poses a challenge in data processing and analysis.

3.3.2 Inconsistent Column Names

Diverse column names are observed across files, even when referring to the same data. For instance, "GPS speed" may be denoted as "WheelBasedVehicleSpeed" or "WheelBasedVehicleSpeed_BB1_X_V" in different files. This inconsistency complicates data integration and interpretation.

3.3.3 Unique Data Logging Patterns

Trucks *Truck 1* and *Truck 2* demonstrate distinct data logging patterns compared to other trucks. Notably, the attribute "ForwardDistanceAlert" exhibits varied representations across trucks. While it corresponds to "ForwardDistanceAlert_Spare3" in other trucks, for *Truck 1* and *Truck 2*, it is represented as "ForwardDistanceAlert_NotAvailable".

3.4 Feature Correlation

In this subsection, we aim to investigate if there are highly correlated features among the selected ones. In general, highly correlated features might negatively affect the performance of a model. Strongly correlated features practically provide the same information, making all but one redundant. Removing these redundant features can improve the model's predictive power. Having strongly correlated features also increases the complexity of the model. By increasing the complexity of the model, they also increase the probability of the model overfitting. Therefore, by removing correlated features, we might achieve better generalization of the data. Additionally, by dropping the highly correlated features, the dimensionality of the dataset decreases, making the model less computationally intensive and faster to train. Moreover, using fewer features makes it easier to interpret which features are important for the model. However, it is important to note that by removing highly correlated features, we cannot be sure if the predictive power of the model will be improved.

Feature $\mathbf{F_i}$ has dimensionality samples x time steps x 1 (numerical feature) / number of categories (categorical feature after one hot encoding). $\mathbf{T_i, k}$ is the long vector of $\mathbf{F_i, k}$ and is vectorized such that the vector contains the time step values of each sample sequentially. Specifically, positions 0 to timesteps-1 of the vector correspond to the values of the feature i for the time steps of the first sample, while positions timesteps to $2 * (timesteps - 1)$ correspond to the values of the second sample, and so on.

If feature $\mathbf{F_1}$ is highly linear correlated with $\mathbf{F_2}$, then there is a matrix β with high norm and a matrix ϵ with small norm such as:

$$\begin{bmatrix} \mathbf{T_2}_{(1,1:k_2)} \\ \mathbf{T_2}_{(2,1:k_2)} \\ \vdots \\ \mathbf{T_2}_{(l,1:k_2)} \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{T_1}_{(1,1:k_1)} \\ 1 & \mathbf{T_1}_{(2,1:k_1)} \\ \vdots & \vdots \\ 1 & \mathbf{T_1^i}_{(l,1:k_1)} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{\beta}_{(1,1:k_2)} \\ \boldsymbol{\beta}_{(2,1:k_2)} \\ \vdots \\ \boldsymbol{\beta}_{(k_1,1:k_2)} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\epsilon}_{(1,1:k_2)} \\ \boldsymbol{\epsilon}_{(2,1:k_2)} \\ \vdots \\ \boldsymbol{\epsilon}_{(l,1:k_2)} \end{bmatrix},$$

where $\mathbf{T_2}$ is the depended variable, $\mathbf{T_1}$ the independent variable, the first column of $\boldsymbol{\beta}$ is the intercept, the rest columns of$\boldsymbol{\beta}$ are the coefficients of the independent variable and $\boldsymbol{\epsilon}$ represent the error term. The intercept and coefficients correspond to the solution of minimization of the sum of the squared residuals over all samples. If the norm of coefficients is close to 0, then there is a weak or no linear relationship between the two-time series. $\mathbf{T_1}$ and $\mathbf{T_2}$ can be vectors or matrices depending on whenever the features are categorical.

We utilize linear regression to calculate the coefficients for all pairs of features. We considered both scenarios: where $\mathbf{F_i}$ is the dependent and $\mathbf{F_j}$ is the independent variable, and vice versa. We also compute the Relative Prediction Error (RPE) as:

$$RPE = \frac{||\mathbf{T}_1^{predicted,\mathbf{T_2}} - \mathbf{T_1}^{\mathbf{true}}||_F}{||\mathbf{T_1}^{\mathbf{true}}||_F},$$

where $\mathbf{T}_1^{predicted,\mathbf{F_2}}$ is the predicted $\mathbf{T}_1$ using $\mathbf{T_2}$ as independent variable.

We observe from Table 3.6, that the # Pre-mitigation brakings of AEB (alert 1), # Mitigation brakings of AEB (alert 2), # Fully mitigation brakings of AEB (alert 3) are correlated. This is expected since an alert 2 requires a preceding alert 1 and an alert 3 requires an alert 1 and 2. This is also the case for # Warning alerts of SA - Level 1 and # Warning alerts of SA - Level 2. Instead of using three different features for the alerts of AEB, we replace them with their summation. We do the same for speed warnings.

Additionally, we observe that the Status AEB, LKS, and FDA can be expressed by Speed. Indeed, the status of these functions is correlated with the Speed. Indeed, their different states, such as active or inactive, are highly dependent on speed. However, the speed is not the only factor. As previously discussed, these functions provide insights about the surrounding condition and the driver's behavior through their categories of alerts and warnings, which cannot be explained by the speed. Therefore, we choose to not remove any of the features from our dataset.

Finally, we note that the Status of the AEB, LKS, and FDA may be expressed by the number of deactivations of the respective function before the two-minute slot. However, this relationship is not very reasonable. A reason for that can be the challenge in effectively calculating the correlation between mixed types of features -categorical and numerical- since there is no standardized methodology.

Depended variable	**Independent variable**	**RPE**	$\|\|\beta(2,:k_1,1:k_2)\|\|_F$	$\|\|\beta(1,1:k_2)\|\|$
# Pre-mitigation AEB	# Fully mitigation AEB	0.2402	0.9698	0.0008
# Mitigation AEB	# Fully mitigation AEB	0.5721	0.8010	0.0088
# Fully mitigation AEB	# Pre-mitigation AEB	0.2403	0.9786	0.0004
# Fully mitigation AEB	# Mitigation AEB	0.5718	0.8101	0.0042
# Warning SA - Level 1	# Warning SA - Level 2	0.0953	0.9924	0.0001
# Warning SA - Level 2	# Warning SA - Level 1	0.0953	0.9978	0.0004
Status of AEB	# Deactivations of AEB	0.4437	1.2723	0.0005
Status of FDA	Speed	0.5398	0.8605	0.0035
Status of FDA	#Deactivations of FDA	0.5276	0.8420	0.0032
Status of LKS	Speed	0.5792	0.8879	0.0023
Status of LKS	# Deactivations of LKS	0.5301	0.8637	0.0056

Table 3.6: Highly correlated features

3.5 Visualisation

We visualize our dataset in 2D in order to gain an insight into how the two classes are spread. Initially, we apply a manual feature extraction from each time series (Table 3.7). Then, we apply t-SNE (t-distributed Stochastic Neighbor Embedding), an unsupervised non-linear dimensionality reduction technique for data exploration and visualization in a low-dimensional space of two or three dimensions.

t-SNE models the high-dimensional data points into low-dimensional points in such a way similar points in high-dimensional space are nearby in the lower-dimensional space, while dissimilar points to initial space are far in the lower-dimensional space [18]. The algorithm calculates the conditional probability of similarity between a pair of points using a Gaussian distribution in high-dimensional space and a t-distribution in lower-dimensional space. t-SNE minimizes the Kullback-Leibler divergence between the probability distribution of the original high-dimensional and the lower-dimensional and in this way, it attempts to maintain the structure of the initial points in the lower dimension space.

Before performing t-SNE, we standardized the features. Then, we examine if there are highly correlated numerical features using the Pearson correlation coefficient. We define a pair of features as highly correlated if the absolute value of the coefficient is larger than 0.8. We apply t-SNE both before and after removing the highly correlated features. We provide the results in Figures 3.8 and 3.9.

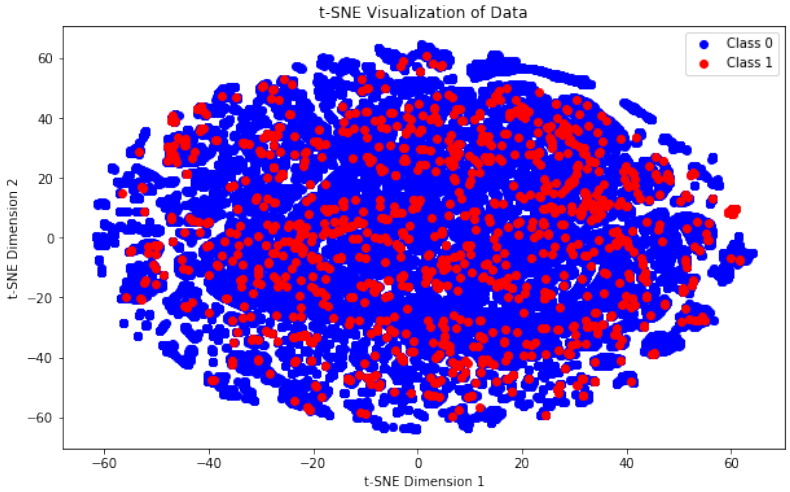


Figure 3.8: t-SNE visualization of Data.

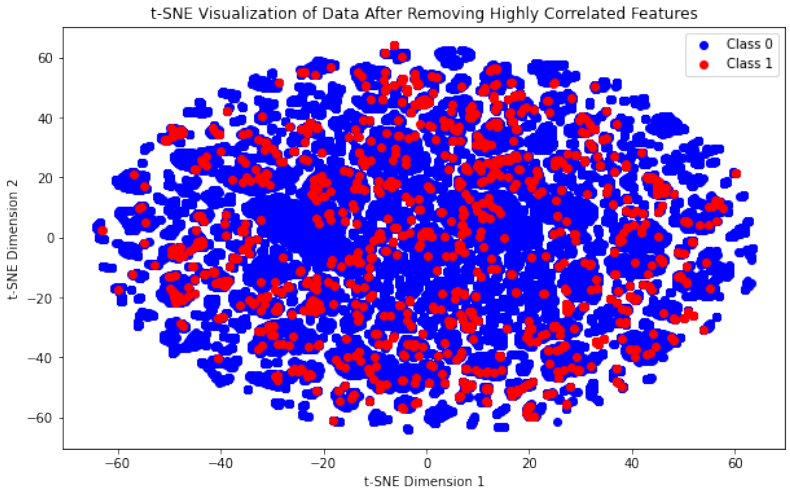


Figure 3.9: t-SNE visualization of data after removing highly correlated features.

Name	Type
Mean Acceleration Pedal Position	Numerical - Scalar
Standard Deviation Acceleration Pedal Position	Numerical - Scalar
Mean Brake Pedal Position	Numerical - Scalar
Standard Deviation Brake Pedal Position	Numerical - Scalar
Mean Speed	Numerical - Scalar
Standard Deviation Speed	Numerical - Scalar
Mean Yaw Rate	Numerical - Scalar
Standard Deviation Yaw Rate	Numerical - Scalar
Duration of Trip	Numerical - Scalar
Status of AEB in 119 sec	Categorical (8 categories) - Scalar
Status of FDA in 119 sec	Categorical (7 categories) - Scalar
Status of LKS in 119 sec	Categorical (5 categories) - Scalar
Status of FSRA in 119 sec	Categorical (7 categories) - Scalar
Status of SA in 119 sec	Categorical (5 categories) - Scalar
# Information Alerts of FSRA in 0 sec	Numerical - Scalar
# Information Alerts of FSRA in two-minute window	Numerical - Scalar
# Warning Alerts of FSRA in 0 sec	Numerical - Scalar
# Warning Alerts of FSRA in two-minute window	Numerical - Scalar
# Alerts of LKS in 0 sec	Numerical - Scalar
# Alerts of LKS in two minute window	Numerical - Scalar
# Deactivations of LKS in 0 sec	Numerical - Scalar
# Deactivations of LKS in two minute window	Numerical - Scalar
# Pre-Mitigation Brakings of AEB in 0 sec	Numerical - Scalar
# Pre-Mitigation Brakings of AEB in two-minute window	Numerical - Scalar
# Mitigation Brakings of AEB in 0 sec	Numerical - Scalar
# Mitigation Brakings of AEB in two-minute window	Numerical - Scalar
# Fully Mitigation Brakings of AEB in 0 sec	Numerical - Scalar
# Fully Mitigation Brakings of AEB in two-minute window	Numerical - Scalar
# Deactivations of AEB in 0 sec	Numerical - Scalar
# Deactivations of AEB in two minute window	Numerical - Scalar
# Information Alerts of FDA in 0 sec	Numerical - Scalar
# Information Alerts of FDA in a minute window	Numerical - Scalar
# Warning Alerts of FDA in 0 sec	Numerical - Scalar
# Warning Alerts of FDA in two-minute window	Numerical - Scalar
# Deactivations of FDA in 0 sec	Numerical - Scalar
# Deactivations of FDA in two minute window	Numerical - Scalar
# Warning Alerts of SA - Level 1 in 0 sec	Numerical - Scalar
# Warning Alerts of SA - Level 1 in two-minute window	Numerical - Scalar
# Warning Alerts of SA - Level 2 in 0 sec	Numerical - Scalar
# Warning Alerts of SA - Level 2 in two-minute window	Numerical - Scalar
Location	Categorical (7 categories) - Scalar
Road Type	Categorical (5 categories) - Scalar
Location	Categorical (7 categories) - Scalar
Day or night	Binary - Scalar
Rainfall	Numerical - Scalar
Snowfall	Numerical - Scalar
Cloud cover	Numerical - Scalar
Dew	Numerical - Scalar
Wind	Numerical - Scalar
Humidity	Numerical - Scalar
Temperature	Numerical - Scalar

Table 3.7: Extracted Features

4

Methods

4.1 Imbalanced dataset

As observed in Chapter 3, our dataset is highly imbalanced. Since the negative samples are much more represented in the dataset than the positive ones, there is a risk for the model to become biased towards predicting the negative class. This can lead to poor generalization for the positive class, meaning that the model may struggle to correctly classify positive instances. In this work, we explored four different approaches to address this issue.

- **Class weights**: The simplest approach to deal with this problem is by using class weighting. The main idea is to assign higher weights to the minority class samples and lower to the majority class during the training process. In this way, the model will pay more attention to the minority class samples and it might be able to classify them more accurately.
- **Oversampling minority class**: In this approach, we duplicate or create new synthetic samples for the minority class into the training set. However, creating synthetic data can be challenging in this kind of application, since we need an expert to verify that the generated data could actually correspond to real truck data. Therefore, in our work, we randomly duplicated minority class samples in the training set until achieving a $90\% - 10\%$ balance between majority and minority classes. It is important to note that we do not alter the validation and test sets.
- **Downsampling majority class**: In this approach, we randomly downsampled the majority class in the training set to achieve a $75\% - 25\%$ balance between the two classes. Once again, we do not alter the validation and test sets.
- **Ensemble Learning with undersampling**: In this approach, we combined ensemble learning with the undersampling of the majority class. In ensemble learning, we aim to enhance the predictive power by aggregating the prediction of multiple models. After splitting the dataset into train and test sets, we divided the majority class samples in the training set into smaller subsets, each with a size equal to three times that of the minority class in the training set. Next, we created pairs of majority class subsets and an entire minority class training set and we trained a model for each pair. During evaluation, we assigned to each sample the one chosen by the majority of the models.

4.2 Data splitting

For all experimental designs, we initially split the positive and negative samples of our dataset into training (tr1) and test sets (te1) with a ratio of 0.77/0.33. Then, we further split the training set into training and validation sets with a ratio of 0.85/0.15, maintaining the class distribution of the entire dataset. For oversampling and downsampling, we randomly duplicate or remove samples from the training set, respectively. For ensemble learning, we create pairs of positive and negative samples.

A similar splitting method is applied for grid search cross-validation, but the dataset being split is tr1.

In the case of the BiLSTM autoencoder, we use the above-mentioned splitting. However, for the training and validation dataset, we use only negative samples and for the testing set, we concatenate the above-mentioned test set with the positive samples from the training and validation sets.

4.3 Training setups

We experimented with three experimental training setups.

1. We trained the network from scratch. In this setup, the encoder component is a simple part of the network, and we train it to encode the input data into a smaller dimension based on the samples' labels. (TS1)
2. In the second setup, we pretrained a symmetrical autoencoder, which has the same encoder part as the classifier. We froze the weights of the encoder part of the classifier to those of the pretrained encoder of the autoencoder. In this setting, the encoder part is not trainable, and it works as a feature extractor trained using the reconstruction error. (TS2)
3. In the last setup, we initialized the weights of the encoder part with the pretrained ones. In this way, the model adjusts them while also considering the labels of the samples. (TS3)

4.4 Baseline models

4.4.1 Logistic regression

As a baseline model, we utilized logistic regression using features the ones described in 3.7. In this case, we split the dataset into training and test sets. The test set is the same as the one used for the other classifiers.

We applied a 5-fold grid search CV to find the best inverse of regularization strength. We experimented with three values: 10, 1, and 0.1. In Table 4.1, the best parameters for each approach are presented.

	Class weighting	Oversampling	Downsampling	Ensemble learning
C	10	10	10	1

Table 4.1: Selected hyperparameters values for logistic regression model.

4.4.2 Trivial compression

In this approach, we simply downsample each time series to achieve different levels of compression. We provide an illustration in Figure 4.1. The compressed time series are fed into a flattened layer, followed by a fully connected network. The fully connected network consists of three dense layers with 256, 128, and 1 neurons, respectively. Dropout layers are added between the dense layers with a dropout rate of 0.2.

We applied a 5-fold grid search CV to find the best hyperparameters. We experimented with compression rate, learning rate, and batch size, in Table 4.2.

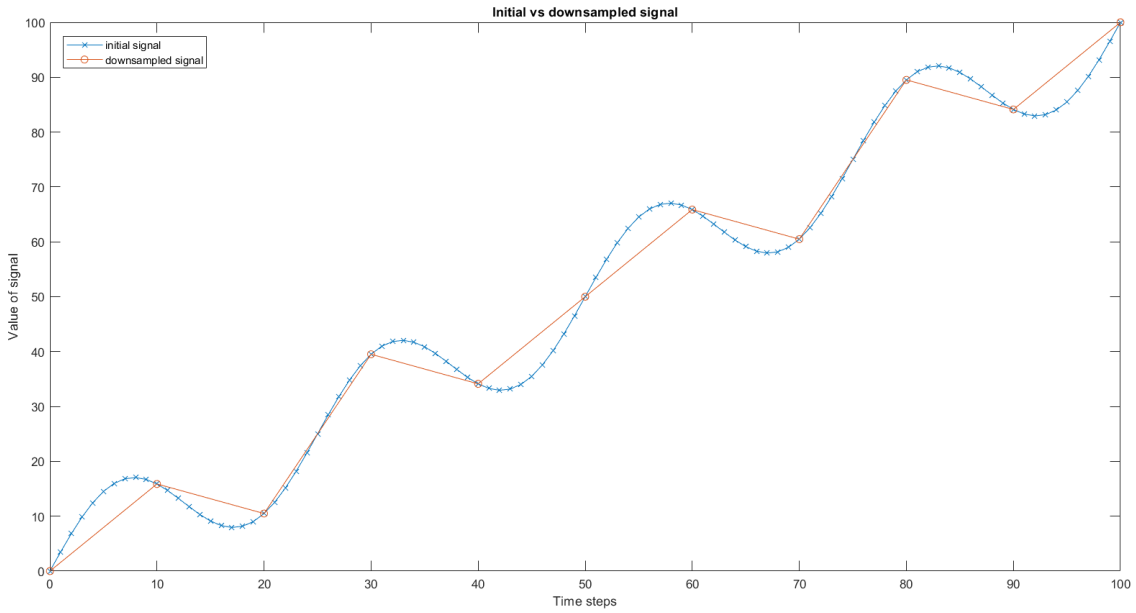


Figure 4.1: Initial vs downsampled signal

Compression rate:	0.2, 0.5, 0.8
Learning rate:	0.001, 0.0001
Batch size:	128, 256
Drop out rate:	0.2
Patience :	5
Number of epochs:	100

Table 4.2: Hyperparameters values used for grid search 5-fold Stratified CV for trivial compression.

In Table 4.3, the best parameters for each approach are presented.

	Class weighting	Oversampling	Downsampling	Ensemble learning
Compression rate	0.8	0.5	0.8	0.8
Learning rate	0.0001	0.0001	0.0001	0.001
Batch size	256	256	256	128

Table 4.3: Selected hyperparameters values for trivial compression model.

4.5 CNN-based classifier

CNN-based classifier consists of two components: A CNN-based encoder, followed by a fully connected network. We experimented with the architecture of the CNN encoder, specifically utilizing two different types of convolutional layers: classical and dilated ones.

The encoder consists of n blocks, with each block consisting of a 1D convolutional layer, followed by a batch normalization layer and then a max pooling layer with a pool size of 2.

The input of the first layer is time steps x features, allowing the model to capture dependencies between different types of features. Regarding the number of filters in each layer, a common practice was followed: the first layer has a smaller number of filters than the second one.

4.5.1 Hyperparameter tuning

We experimented with the number of layers, the number of filters in each layer, the learning rate, and batch size. In Table 4.4, we present the different values for each parameter. Specifically, the number of filters in each layer was chosen to achieve four different percentages of compression of the input: 0.13,0.25, 0.5, 0.7.

Number of layers:	1, 2
Number of filters for 1 layers:	16 (0.13), 32(0.25), 64(0.5), 86 (0.7)
Number of filters for 2 layers:	16-32(0.13), 32-64 (0.25), 64-128 (0.5), 128-174 (0.7)
Learning rate:	0.001, 0.0001
Dilation rate:	2, 3
Batch size:	128, 256
Kernel size:	3
Drop out rate:	0.2
Patience :	5
Number of epochs:	100
pool size:	2

Table 4.4: Hyperparameters values used for grid search 5-fold Stratified CV for CNN.

Standard CNN	Class weighting	Oversampling	Downsampling	Ensemble learning
Number of filters	86	128-174	86	86
Learning rate	0.0001	0.001	0.0001	0.001
Batch size	256	256	256	128

Table 4.5: Selected hyperparameters values for standard CNN-based model.

Dilated CNN	Class weighting	Oversampling	Downsampling	Ensemble learning
Dilation rate	2	2	3	2
Number of filters	32-64	64-128	64	32
Learning rate	0.001	0.001	0.001	0.001
Batch size	128	256	128	128

Table 4.6: Selected hyperparameters values for dilated CNN-based model.

4.5.2 CNN-based autoencoders

A symmetrical CNN-based autoencoder was built for each of the four approaches, as each approach exhibited better performance during cross-validation for different architectures. The ReLU activation function was used for all layers except the last one, where a linear activation was utilized because the input signals have both negative and positive values after standardization. The encoder part of the autoencoder is the same as the encoder part of our classifier.

We also compare the reconstruction power of the autoencoder with a trivial case. In the trivial case, the input is compressed as described in the trivial compression. During reconstruction, the signals are upsampled by repeating each value of the compressed signal multiple times to obtain the desired length.

4.6 RNN-based classifier

The RNN-based classifier consists of two components: an RNN-based encoder followed by a fully connected network. We experimented with the architecture of the RNN encoder, specifically utilizing different variations of RNNs to determine which one best fits our data. We used LSTM, GRU, and Bidirectional LSTM.

The encoder consists of n blocks, with each block consisting of an RNN-based layer followed by a drop-out layer.

4.6.1 Hyperparameter tuning

We experimented with the number of layers, the number of filters in each layer, the learning rate, and batch size. In Table 4.8, we present the different values for each parameter. We used tanh as the activation function for all layers except the last one, where we used sigmoid.

Number of layers:	1, 2
Number of filters for 1 layers:	32, 64, 128
Number of filters for 2 layers:	32-32, 64-64, 128-128
Learning rate:	0.001, 0.0001
Batch size:	128, 256
Drop out rate:	0.2
Patience :	5
Number of epochs:	100

Table 4.7: Hyperparameters values used for grid search 5-fold Stratified CV for RNN.

LSTM	Class weighting	Oversampling	Downsampling	Ensemble Learning
Number of layers	1	1	1	1
Number of filters	64	128	128	64
Learning rate	0.001	0.0001	0.001	0.001
Batch size	128	256	256	128
GRU	Class weighting	Oversampling	Downsampling	Ensemble Learning
Number of layers	1	2	1	1
Number of filters	64	128-128	64	128
Learning rate	0.0001	0.0001	0.001	0.001
Batch size	128	128	128	128
BiLSTM	Class weighting	Oversampling	Downsampling	Ensemble Learning
Number of layers	1	1	1	1
Number of filters	32	128	128	64
Learning rate	0.001	0.001	0.001	0.001
Batch size	128	128	128	256

Table 4.8: Selected hyperparameters values for RNN-based models.

4.7 Parallel CNN-LSTM-based classifier

The Parallel CNN-LSTM-based classifier consists of three components: a CNN-based encoder, an LSTM-based encoder, and a fully connected network. The CNN-based encoder and the LSTM-based encoder have the same input (time steps x features). The output of the two encoders is concatenated and used as the input of the fully connected network.

Each encoder consists of n blocks. For the LSTM-base encoder, each block consists of an RNN-based layer followed by a drop-out layer. In the case of a CNN-based encoder, each block consists of a 1D convolutional layer, followed by a batch normalization layer and then a max pooling layer with a pool size of 2.

4.7.1 Hyperparameter tuning

We experimented with the number of layers, the number of filters in each layer, the learning rate, and batch size. In Table 4.9, we present the different values for each parameter. We used tanh as the activation function for lstm layers, and ReLu for all other layers except the last one, where we used sigmoid.

Number of layers:	1, 2
Number of filters for 1 layer :	CNN: 64 - LSTM: 64, CNN: 86 - LSTM: 128
Number of filters for 2 layers:	CNN: 64-128 - LSTM: 64-64, CNN: 128-174 - LSTM: 128-128
Learning rate:	0.001, 0.0001
Dilation rate:	2, 3
Batch size:	128, 256
Kernel size:	3
Drop out rate:	0.2
Patience :	5
Number of epochs:	100
pool size:	2

Table 4.9: Hyperparameters values used for grid search 5-fold Stratified CV for Parallel CNN-LSTM-based classifier.

Parallel CNN-LSTM	Class weighting	Oversampling	Downsampling	Ensemble learning
Number of filters	86, 128	86, 128	64, 64	86, 128
Learning rate	0.0001	0.0001	0.001	0.001
Batch size	256	256	128	128

Table 4.10: Selected hyperparameters values for Parallel CNN-LSTM-based classifier.

4.8 BiLSTM autoencoder

In this approach, we train a BiLSTM autoencoder on the majority of class samples. Then, we evaluate the autoencoder on majority and minority samples. We hypothesize that the minority class samples follow a different distribution than the samples of the majority class and thus, the autoencoder will yield a high reconstruction error for the minority class samples. If the reconstruction error of a sample is larger than a threshold, we classify this sample as a positive one.

After the training of the autoencoder on the majority of class samples. We set two different thresholds:

1. Threshold 1: mean value of RRE plus the standard deviation of RRE on the training set.
2. Threshold 2: third quartile (Q3) + IQR on training set, where IQR= Q3-Q1.

The BiLSTM autoencoder consists of two components: an encoder and a decoder part. We used a symmetrical autoencoder. The encoder consists of n blocks, with each block consisting of a BiLSTM layer followed by a drop-out layer.

4.8.1 Hyperparameter tuning

We experimented with the number of layers, the number of filters in each layer, the learning rate, and batch size. In Table 4.11, we present the different values for each parameter. We also experiment with the transformation of the data: Min-Max normalization and z-score standardization. We used tanh as the activation function for all layers except the last one, where we used sigmoid (min-max) / linear (z-score standardization).

We used a hold-out CV instead of a 5-fold CV due to the shorter time of execution. We selected the set of hyperparameters that yields the best ARRE.

Number of layers:	1, 2
Number of filters for 1 layers:	128, 256
Number of filters for 2 layers:	128-128, 256-256
Learning rate:	0.001, 0.0001
Batch size:	128, 256
Drop out rate:	0.2
Patience :	5
Number of epochs:	150

Table 4.11: Hyperparameters values used for grid search hold-out CV for BiLSTM autoencoder.

	z-score	MinMax
Number of layers:	2	2
Number of filters for 1 layers:	256	256
Number of filters for 2 layers:	256	256
Learning rate:	0.001	0.001
Batch size:	128	128

Table 4.12: Selected hyperparameters values for BiLSTM autoencoder.

5
Results

We split the dataset into three sets: training, validation, and test set, as described above. Table 5.1 illustrates the number of samples for each class in each set.

Samples	All	Positive	Negative
Training	48550	736	47814
Validation	8569	131	8438
Test	28135	428	27707

Table 5.1: Split of the dataset.

For each model, the performance on training and test sets is provided. In the case of DNNs, each model is trained three times. In addition to the best performance, the mean performances of the three trials are also provided.
Regarding the permutation feature importance scores, the precision-recall AUC was used as the metric for comparing the model's performance on the initial test set and the shuffled ones. Additionally, a brief description of the results for each model is provided, and the corresponding images can be found in the appendix.

5.1 Logistic regression

Table 5.13, shows that the model exhibits poor performance across all approaches. The best score is achieved with the ensemble learning approach. Additionally, in the cases of oversampling the minority class and downsampling the majority class, we observe that the model overfits.
In Figure A.2, we see that only when we use class weights, does the model correctly classify more than half of the positive samples. However, this approach also yields the highest number of false positives. We also observe a trade-off: as the model achieves a smaller number of false positives, the number of false negatives increases. There is no case where the number of false positives is lower than the number of true positives.
Regarding the coefficients, across all the approaches, we observe that 9 coefficients have the most significant values (1.5-0.39), 31 have a moderate value from 0.3 to 0.1, and 36 have values smaller than 0.099. The most important features are almost the same for all approaches: FSRA not available in the last timestep, region US, FSRA enable and inactive in the last timestep, # of deactivation of FDA 0S, FDA deactivated in the last timestep, mean value of Acceleration pedal position, and FSRA enable and active in last time step.

		Precision	Recall	F1	ROC-AUC	PR-AUC
Class weighting	Training	0.0429	0.6840	0.0807	0.8044	0.1739
Class weighting	Test	0.0405	0.6495	0.0763	0.7844	0.1819
Oversampling	Training	0.7432	0.2852	0.4122	0.8033	0.5909
Oversampling	Test	0.1501	0.2897	0.1978	0.7771	0.1940
Downsampling	Training	0.7846	0.1765	0.2881	0.8018	0.4374
Downsampling	Test	0.3400	0.1986	0.2507	0.7765	**0.2077**
Ensemble learning	Training	0.1225	0.3403	0.1802	0.8034	0.1941
Ensemble learning	Test	0.1147	0.3294	0.1702	0.7829	0.2004

Table 5.2: Performance of Logistic Regression for the different approaches.

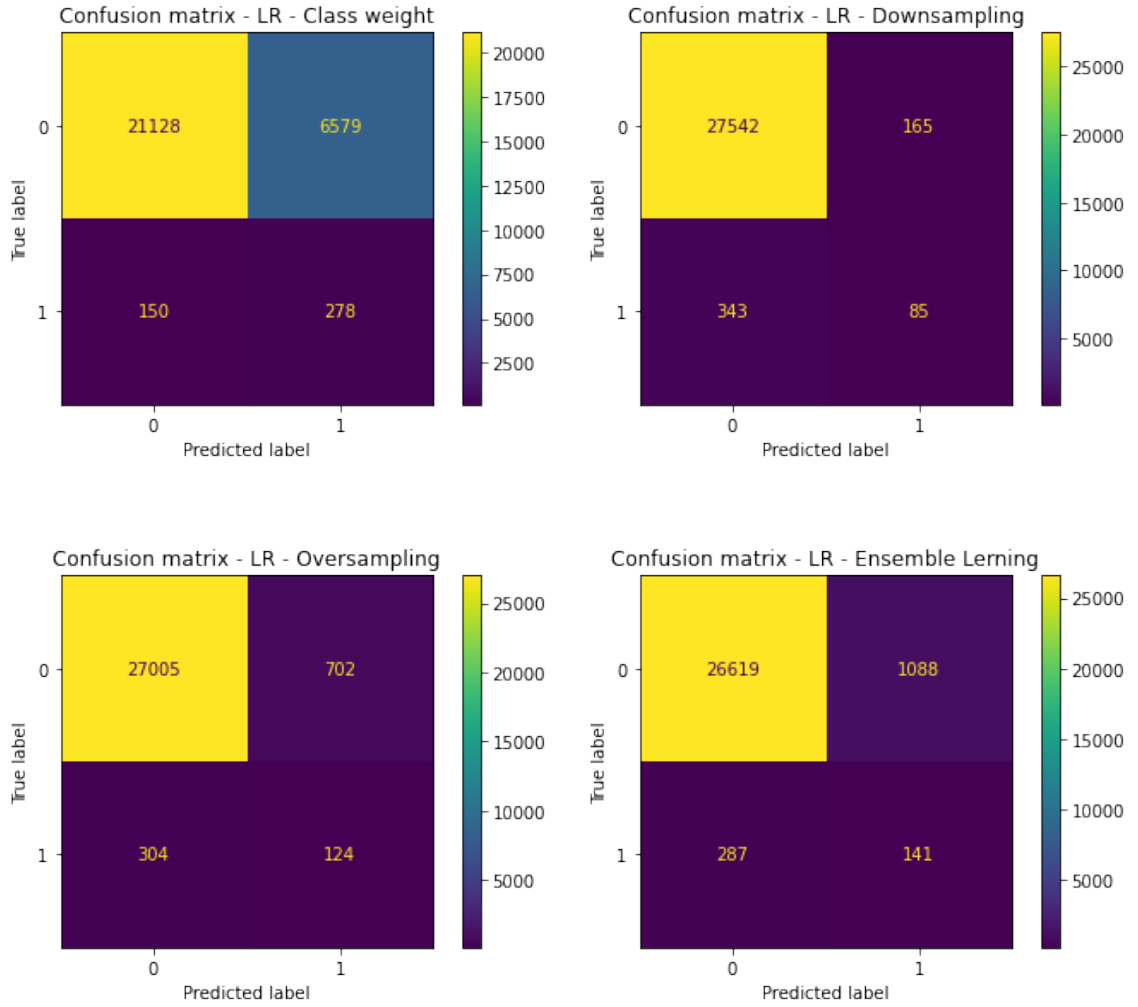


Figure 5.1: Confusion matrices of the test set for the different approaches.

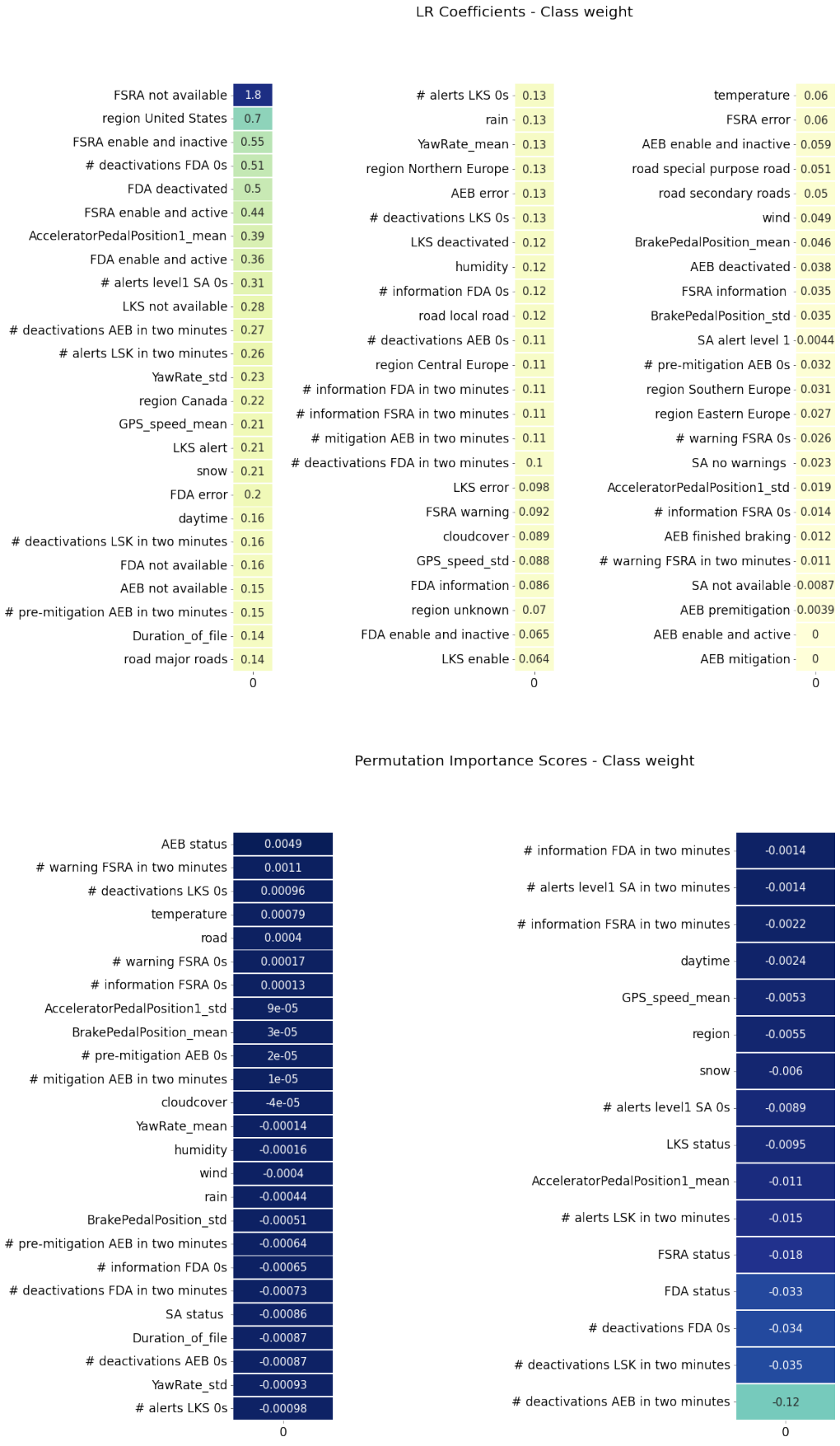


Figure 5.2: Logistic Regression coefficients and permutation importance scores for class weight approach.

In feature permutation importance, we treat the categories of the one-hot encoded categorical features as a single feature. Given that the best performance is 0.2, a feature importance score (FIS) less negative than -0.01 corresponds to an impact to the performance smaller than 5% and a FIS less negative -0.03 to an impact smaller than 25%. Using a threshold of an FIS more negative than -0.03 the most important features are: # deactivations of AEB in two minutes, # deactivations of LSK in two minutes, # deactivations of FDA 0s and FDA status at the last timestep.

5.2 Trivial compression

Similar to logistic regression, we observe that the model yields poor performance, though it exhibits higher performance than logistic regression across all cases. Additionally, the model overfits more severely than logistic regression across all approaches.

In Figure 5.3, we see that similar to LR, only when we use class weights can the model correctly classify more than half of the positive samples. In this case, the model yields a smaller number of false positives for class weights than LR but a larger number for false negatives. We also observe the same trade-off: as the number of false positives decreases, the number of false negatives increases. Again, there is no case where the number of false positives is lower than the number of true positives. For the trivial compression model, the ensemble learning approach exhibits the best PR-AUC score.

In Figures 5.4 and A.3, we observe that the importance ranking of the features may differ among the approaches. However, AEB status, LKS status, FSRA status, # deactivation of FDA, FDA status, and region exhibit high scores across all approaches. Speed, Acceleration Pedal Position, temperature, # LKS warning, and road type have a moderate impact on the performance of the model. Notably, in the case of downsampling, # FSRA warnings, and # FSRA information have moderate importance scores. They have very small, albeit positive, scores in all other cases.

		Precision	Recall	ROC-AUC	F1	PR-AUC
Class weighting	Training	0.1124	0.9389	0.9711	0.2	0.5205
Class weighting	Test	0.0695	0.5732	0.7989	0.1234	0.2122
Oversampling	Training	0.9267	0.8542	0.9863	0.8883	0.9555
Oversampling	Test	0.2176	0.3614	0.8035	0.2698	0.2775
Downsampling	Training	0.9769	0.7215	0.9876	0.8299	0.9362
Downsampling	Test	0.2531	0.3178	0.8068	0.2811	0.2201
Ensemble learning	Training	0.2298	0.875	0.9804	0.3638	0.6811
Ensemble learning	Test	0.138	0.4743	0.8358	0.2136	**0.2922**

Table 5.3: The mean performance of Trivial Compression for the different approaches.

		Precision	Recall	ROC-AUC	F1	PR-AUC
Class weighting	Test	0.08296	0.5561	0.7962	0.1442	0.2364
Oversampling	Test	0.2301	0.3318	0.7966	0.2718	0.2916
Downsampling	Test	0.2326	0.3364	0.8084	0.2751	0.2279
Ensemble learning	Test	0.1285	0.4790	0.8371	0.2027	**0.2960**

Table 5.4: The best performance of Trivial Compression for the different approaches.

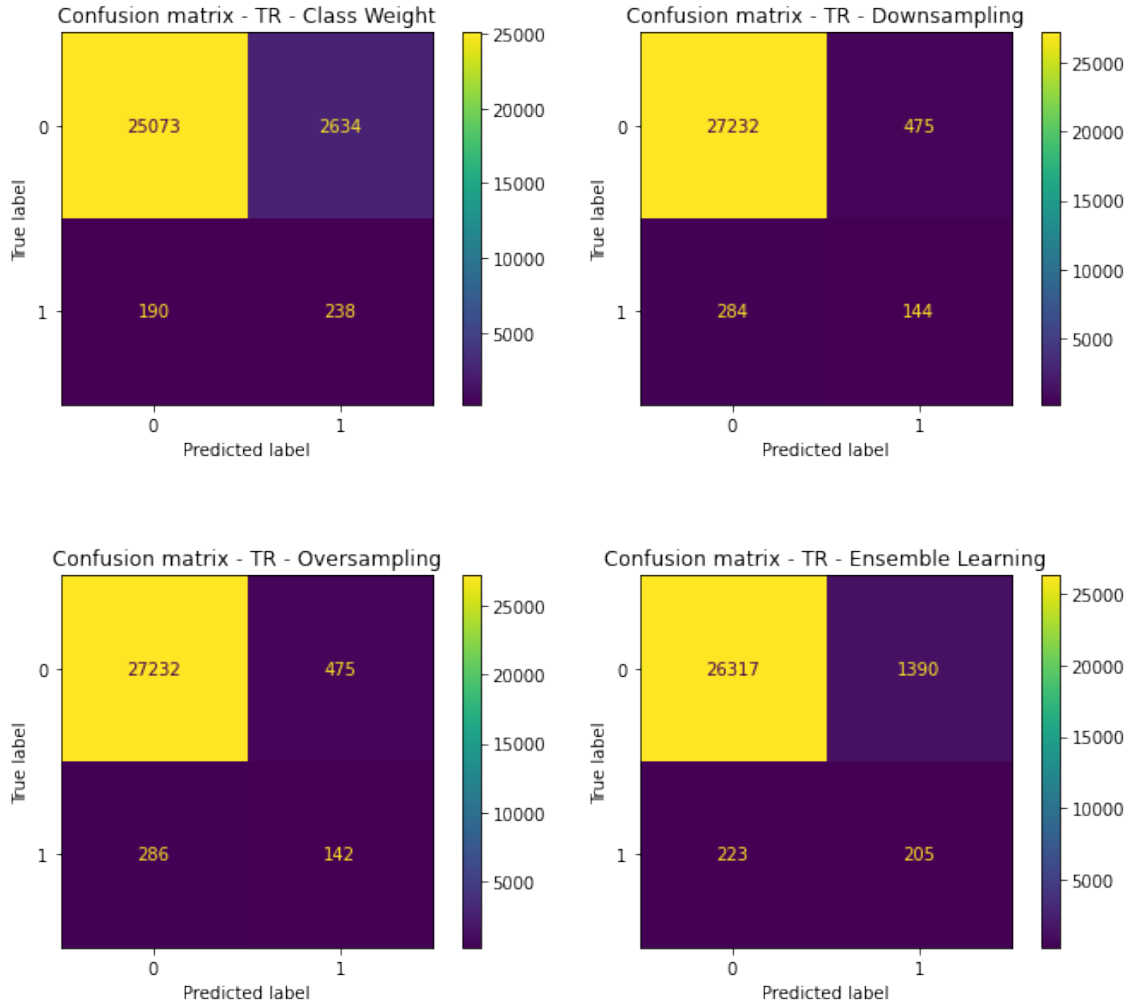


Figure 5.3: Confusion matrices of the test set for the different approaches.

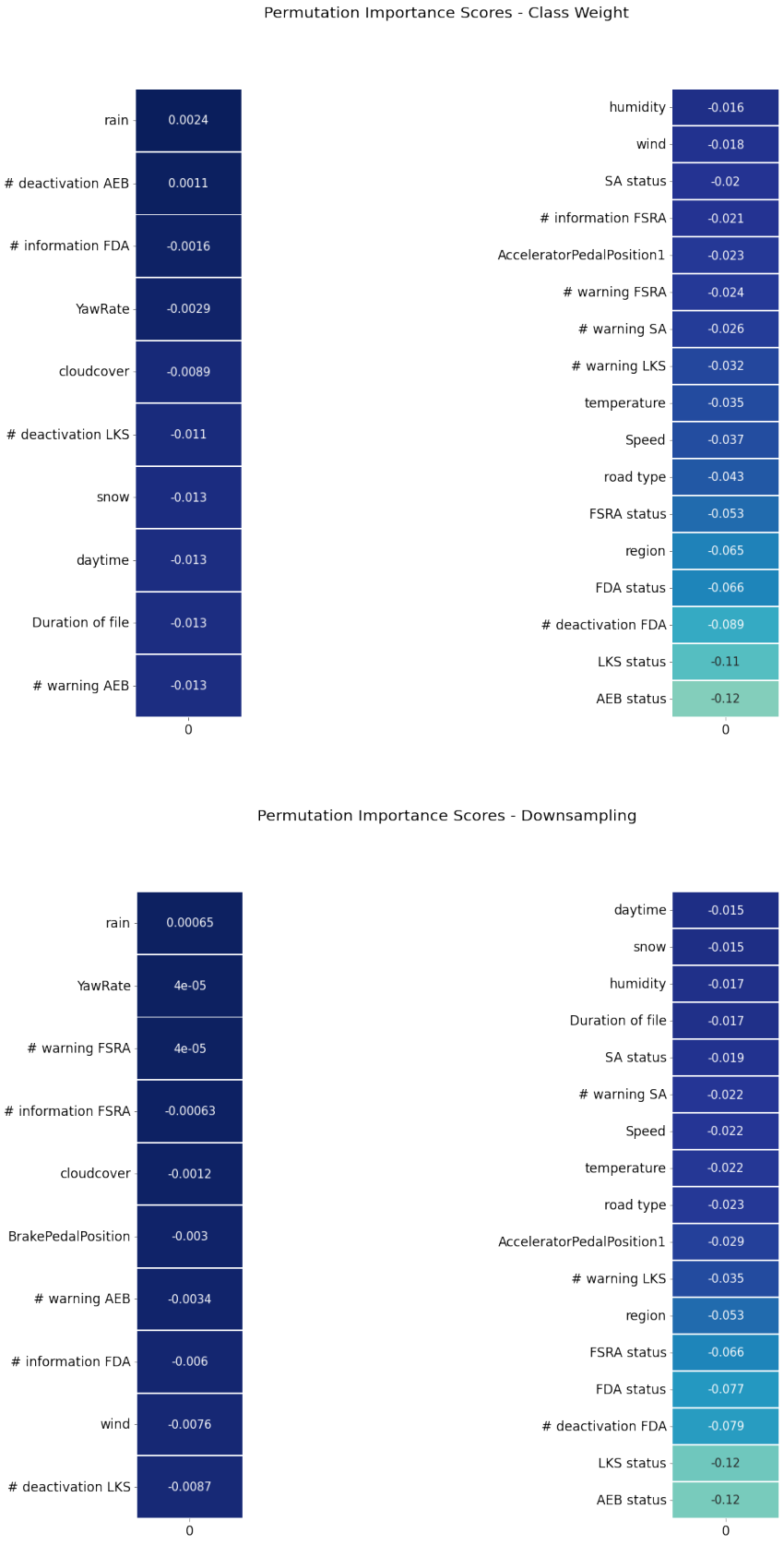


Figure 5.4: Trivial compression's permutation importance scores for class weight approach and downsampling.

5.3 Standard CNN

Like previous models, the model exhibits poor performance. It exhibits higher average performance than trivial compression for the cases of class weight and downsampling. In the case of ensemble learning, only for TS3 does it achieve a better average score than trivial compression. For oversampling, trivial compression achieves a better score on average. Additionally, we observe that the model overfits across all approaches.

For the case of class weighting, TS3 achieves better results than TS2, and TS2 achieves better results than TS1 on average. However, TS1 achieves a higher best performance than TS2. For oversampling, TS2 achieves better results than TS1 and TS1 achieves better results than TS3. For downsampling, TS2 achieves better results than TS3 and TS3 achieves better results than TS1. For ensemble learning, TS3 achieves better results than TS2 and TS2 achieves better results than TS1. Lastly, we observe that the autoencoders exhibit similar performances.

In Figures 5.5, A.4, and A.5, we can see, as in previous models, only when using class weights can the model correctly classify more than the half of the positive samples. As in trivial compression, the model yields a smaller number of false positives for class weights than LR, but a larger number for false negative. We also observe the same trade-off: as the number of false positives decreases, the number of false negatives increases. As in previous models, there is no case where the number of false positives is lower than the number of true positives. In this case, we can observe that the ensemble learning approach has an almost equal number of true and false positives, with false positives slightly exceeding the number of true positives.

In Figures 5.6, A.6, A.7, A.8, A.9, and A.10 we observe that the importance ranking of the features may differ among the approaches and training setups. However, AEB status, LKS status, # deactivation of FDA, and FDA status exhibit high scores across all approaches. Other features with a high score for some of the approaches and training setups are region, FSRA status, Speed, Acceleration Pedal Position, temperature, and road type. Notably, in any case, # FSRA warnings, and # FSRA information have small importance scores.

Number of filters	Training ARRE	Validation ARRE	Test ARRE
86	0.0230	0.0232	0.0231
128-174	0.03	0.0306	0.0307

Table 5.5: Performance of autoencoders for standard CNN-based model.

		Precision	Recall	ROC-AUC	F1	PR-AUC
Class weighting (TS1)	Test	0.1322	0.5000	0.8368	0.2091	0.2553
Class weighting (TS2)	Test	0.0760	0.5771	0.8367	0.1343	0.2528
Class weighting (TS3)	Test	0.0773	0.6355	0.8397	0.1315	0.2823
Oversampling (TS1)	Test	0.2135	0.3107	0.8175	0.2531	0.2769
Oversampling (TS2)	Test	0.4111	0.2430	0.8227	0.3054	0.2812
Oversampling (TS3)	Test	0.3900	0.2360	0.8268	0.2940	0.2660
Downsampling (TS1)	Test	0.1870	0.3621	0.8339	0.2466	0.2441
Downsampling (TS2)	Test	0.2744	0.3224	0.8483	0.2965	0.2656
Downsampling (TS3)	Test	0.3165	0.2640	0.8347	0.2879	0.2502
Ensemble learning (TS1)	Test	0.1367	0.4953	0.8500	0.2142	0.2933
Ensemble learning (TS2)	Test	0.1358	0.4977	0.8548	0.3124	0.2951
Ensemble learning (TS3)	Test	0.1319	0.5	0.85559	0.2087	**0.3017**

Table 5.6: The best performance of standard CNNs for the different approaches.

		Precision	Recall	ROC-AUC	F1	PR-AUC
Class weighting (TS1)	Training	0.1758	0.976	0.9853	0.2931	0.6672
Class weighting (TS1)	Test	0.0971	0.5537	0.8313	0.162	0.2388
Class weighting (TS2)	Training	0.1165	0.9524	0.972	0.2066	0.5195
Class weighting (TS2)	Test	0.0742	0.6145	0.8359	0.1316	0.2422
Class weighting (TS3)	Training	0.1011	0.9669	0.9761	0.1829	0.5668
Class weighting (TS3)	Test	0.0675	0.6464	0.8381	0.1221	0.2645
Oversampling (TS1)	Training	0.9011	0.9794	0.9965	0.9383	0.9864
Oversampling (TS1)	Test	0.1606	0.384	0.824	0.2196	0.2665
Oversampling (TS2)	Training	0.9892	0.9938	0.9999	0.9915	0.9995
Oversampling (TS2)	Test	0.3606	0.2562	0.8258	0.2926	0.2743
Oversampling (TS3)	Training	0.9906	0.9897	0.9996	0.9901	0.9988
Oversampling (TS3)	Test	0.3694	0.2453	0.8218	2914	0.2615
Downsampling(TS1)	Training	0.964	0.7446	0.9867	0.838	0.9323
Downsampling (TS1)	Test	0.2119	0.3326	0.8322	0.2564	0.2346
Downsampling(TS2)	Training	0.9602	0.5906	0.9762	0.7305	0.8821
Downsampling (TS2)	Test	0.2929	0.3162	0.8438	0.3038	0.2519
Downsampling(TS3)	Training	0.9611	0.5349	0.973	0.687	0.8631
Downsampling (TS3)	Test	0.3042	0.2897	0.8354	0.2961	0.2477
Ensemble learning (TS1)	Training	0.2265	0.8895	0.9802	0.361	0.6727
Ensemble learning (TS1)	Test	0.1356	0.4883	0.8497	0.2122	0.2895
Ensemble learning (TS2)	Training	0.2024	0.8333	0.9694	0.3257	0.5552
Ensemble learning (TS2)	Test	0.1316	0.50008	0.8536	0.2084	0.2918
Ensemble learning (TS3)	Training	0.2161	0.8768	0.9785	0.3467	0.6288
Ensemble learning (TS3)	Test	0.1321	0.4953	0.8571	0.2085	**0.3006**

Table 5.7: The mean performance of standard CNNs for the different approaches.

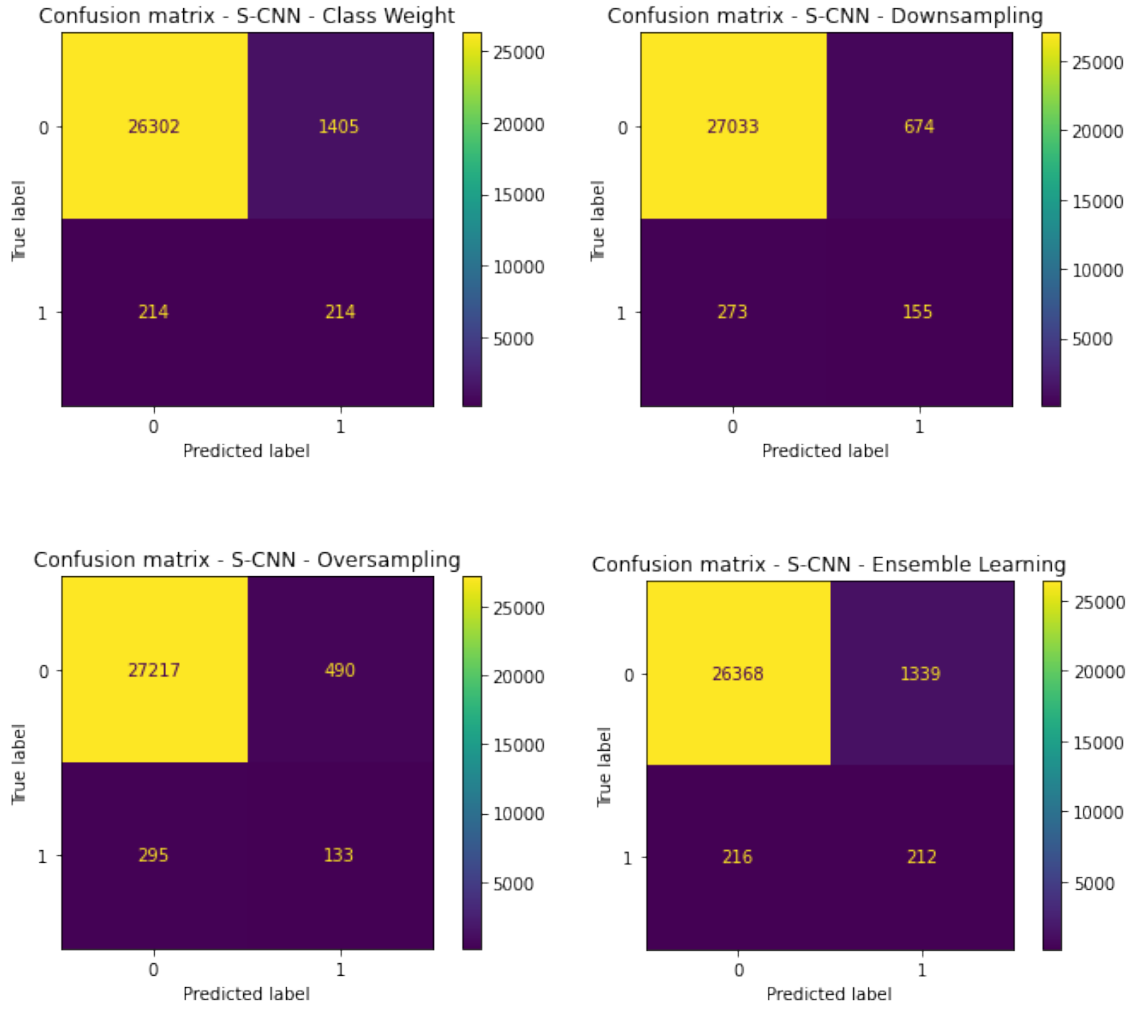


Figure 5.5: Confusion matrices of the test set for the different approaches, TS1.

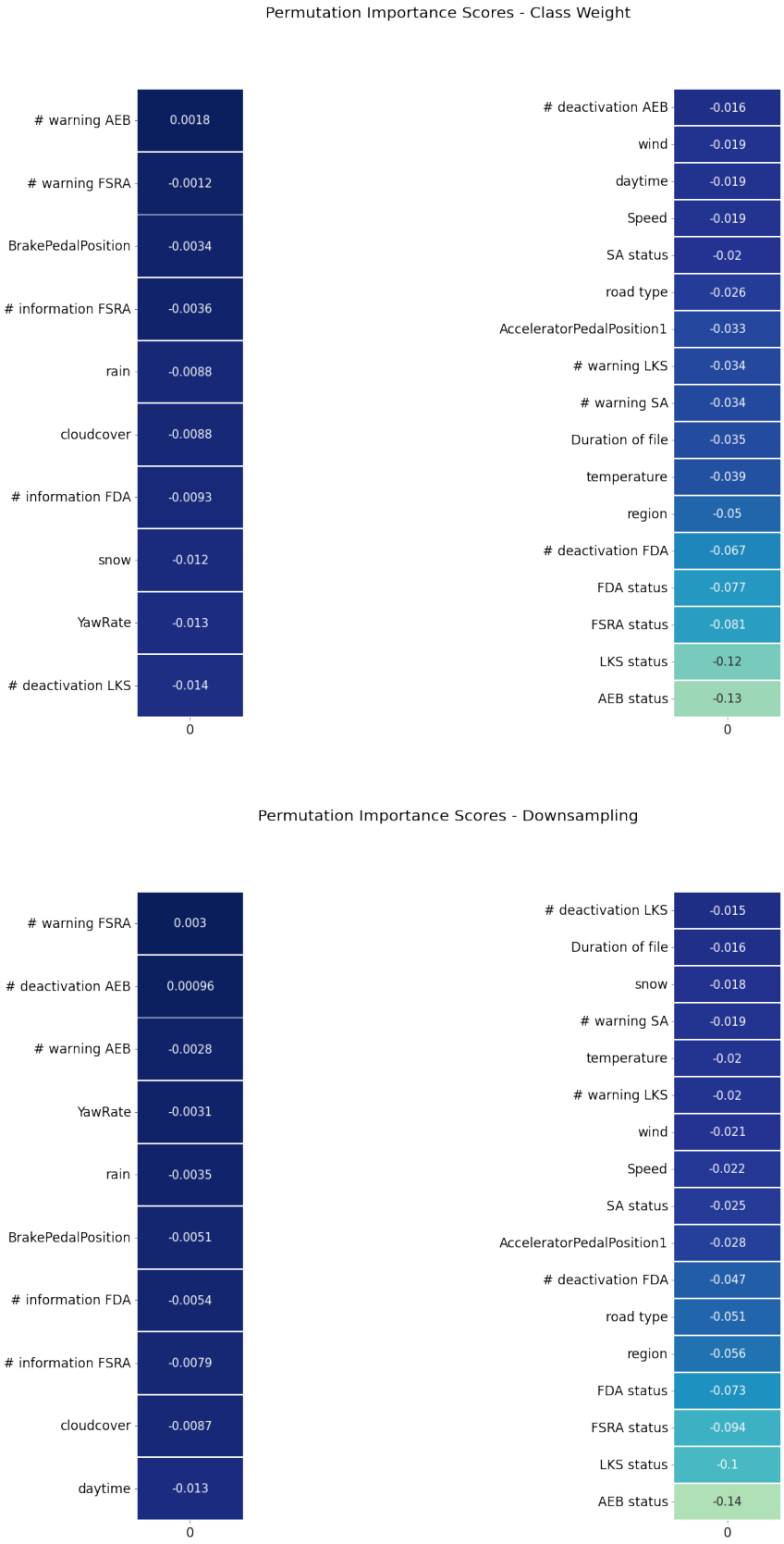


Figure 5.6: Standard CNN's permutation importance scores for class weighting and downsampling, TS1.

5.4 Dilated CNN

Similar to the previous models, the Dilated CNN classifier exhibits weak predictive power. It exhibits higher average performance than trivial compression for the cases of class weight and downsampling. In the case of ensemble learning, only for TS1 does it achieve a better average score than trivial compression. For oversampling, trivial compression achieves a better score on average. Additionally, we observe that the model overfits across all approaches. There are approaches and training setups where the dilated behave better or worse than standard CNNs. So, it is not clear which type of CNN is better for the specific task.

For the case of class weighting, TS2 achieves better results than TS1, and TS1 achieves better results than TS3 on average. However, TS3 achieves a higher best performance than TS2. For oversampling, TS3 achieves better results than TS1 and TS1 achieves better results than TS2. However, TS1 achieves a higher best performance than TS3. For downsampling, TS1 achieves better results than TS2 and TS2 achieves better results than TS3. For ensemble learning, TS1 achieves better results than TS2 and TS2 achieves better results than TS3.

For the autoencoders, we observe that the higher the compression, the larger the ARRE. Also, in cases of one layer of 64 filters and two layers with a number of 64-128 filters, the compression is the same. However, we observe different performances which can be attributed to the different dilation rates. In the other two cases, the ARRE increases significantly.

In Figures 5.7, A.11, and A.12, we can see, as in previous models, only when using class weights can the model correctly classify more than the half of the positive samples. In dilated CNNs, the maximum number of true positives have been achieved for the class weighting approach, but also a large number of false positives have occurred. We also observe the same trade-off: as the number of false positives decreases, the number of false negatives increases. There is no case where the number of false positives is lower than the number of true positives. As in standard CNNs, we can observe that the ensemble learning approach has an almost equal number of true and false positives, with false positives slightly exceeding the number of true positives.

In Figures 5.8, A.13, A.14, A.15, A.16, and A.17 we observe that the importance ranking of the features may differ among the approaches and training setups. However, AEB status and LKS status exhibit high scores across all approaches and training setups. Other features with high scores for some of the approaches and training setups are region, FSRA status, FDA status, Speed, temperature, road type, and # deactivations FDA. Notably, in any case, # FSRA warnings, and # FSRA information have small importance scores.

		Precision	Recall	ROC-AUC	F1	PR-AUC
Class weighting (TS1)	Training	0.0575	0.9547	0.9474	0.1081	0.4562
Class weighting (TS1)	Test	0.0445	0.7438	0.8351	0.0837	0.2538
Class weighting (TS2)	Training	0.0654	0.9615	0.9623	0.1222	0.543
Class weighting (TS2)	Test	0.0489	0.7235	0.8365	0.0914	0.2565
Class weighting (TS3)	Training	0.0712	0.9108	0.9434	0.132	0.4251
Class weighting (TS3)	Test	0.0522	0.6667	0.827	0.0968	0.253
Oversampling (TS1)	Training	0.875	0.9196	0.9883	0.8964	0.9589
Oversampling (TS1)	Test	0.1465	0.3894	0.8222	0.2086	0.2684
Oversampling (TS2)	Training	0.9305	0.9599	0.9961	0.9449	0.9863
Oversampling (TS2)	Test	0.198	0.3458	0.826	0.242	0.2526
Oversampling (TS3)	Training	0.9375	0.9743	0.9967	0.9553	0.9884
Oversampling (TS3)	Test	0.2219	0.345	0.8265	0.2555	0.2708
Downsampling(TS1)	Training	0.9578	0.7876	0.9891	0.8595	0.9413
Downsampling (TS1)	Test	0.2071	0.3388	0.8259	0.2527	0.2479
Downsampling(TS2)	Training	0.947	0.6495	0.9803	0.7622	0.8947
Downsampling (TS2)	Test	0.24	0.3162	0.826	0.2609	0.2452
Downsampling(TS3)	Training	0.9383	0.62	0.9752	0.7432	0.874
Downsampling (TS3)	Test	0.2388	0.3061	0.823	0.2655	0.2347
Ensemble learning (TS1)	Training	0.2565	0.8958	0.9843	0.3988	0.7234
Ensemble learning (TS1)	Test	0.1484	0.4727	0.8577	0.2259	0.2973
Ensemble learning (TS2)	Training	0.2228	0.8166	0.9687	0.35	0.5625
Ensemble learning (TS2)	Test	0.1397	0.4603	0.8413	0.2143	0.2605
Ensemble learning (TS3)	Training	0.2275	0.8311	0.9718	0.3566	0.5934
Ensemble learning (TS3)	Test	0.1364	0.4587	0.8434	0.2097	0.2604

Table 5.8: The mean performance of dilated CNNs for the different approaches.

		Precision	Recall	ROC-AUC	F1	PR-AUC
Class weighting (TS1)	Test	0.0543	0.7056	0.8437	0.1008	0.2848
Class weighting (TS2)	Test	0.0572	0.6776	0.8406	0.1056	0.2693
Class weighting (TS3)	Test	0.0476	0.6893	0.8269	0.0890	0.2718
Oversampling (TS1)	Test	0.1931	0.3388	0.8104	0.2460	0.2885
Oversampling (TS2)	Test	0.2763	0.2827	0.8260	0.2794	0.2602
Oversampling (TS3)	Test	0.3102	0.2921	0.8273	0.3008	0.2840
Downsampling (TS1)	Test	0.1743	0.3995	0.8409	0.2427	0.2681
Downsampling (TS2)	Test	0.2268	0.3318	0.8293	0.2694	0.2505
Downsampling (TS3)	Test	0.2577	0.3131	0.8203	0.2827	0.2485
Ensemble learning (TS1)	Test	0.1550	0.4766	0.8557	0.2339	0.3009
Ensemble learning (TS2)	Test	0.1382	0.4860	0.8425	0.2152	0.2642
Ensemble learning (TS3)	Test	0.1281	0.4766	0.8446	0.2020	0.2629

Table 5.9: The best performance of dilated CNNs for the different approaches.

Number of filters	Training ARRE	Validation ARRE	Test ARRE
64	0.0308	0.0309	0.031
32	0.1613	0.161	0.1623
64-128	0.043	0.0434	0.044
32-64	0.1831	0.1831	0.1845

Table 5.10: Performance of autoencoders for dilated CNN-based model.

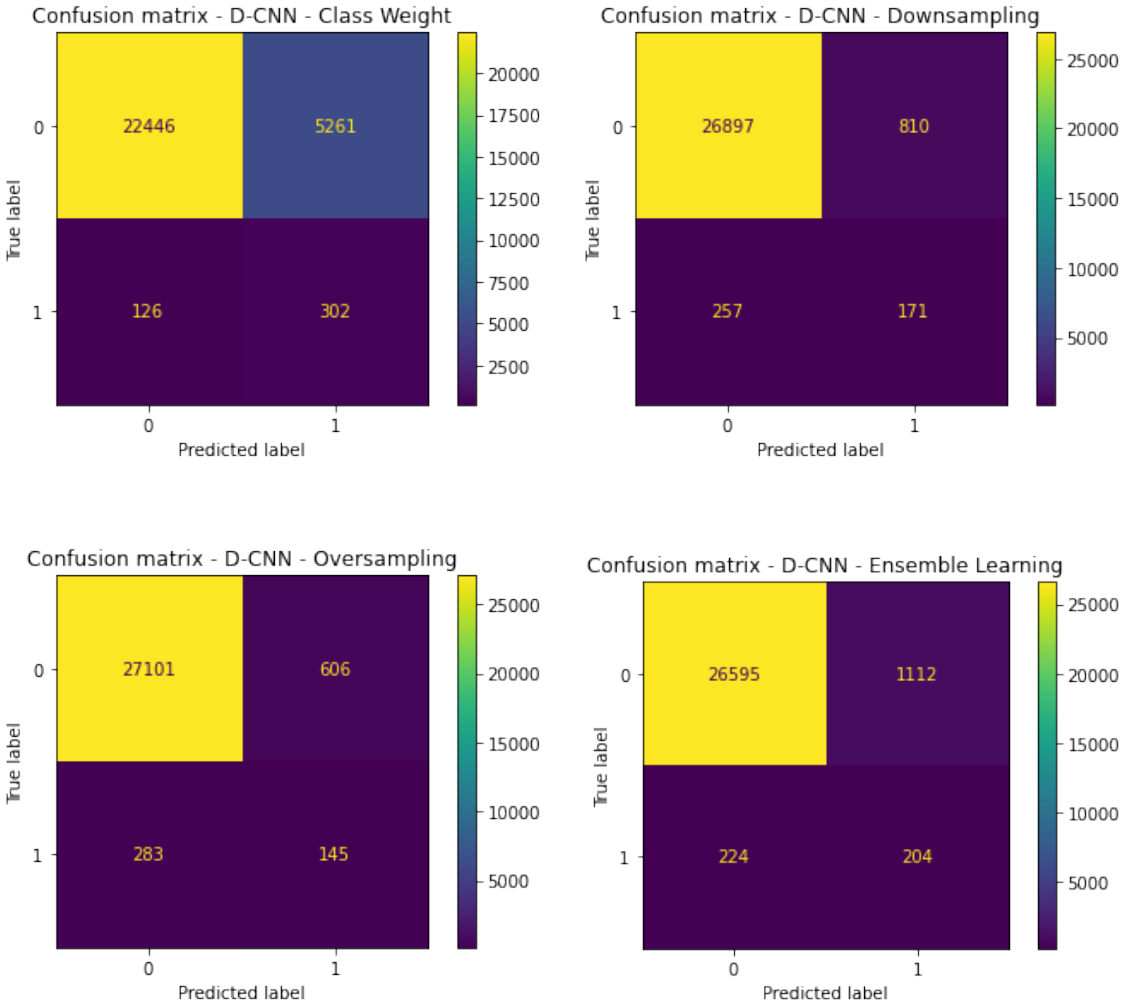


Figure 5.7: Confusion matrices of the test set for the different approaches, TS1.

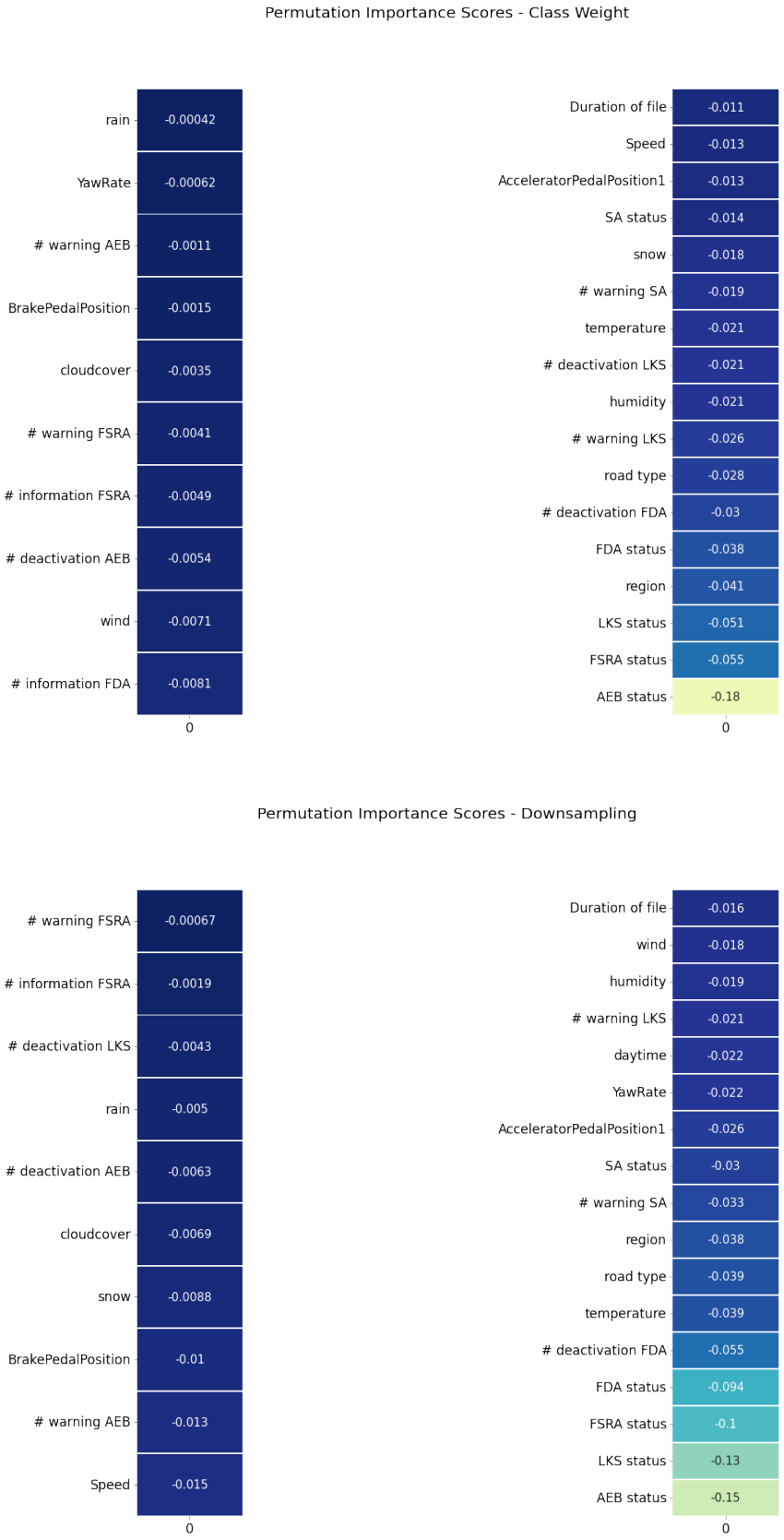


Figure 5.8: Dilated CNN's permutation importance scores for class weighting and downsampling, TS1.

5.5 Trivial compression vs CNN-based autoencoders

We observe that trivial compression outperforms the CNN-based autoencoders. In Figures 5.10, and 5.9, we plot the ARRE versus the compression rate. We define the compression rate as $\frac{\text{size of compressed signal}}{\text{size of original signal}}$. It is important to highlight that in the dataset, many features correspond to constant signals, meaning that by down-sampling, we can reconstruct the signal losslessly using only one value. This could explain why trivial compression exhibits a better reconstruction error.

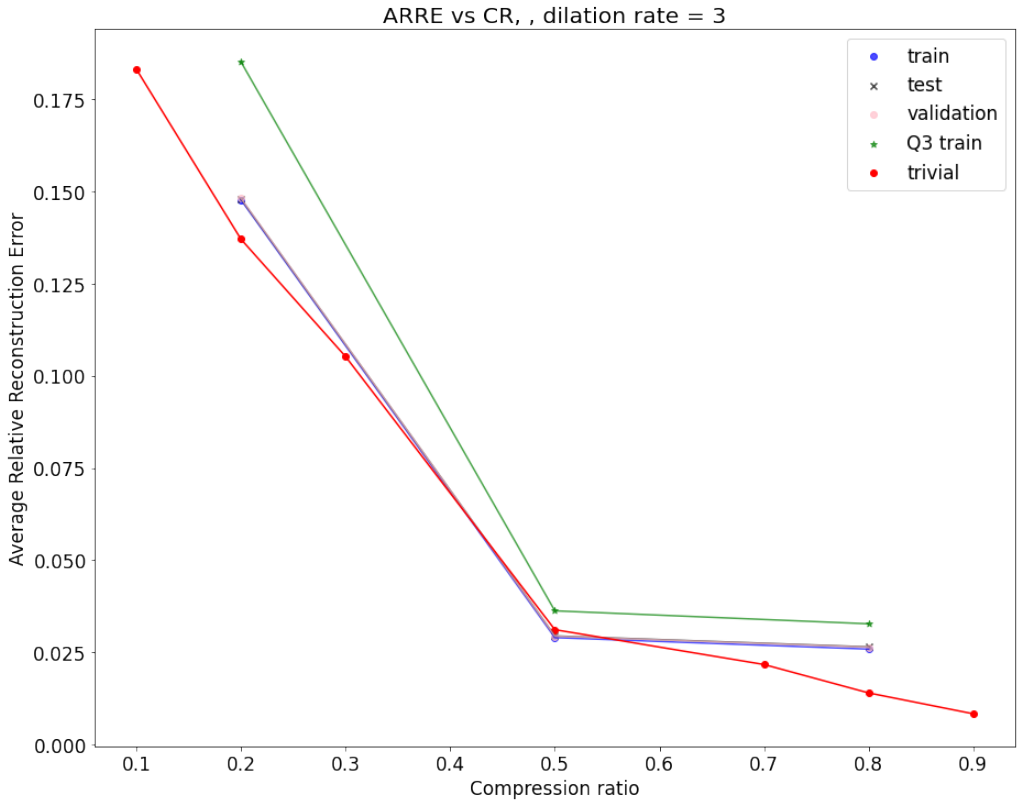


Figure 5.9: Trivial compression vs CNN-based autoencoders.

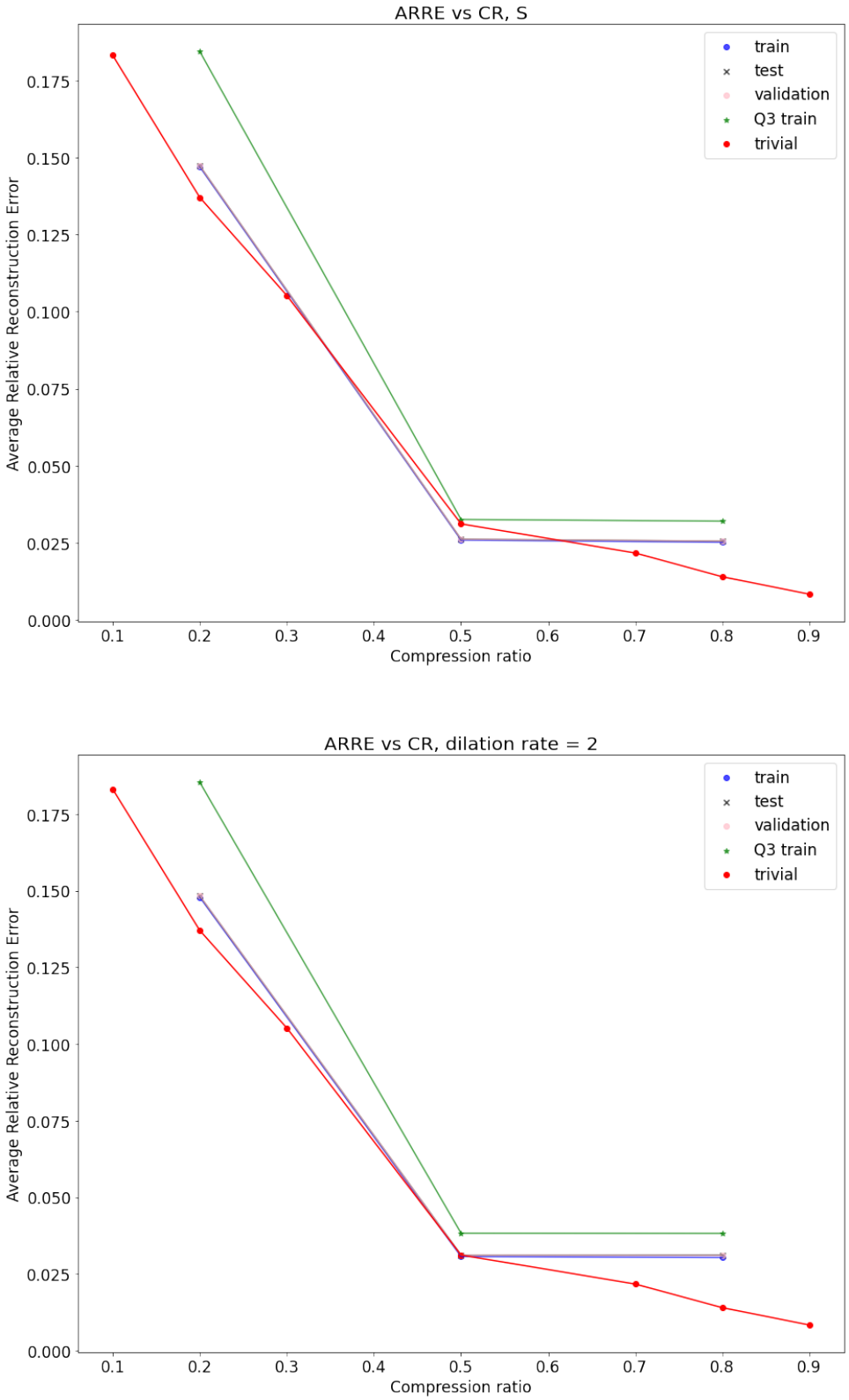


Figure 5.10: Trivial compression vs CNN-based autoencoders.

5.6 LSTM

As observed with previous models, the LSTM-based classifier shows weak predictive performance. It exhibits higher average performance than trivial compression for the cases of class weight and downsampling. In comparison with the CNN approach, there are cases where CNN outperforms and other cases where LSTM does. In general, both approaches yield similar performance based on the PR-AUC score. Additionally, we observe that the model overfits across all approaches.

For the case of class weighting, TS2 achieves better results than TS3, and TS3 achieves better results than TS1 on average. For oversampling, TS3 achieves better results than TS2 and TS2 achieves better results than TS1. For downsampling, TS2 achieves better results than TS3 and TS3 achieves better results than TS1. For ensemble learning, TS3 achieves better results than TS1 and TS1 achieves better results than TS2. Lastly, we observe that the autoencoders exhibit poorer performance than the ones used for CNNs, but the compression is much higher in this case.

In Figures 5.11, A.18, and A.19, we can see, as in previous models, only when using class weights can the model correctly classify more than the half of the positive samples and in the case of TS3 for ensemble learning. We also observe the same trade-off: as the number of false positives decreases, the number of false negatives increases. As in previous models, there is no case where the number of false positives is lower than the number of true positives.

In Figures 5.12, A.20, A.21, A.22, A.23, and A.24 we observe that the importance ranking of the features may differ among the approaches and training setups. However, AEB status, LKS status, FSRA status, # deactivation of FDA, and FDA status exhibit high scores across all approaches. Other features with a moderate score for some of the approaches and training setups are region, Speed, Acceleration Pedal Position, # deactivation of LKS, and road type. Notably, in any case, # FSRA warnings, and # FSRA information have small importance scores.

Number of filters	Training ARRE	Validation ARRE	Test ARRE
64	0.203	0.204	0.204
128	0.142	0.145	0.144

Table 5.11: Performance of autoencoders for LSTM-based model.

		Precision	Recall	ROC-AUC	F1	PR-AUC
Class weighting (TS1)	Training	0.0687	0.942	0.9522	0.1281	0.4535
Class weighting (TS1)	Test	0.0491	0.6838	0.8225	0.0916	0.2796
Class weighting (TS2)	Training	0.0993	0.9633	0.9707	0.1796	0.522
Class weighting (TS2)	Test	0.0682	0.676	0.848	0.1236	0.2928
Class weighting (TS3)	Training	0.0761	0.9293	0.9437	0.14	0.4307
Class weighting (TS3)	Test	0.056	0.7017	0.8401	0.1032	0.291
Oversampling (TS1)	Training	0.7978	0.7249	0.9561	0.7592	0.853
Oversampling (TS1)	Test	0.1256	0.4525	0.8249	0.1962	0.2556
Oversampling (TS2)	Training	0.8119	0.8117	0.9674	0.8113	0.8883
Oversampling (TS2)	Test	0.1232	0.4618	0.8222	0.1945	0.2803
Oversampling (TS3)	Training	0.8287	0.8484	0.9757	0.8382	0.9106
Oversampling (TS3)	Test	0.1291	0.4572	0.8106	0.2011	0.2815
Downsampling(TS1)	Training	0.8451	0.5725	0.9506	0.6725	0.775
Downsampling (TS1)	Test	0.212	0.3731	0.8233	0.2647	0.2643
Downsampling(TS2)	Training	0.8669	0.5575	0.9654	0.6756	0.8152
Downsampling (TS2)	Test	0.2277	0.3489	0.8428	0.2718	0.2864
Downsampling(TS3)	Training	0.8414	0.5815	0.9605	0.06867	0.802
Downsampling (TS3)	Test	0.1981	0.3816	0.8352	0.26	0.2855
Ensemble learning (TS1)	Training	0.1432	0.6096	0.9179	0.2318	0.328
Ensemble learning (TS1)	Test	0.1172	0.4751	0.8315	0.188	0.259
Ensemble learning (TS2)	Training	0.1571	0.5376	0.8898	0.2423	0.2892
Ensemble learning (TS2)	Test	0.134	0.4533	0.8223	0.206	0.2503
Ensemble learning (TS3)	Training	0.144	0.5996	0.9064	0.2321	0.3355
Ensemble learning (TS3)	Test	0.1251	0.5023	0.8317	0.2002	0.2749

Table 5.12: The mean performance of the LSTM-based classifier for the different approaches.

		Precision	Recall	ROC-AUC	F1	PR-AUC
Class weighting (TS1)	Test	0.04810	0.6919	0.8314	0.0899	0.2885
Class weighting (TS2)	Test	0.0630	0.7220	0.8572	0.1159	0.2976
Class weighting (TS3)	Test	0.0547	0.7360	0.8508	0.1019	0.2969
Oversampling (TS1)	Test	0.1211	0.4696	0.8310	0.1925	0.2663
Oversampling (TS2)	Test	0.1215	0.4579	0.8068	0.1921	0.2903
Oversampling (TS3)	Test	0.1272	0.4977	0.8460	0.2026	0.2957
Downsampling (TS1)	Test	0.1977	0.4019	0.8380	0.2650	0.2736
Downsampling (TS2)	Test	0.1844	0.3972	0.8484	0.2519	0.2920
Downsampling (TS3)	Test	0.1820	0.4019	0.8397	0.2505	0.2915
Ensemble learning (TS1)	Test	0.1142	0.4766	0.8326	0.182	0.2608
Ensemble learning (TS2)	Test	0.1195	0.4930	0.8256	0.1923	0.2592
Ensemble learning (TS3)	Test	0.1233	0.5140	0.8312	0.1989	0.2782

Table 5.13: The best performance of the LSTM-based classifier for the different approaches.

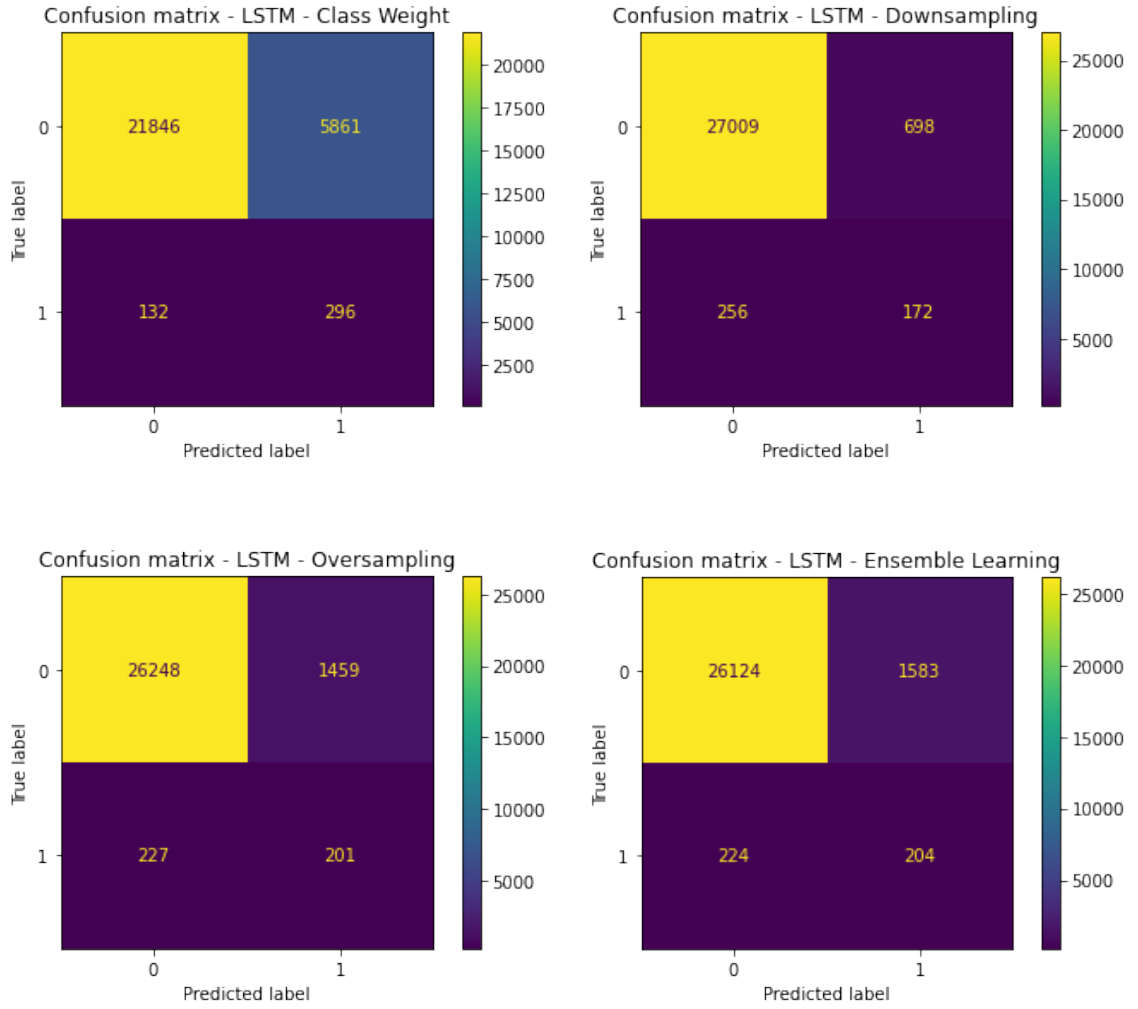


Figure 5.11: Confusion matrices of the test set for the different approaches, TS1.

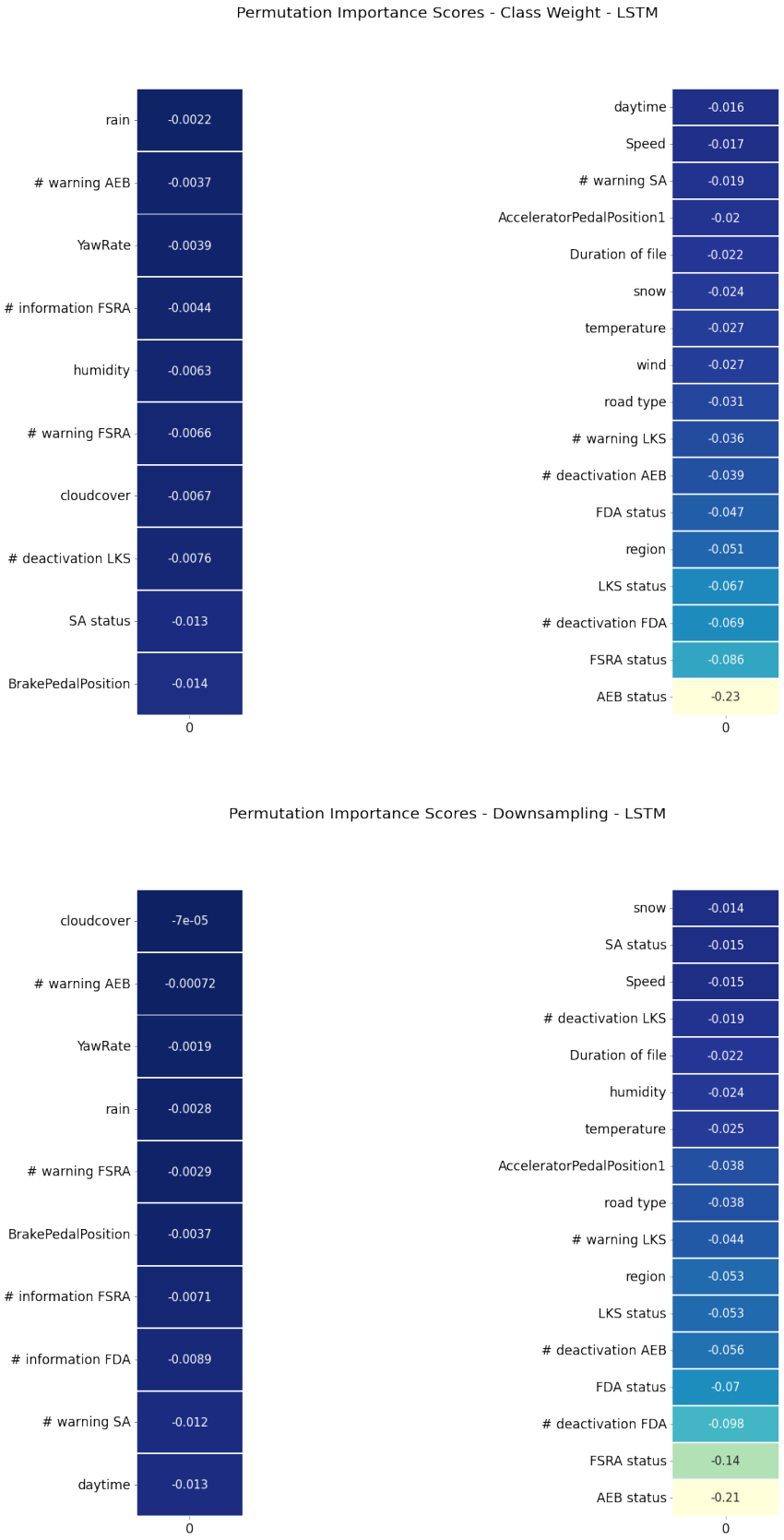


Figure 5.12: LSTM's permutation importance scores for class weighting and downsampling, TS1.

5.7 GRU

The model exhibits poor predictive power. It exhibits higher average performance than trivial compression for the cases of class weight and downsampling. Also, it has a better performance in the case of oversampling on average. In comparison with CNN and LSTM approaches there are cases where CNN and LSTM outperform and other cases where GRU does. In general, CNN, LSTM, and GRU approaches yield similar performance based on PR-AUC score. Additionally, we observe that the model overfits across all approaches.

For the case of class weighting, TS2 achieves better results than TS3, and TS3 achieves better results than TS1 on average. However, TS3 achieves a higher best performance than TS2. For oversampling, TS2 achieves better results than TS3 and TS3 achieves better results than TS1 on average. However, TS1 achieves a higher best performance than TS2 and TS2 than TS3. For downsampling, TS2 achieves better results than TS1 and TS1 achieves better results than TS3 on average. However, TS1 achieves a higher best performance than TS2 and TS2 than TS3. For ensemble learning, TS3 achieves better results than TS2 and TS2 achieves better results than TS1. Lastly, we observe that the autoencoders exhibit poorer performance than the ones used for CNNs, but the compression is much higher in this case.

In Figures 5.13, A.25, and A.26, we can see, as in previous models, only when using class weights can the model correctly classify more than the half of the positive samples. We also observe the same trade-off: as the number of false positives decreases, the number of false negatives increases. As in previous models, there is no case where the number of false positives is lower than the number of true positives.

In Figures 5.14, A.27, A.28, A.29, A.30, and A.31, we observe that the importance ranking of the features may differ among the approaches and training setups. However, AEB status, LKS status, FSRA status, # deactivation of FDA, FDA status, and # warnings of LKS exhibit high scores across all approaches. Other features with a moderate score for some of the approaches and training setups are region, # warnings of FDA, and road type. Notably, in any case, # FSRA warnings, and # FSRA information have small importance scores.

Number of filters	Training ARRE	Validation ARRE	Test ARRE
64	0.201	0.203	0.203
128	0.13	0.140	0.1399
128-128	0.105	0.107	0.107

Table 5.14: Performance of autoencoders for GRU-based model.

		Precision	Recall	ROC-AUC	F1	PR-AUC
Class weighting (TS1)	Train	0.0642	0.8569	0.9163	0.1193	0.3243
Class weighting (TS1)	Test	0.05009	0.676	0.8215	0.0946	0.2603
Class weighting (TS2)	Training	0.0735	0.8886	0.9347	0.1357	0.3709
Class weighting (TS2)	Test	0.0593	0.7079	0.8456	0.1094	0.2817
Class weighting (TS3)	Training	0.0698	0.9221	0.9386	0.1297	0.382
Class weighting (TS3)	Test	0.0558	0.729	0.8487	0.1036	0.2773
Oversampling (TS1)	Training	0.8022	0.6301	0.9365	0.7051	0.8195
Oversampling (TS1)	Test	0.1473	0.4369	0.8196	0.2199	0.2778
Oversampling (TS2)	Training	0.833	0.7689	0.9702	0.789	0.8942
Oversampling (TS2)	Test	0.1539	0.4611	0.8423	0.2289	0.2982
Oversampling (TS3)	Training	0.7987	0.707	0.9508	0.7489	0.8456
Oversampling (TS3)	Test	0.1385	0.4727	0.8479	0.2139	0.2802
Downsampling(TS1)	Training	0.8825	0.4629	0.9332	0.5953	0.7354
Downsampling (TS1)	Test	0.2669	0.3248	0.8236	0.2827	0.2763
Downsampling(TS2)	Training	0.8745	0.4964	0.95533	0.6295	0.773
Downsampling (TS2)	Test	0.2314	0.3201	0.8538	0.2618	0.2793
Downsampling(TS3)	Training	0.8272	0.4049	0.9243	0.5337	0.6798
Downsampling (TS3)	Test	0.2838	0.3107	0.842	0.2839	0.2684
Ensemble learning (TS1)	Training	0.1505	0.5118	0.8942	0.2326	0.3194
Ensemble learning (TS1)	Test	0.1334	0.4408	0.8288	0.2041	0.2672
Ensemble learning (TS2)	Training	0.1443	0.6322	0.9251	0.2349	0.3562
Ensemble learning (TS2)	Test	0.1183	0.4875	0.8543	0.1903	0.2813
Ensemble learning (TS3)	Training	0.1485	0.6277	0.9255	0.2402	0.3581
Ensemble learning (TS3)	Test	0.1188	0.4766	0.853	0.1902	0.2815

Table 5.15: The mean performance of GRU-based classifiers for the different approaches.

		Precision	Recall	ROC-AUC	F1	PR-AUC
Class weighting (TS1)	Test	0.0548	0.6752	0.8255	0.1014	0.2706
Class weighting (TS2)	Test	0.0566	0.7150	0.8487	0.1049	0.2831
Class weighting (TS3)	Test	0.0557	0.7290	0.8495	0.1036	0.2877
Oversampling (TS1)	Test	0.1676	0.4346	0.8226	0.2419	0.3142
Oversampling (TS2)	Test	0.1393	0.4836	0.8414	0.2163	0.3111
Oversampling (TS3)	Test	0.1335	0.4813	0.8443	0.2090	0.2906
Downsampling (TS1)	Test	0.1846	0.3972	0.8257	0.2420	0.2791
Downsampling (TS2)	Test	0.2864	0.2850	0.8507	0.2857	0.2831
Downsampling (TS3)	Test	0.2901	0.3341	0.8535	0.3105	0.2899
Ensemble learning (TS1)	Test	0.1325	0.4346	0.83	0.2031	0.2723
Ensemble learning (TS2)	Test	0.1181	0.4907	0.8543	0.1904	0.2843
Ensemble learning (TS3)	Test	0.1163	0.4766	0.8549	0.1870	0.2844

Table 5.16: The best performance of GRU-based classifiers for the different approaches.

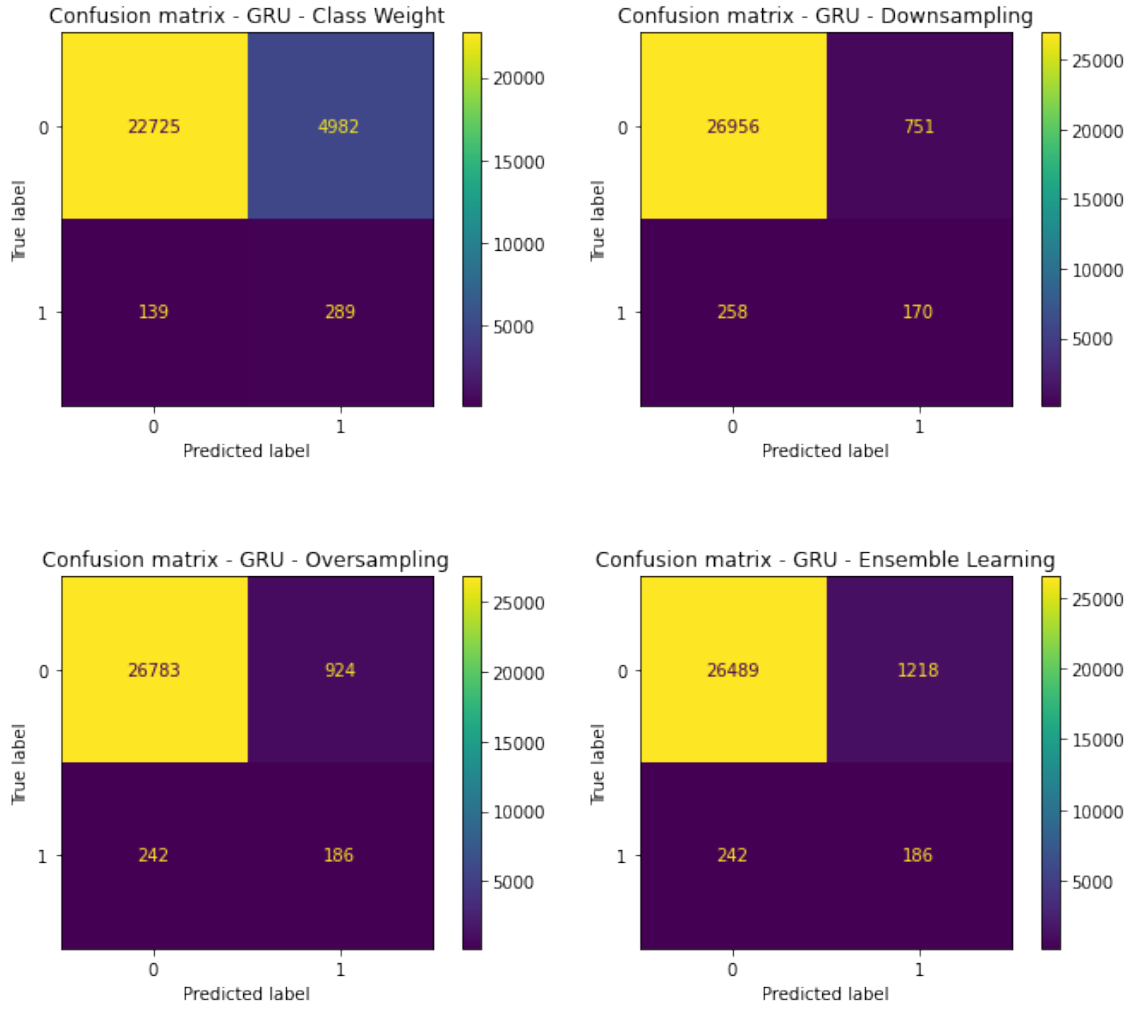


Figure 5.13: Confusion matrices of the test set for the different approaches, TS1.

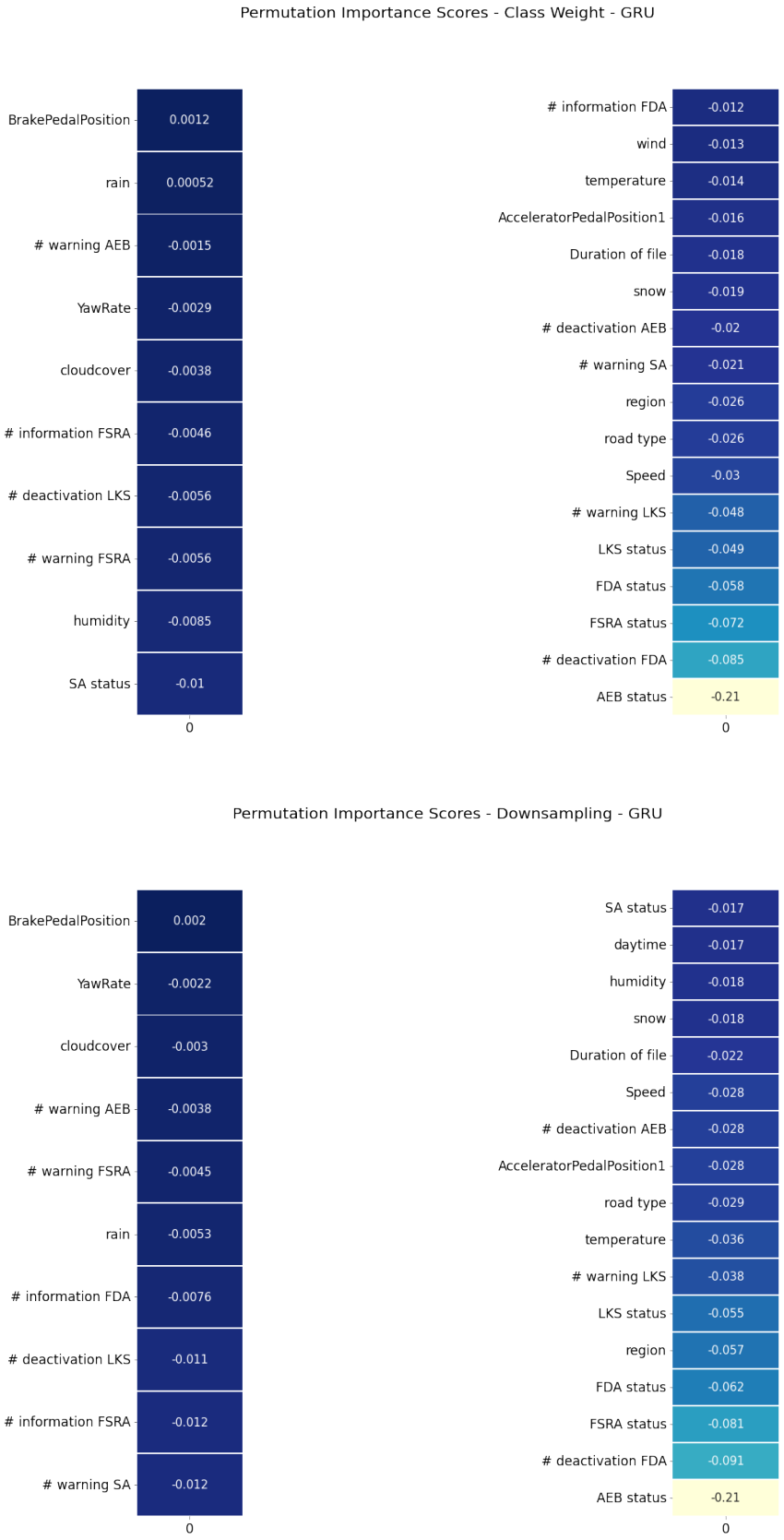


Figure 5.14: GRU's permutation importance scores for class weighting and downsampling, TS1.

5.8 Bi-LSTM

The Bi-LSTM-based classifier yields weak predictive power. It exhibits higher average performance than trivial compression for the cases of class weight and downsampling. Additionally, it shows better performance in the case of oversampling on average. In comparison with CNN and other RNN approaches, there are cases where CNN and other RNN outperform and other cases where GRU does. In general, CNN and RNN approaches yield similar performance based on PR-AUC score. Additionally, we observe that the model overfits across all approaches.

For the case of class weighting, TS2 achieves better results than TS3, and TS3 achieves better results than TS1. For oversampling, TS1 achieves better results than TS2 and TS2 achieves better results than TS3. For downsampling, TS2 achieves better results than TS1 and TS1 achieves better results than TS3. For ensemble learning, TS1 achieves better results than TS3 and TS3 achieves better results than TS2. Lastly, we observe that the autoencoders exhibit poorer performance than the ones used for CNNs, but the compression is much higher in this case.

In Figures 5.15, A.32, and A.33, we can see, in the of class weighting and ensemble learning (TS1, TS3) the model can correctly classify more than the half of the positive samples. We also observe the same trade-off: as the number of false positives decreases, the number of false negatives increases. As in previous models, there is no case where the number of false positives is lower than the number of true positives.

In Figures 5.16, A.34, A.35, A.36, A.37, and A.38 we observe that the importance ranking of the features may differ among the approaches and training setups. However, AEB status, LKS status, FSRA status, # deactivation of FDA, and FDA status exhibit high scores across all approaches. Other features with moderate scores for some of the approaches and training setups are region, temperature, Duration file, and road type. Notably, in any case, # FSRA warnings, and # FSRA information have small importance scores.

Number of filters	Training ARRE	Validation ARRE	Test ARRE
32	0.5902	0.5903	0.5904
64	0.574	0.575	0.575
128	0.5684	0.5685	0.5682

Table 5.17: Performance of autoencoders for BiLSTM-based model.

		Precision	Recall	ROC-AUC	F1	PR-AUC
Class weighting (TS1)	Training	0.0828	0.933	0.9584	0.1518	0.4626
Class weighting (TS1)	Test	0.0598	0.6768	0.8447	0.1097	0.2708
Class weighting (TS2)	Training	0.1034	0.9479	0.9683	0.186	0.4973
Class weighting (TS2)	Test	0.0729	0.6674	0.8534	0.1311	0.2923
Class weighting (TS3)	Training	0.0725	0.9434	0.9521	0.1343	0.4189
Class weighting (TS3)	Test	0.0556	0.7235	0.8574	0.103	0.277
Oversampling (TS1)	Training	0.9243	0.9521	0.9957	0.9377	0.9839
Oversampling (TS1)	Test	0.1904	0.4003	0.8399	0.2545	0.2997
Oversampling (TS2)	Training	0.916	0.9282	0.9938	0.922	0.9764
Oversampling (TS2)	Test	0.1731	0.3793	0.8237	0.2374	0.2838
Oversampling (TS3)	Training	0.9252	0.9444	0.9951	0.9346	0.9815
Oversampling (TS3)	Test	0.1842	0.3699	0.8284	0.2425	0.2858
Downsampling(TS1)	Training	0.8613	0.6001	0.9632	0.7062	0.8213
Downsampling (TS1)	Test	0.2027	0.3886	0.8527	0.2655	0.2743
Downsampling(TS2)	Training	0.9036	0.668	0.9767	0.7677	0.8767
Downsampling (TS2)	Test	0.1987	0.3855	0.8494	0.2613	0.286
Downsampling(TS3)	Training	0.8755	0.5793	0.9595	0.6609	0.8157
Downsampling (TS3)	Test	0.2291	0.3388	0.8449	0.2425	0.2615
Ensemble learning (TS1)	Training	0.1558	0.7636	0.95	0.2588	0.4306
Ensemble learning (TS1)	Test	0.1095	0.5358	0.8658	0.1819	0.2886
Ensemble learning (TS2)	Training	0.1424	0.6884	0.9354	0.2359	0.3828
Ensemble learning (TS2)	Test	0.1051	0.4922	0.8492	0.1732	0.2751
Ensemble learning (TS3)	Training	0.1517	0.7292	0.9439	0.2511	0.4163
Ensemble learning (TS3)	Test	0.1078	0.5016	0.8498	0.1774	0.2824

Table 5.18: The mean performance of BiLSTM-based classifiers for the different approaches.

		Precision	Recall	ROC-AUC	F1	PR-AUC
Class weighting (TS1)	Test	0.069	0.6565	0.8487	0.1249	0.2841
Class weighting (TS2)	Test	0.0767	0.6729	0.8523	0.1376	0.3032
Class weighting (TS3)	Test	0.0508	0.7547	0.8585	0.0952	0.2869
Oversampling (TS1)	Test	0.2225	0.3598	0.8427	0.2750	0.3069
Oversampling (TS2)	Test	0.1587	0.3855	0.8244	0.2248	0.2933
Oversampling (TS3)	Test	0.1637	0.4065	0.8341	0.2334	0.3020
Downsampling (TS1)	Test	0.2165	0.3995	0.8548	0.2808	0.2841
Downsampling (TS2)	Test	0.2014	0.3925	0.8530	0.2662	0.2920
Downsampling (TS3)	Test	0.2332	0.3154	0.8513	0.2681	0.2784
Ensemble learning (TS1)	Test	0.1122	0.5444	0.8658	0.1860	0.2903
Ensemble learning (TS2)	Test	0.1066	0.4953	0.8515	0.1755	0.2797
Ensemble learning (TS3)	Test	0.1077	0.5187	0.8542	0.1784	0.2837

Table 5.19: The best performance of BiLSTM-based for the different approaches.

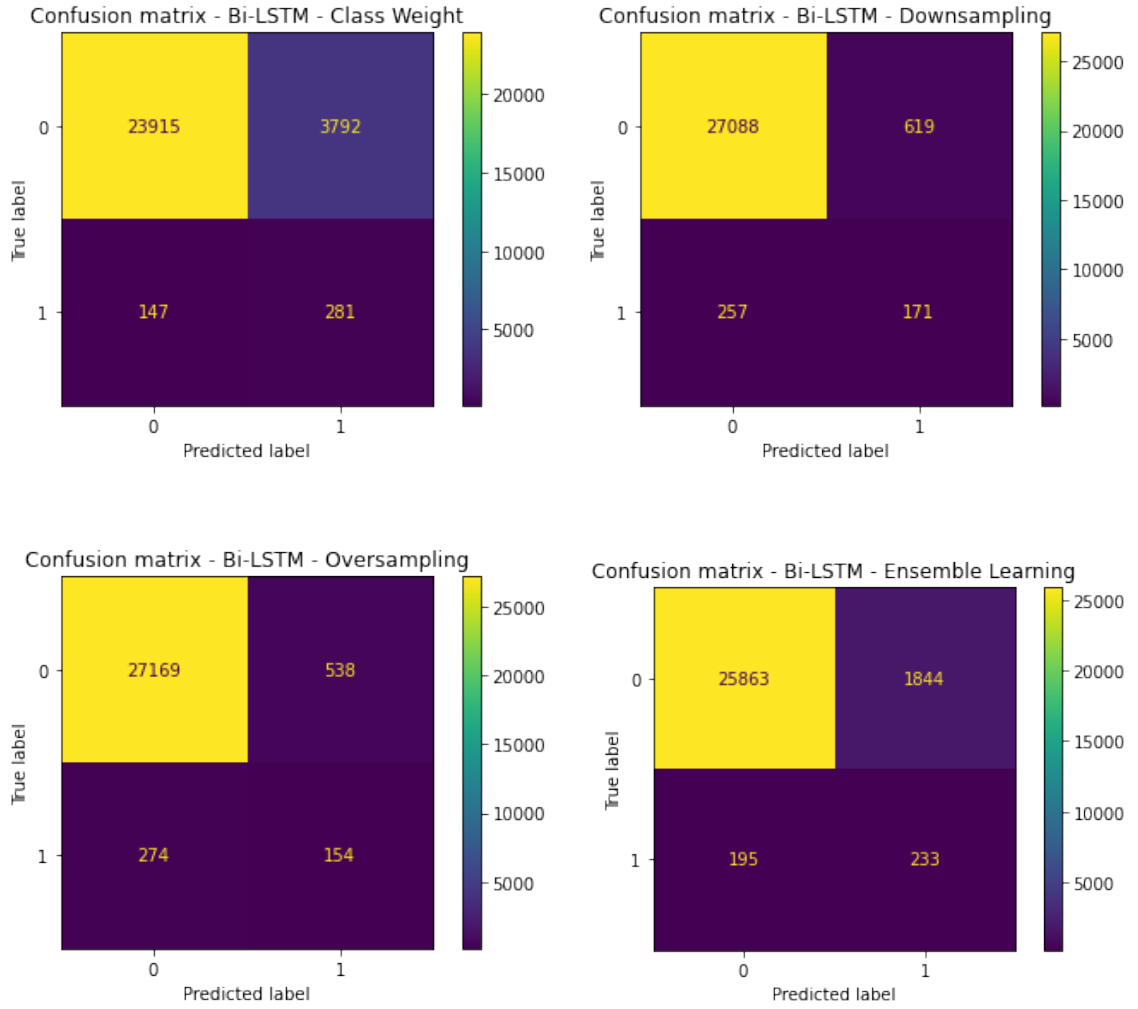


Figure 5.15: Confusion matrices of the test set for the different approaches, TS1.

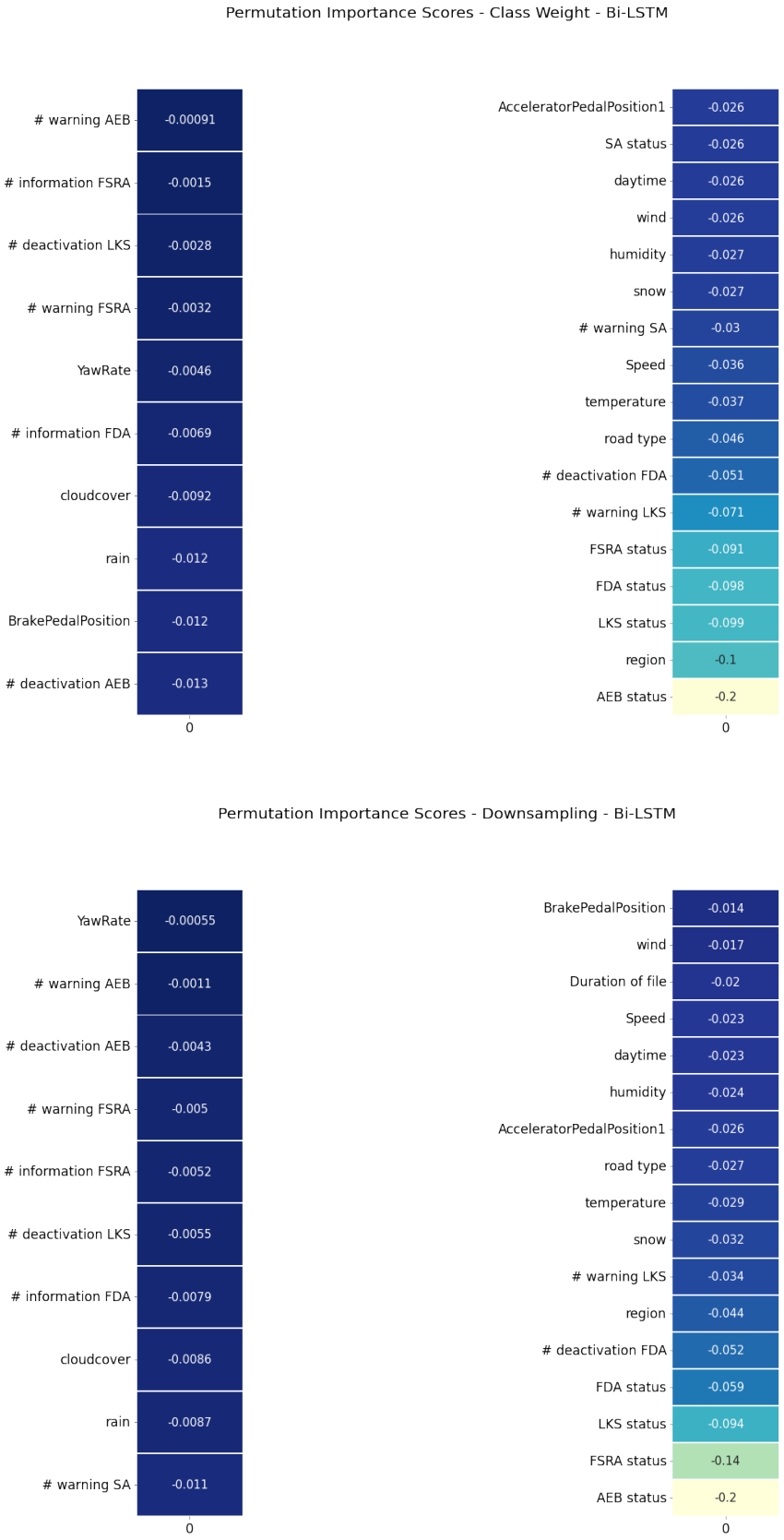


Figure 5.16: BiLSTM's permutation importance scores for class weighting and downsampling, TS1.

5.9 Parallel CNN-LSTM-based classifier

The Parallel CNN-LSTM-based classifier has limited predictive power. It exhibits higher performance than trivial compression for all cases apart from the average performance of class weighting TS3. Additionally, it shows better performance than standard CNN-based classifiers. It outperforms the LSTM-based classifier only for the cases of oversampling and ensemble learning. In comparison with dilated CNN and other RNN approaches, there are cases where Parallel CNN-LSTM outperforms them and others it does not. In general, in the cases of oversampling and ensemble learning, it exhibits a better performance. However, it yields a similar performance based on the PR-AUC score with other models. Additionally, we observe that the model overfits across all approaches.

For the case of class weighting, TS2 achieves better results than TS3, and TS3 achieves better results than TS1 on average. However, TS1 achieves a higher best performance than TS3. For oversampling, TS1 achieves better results than TS2 and TS2 achieves better results than TS3. For downsampling, TS3 achieves better results than TS2 and TS2 achieves better results than TS1. For ensemble learning, TS3 achieves better results than TS2, and TS2 achieves better results than TS1.

In Figures 5.17, A.39, and A.40, we can see, in the class weighting and ensemble learning (TS1) the model can correctly classify more than the half of the positive samples. We also observe the same trade-off: as the number of false positives decreases, the number of false negatives increases. As in previous models, there is no case where the number of false positives is lower than the number of true positives.

In Figures 5.18, A.41, A.42, A.43, A.44, and A.45 we observe that the importance ranking of the features may differ among the approaches and training setups. However, AEB status, LKS status, FSRA status, # deactivation of FDA, and FDA status exhibit high scores across all approaches. Other features with moderate scores for some of the approaches and training setups are region, temperature, Duration file, road type, speed, and acceleration pedal position. Notably, # FSRA warnings and # FSRA information have small importance scores. Only in the case of class weighting TS3, they do have important scores.

Type	Number of filters	Training ARRE	Validation ARRE	Test ARRE
LSTM	64	0.203	0.204	0.204
LSTM	128	0.142	0.145	0.144
CNN	86	0.0230	0.0232	0.0231
CNN	128	0.0323	0.0326	0.325

Table 5.20: Performance of autoencoders for Parallel CNN-LSTM-based classifiers.

		Precision	Recall	ROC-AUC	F1	PR-AUC
Class weighting (TS1)	Training	0.1699	0.9855	0.9901	0.2859	0.7229
Class weighting (TS1)	Test	0.0934	0.5701	0.8336	0.1581	0.2659
Class weighting (TS2)	Training	0.1222	0.9787	0.9822	0.2162	0.6071
Class weighting (TS2)	Test	0.0756	0.6145	0.838	0.1339	0.2763
Class weighting (TS3)	Training	0.1607	0.9751	0.987	0.2754	0.6562
Class weighting (TS3)	Test	0.0963	0.5576	0.8326	0.1591	0.2662
Oversampling (TS1)	Training	0.9739	0.9873	0.9994	0.9805	0.9975
Oversampling (TS1)	Test	0.293	0.3411	0.8272	0.312	0.3099
Oversampling (TS2)	Training	0.9076	0.8528	0.9855	0.8789	0.9506
Oversampling (TS2)	Test	0.2081	0.4026	0.8337	0.2736	0.3061
Oversampling (TS3)	Training	0.923	0.9361	0.9936	0.9294	0.9777
Oversampling (TS3)	Test	0.1998	0.3871	0.8308	0.2624	0.2989
Downsampling(TS1)	Training	0.9406	0.8012	0.9888	0.865	0.9419
Downsampling (TS1)	Test	0.1758	0.3614	0.8322	0.2365	0.2352
Downsampling(TS2)	Training	0.9556	0.692	0.9792	0.7991	0.905
Downsampling (TS2)	Test	0.2327	0.3419	0.8301	0.2729	0.2528
Downsampling(TS3)	Training	0.9533	0.7178	0.9844	0.8183	0.9191
Downsampling (TS3)	Test	0.2283	0.303	0.83	0.2597	0.2532
Ensemble learning (TS1)	Training	0.2242	0.8827	0.9788	0.3576	0.6561
Ensemble learning (TS1)	Test	0.1362	0.4922	0.8493	0.2133	0.2915
Ensemble learning (TS2)	Training	0.201	0.8184	0.9682	0.3226	0.5481
Ensemble learning (TS2)	Test	0.1334	0.5055	0.855	0.211	0.294
Ensemble learning (TS3)	Training	0.2056	0.8342	0.9696	0.3299	0.5591
Ensemble learning (TS3)	Test	0.1335	0.5047	0.8535	0.211	0.2948

Table 5.21: The mean performance of Parallel CNN-LSTM-based classifiers for the different approaches.

		Precision	Recall	ROC-AUC	F1	PR-AUC
Class weighting (TS1)	Test	0.0976	0.5327	0.8318	0.1650	0.2779
Class weighting (TS2)	Test	0.0867	0.5864	0.8407	0.1510	0.2821
Class weighting (TS3)	Test	0.1003	0.5397	0.8331	0.1691	0.2728
Oversampling (TS1)	Test	0.3248	0.3271	0.8256	0.3260	0.3151
Oversampling (TS2)	Test	0.1894	0.4276	0.8320	0.2626	0.3100
Oversampling (TS3)	Test	0.2285	0.3902	0.8310	0.2882	0.3068
Downsampling (TS1)	Test	0.1879	0.3855	0.8338	0.2527	0.2558
Downsampling (TS2)	Test	0.2869	0.3131	0.8235	0.2994	0.2593
Downsampling (TS3)	Test	0.2161	0.3318	0.8337	0.2618	0.2674
Ensemble learning (TS1)	Test	0.1372	0.4977	0.8475	0.2150	0.2936
Ensemble learning (TS2)	Test	0.1292	0.5164	0.8566	0.2066	0.2960
Ensemble learning (TS3)	Test	0.1333	0.5093	0.8533	0.2112	0.2961

Table 5.22: The best performance of Parallel CNN-LSTM-based for the different approaches.

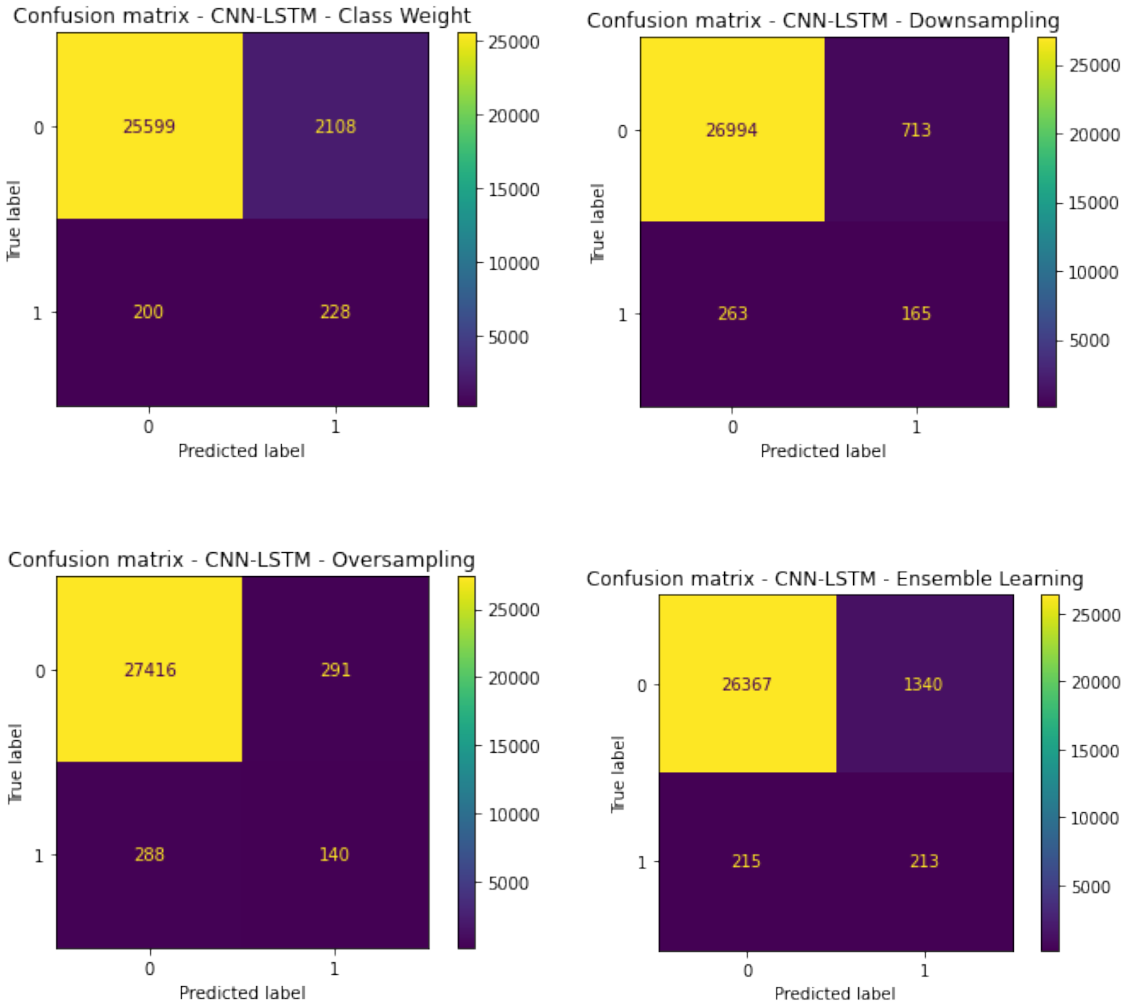


Figure 5.17: Confusion matrices of the test set for the different approaches, TS1.

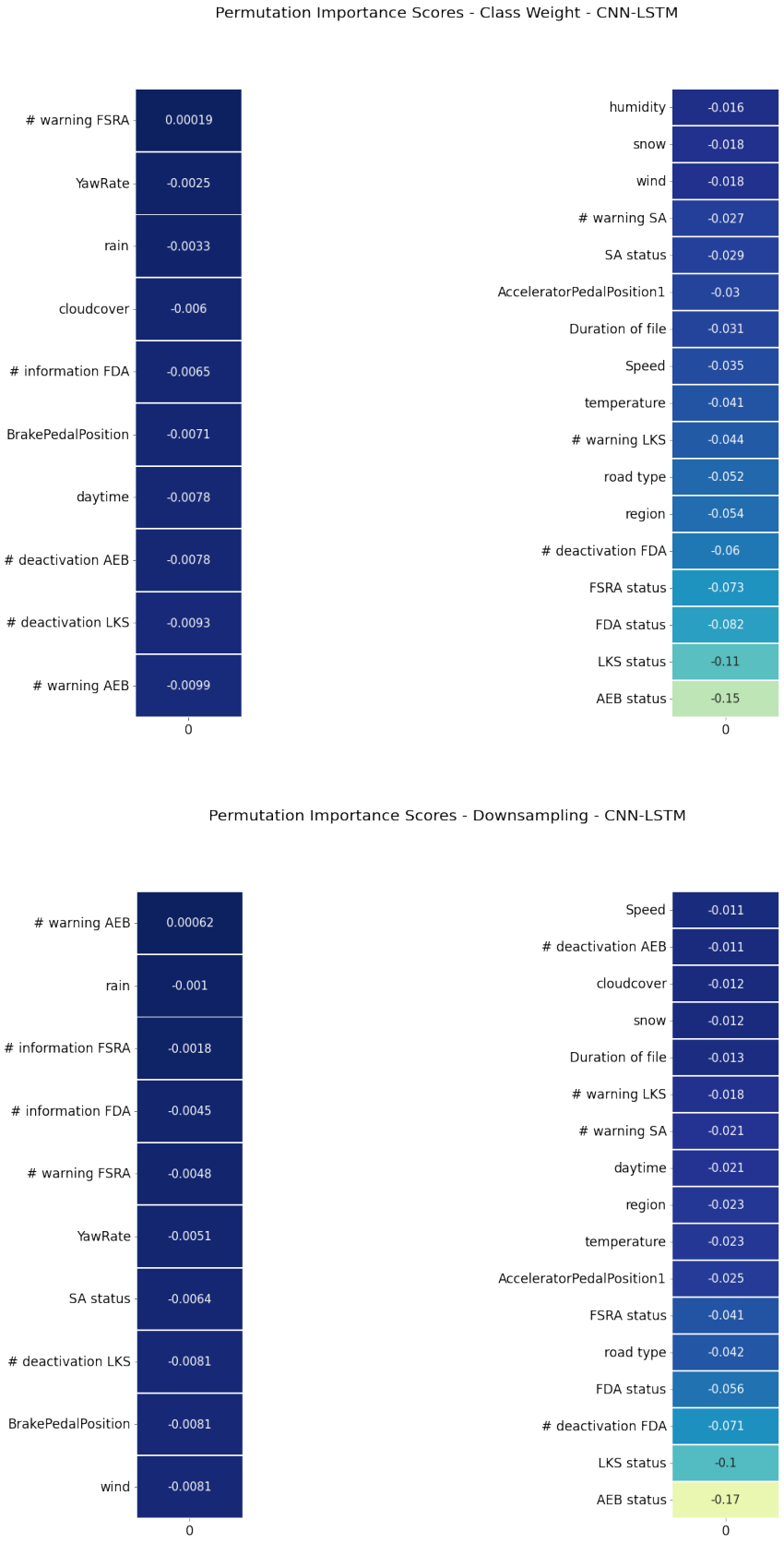


Figure 5.18: Parallel CNN-LSTM's permutation importance scores for class weighting and downsampling, TS1.

5.10 BiLSTM autoencoder

In Figure 5.19, we illustrate the histogram of the RRE of negative and positive samples on the test set. We also depict the two thresholds, if the value of RRE is greater than the threshold, we classify the sample as positive. We can observe, that in both cases the distribution of RRE for negative and positive samples overlap. Additionally, in Tables 5.25 and 5.24, we observe that we cannot detect the positive samples.

In Table 5.23, we observe that the autoencoder's performance using MinMax normalization outperforms the performance using z-score standardization. Using MinMax normalization the signals take values between 0 and 1. In this case, the output layer of the autoencoder uses a sigmoid activation function which restricts the output values between 0 and 1. In this way, the input and output have values in the same range and that can be helpful for the autoencoder to learn the reconstruction task. On the other hand, in the case of z-score standardization, the values of the signals are not restricted between a specific range, thus the output layer uses a linear activation function, which also does not restrict the output values in a specific range. However, the fact that the inputs and the outputs do not take values in a specific range can make the reconstruction problem more challenging, and thus the autoencoder exhibits a poorer performance.

Threshold 1 (mean+std)	Total number of samples	predicted as positive
Positive samples:	1295	49
Negative samples:	27707	214
Threshold 2 (Q3+IQR)	Total number of samples	predicted as positive
Positive samples:	1295	15
Negative samples:	27707	50

Table 5.23: Results of BiLSTM autoencoder for z-score standardization.

Threshold 1 (mean+std)	Total number of samples	predicted as positive
Positive samples:	1295	6
Negative samples:	27707	11
Threshold 2 (Q3+IQR)	Total number of samples	predicted as positive
Positive samples:	1295	9
Negative samples:	27707	27

Table 5.24: Results of BiLSTM autoencoder for MinMax normalization.

	Train ARRE	Validation ARRE	Test ARRE
MinMax:	0.0028	0.0033	0.0034
Z-score:	0.0502	0.0529	0.0534

Table 5.25: Performance of BiLSTM autoencoder.

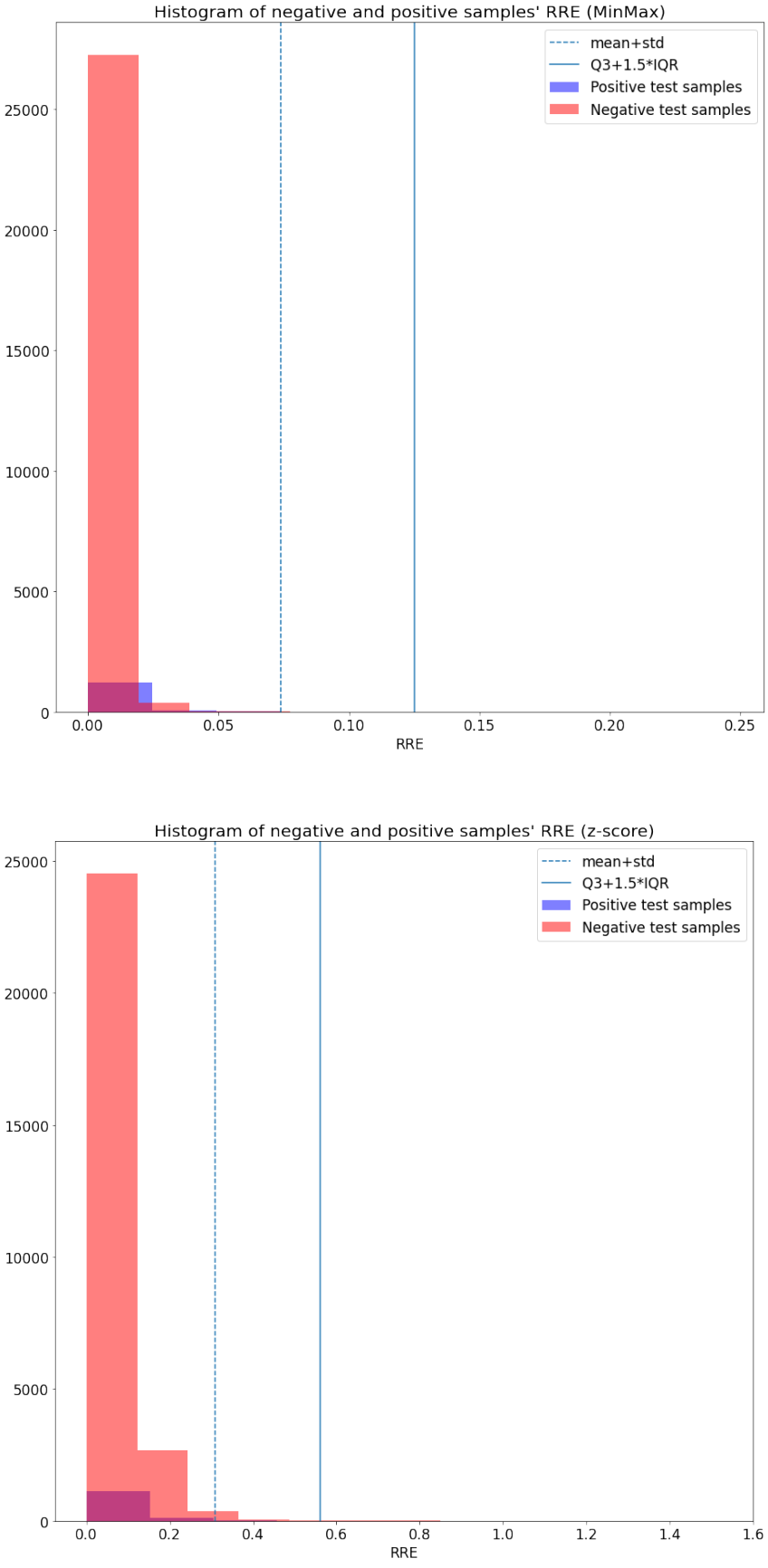


Figure 5.19: BiLSTM's permutation importance scores for class weighting and downsampling, TS1.

6
Conclusion

6.1 Discussion

6.1.1 Model performance

We were dealing with a severely imbalanced dataset, which presents a challenging problem. Due to the imbalance, one can be more flexible in defining a good performance. A model with good performance should provide predictions that are more likely to be correct than wrong for both classes. Specifically, the number of true positives/negatives should be greater than the number of false positives/negatives. This scenario typically occurs in two cases: the two classes do not overlap at all, or they overlap, but the overlap is relatively small and the sizes of the two classes are comparable. In real-life data, non-overlapping classes are uncommon.

When the dataset is severely imbalanced in the regions where the two classes overlap or the boundaries are ambiguous, the number of false positives will likely be larger than the number of true positives, even when attempting various techniques to mitigate imbalance. This results in a classifier whose predictions for the minority class cannot be trusted, as what the model classifies as the minority class probably belongs to the majority class. However, if the classifier can distinguish the minority or even a fair proportion of the minority class with a number of false positives that is small relative to the size of the majority class, it could indicate that there might be a distinguishing pattern between the two classes.

For all classifiers, despite their overall poor performance, we observed in the confusion matrices that more than half of the positive samples in the test set were classified correctly and the number of false positives is relatively small compared to the size of the majority class in the case of the class weighting approach. These instances could indicate that there might be a discernible pattern to distinguish the classes. For the other approaches, the classifiers can correctly classify less than half of the positive samples in the test set with a much smaller number of false positives. While the percentage of correctly classified positive samples is not so small that one can confidently claim that the classifier predicts almost everything as a negative class, it is also smaller than 50%, making it challenging to conclude the existence of a distinguishing pattern.

We observed that our models overfit, even in the case of logistic regression, the simplest model used. Also, we noticed that with class weighting and ensemble learning,

the models overfit less compared to oversampling and downsampling. One reason behind this could be that when the minority class is oversampled, duplicates are used, which do not add new information but rather try to balance the distribution of the two classes forcing the model to pay attention to the minority class samples. In the case of downsampling, a large portion of the majority class is removed, which can lead to losing important information, resulting in poor generalization. On the other hand, with class weighting, we do not alert the distribution of the classes; we simply use weights in the loss function to make the model pay more attention to minority class samples during the training process. Lastly, in the case of ensemble learning combined with downsampling, we mitigate the loss of information from downsampling by combining predictions from multiple models.

For the cases of DNNs, we utilized early stopping and added dropout layers to mitigate overfitting. However, none of these techniques seemed to work efficiently enough. Since DNNs are data-hungry models, more data could mitigate the overfitting.

6.1.2 Pretrained encoders

The main idea behind using frozen pretrained encoders is to extract features from the data that can reconstruct a significant portion of the initial data irrespective of the label. Even if the autoencoder performs sufficiently well, these features are not guaranteed to be good features for the classification task, since the autoencoder was trained to reconstruct the initial data. So, we do not always expect that it will yield better performance than a classifier that does not use a frozen pretrained encoder.

When we initialize the weights of the encoder with those pretrained by the autoencoder, we expect that the classifier will start training from a better initial point than with random initialization, making the classifier less vulnerable to converging to a local minimum. We actually provide extra information to the classifier, which can be more beneficial in the cases of downsampling and ensemble learning where we introduce information to the classifier from all data, even if it is trained on only a part of the majority class. However, it is important to note that when the autoencoder performs poorly, the learned weights may not be beneficial for the classifier.

For standard CNNs, the pretrained encoders in ensemble learning, downsampling, and class weighting improved the performance of the model, with initialized weights outperforming the frozen weights. However, in the case of oversampling, frozen weights outperformed the initialized ones. One reason for this could be that the learned features in the imbalanced dataset capture the distinctive patterns of the minority class more effectively, and in a more balanced dataset, this crucial information seems to get lost.

For dilated CNNs, we observe that in the case of oversampling and ensemble learning, the pretrained encoders exhibit poorer performance. This is likely because the performance of the corresponding autoencoder was very poor, and it probably could

not capture essential features from the data. In the cases of class weighting and downsampling, frozen weights seem to yield better performance on average than initialized ones. A hypothesis for this is that the features learned by the autoencoder can capture the distinctive patterns of the minority class, and by retraining them using the balanced distribution techniques, some of this crucial information seems to get lost.

In the case of RNN-based classifiers, we observed that there are cases where the model which uses frozen weights outperforms the one using initialized weights. Additionally, there are instances where the model without pretrained weights yields better results than when it uses pretrained encoder parts. In RNN-based autoencoders, the compression rate is much higher than in the case of CNNs, and they exhibit a poorer performance. However, there are cases where the pretrained RNN-based encoder improves the classifier's performance, indicating that the learned features can capture the distinctive patterns of the minority class.

Similar behavior was observed also for the case of Parallel CNN-LSTM. Namely, in the cases of downsampling, class weighting and ensemble learning the pretrained weight improved the performance of the model. However, in the case of class weighting the model which uses frozen weights outperforms the one using initialized weights. In the case of oversampling the model exhibits worse performance using the pretrained weights.

6.1.3 Feature importance

We employed feature permutation importance to identify which features significantly contribute to the model's predictions. In our case, the models yielded a poor performance based on the used metrics. However, we observed certain features with high negative feature permutation importance scores relative to model performance. Howbeit, given the overall weak predictive capability of the classifier, caution is advised in interpreting the results.

Additionally, a limitation of feature permutation importance is that if there are correlated features, the importance of the associated feature may be reduced and distributed among these correlated features. As mentioned earlier, we utilized feature correlation analysis to identify highly linearly correlated features. However, since there is no standardized methodology for computing correlations among mixed types of features, readers need to consider this when interpreting our findings.

6.2 Conclusion

As discussed above, none of the four approaches could provide clear evidence of a pattern behind the drivers' decision to deactivate the FSRA. Only in the cases of class weighting were the models able to correctly classify more than half of the positive samples in the test set. However, this resulted in a high number of false

positives relative to the size of the positive class. Given the poor performance of the model, it is unclear if the observed results are due to an actual pattern or the methods used to address the imbalance between the two classes. Additionally, in the case of the autoencoder, we observed that the distribution of positive and negative samples is similar. Moreover, in Figures 3.8 and 3.9, after converting the features to scalars, we used t-SNE to visualize them to 2D. We saw that the two classes overlap. which indicates that given the features the classes may not be separable.

Utilizing feature permutation importance, we observed that the most important features are AEB, FSRA, LKS, and FDA Status and # FDA deactivation across all methods, with AEB dominating in most cases. Region, speed, acceleration pedal position, and road type exhibit moderate importance for the majority of the models. Apart from, # LKS warnings and # FDA deactivations, no other feature related to the behavior of the driver before the observed two-minute slot seems to affect the performance of the model. Notably, the number of FRSA warnings and information alerts before the observed two-minute window seems does not seem to affect the model's performance. Additionally, weather conditions do not appear to be important for the model. As highlighted in the discussion section, caution must be taken when interpreting these results due to the weak predictive power of the model.

In Chapter 3, we observed that approximately 25% of the deactivations could not have been performed by the driver, as they occurred in the first 3 seconds of the file, or the time gap between reactivation of the function and the next deactivation was smaller than 3 seconds. Furthermore, 27% of the deactivations occurred between the first 3 seconds to 1 minute. For the deactivations that occurred after the first 3 seconds of the file, more than 50% had a duration smaller than 10 minutes. Moreover, only 24% of them did the driver not reactivate the FSRA until the end of the file. Additionally, 7% of them occurred when the truck was moving at high speed. We expected that since the driver decided to deactivate the function, they would not reactivate it. Apart from the reactivation, it also seems odd to deactivate the function for such a short time period and to decide to deactivate it under circumstances where the function would not yield an alert. However, we could not confidently conclude that these instances are due to a logger error and not the actions of the driver, unlike the 25% we already excluded from our dataset. Given these observations and the weak performance of the models, it is possible that a significant part of the dataset may not accurately represent the drivers' actions but instead could be attributed to a logger error. Under this hypothesis, it is expected for the models to exhibit a weak performance since a significant part of positive samples is unrelated to the drivers' behavior.

Another hypothesis for the weak performance of the models could be the severely imbalanced dataset. More positive samples could provide a better representation of the minority class, enabling the models to find the discernible patterns between the two classes.

As mentioned in the limitation section, we assumed that we could infer the behavior

of the driver using data collected by the sensors. These features provide information about the driving behavior, such as whether the driver over-speeds or if there are steep changes in speed before deactivation. They provide information on the surrounding environment of the truck, such as the presence of pedestrians in close range while the truck is moving at low speed or other vehicles. However, we do not have a clear image of the driver's emotional state and the surrounding environment. Since the models perform weakly, they do need extra features beyond the ones currently provided.

Finally, it is possible, for each driver to exhibit their own behavior and preference for the function, and the patterns behind the deactivations are distinct among different drivers. If this is the case, then we should train a model for each driver. However, we do not have enough data on individual drivers to test this hypothesis.

6.3 Future work

The future work could follow multiple directions. A self-attention model could be utilized with the existing features and dataset. In many cases, self-attention models have managed to outperform models without self-attention layers. Also, the existing model could be retrained on a larger dataset that contains more minority class samples.

Another direction could involve collecting more data from sensors, like a one-minute video capturing 30 seconds before and 30 seconds after the deactivation. This could help identify common conditions under which drivers decide to deactivate the function. Furthermore, direct communication with drivers could be beneficial, gathering information about their personal experience with FSRA and their motivation behind the deactivation of the system. If the number of drivers is relatively large, a Large Language Model approach could be utilized to process the responses and find patterns. Based on the findings, the research could follow new directions.

Finally, it would be meaningful to investigate whether some deactivations could be an error of the logger. In this case, a more clean dataset could be available for the training of the model.

Bibliography

[1] World Health Organization. (2019). 'Road traffic injuries'.
URL: https://www.who.int/health-topics/road-safety#tab=tab_1

[2] 'Front Short Range Assist'. *Volvo Trucks Driver Guide.*
URL: https://driverguide.volvotrucks.com/lang/en/chassi/Z0000FE/topic/319388/.

[3] Zheng, Yi and Liu, Qi and Chen, Enhong and Ge, Yong and Zhao, J. 'Exploiting multi-channels deep convolutional neural networks for multivariate time series classification', *Frontiers of Computer Science*, 8485, 09 2015. doi: 10.1007/s11704-015-4478-2

[4] F. Karim, S. Majumdar, H. Darabi and S. Chen. 'LSTM Fully Convolutional Networks for Time Series Classification', *IEEE Access*, 6:1662-1669, 2018, doi: 10.1109/ACCESS.2017.2779939.

[5] Fazle Karim, Somshubra Majumdar, Houshang Darabi, Samuel Harford. 'Multivariate LSTM-FCNs for time series classification', *Neural Networks*, 116:237-245, 2019, ISSN 0893-6080,
URL: https://doi.org/10.1016/j.neunet.2019.04.014.

[6] Rahman, Md Mushfiqur and Farahani, Mojtaba Askarzadeh and Wuest, Thorsten. 'Multivariate Time-Series Classification of Critical Events from Industrial Drying Hopper Operations: A Deep Learning Approach', *Journal of Manufacturing and Materials Processing*, 7(5):164, 2023, ISSN 2504-4494, doi: 10.3390/jmmp7050164.
URL: https://www.mdpi.com/2504-4494/7/5/164

[7] Japkowicz, Nathalie and Myers, Catherine and Gluck, Mark. 'A Novelty Detection Approach to Classification', *Proceedings of the Fourteenth Joint Conference on Artificial Intelligence*, (1999).

[8] Shivam Bansal, **URL:** https://www.kaggle.com/code/shivamb/3d-convolutions-understanding-use-case/notebook

[9] Mustaqeem, Soonil Kwon. 'MLT-DNet: Speech emotion recognition using 1D dilated CNN based on multi-learning trick approach', *Expert Systems with Applications*,167, 2021, 114177, ISSN 0957-4174,
URL: https://doi.org/10.1016/j.eswa.2020.114177.

[10] R. M. Schmidt, 'Recurrent neural networks (rnns): A gentle introduction and overview',
URL: https://arxiv.org/abs/ 1912.05911.

[11] Hochreiter, S. and Schmidhuber, J.. 'Long short-term memory', *Neural Computation*, 9(8): 1735–1780, 1997.

[12] Christopher Olah. ‘Understanding LSTM Networks’, 2015,
URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[13] Ellis Stewart. ‘What is a Gated Recurrent Unit (GRU) and How Does it Work?’. 2024,
URL: https://em360tech.com/tech-article/gated-recurrent-unit-gru

[14] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. ‘On the properties of neural machine translation: Encoder-decoder approaches’. *arXiv preprint arXiv*:1409.1259, 2014.

[15] M. Schuster and K. K. Paliwal. ‘Bidirectional recurrent neural networks’, *IEEE Transactions on Signal Processing*, 45(11): 2673-2681, Nov. 1997, doi: 10.1109/78.650093.

[16] Michelucci, Umberto. ‘An Introduction to Autoencoders’, 2022,
URL: https://www.researchgate.net/publication/357766280_An_Introduction_to_Autoencoders

[17] Authors: Cord Dankers, Veronika Kronseder, Moritz Wagner Supervisor: Giuseppe Casalicchio.‘Chapter 8 Introduction to Feature Importance’,
URL: https://slds-lmu.github.io/iml_methods_limitations/pfi.htmlpermutation-feature-importance-pfi

[18] van der Maaten, Laurens, and Geoffrey Hinton. ‘Visualizing data using t-SNE.’, Journal of Machine Learning Research 9, no. Nov (2008): 2579-2605.

A

Appendix 1

A.1 Results

A.1.1 Logistic Regression

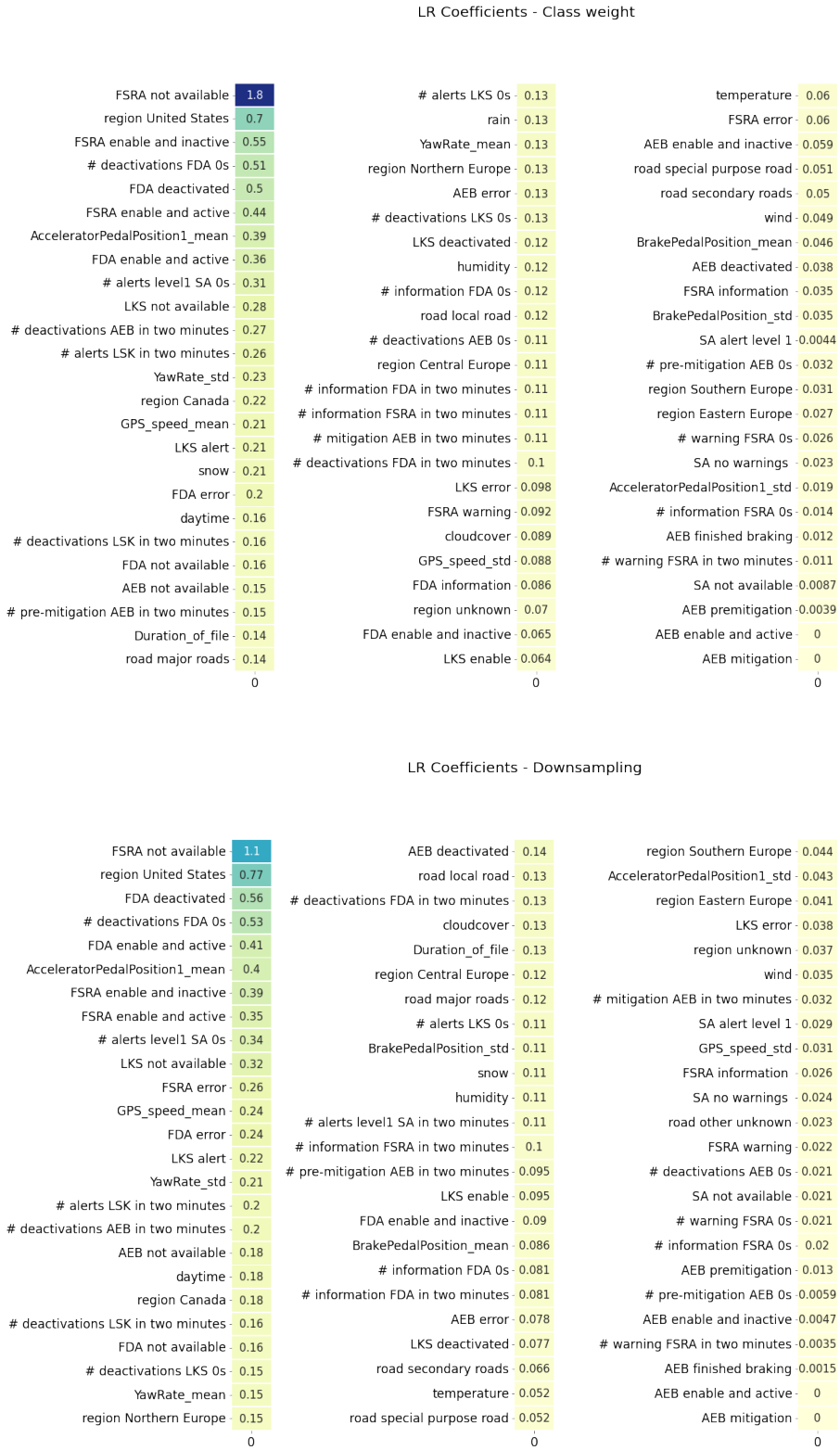


Figure A.1: Logistic Regression coefficients for all approaches

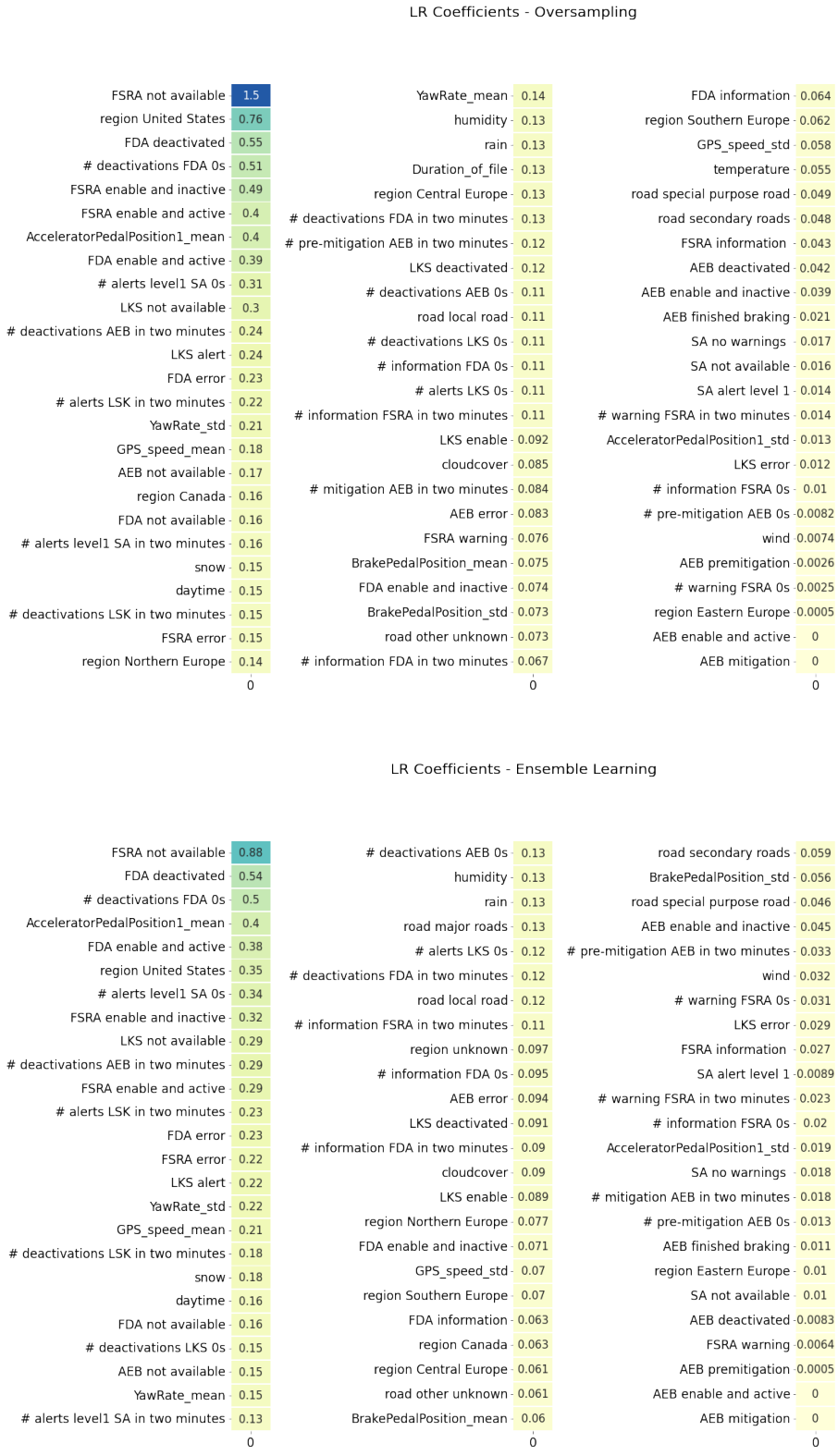


Figure A.2: Logistic Regression coefficients for all approaches

A.1.2 Trivial Compression

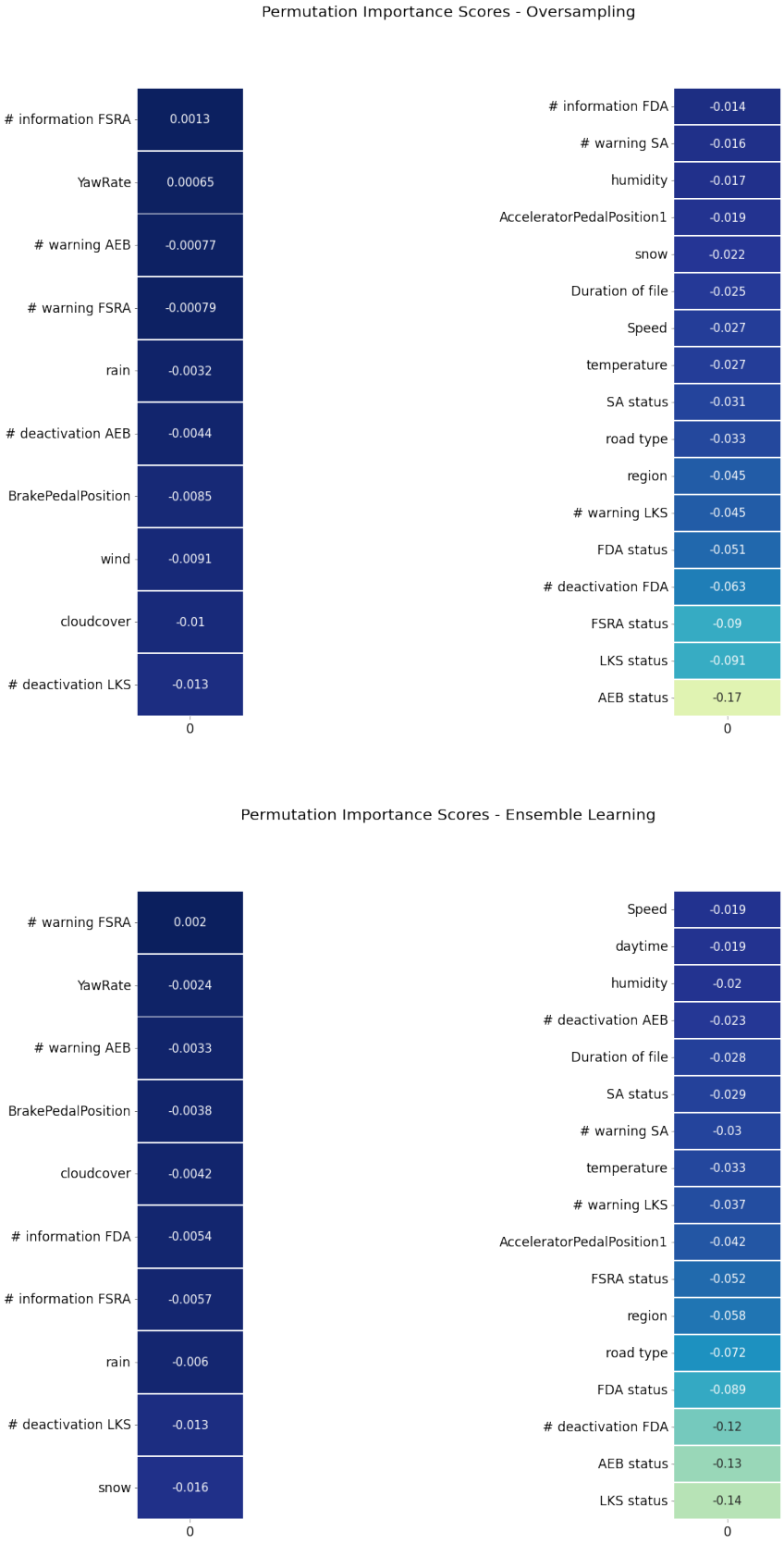


Figure A.3: Trivial compression's permutation importance scores for oversampling and ensemble learning.

A.1.3 Standard CNNs

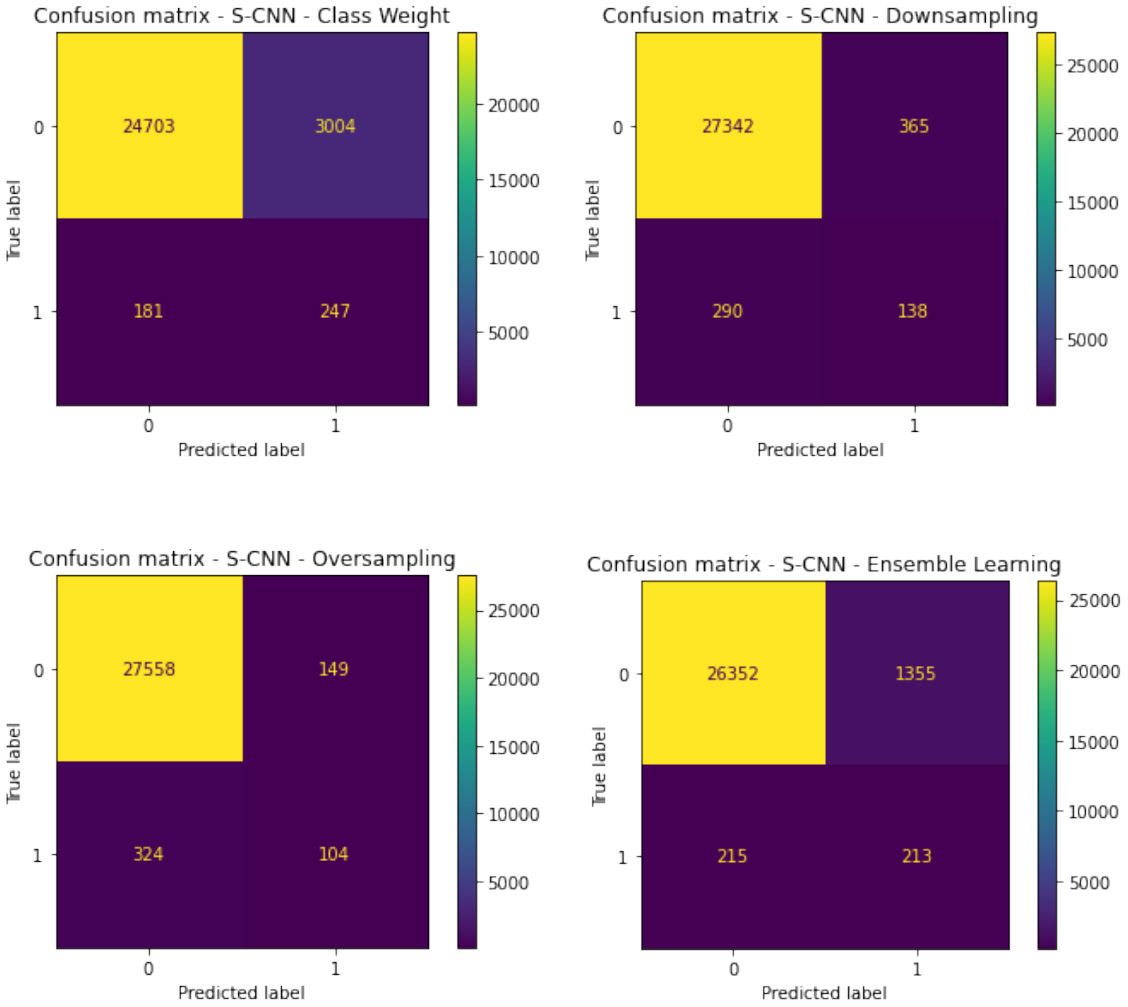


Figure A.4: Confusion matrices of the test set for the different approaches, TS2.

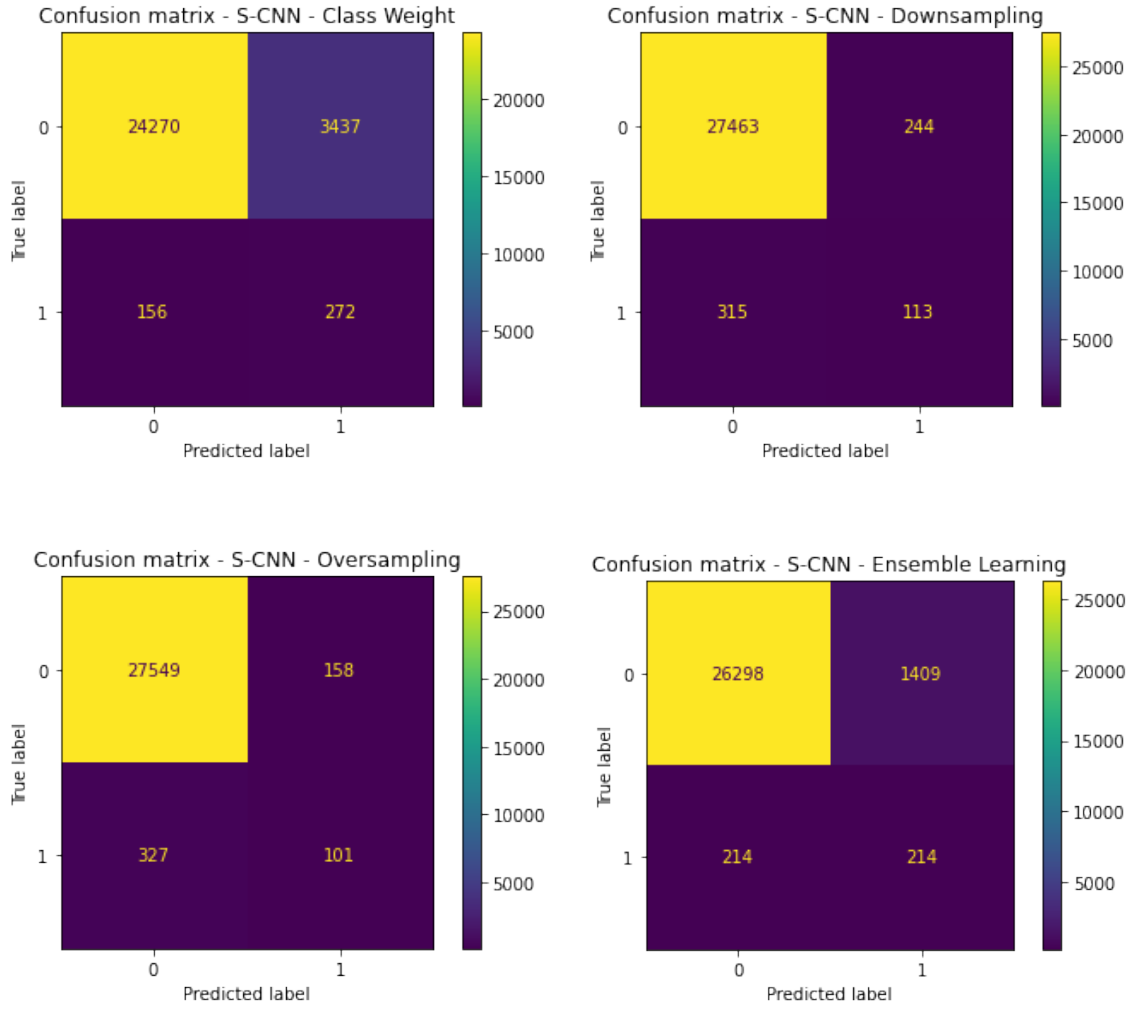


Figure A.5: Confusion matrices of the test set for the different approaches, TS3.

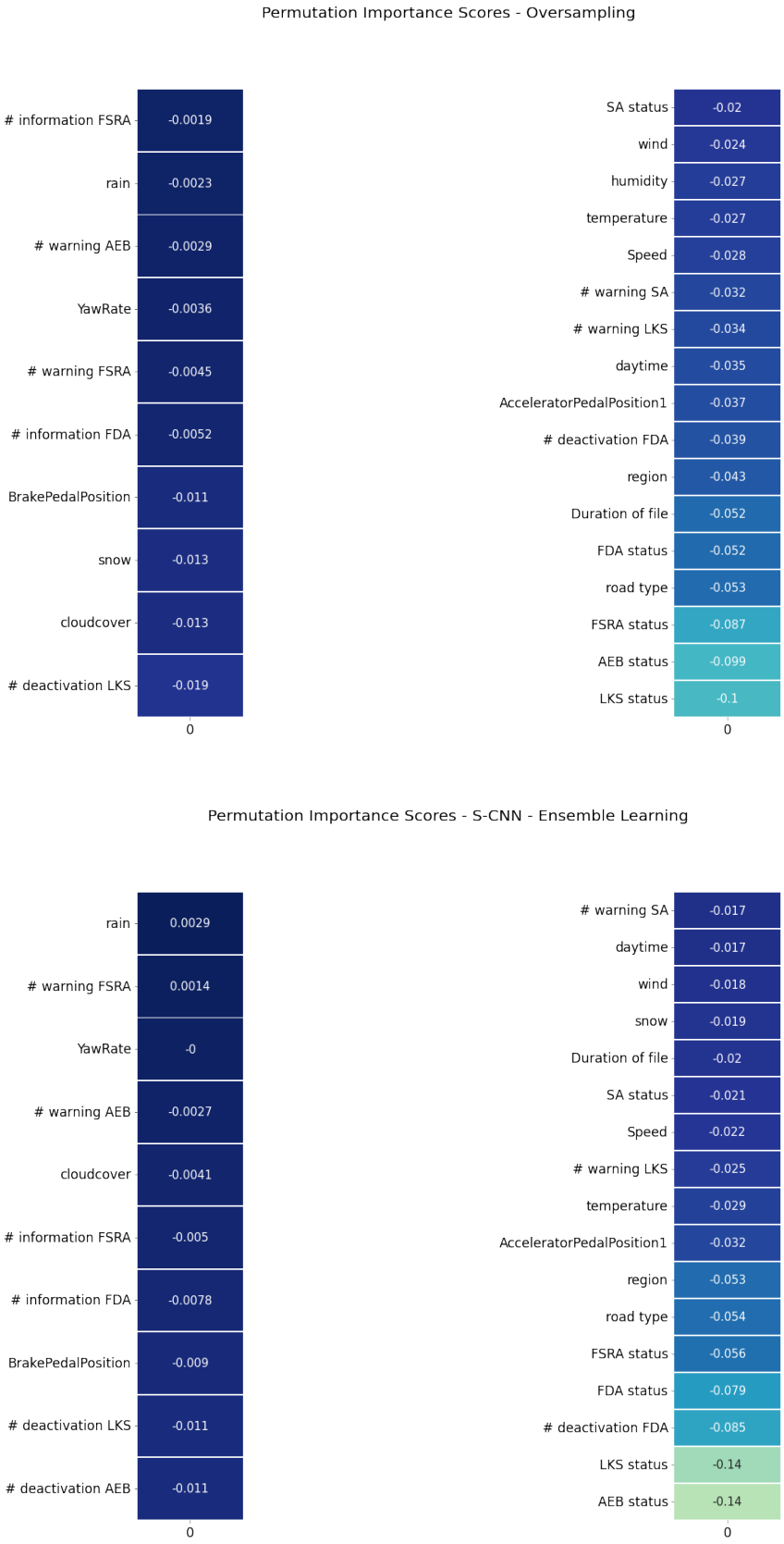


Figure A.6: Standard CNN's permutation importance scores for oversampling and ensemble learning, TS1.

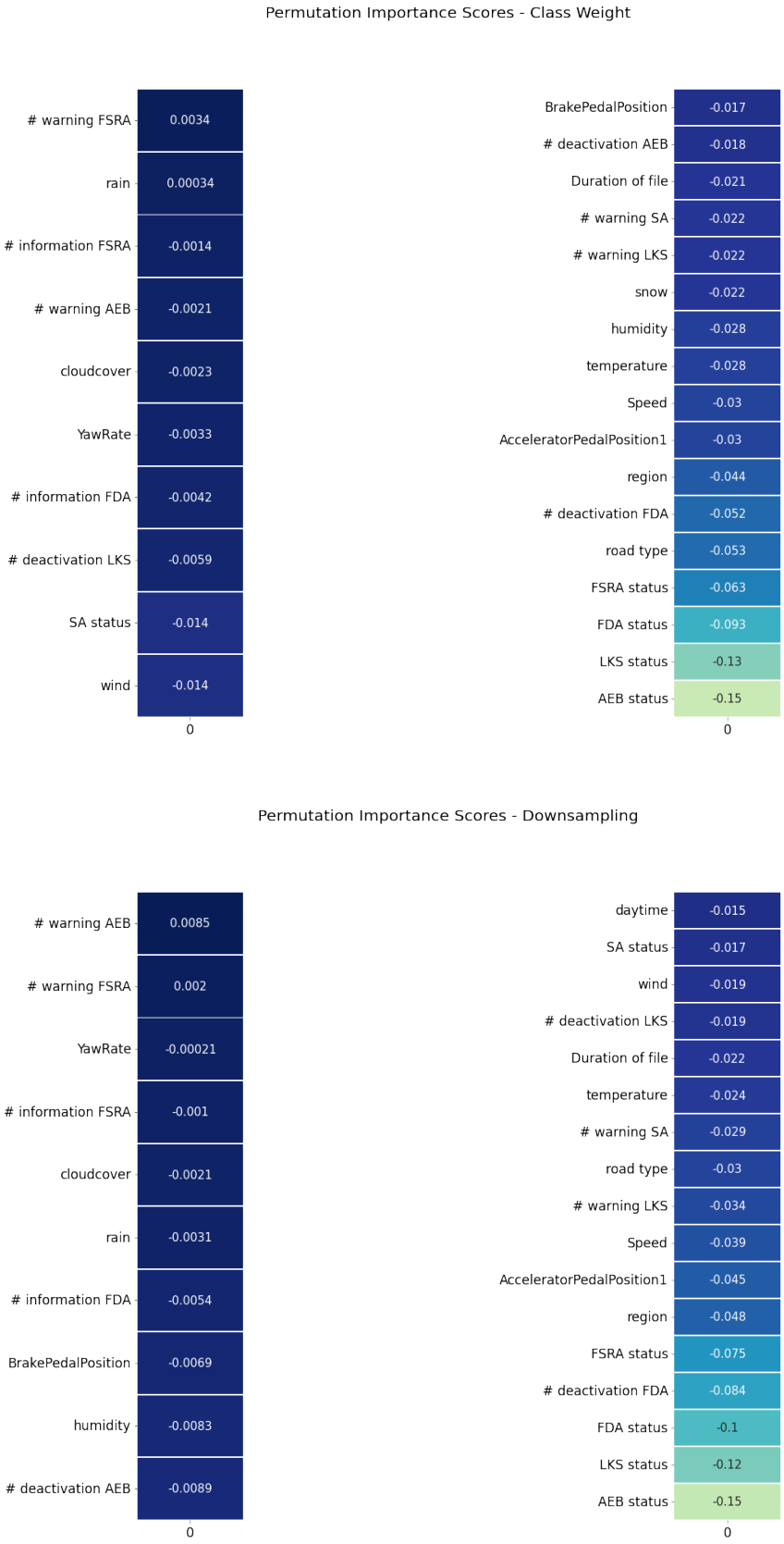


Figure A.7: Standard CNN's permutation importance scores for class weighting and downsampling, TS2.

X

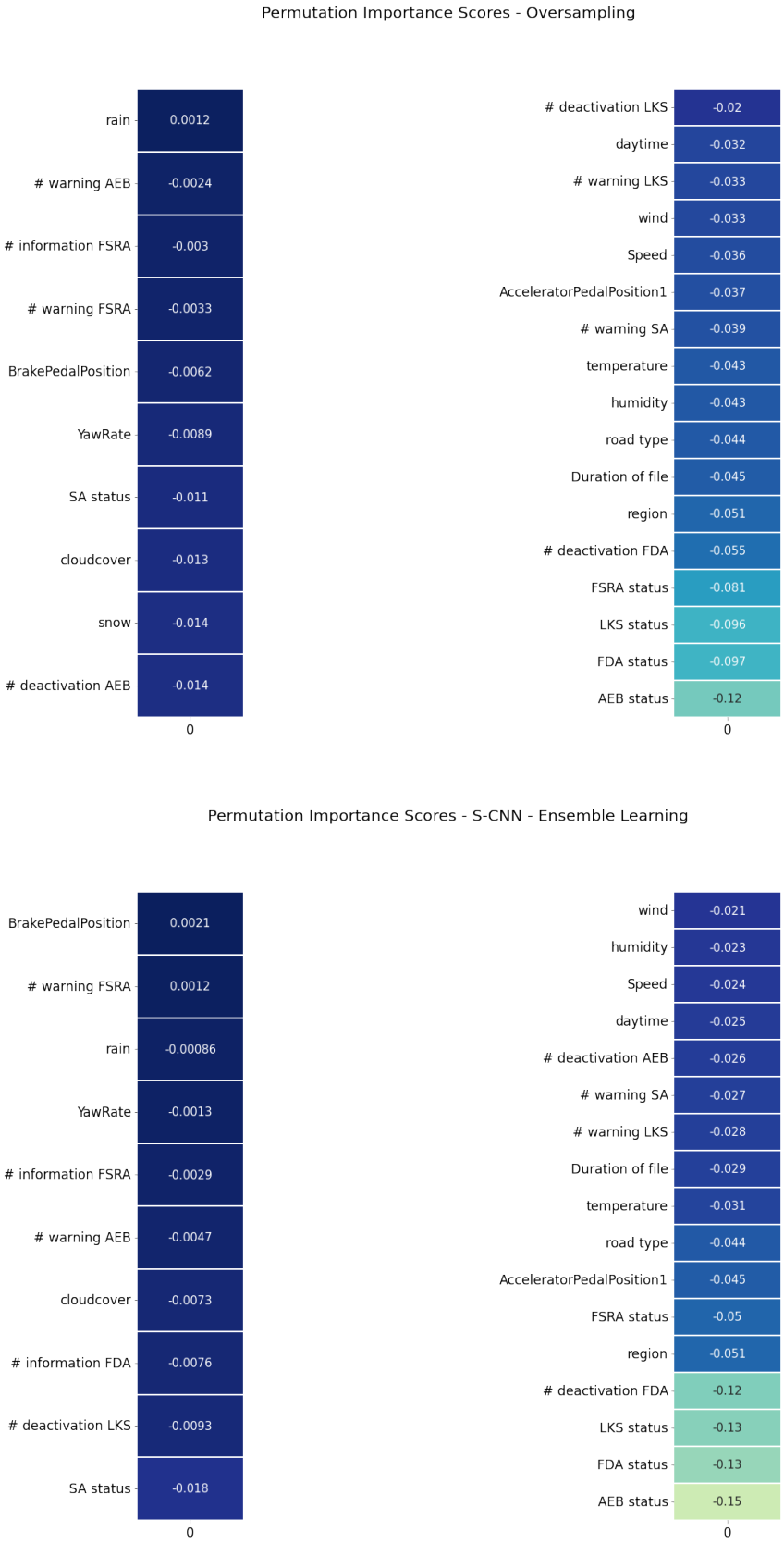


Figure A.8: Standard CNN's permutation importance scores for oversampling and ensemble learning, TS2.

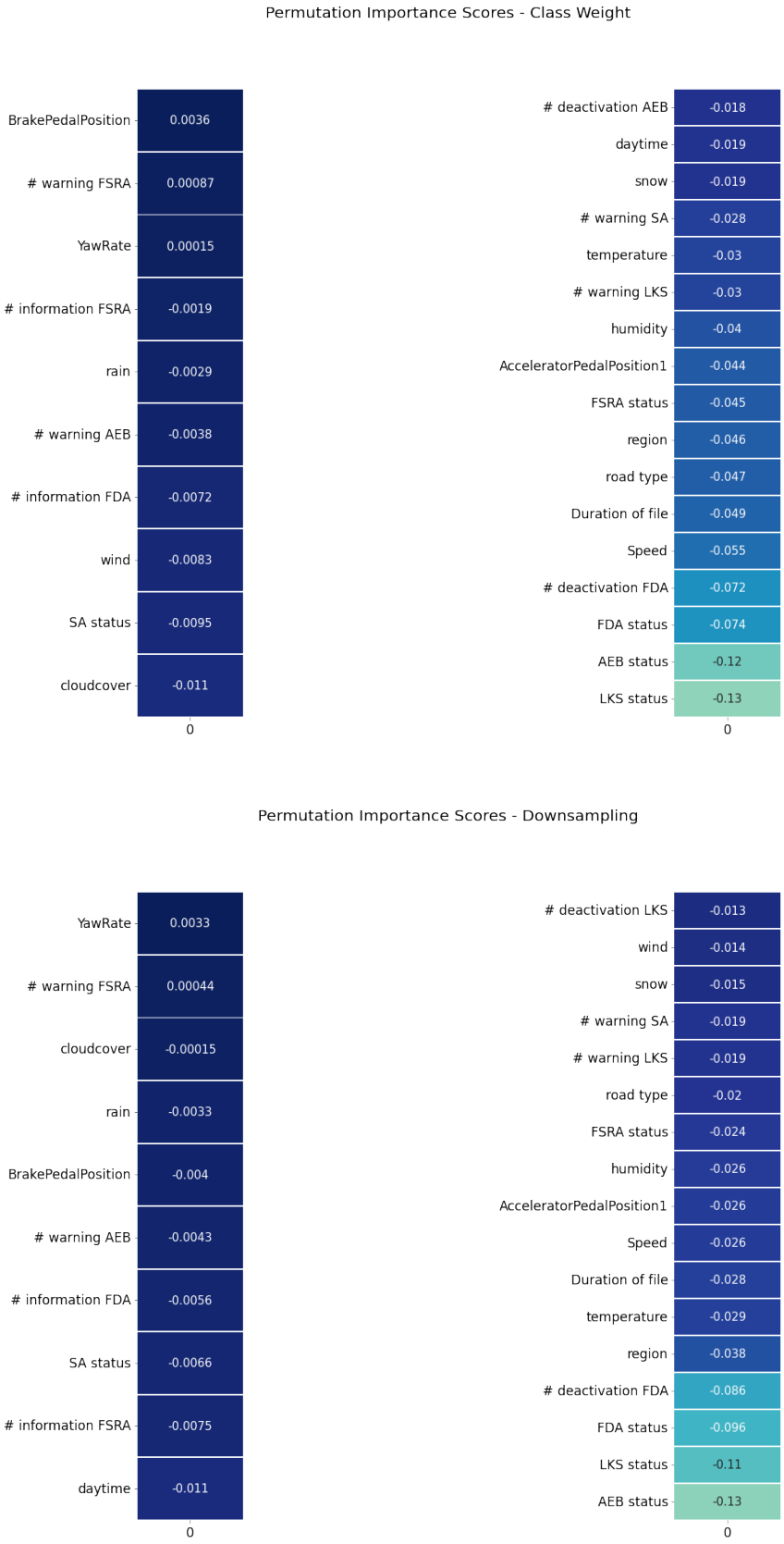


Figure A.9: Standard CNN's permutation importance scores for class weighting and downsampling, TS3.

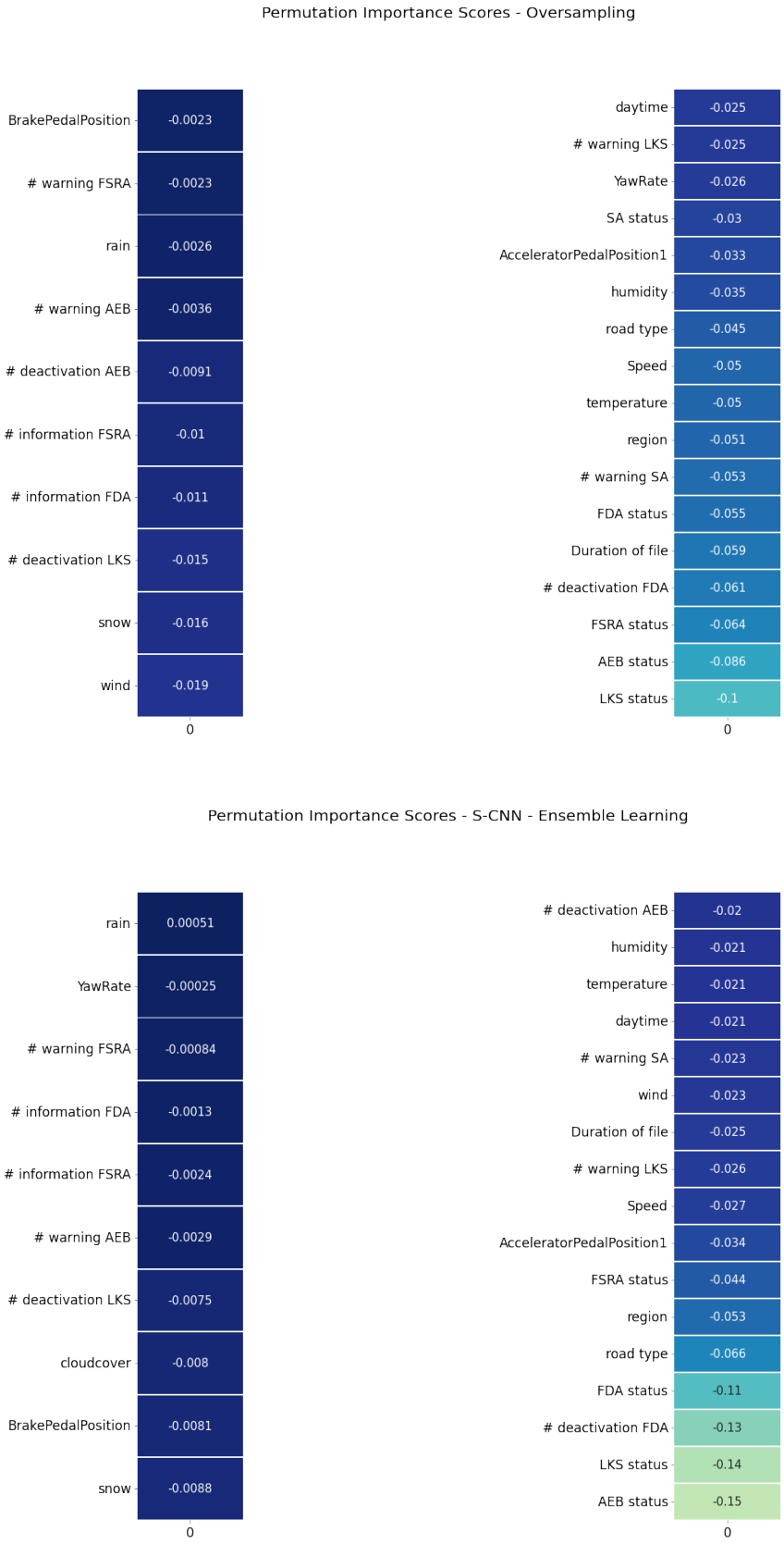


Figure A.10: Standard CNN's permutation importance scores for oversampling and ensemble learning, TS3.

A.1.4 Dilated CNNs

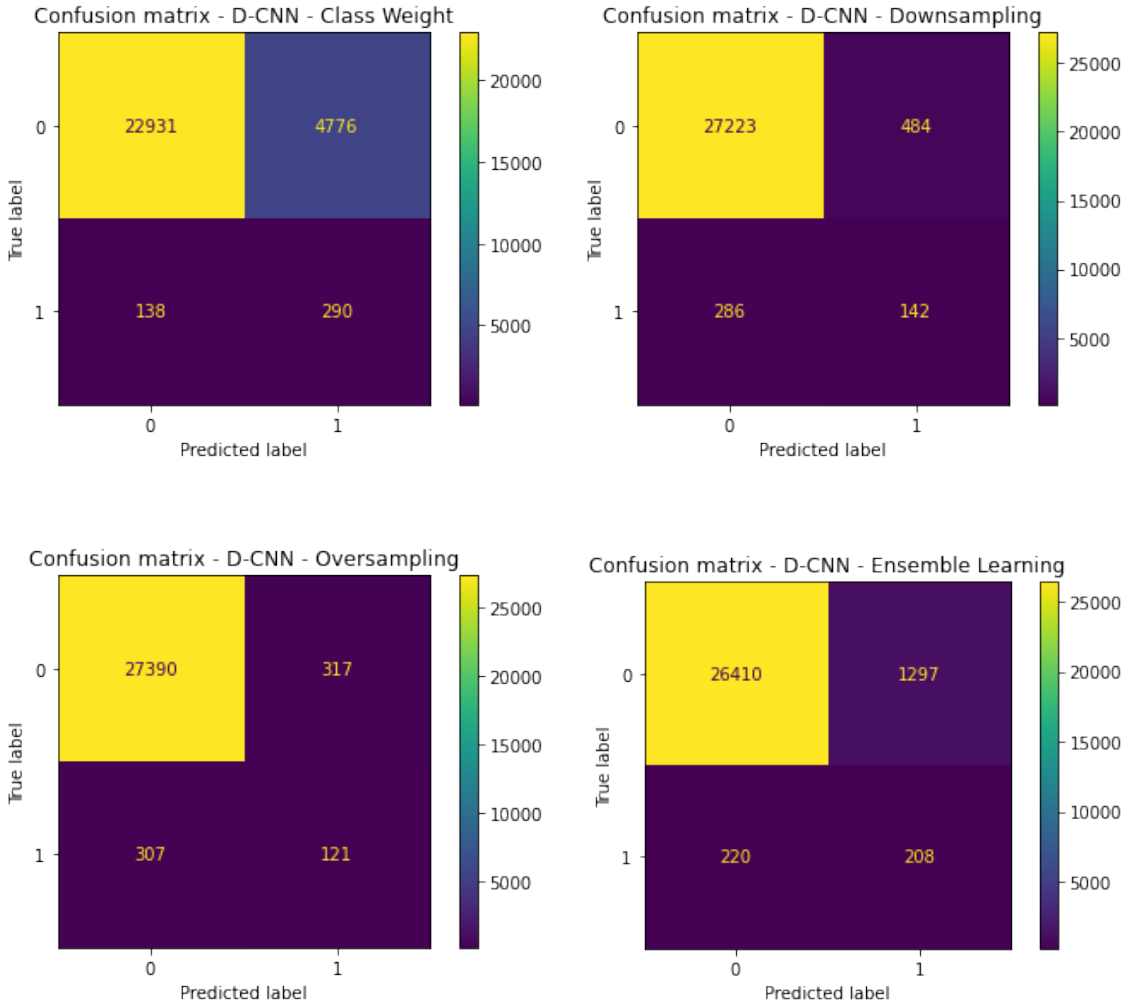


Figure A.11: Confusion matrices of the test set for the different approaches, TS2.

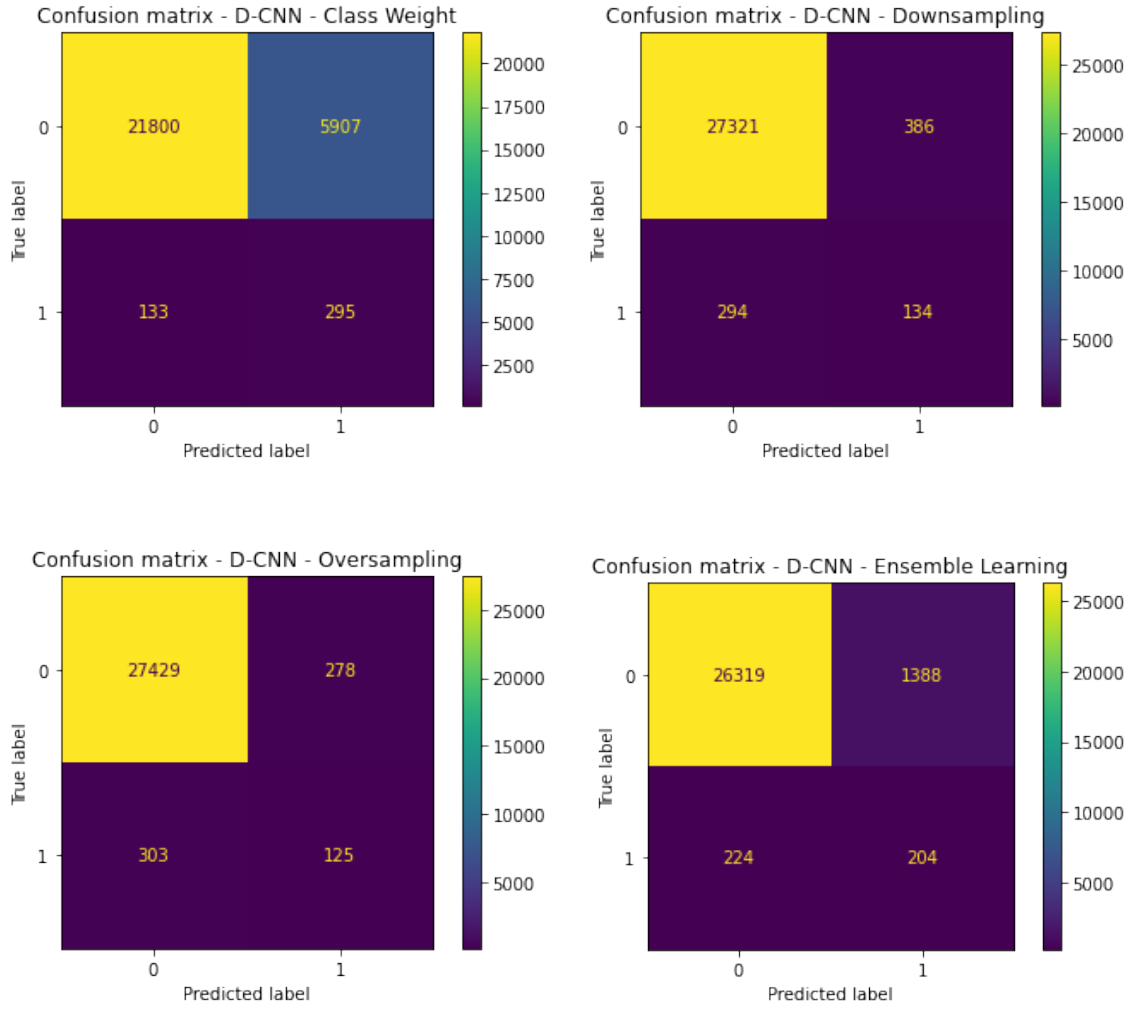


Figure A.12: Confusion matrices of the test set for the different approaches, TS3.

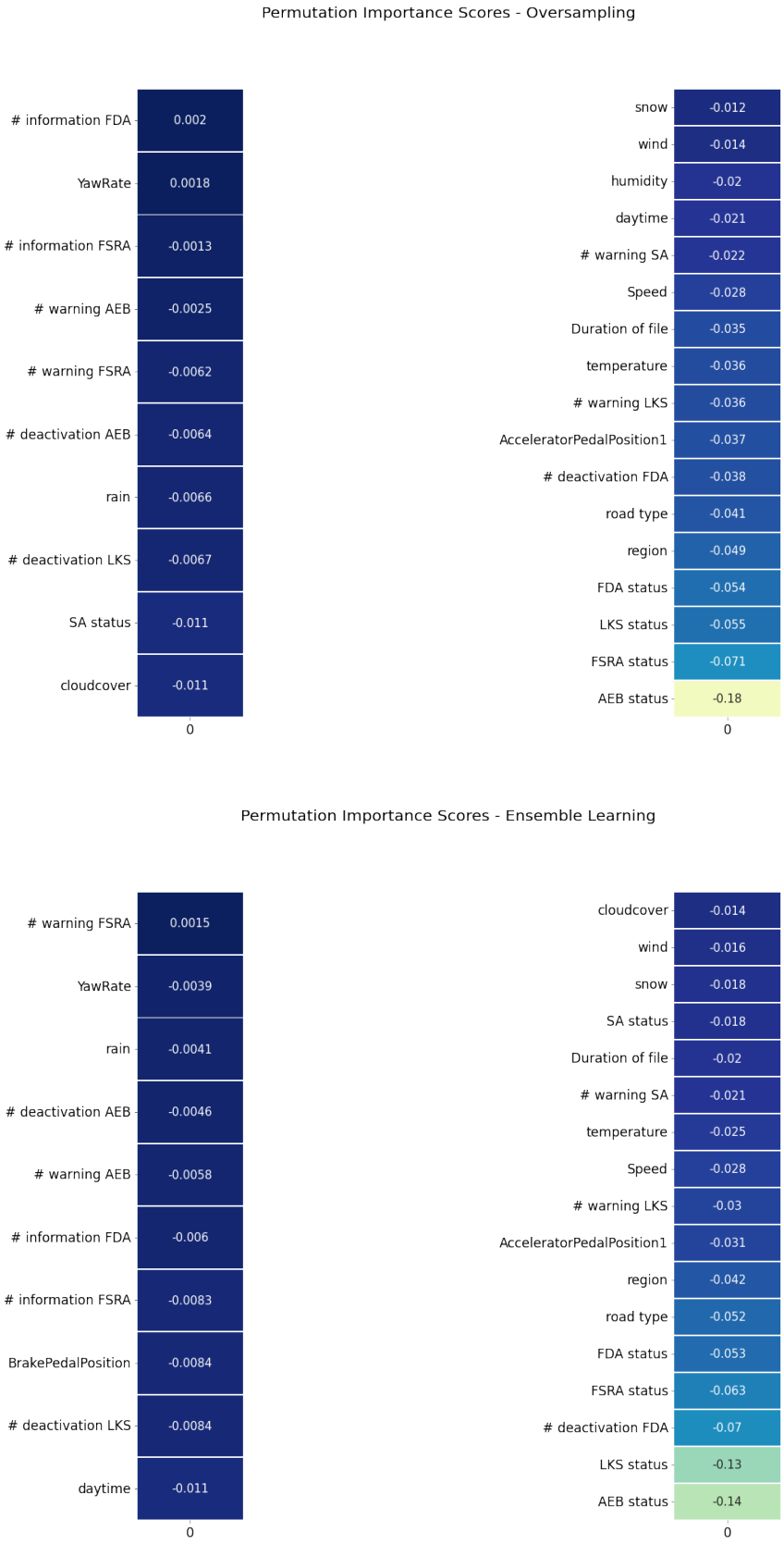


Figure A.13: Dilated CNN's permutation importance scores for oversampling and ensemble learning, TS1.

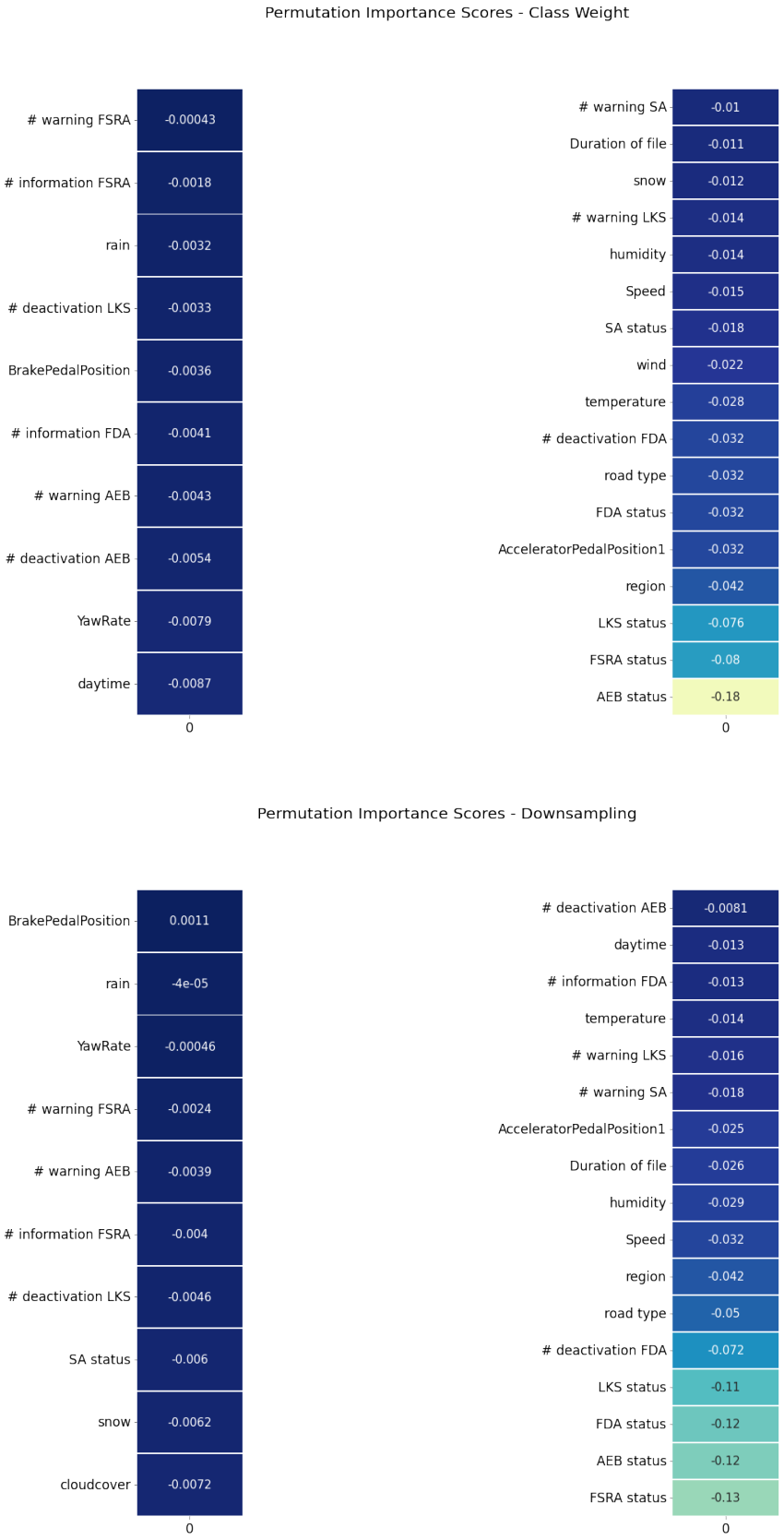


Figure A.14: Dilated CNN's permutation importance scores for class weighting and downsampling, TS2.

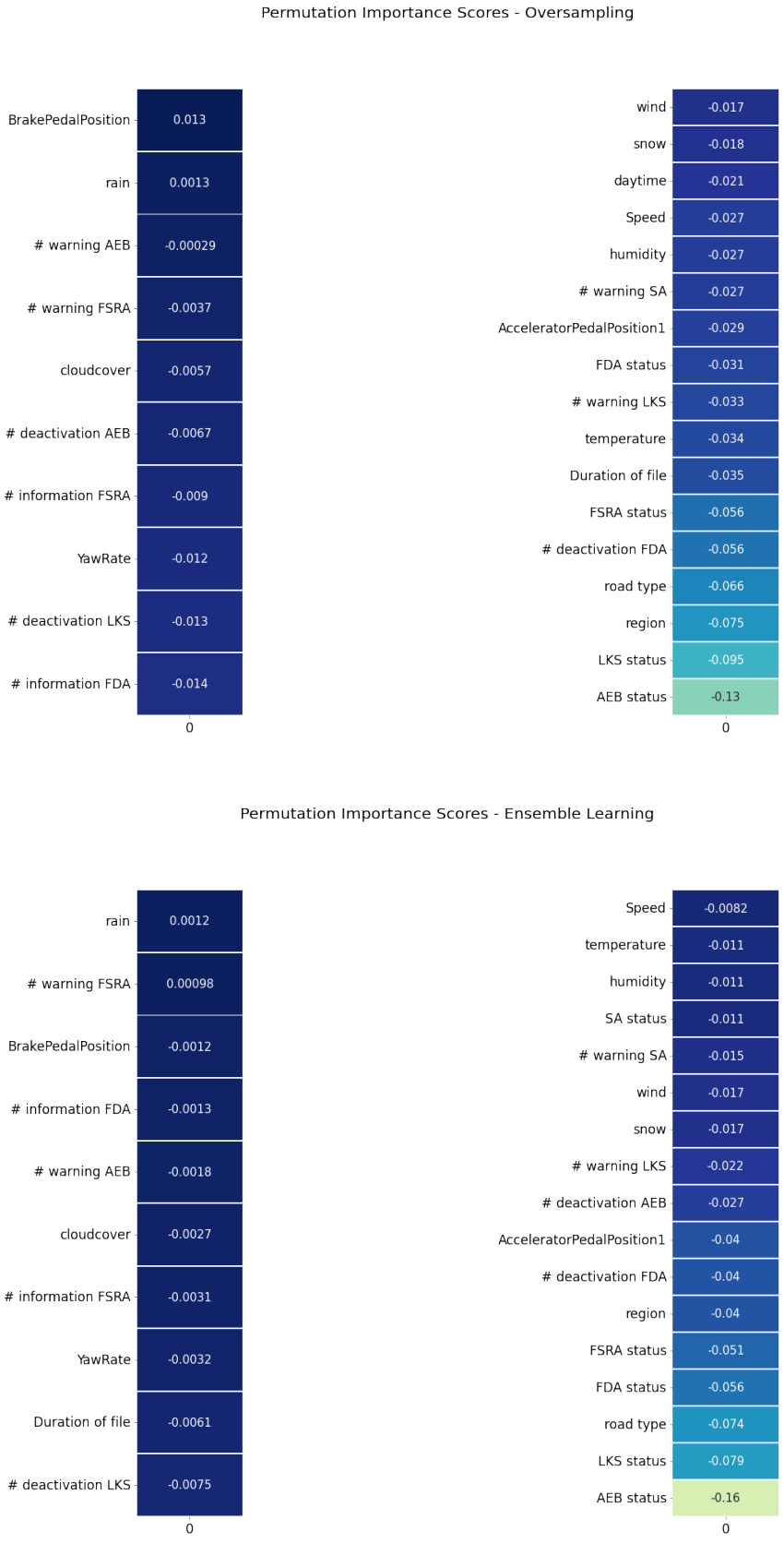


Figure A.15: Dilated CNN's permutation importance scores for oversampling and ensemble learning, TS2.

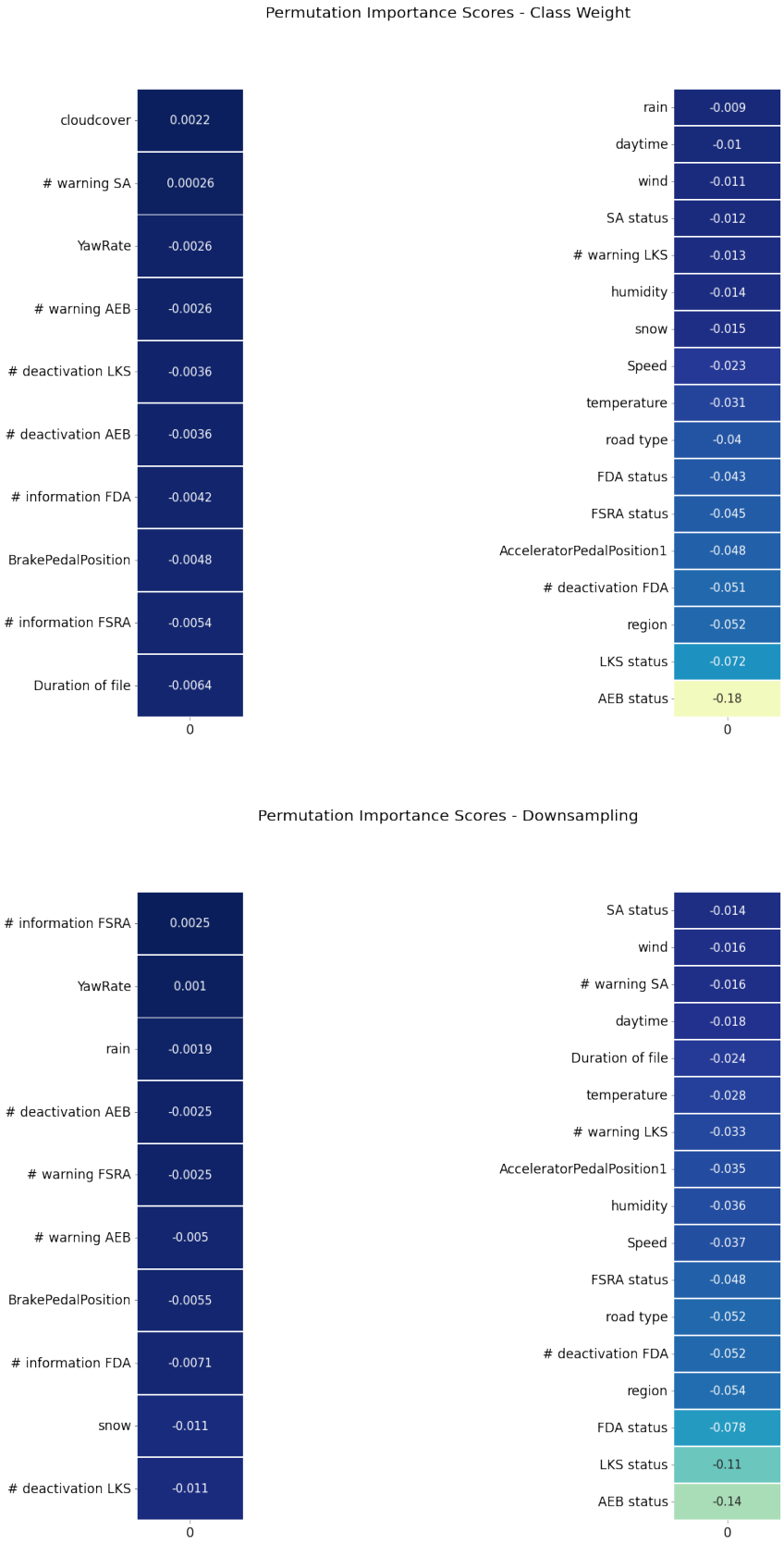


Figure A.16: Dilated CNN's permutation importance scores for class weighting and downsampling, TS3.

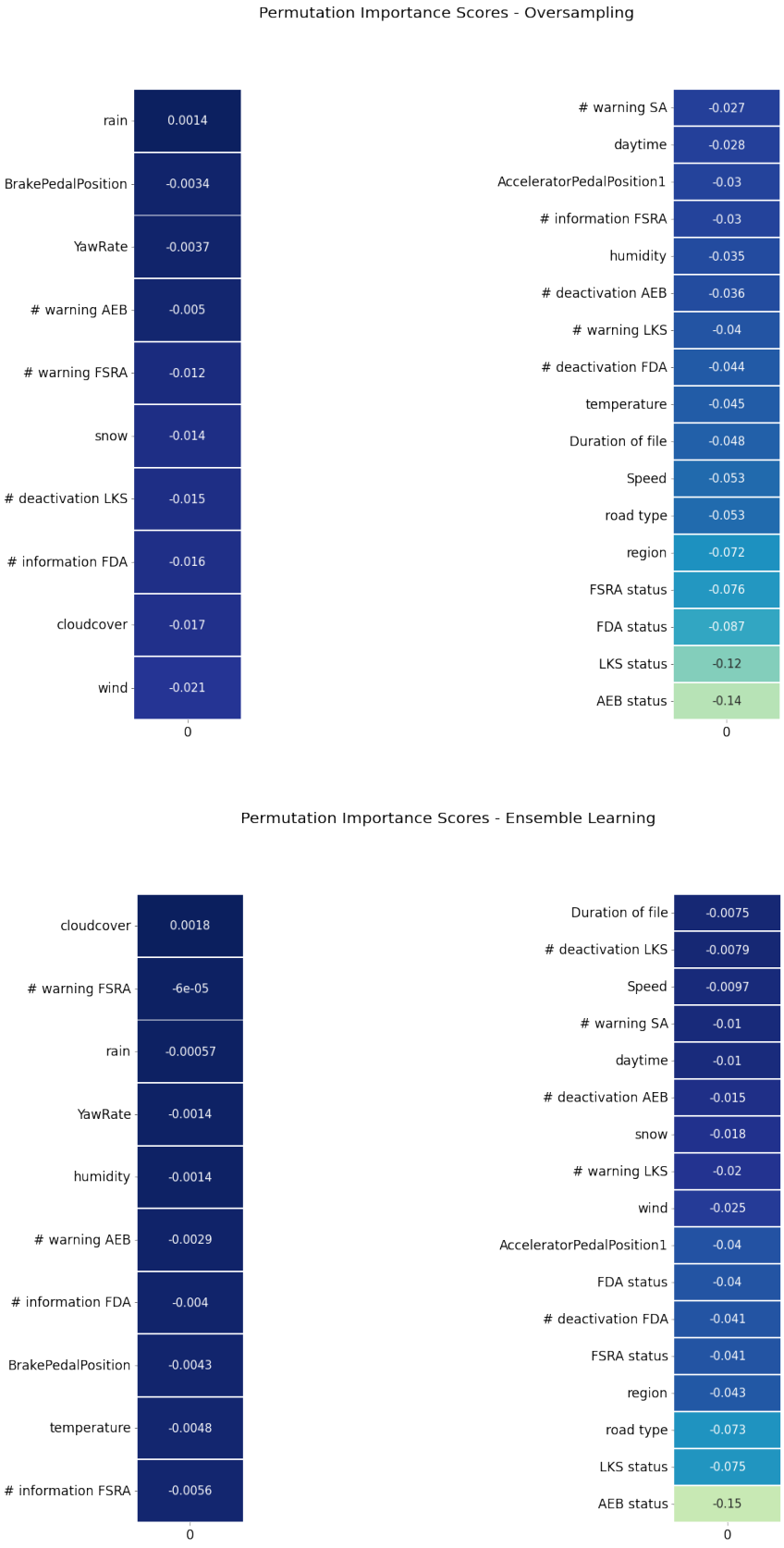


Figure A.17: Dilated CNN's permutation importance scores for oversampling and ensemble learning, TS3.

A.1.5 LSTM

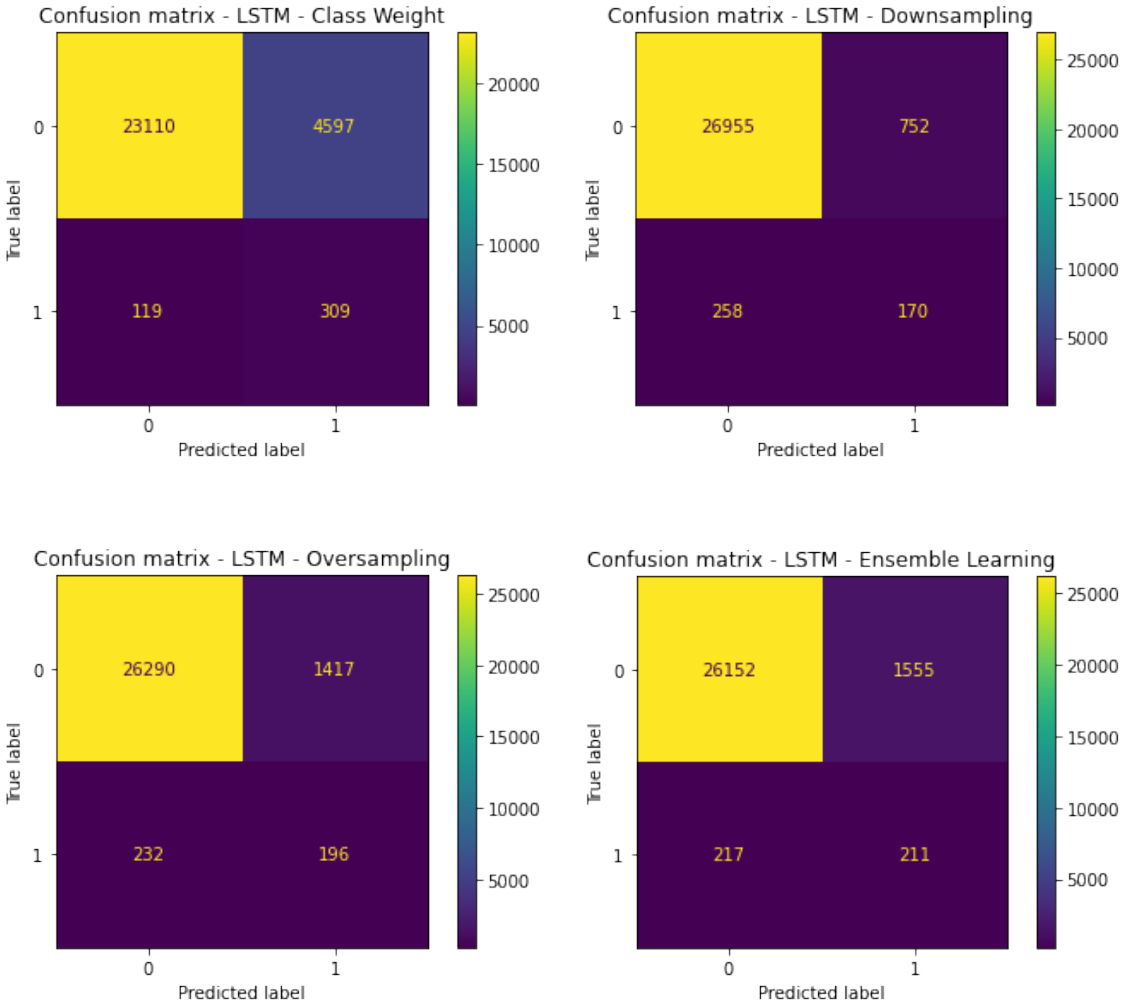


Figure A.18: Confusion matrices of the test set for the different approaches, TS2.

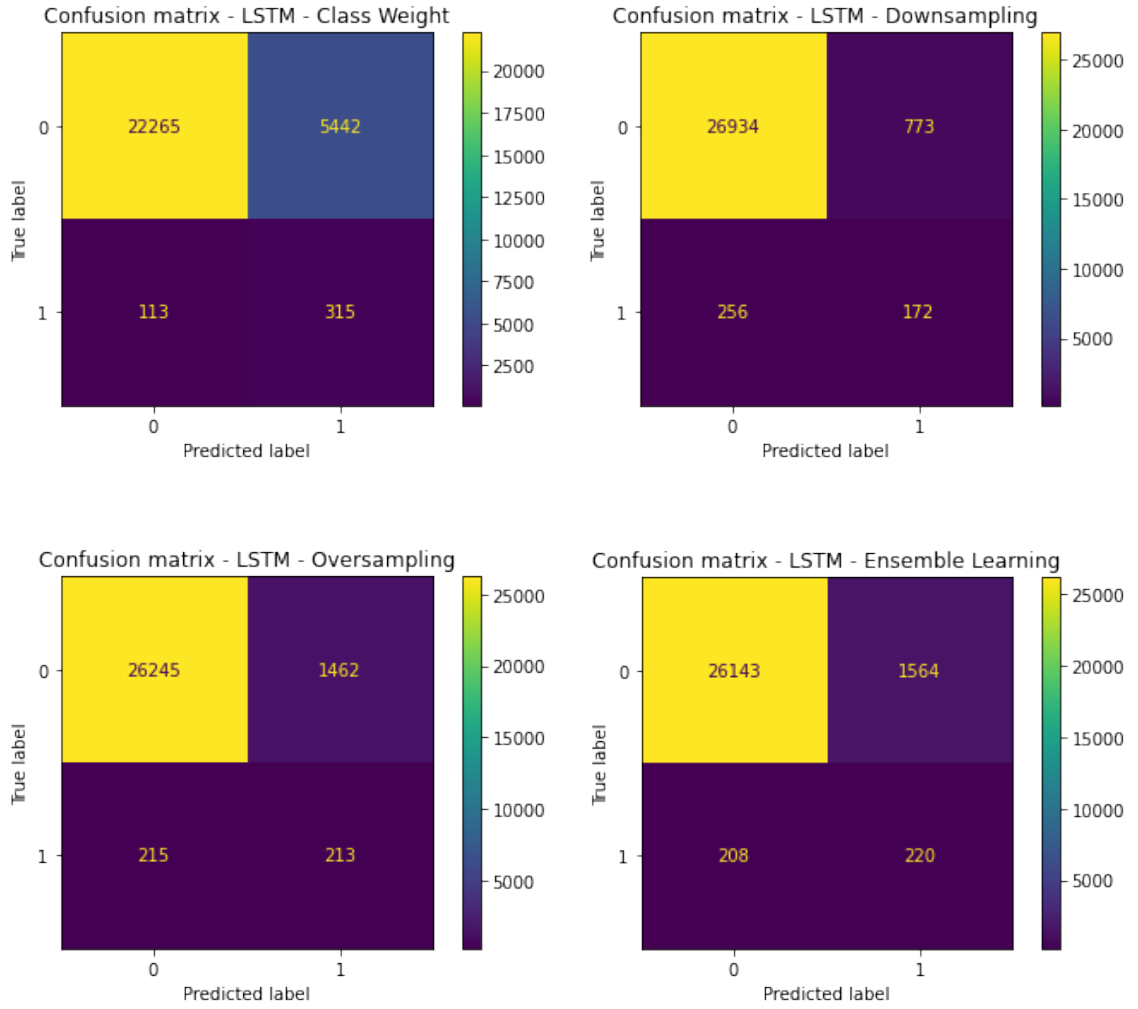


Figure A.19: Confusion matrices of the test set for the different approaches, TS3.

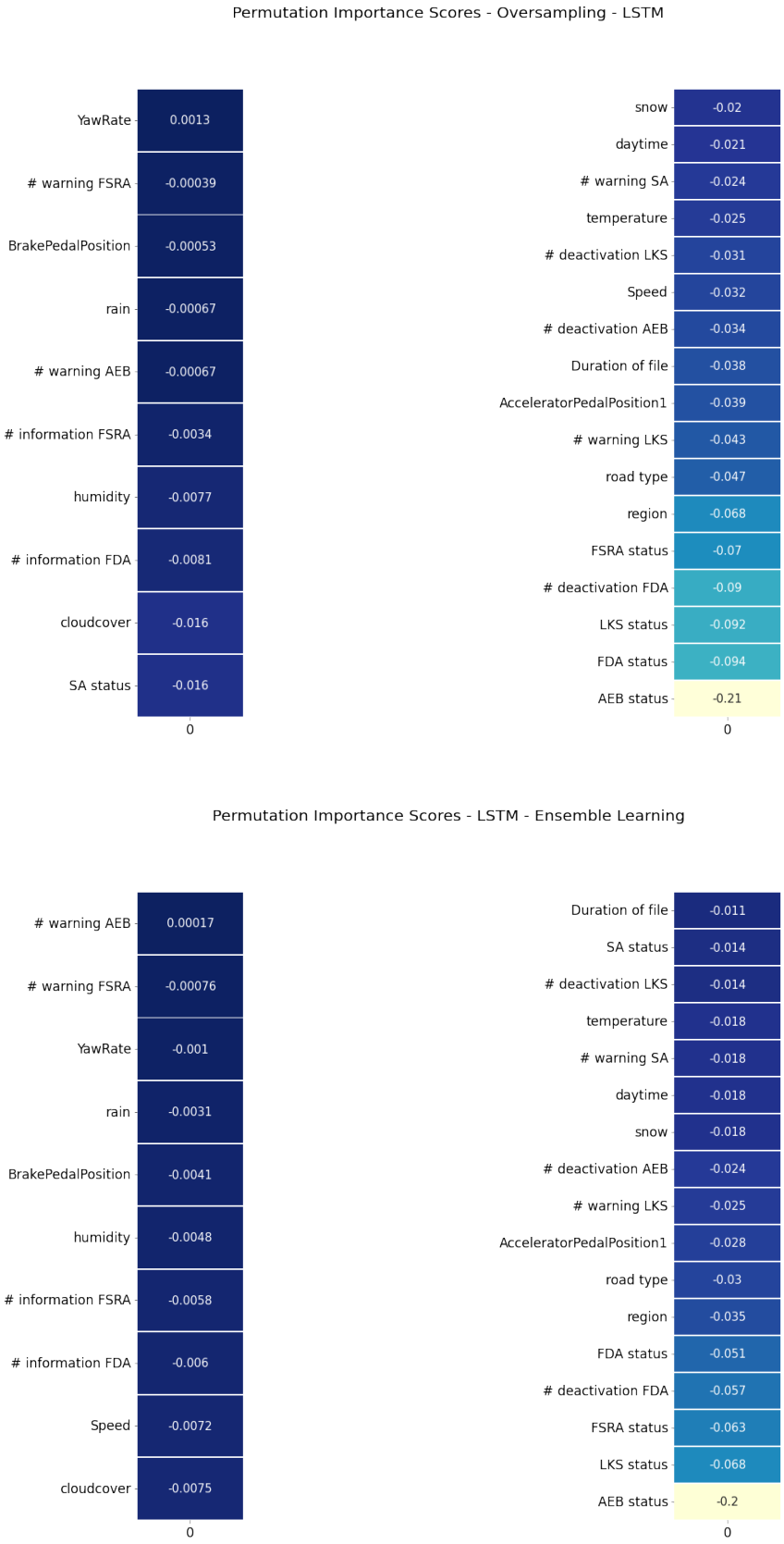


Figure A.20: LSTM's permutation importance scores for oversampling and ensemble learning, TS1.

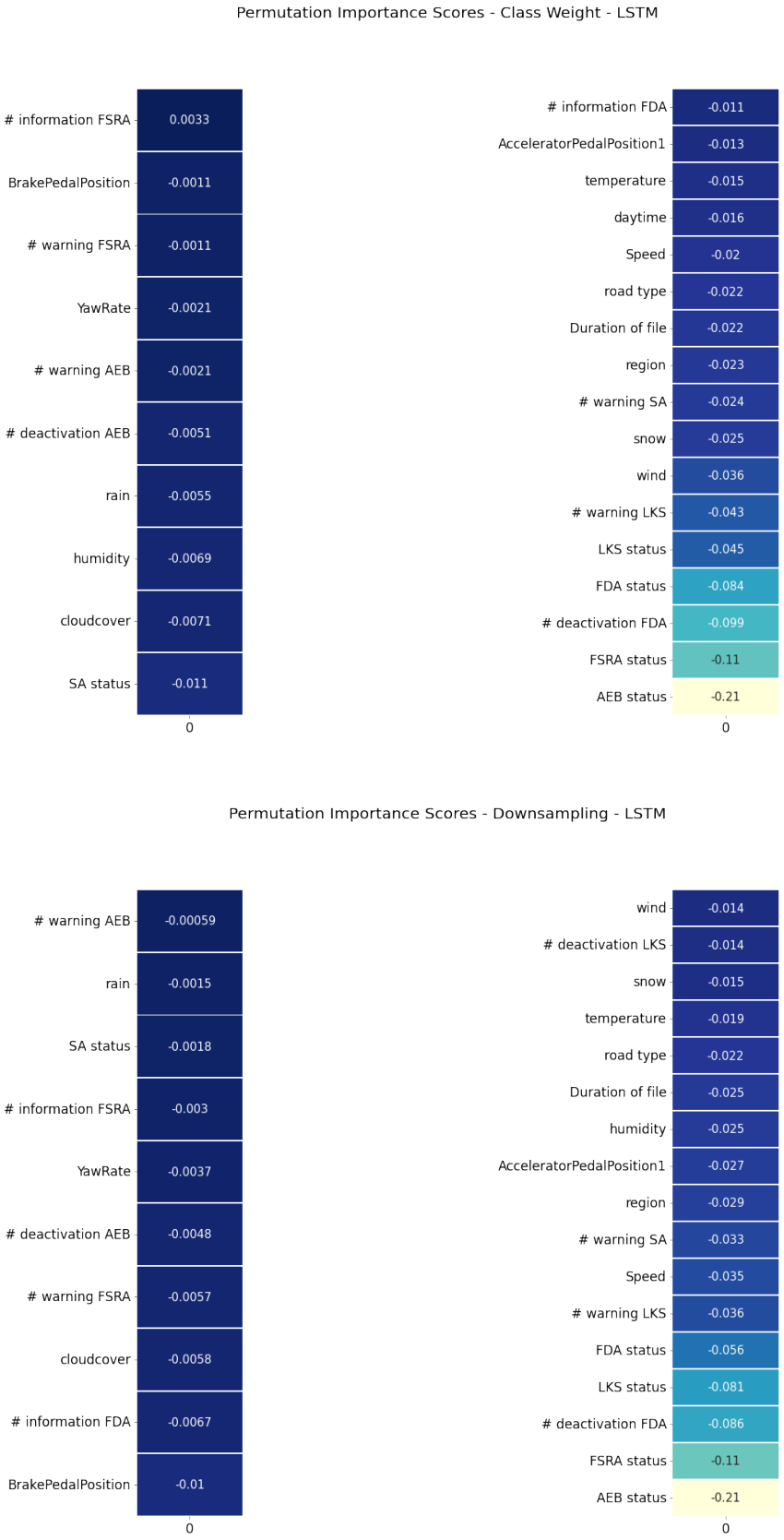


Figure A.21: LSTM's permutation importance scores for class weighting and downsampling, TS2.

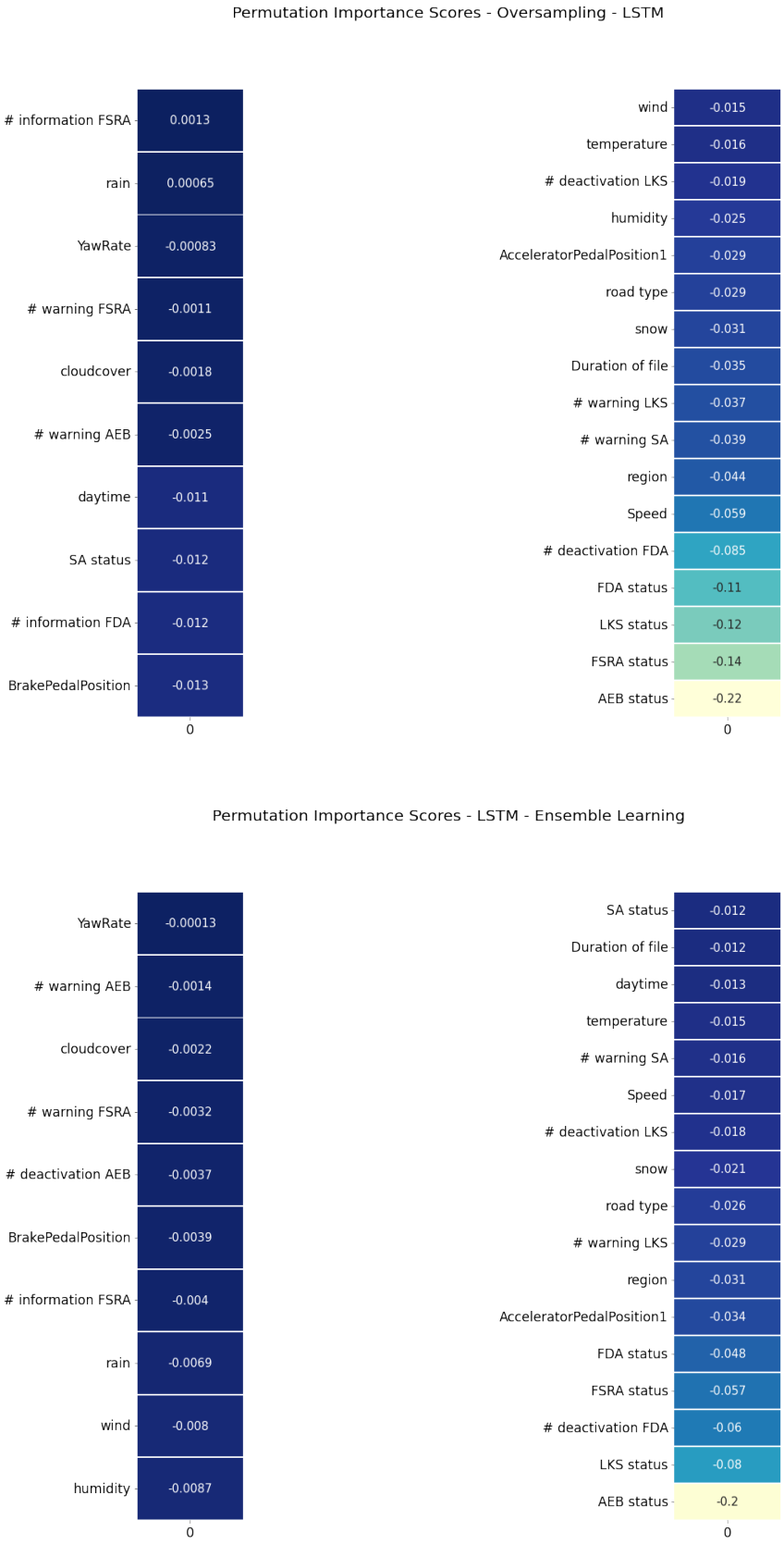


Figure A.22: LSTM's permutation importance scores for oversampling and ensemble learning, TS2.

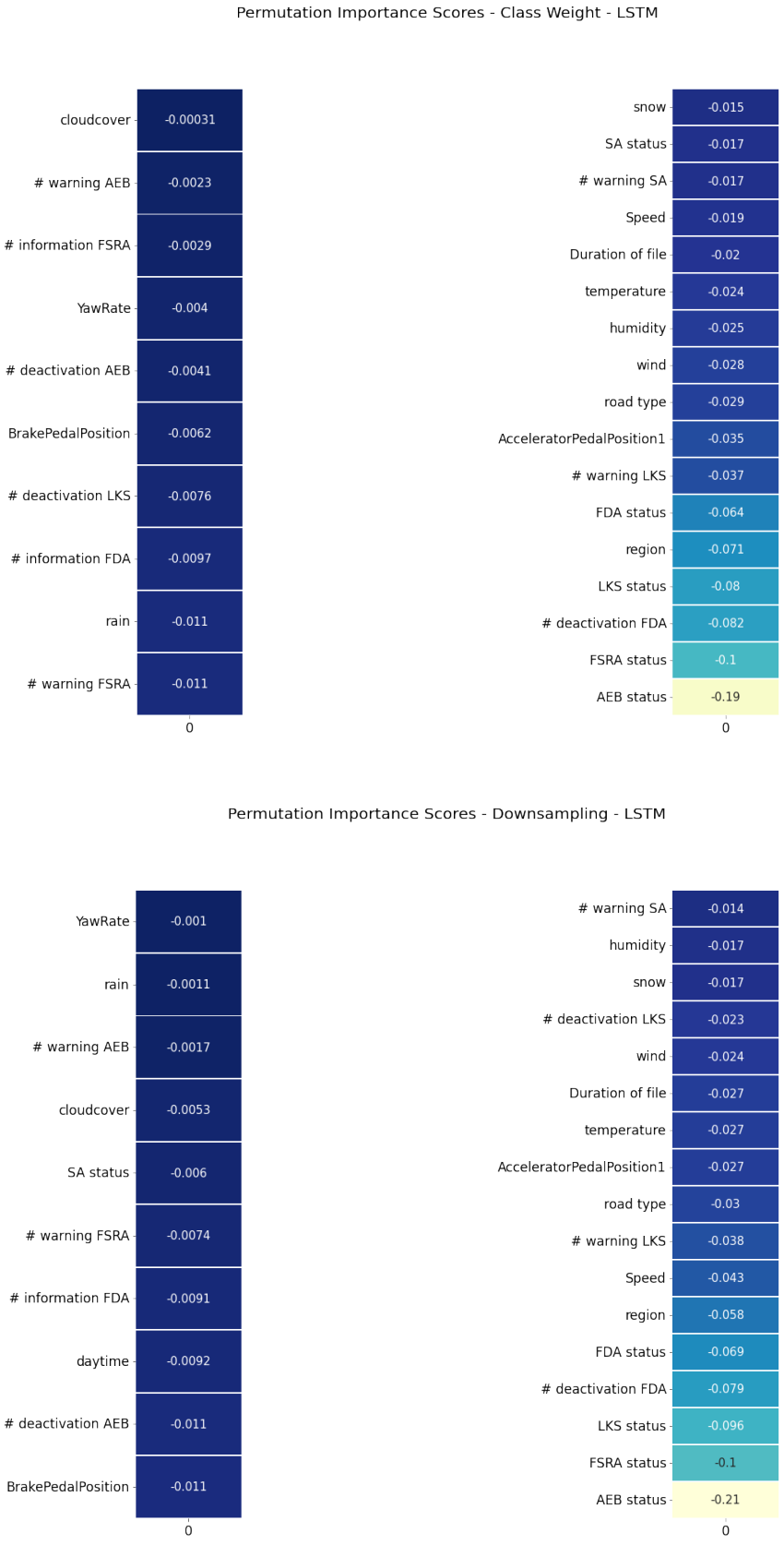


Figure A.23: LSTM's permutation importance scores for class weighting and downsampling, TS3.

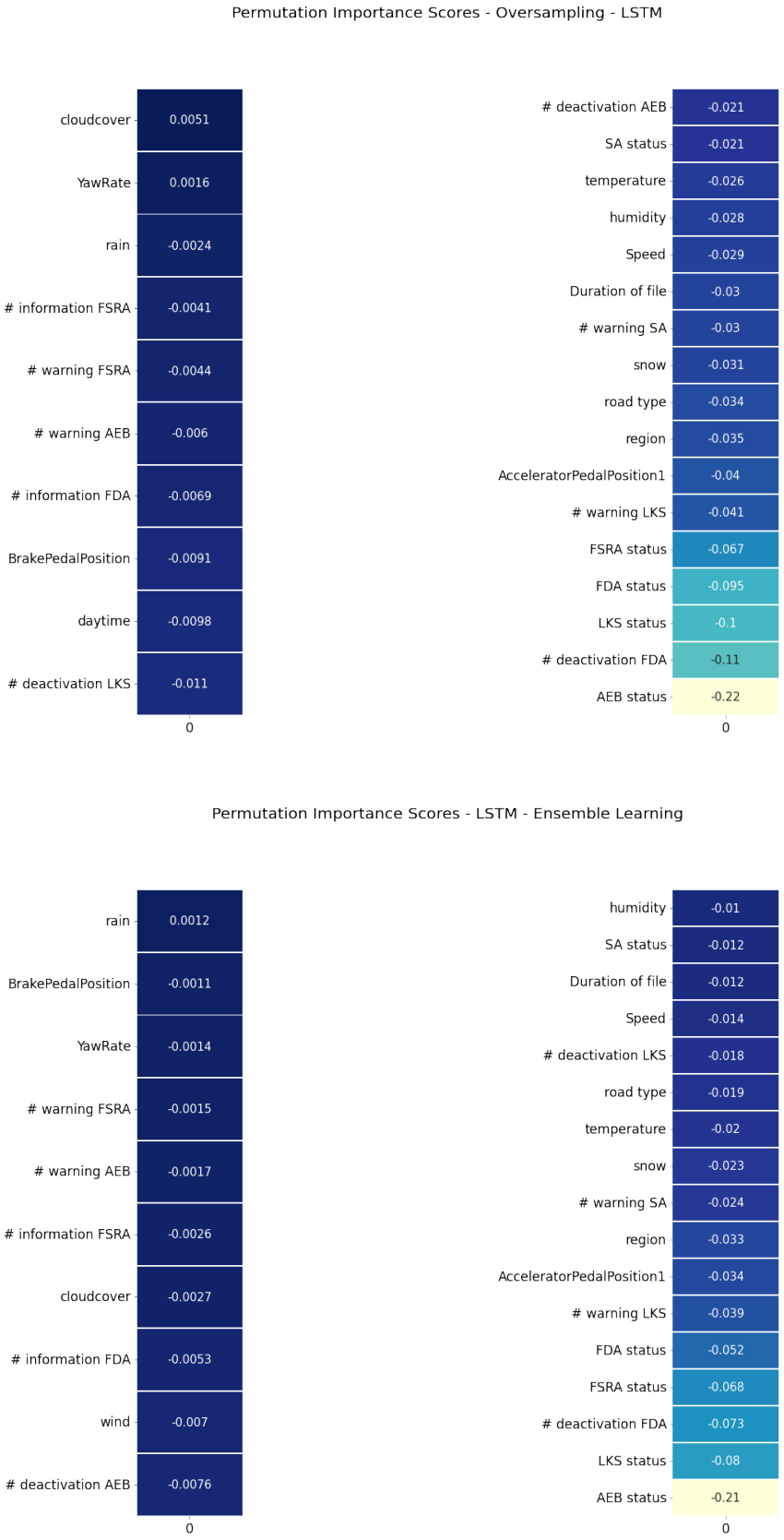


Figure A.24: LSTM's permutation importance scores for oversampling and ensemble learning, TS3.

A.1.6 GRU

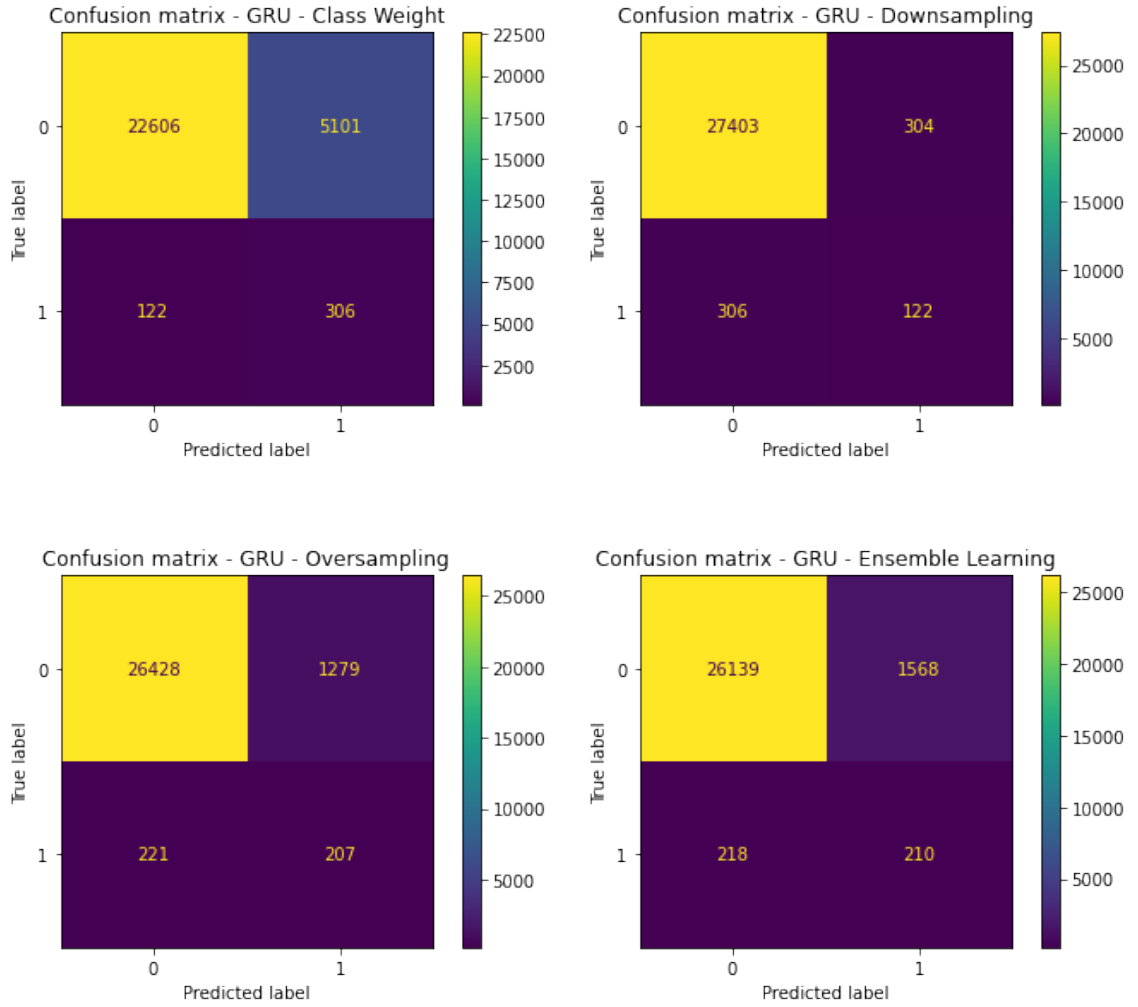


Figure A.25: Confusion matrices of the test set for the different approaches, TS2.

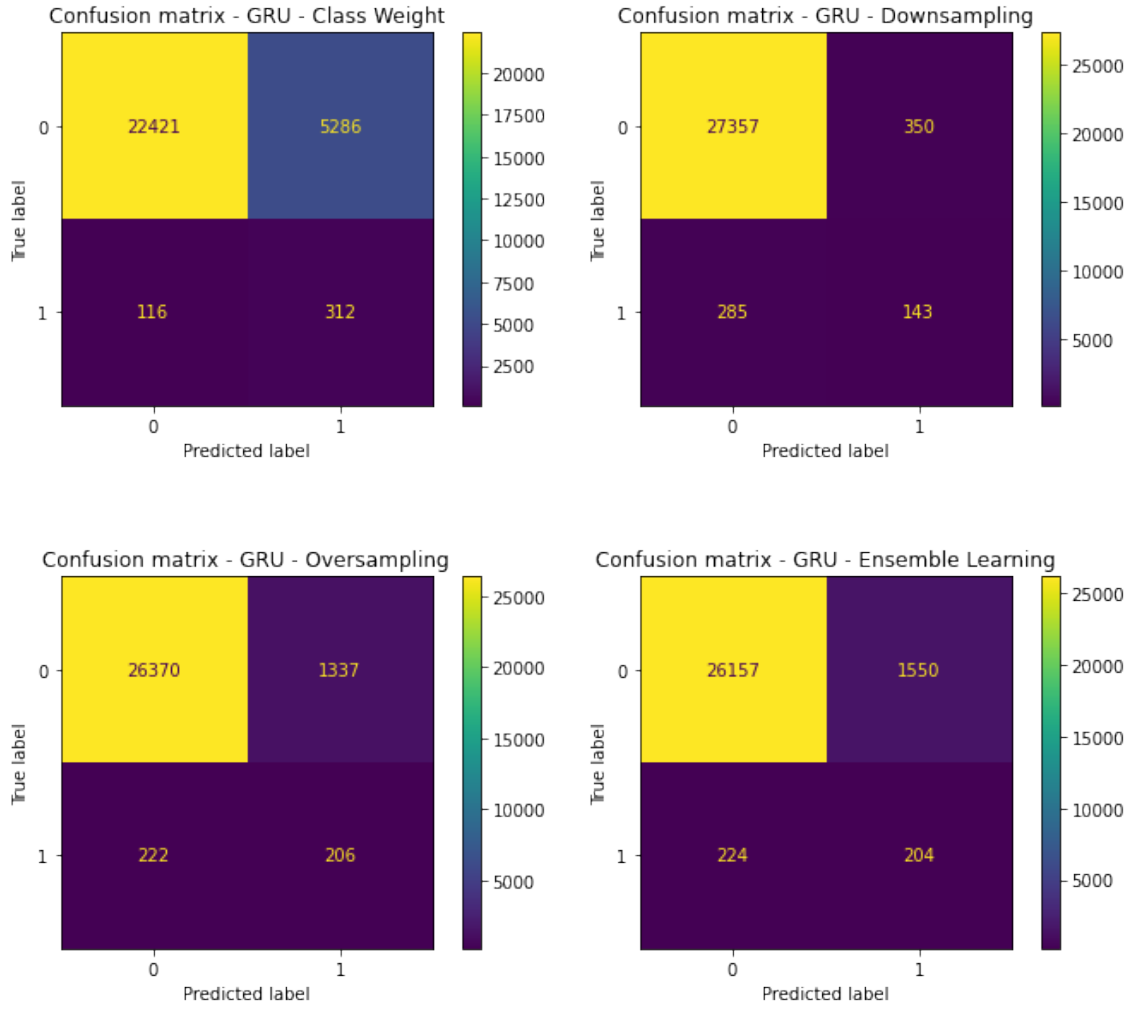


Figure A.26: Confusion matrices of the test set for the different approaches, TS3.

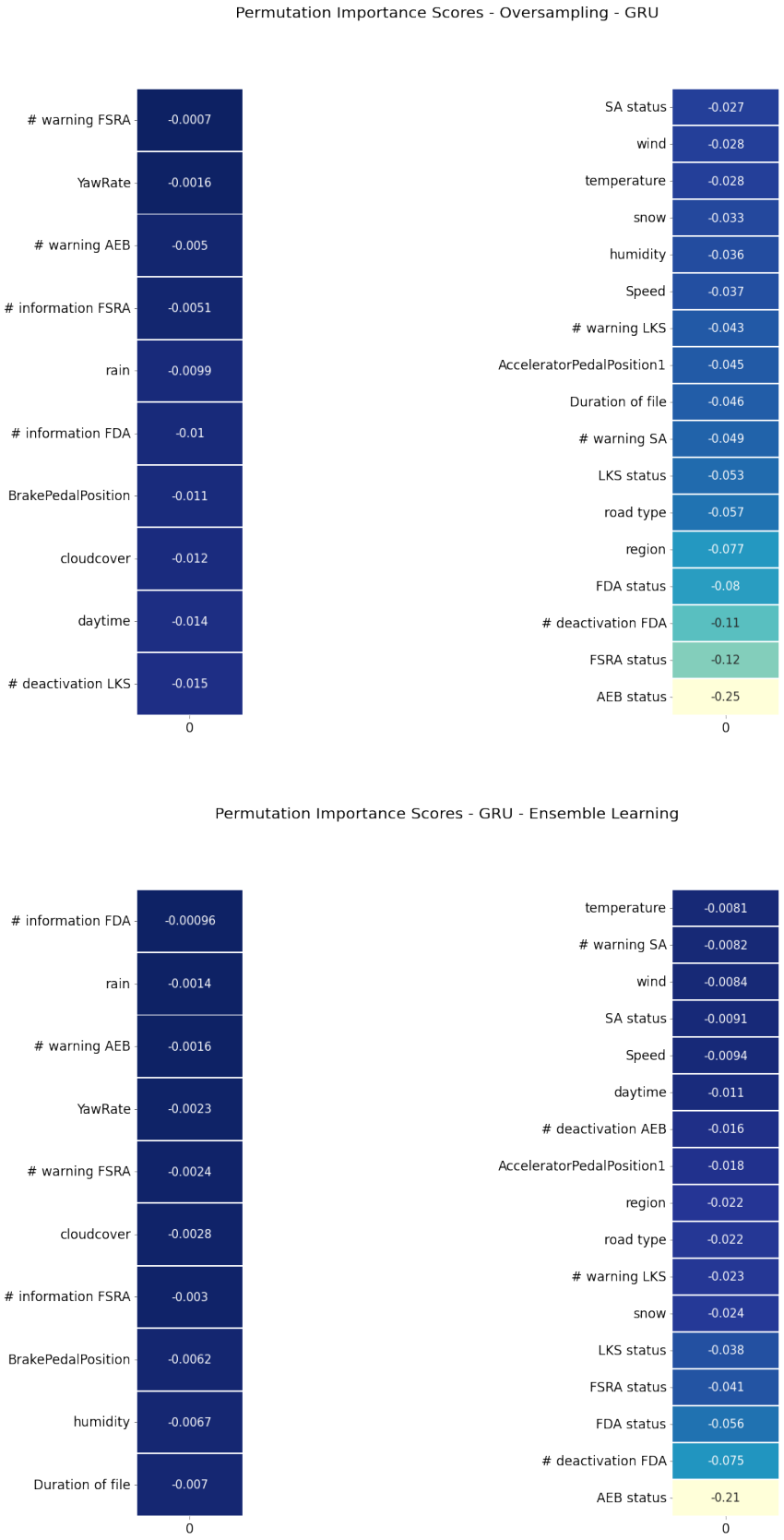


Figure A.27: GRU's permutation importance scores for oversampling and ensemble learning, TS1.

XXX

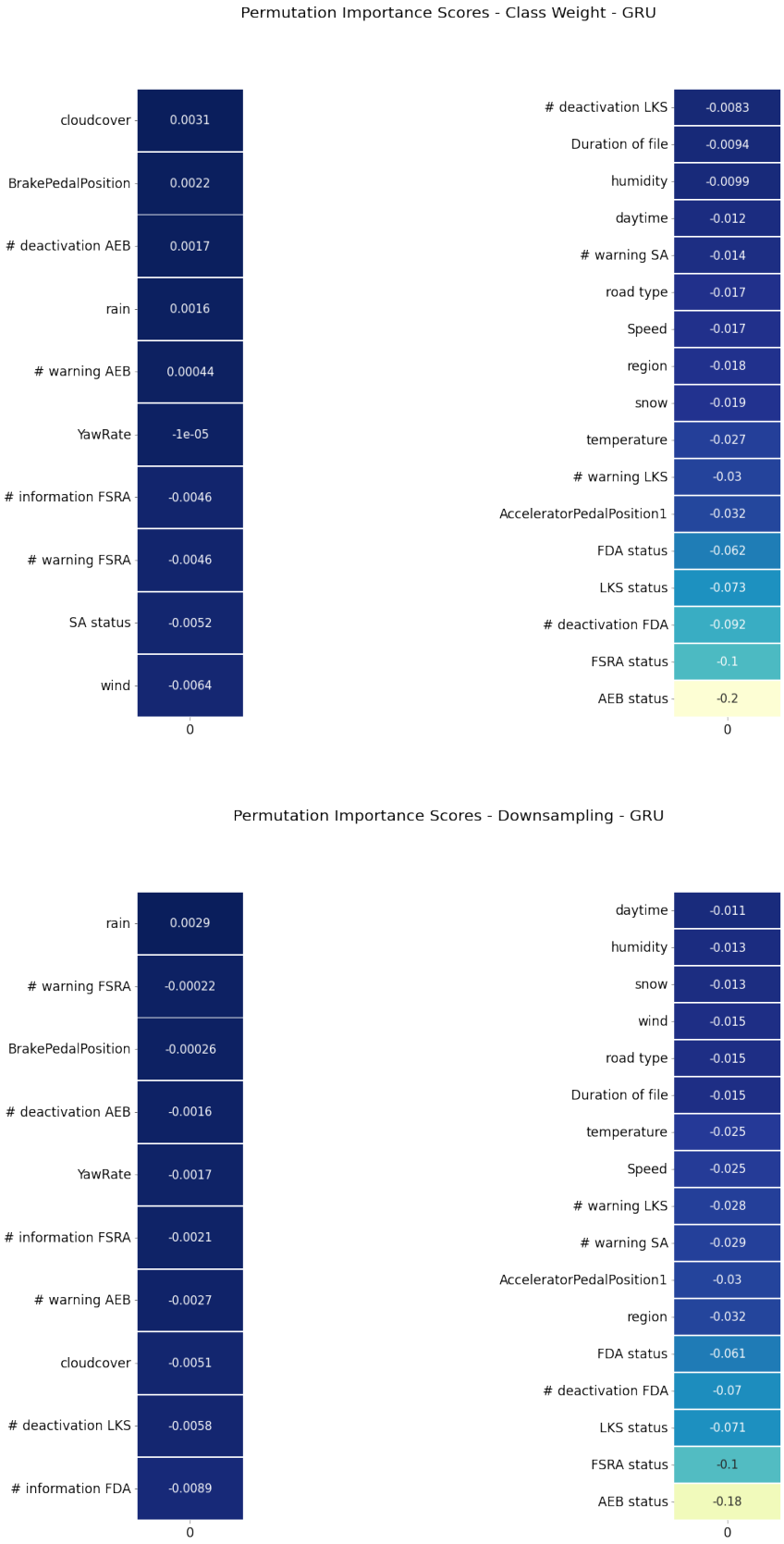


Figure A.28: GRU's permutation importance scores for class weighting and downsampling, TS2.

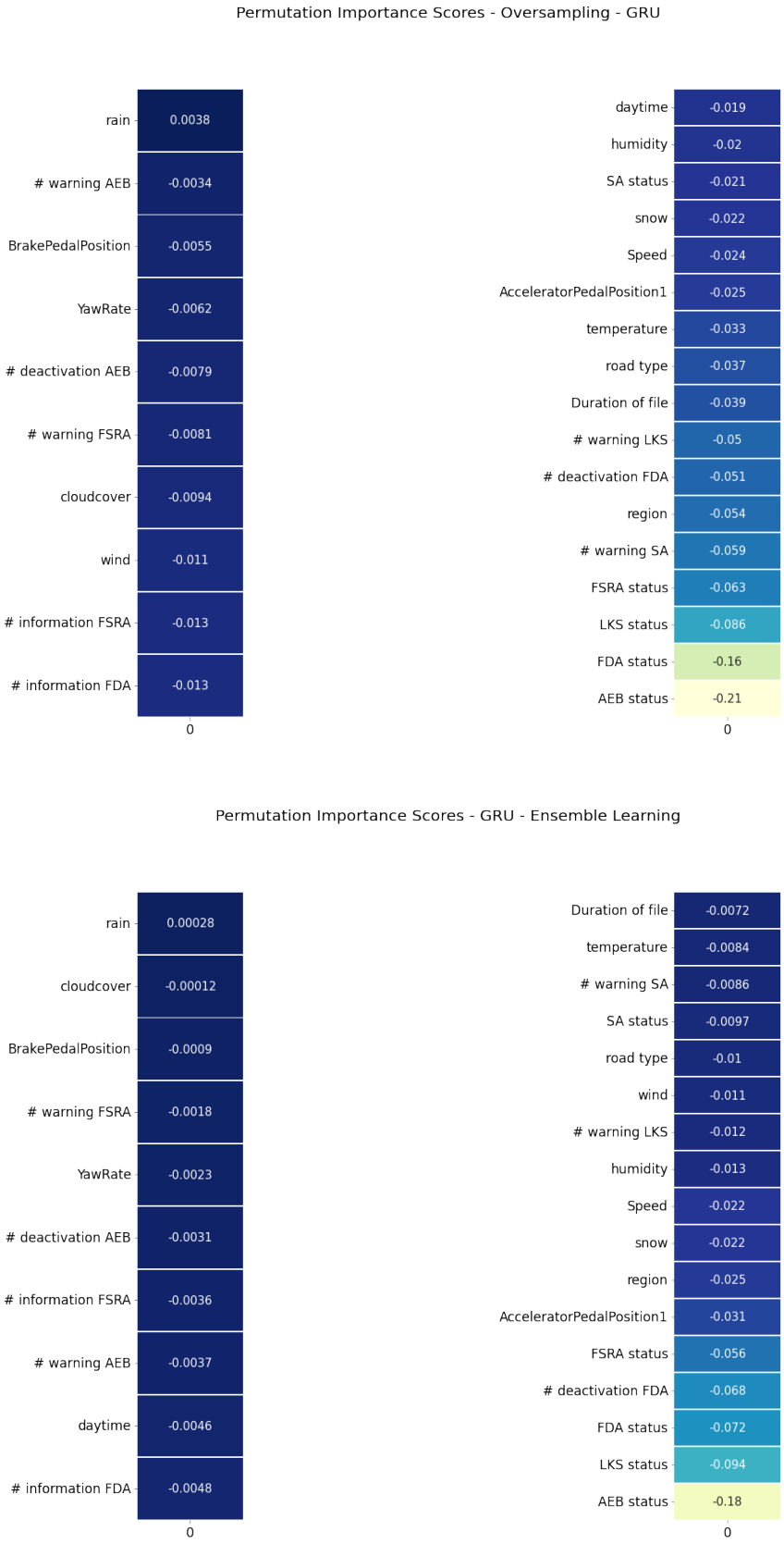


Figure A.29: GRU's permutation importance scores for oversampling and ensemble learning, TS2.

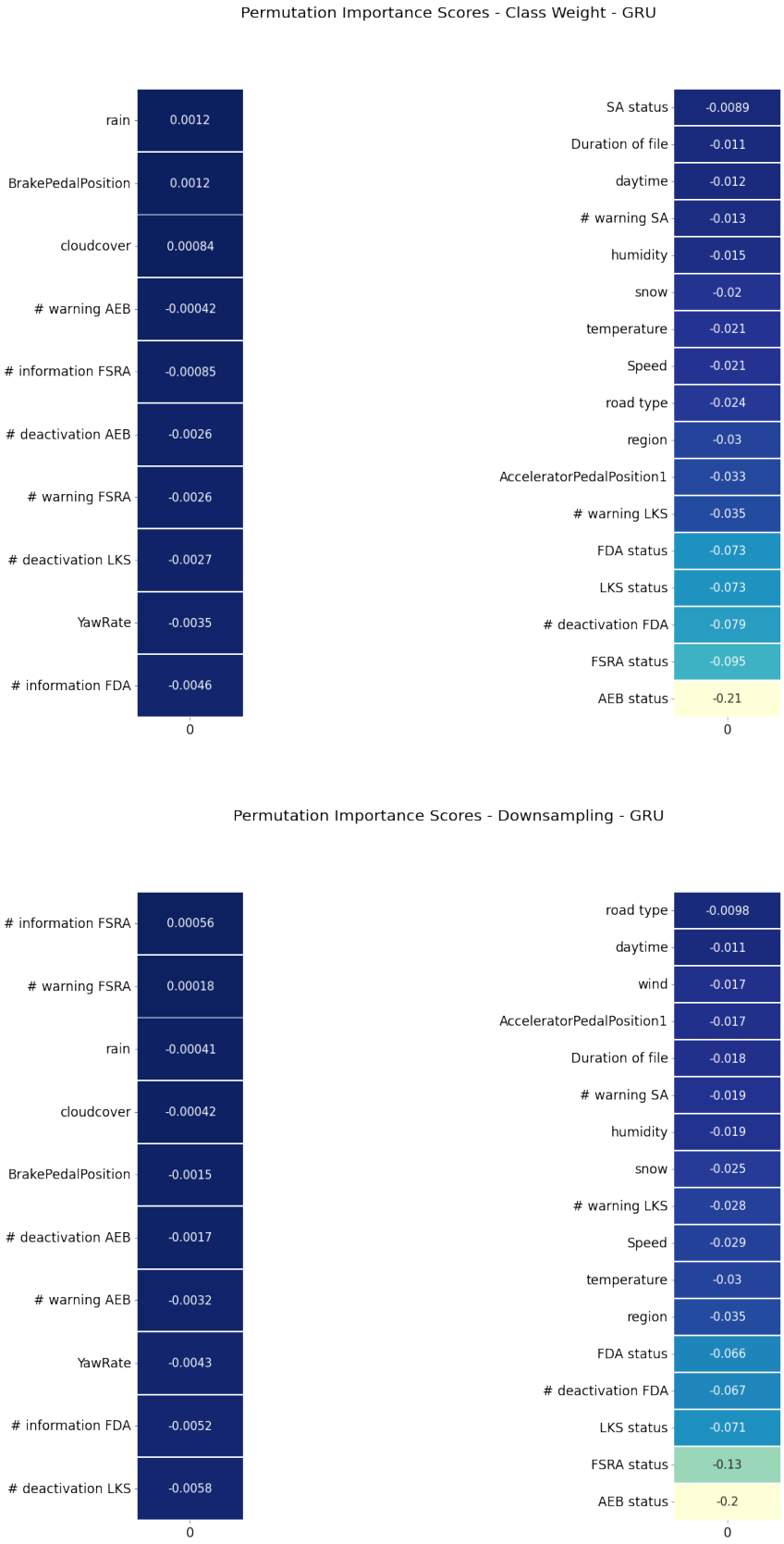


Figure A.30: GRU's permutation importance scores for class weighting and downsampling, TS3.

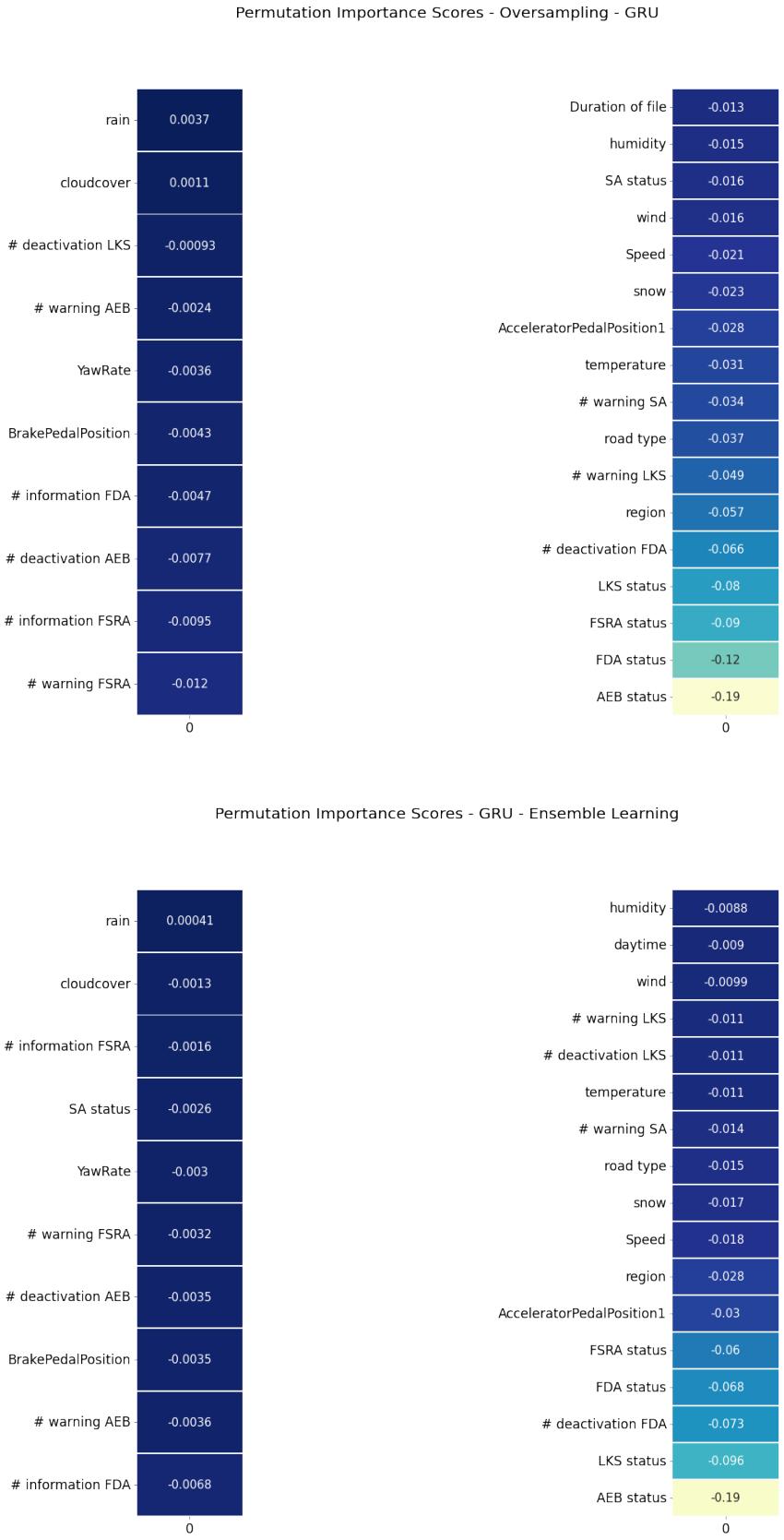


Figure A.31: GRU's permutation importance scores for oversampling and ensemble learning, TS3.

A.1.7 BiLSTM

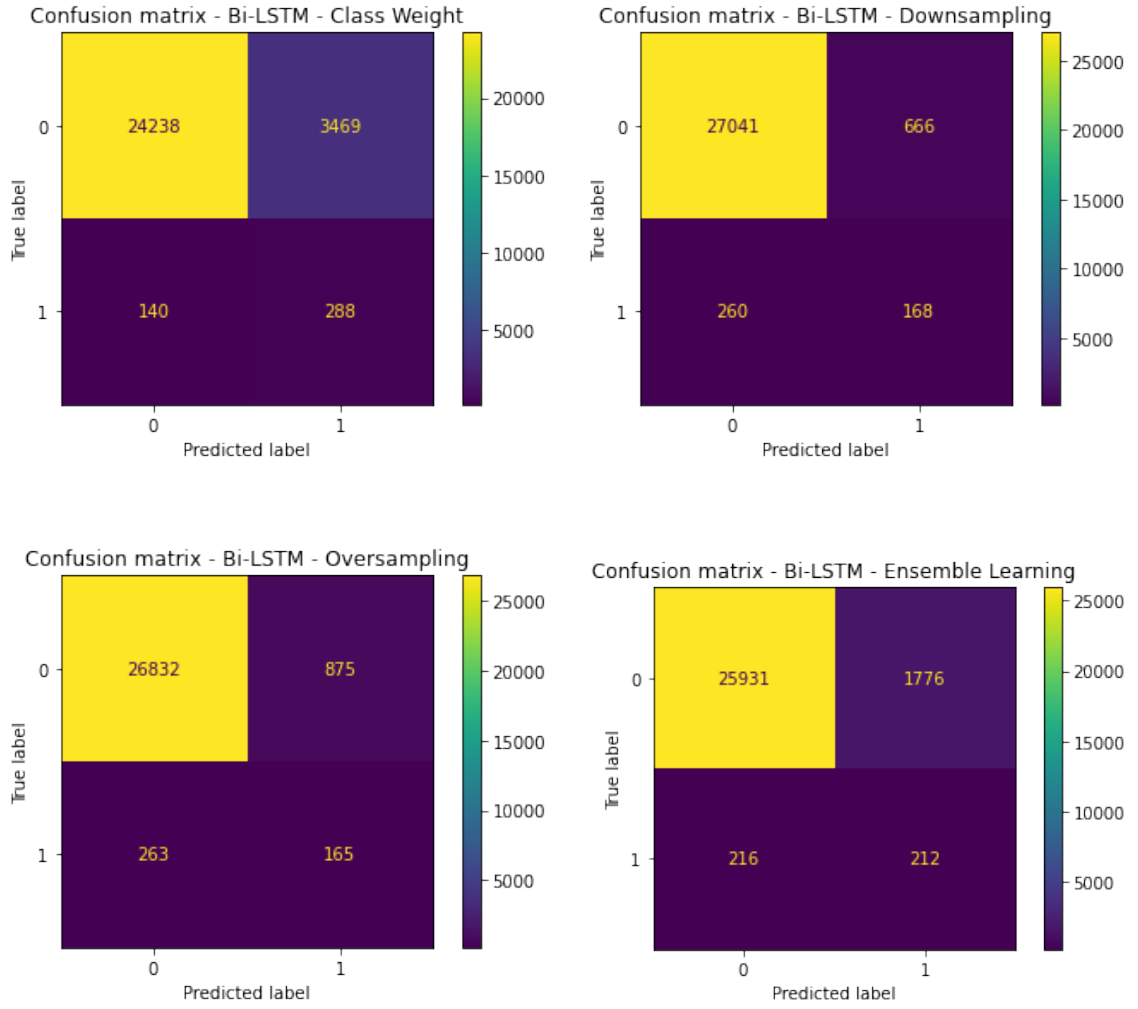


Figure A.32: Confusion matrices of the test set for the different approaches, TS2.

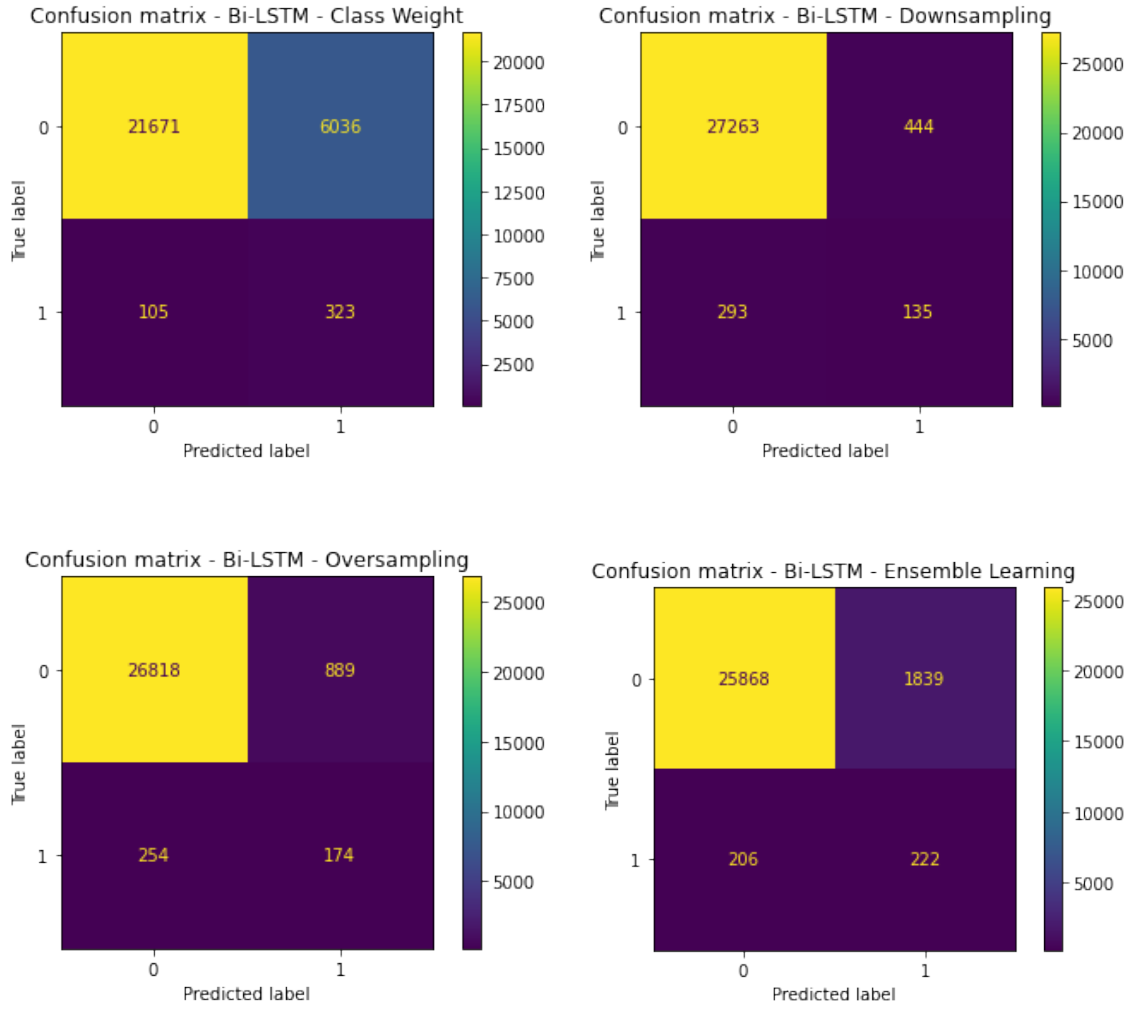


Figure A.33: Confusion matrices of the test set for the different approaches, TS3.

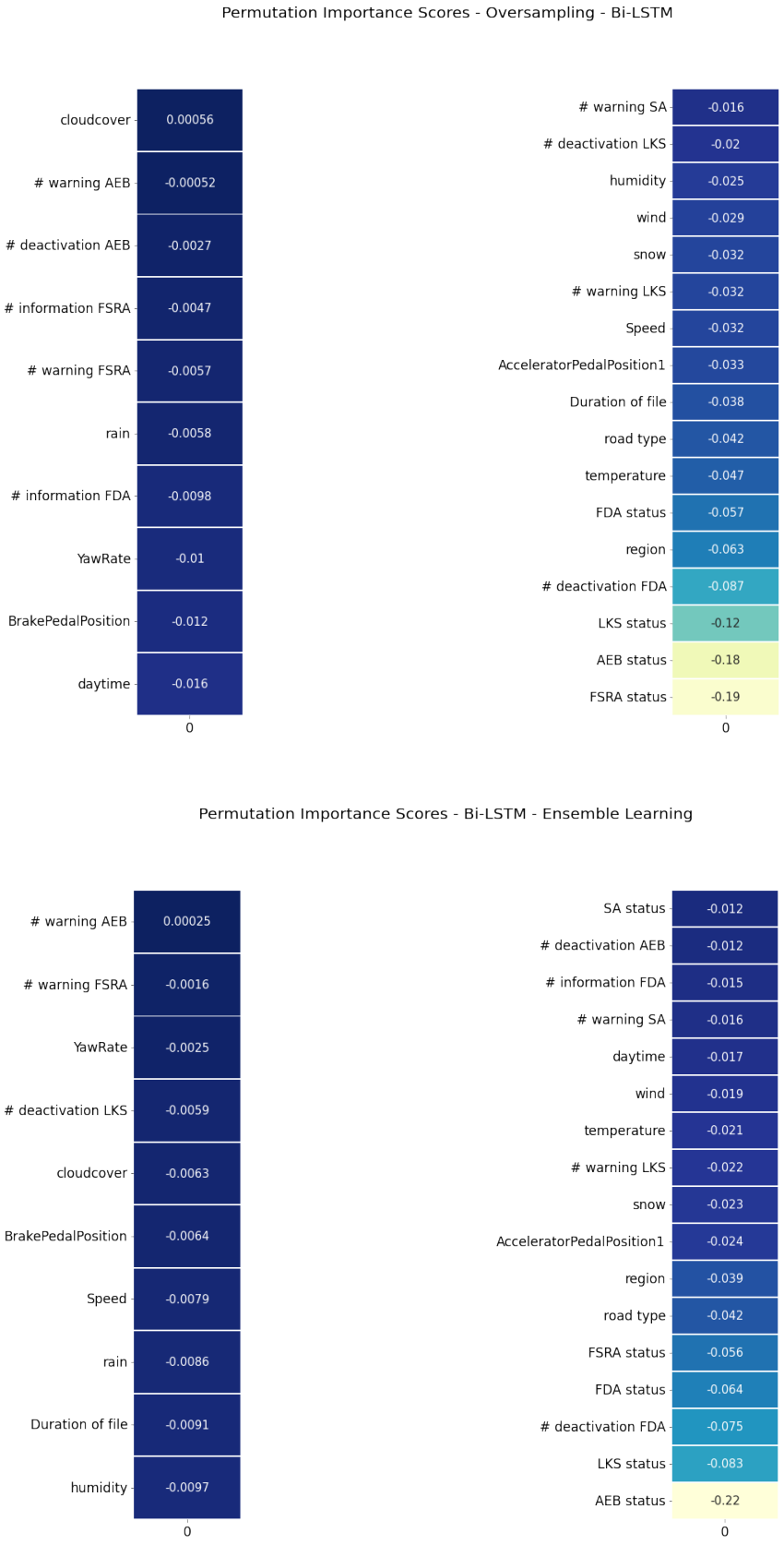


Figure A.34: BiLSTM's permutation importance scores for oversampling and ensemble learning, TS1.

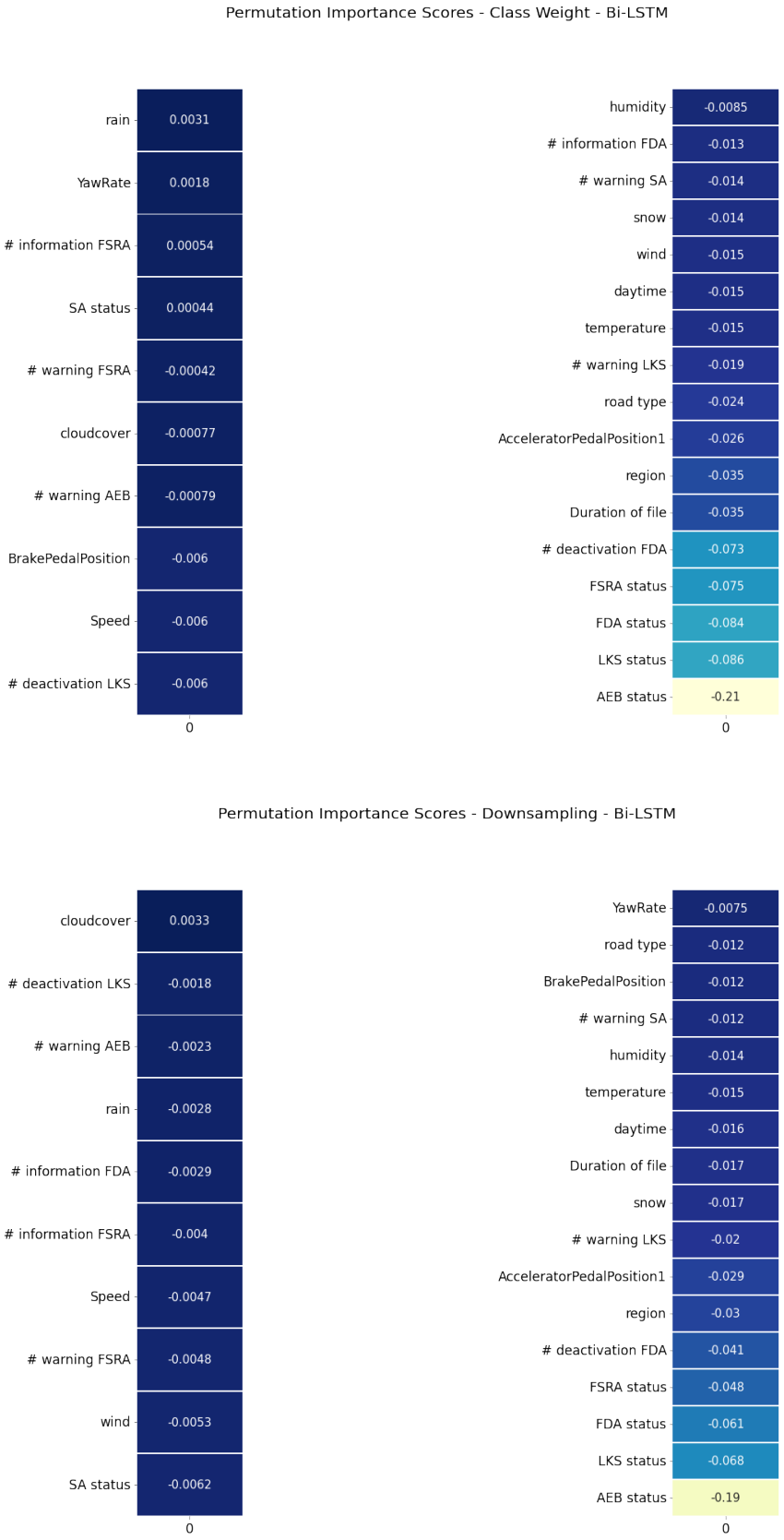


Figure A.35: BiLSTM's permutation importance scores for class weighting and downsampling, TS2.

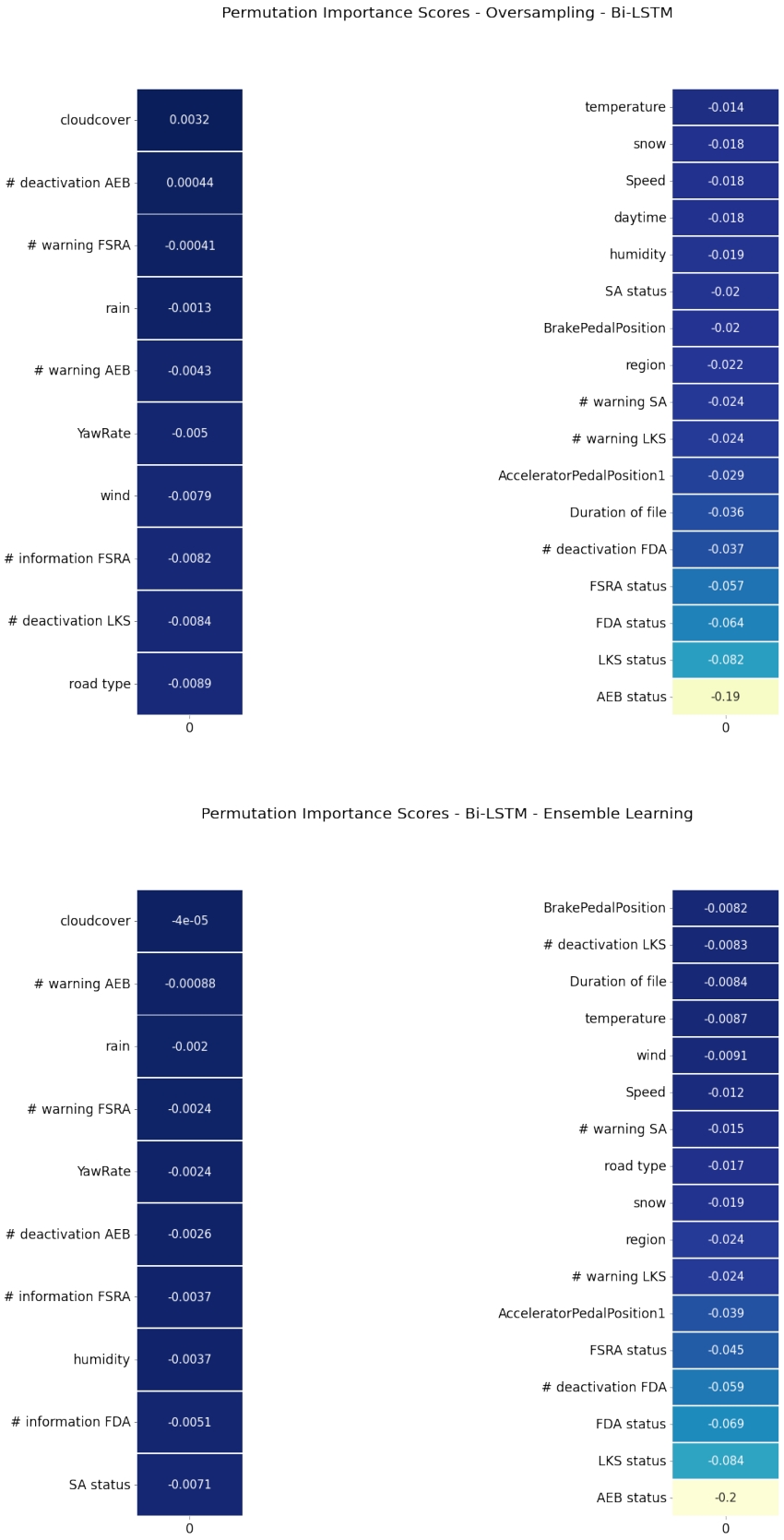


Figure A.36: BiLSTM's permutation importance scores for oversampling and ensemble learning, TS2.

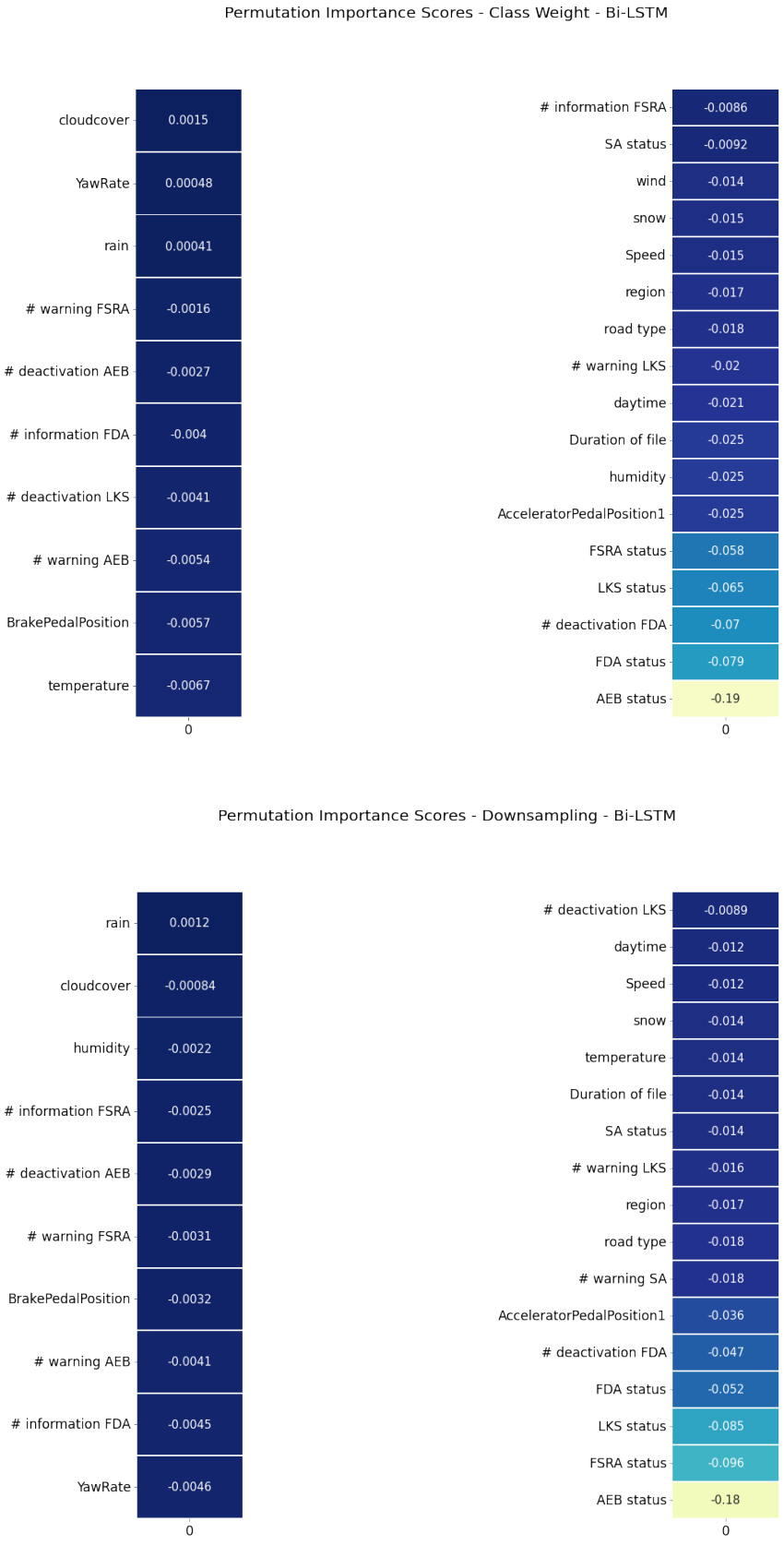


Figure A.37: BiLSTM's permutation importance scores for class weighting and downsampling, TS3.

XL

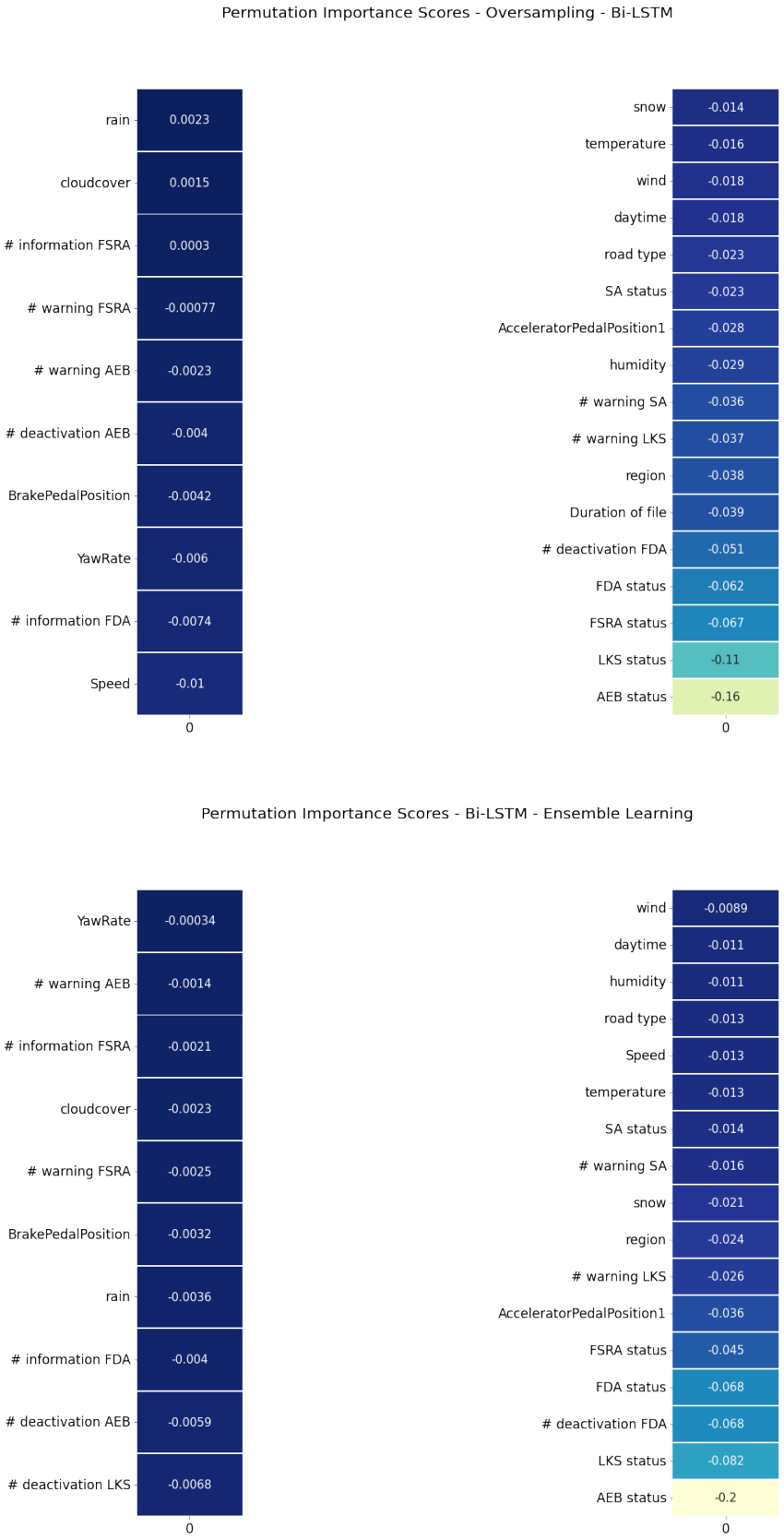


Figure A.38: BiLSTM's permutation importance scores for oversampling and ensemble learning, TS3.

A.1.8 Parallel CNN-LSTM

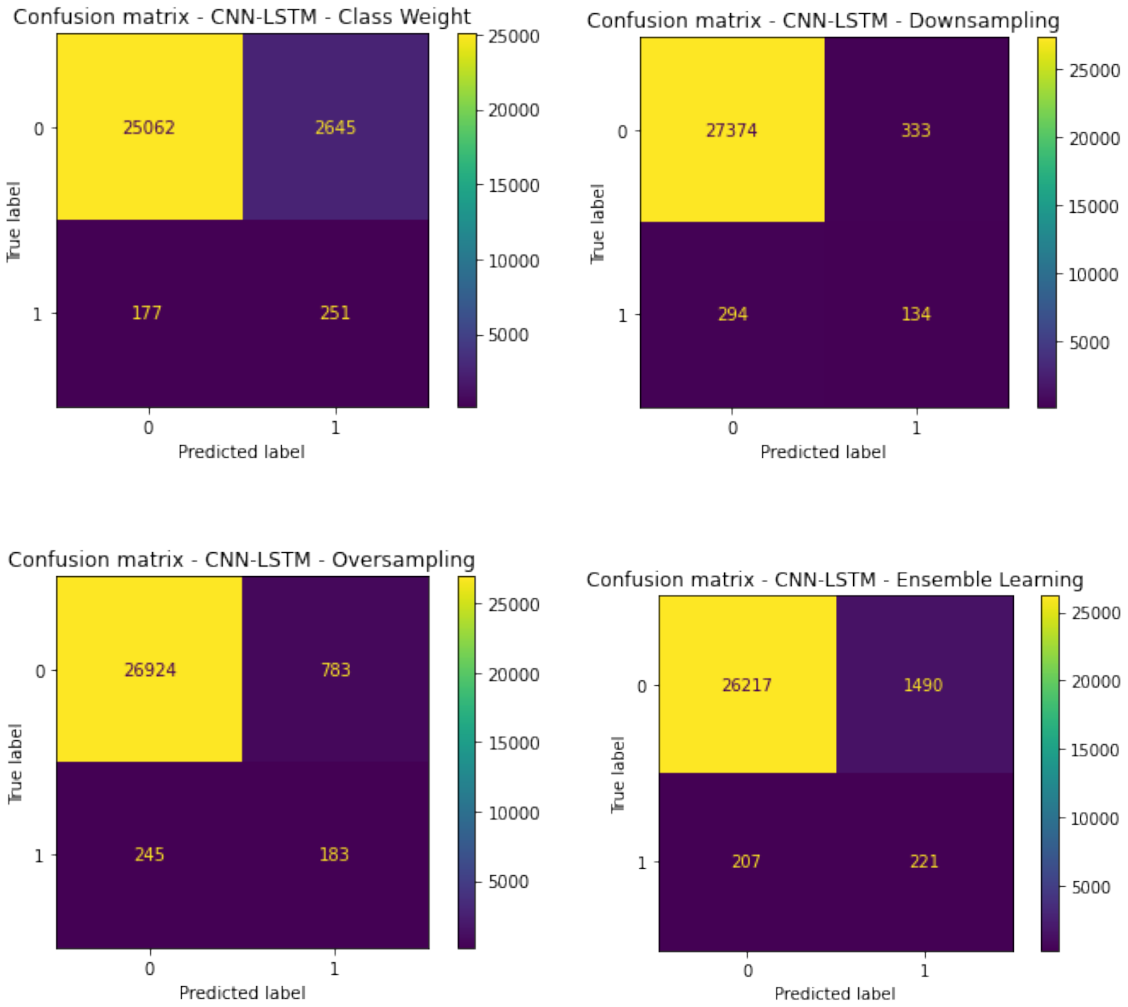


Figure A.39: Confusion matrices of the test set for the different approaches, TS2.

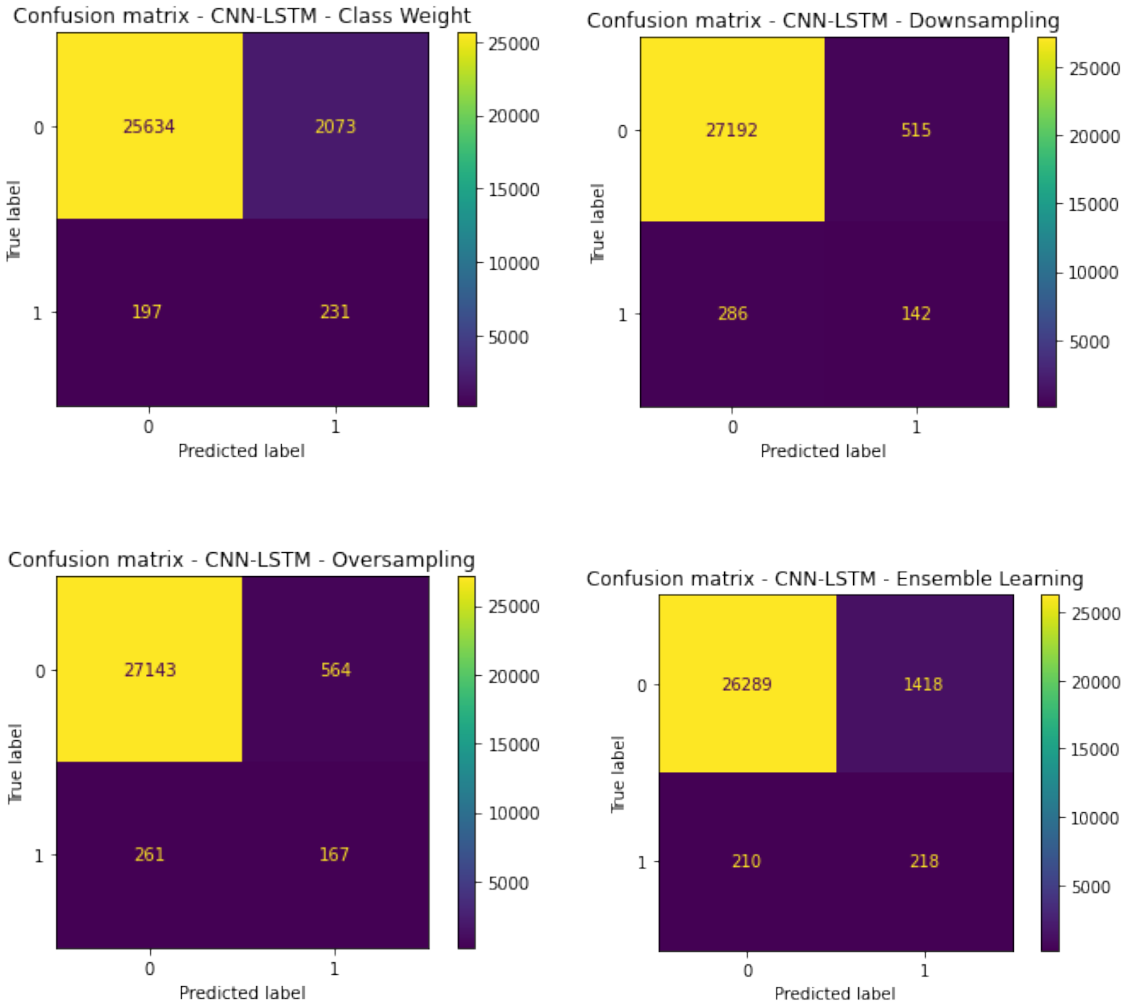


Figure A.40: Confusion matrices of the test set for the different approaches, TS3.

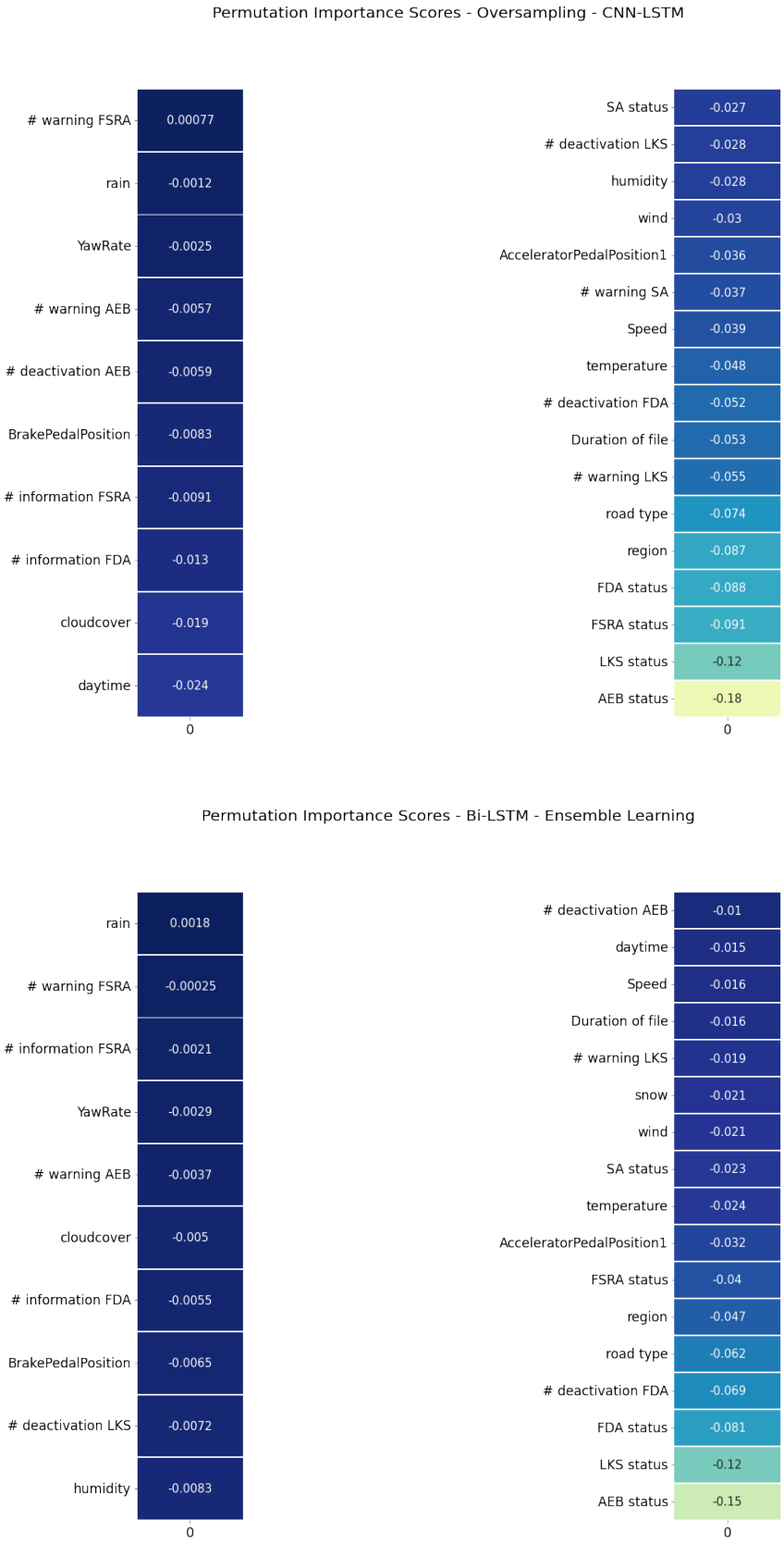


Figure A.41: Parallel CNN-LSTM's permutation importance scores for oversampling and ensemble learning, TS1.

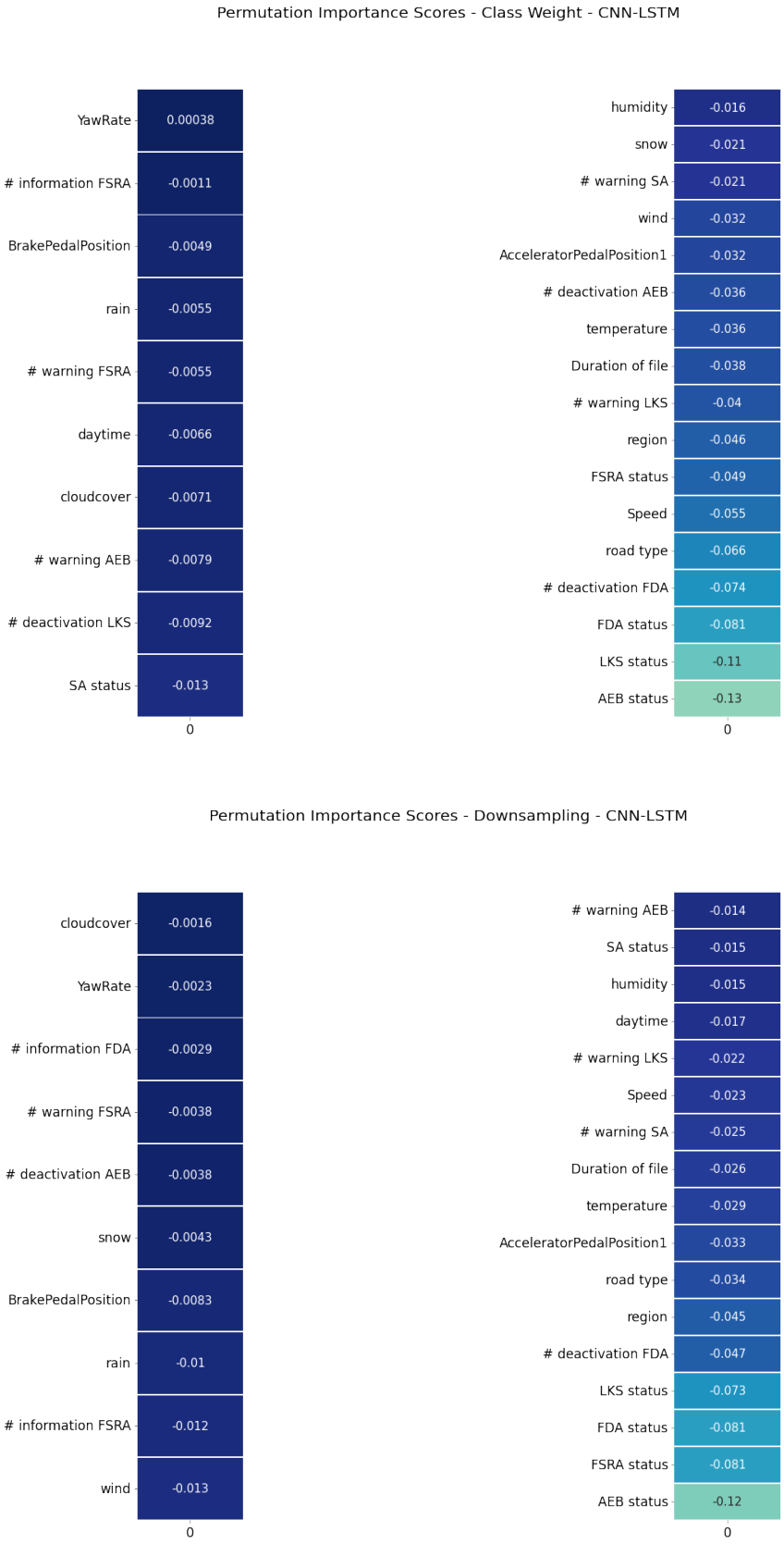


Figure A.42: Parallel CNN-LSTM's permutation importance scores for class weighting and downsampling, TS2.

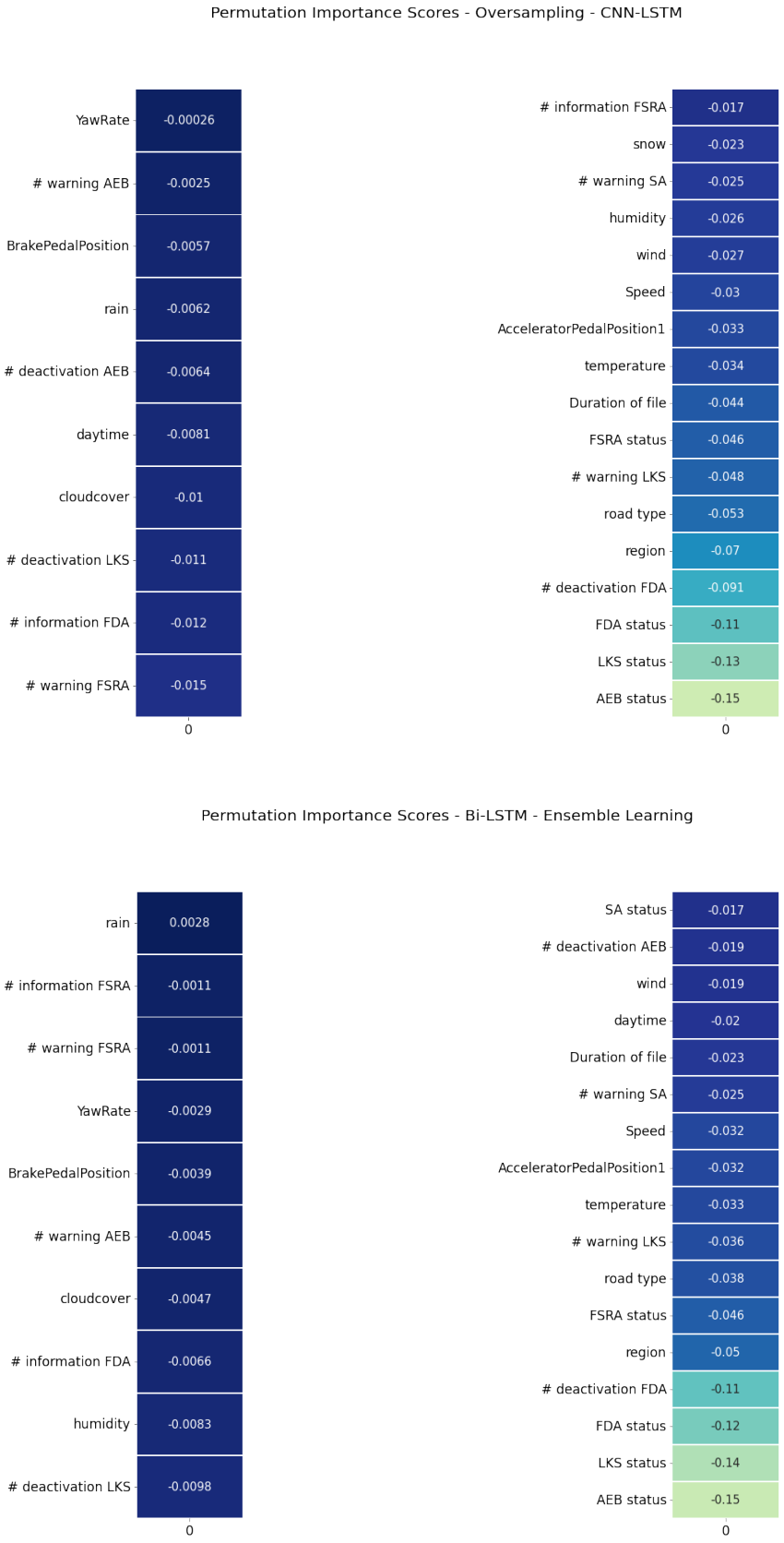


Figure A.43: Parallel CNN-LSTM's permutation importance scores for oversampling and ensemble learning, TS2.

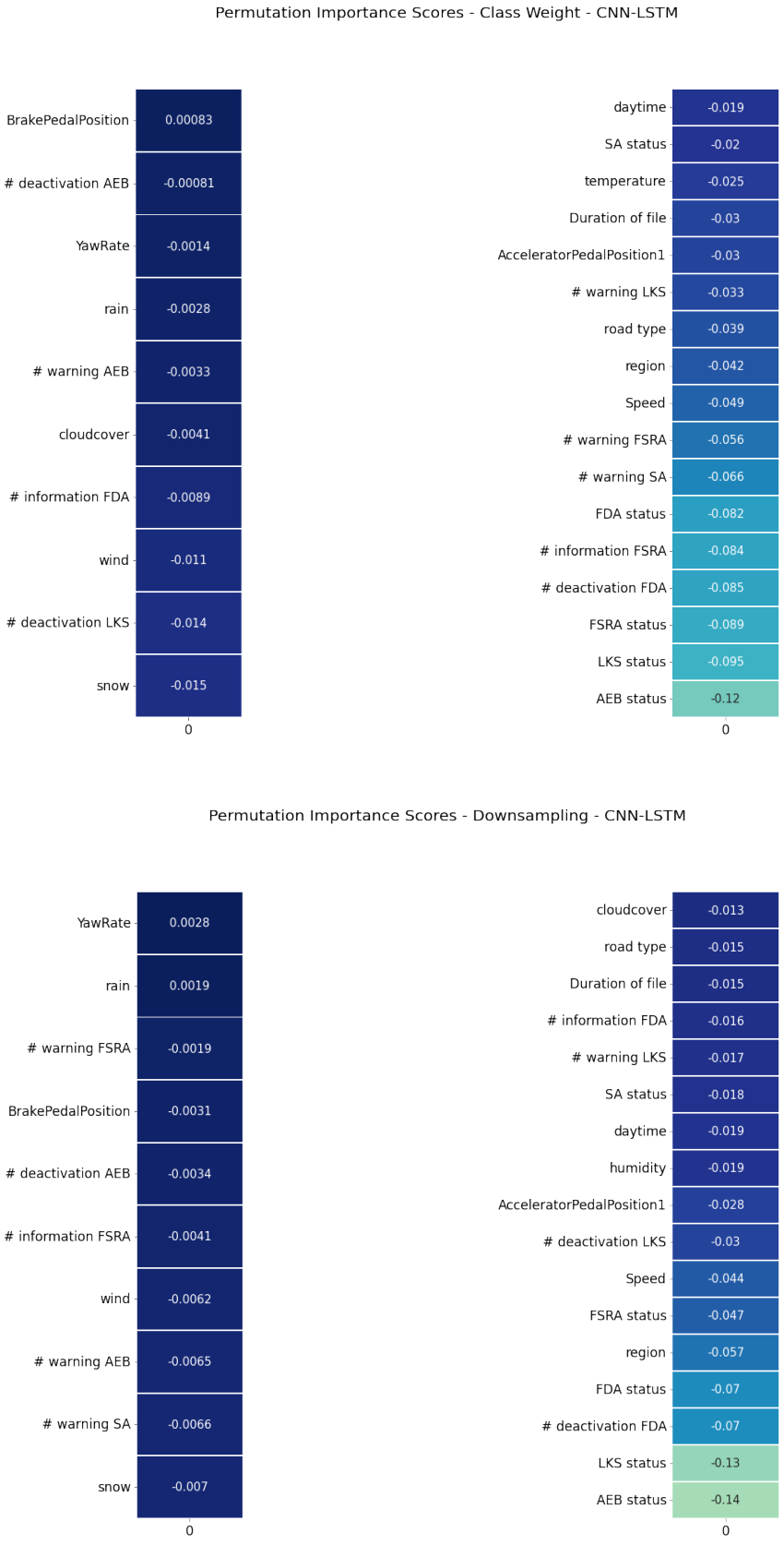


Figure A.44: Parallel CNN-LSTM's permutation importance scores for class weighting and downsampling, TS3.

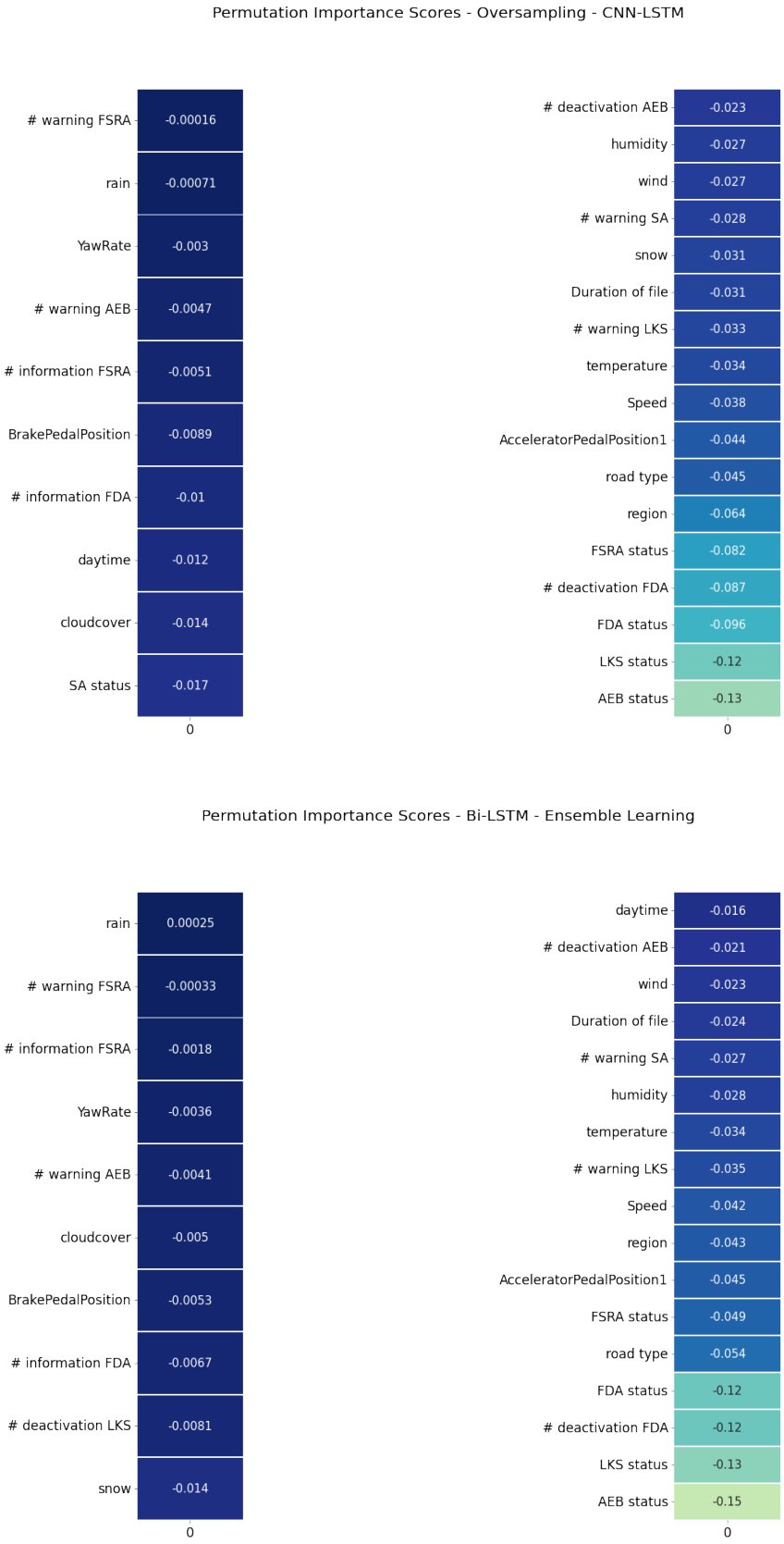


Figure A.45: Parallel CNN-LSTM's permutation importance scores for oversampling and ensemble learning, TS3.