UiO : Department of Physics
University of Oslo

FYS-STK3155/4155 -

APPLIED DATA ANALYSIS AND MACHINE LEARNING

Machine Learning Models for Sleep Apnea Classification: A Comparative Study Using The SHHS Dataset

Authors:
Anders Rudi Bråthen Bakir
Nadia Elise Helene Ørning
Trond Victor Qian

14th December 2024

Abstract

This report examined three machine learning techniques: Random Forest, Feedforward Neural Networks, and Convolutional Neural Networks and their performance in classifying sleep apnea events. The Sleep Heart Health Study (SHHS) dataset, conducted by the National Heart, Lung, and Blood Institute of the US, provided the basis for our study. We specifically used the SHHS1 dataset, which includes polysomnogram data from 6441 participants.

Our approach involved data preprocessing, model training, and evaluation of performance metrics for each machine learning model. As the SHHS1 dataset has class imbalance, we trained the models on both balanced and imbalanced SHHS1 datasets, and used multiple metrics, accuracy, precision, recall, AUC and F1 to assess their performance. The models were evaluated on the unbalanced SHHS1 dataset to reflect the real-world application.

Balancing the dataset before training proved beneficial in detecting the minority class (Apnea/Hypopnea). The Random Forest model achieved an F1 score of 0.46 on the balanced dataset, compared to 0.29 on the unbalanced dataset for the minority class. The FFNN attained an F1 score of 0.44 on the balanced dataset, but only 0.21 on the unbalanced. The CNN performed less effectively, with an F1 score of 0.31 on the balanced dataset and just 0.02 on the unbalanced dataset.

These results highlight the importance of handling class imbalance in time-series datasets. While the models' performance across the board was limited, particularly in classifying the minority class, the Random Forest and FFNN models demonstrated more consistent results. Neural networks, however, are sensitive to hyperparameter tuning and typically require more data and computational resources to achieve optimal performance. Due to limited access to powerful computing resources, extensive hyperparameter tuning for FFNN and CNN was constrained, which may have impacted its performance. Although studies like ones by Wang et al. state CNN having optimal performance for sleep apnea classification [1], in the end, the Random Forest model achieved the highest F1 score (0.46) for the minority class, FFNN the second highest (0.44) and CNN last (0.31).

Future work will focus on exploring advanced architectures, such as LSTM, more extensive hyperparameter tuning, and further improving class imbalance.

Table of Contents

II Appendix **33**

1 Introduction

Sleep apnea, a prevalent sleep disorder characterized by interruptions in breathing during sleep, has been the subject of significant research due to its potential links to serious health conditions such as coronary heart disease, stroke, all-cause mortality, and hypertension. It is estimated that 10% of the global adult population suffers from severe sleep apnea, while up to 80% of sleep apnea cases remain undiagnosed [2]. The current gold standard to diagnose sleep apnea is through polysomnography, a comprehensive sleep study conducted in specialized labs or hospitals. This process, however, is resource-intensive, time-consuming, and often inaccessible to many patients due to its high cost and limited availability. To address these challenges, there has been growing interest in using machine learning to improve the accuracy, accessibility, and efficiency of sleep apnea diagnostics.

This report analyzes polysomnogram data from the Sleep Heart Health Study dataset (SHHS), conducted by the National Heart, Lung, and Blood Institute of the US [3]. We will specifically focus on the SHHS1 data set, which includes detailed physiological recordings. We will focus on the classification of sleep apnea, where we compare three machine learning techniques: Random Forest, Feedforward Neural Networks, and Convolutional Neural Networks, in classifying sleep apnea events. For this project, Random Forests were chosen due to their robustness and ability to handle imbalanced datasets. They offer a simpler and more interpretable alternative to neural networks, making them an ideal baseline for comparison. Feedforward Neural Network was included due to its implementation in a prior project (Project 2), in which we can assess its performance in this context. Lastly, Convolutional Neural Networks (CNNs) were selected based on their state-of-the-art performance in sleep apnea classification tasks, as highlighted by this study by Wang et al. [1].

The goal of this study is to evaluate the effectiveness of the different machine learning models in detecting sleep apnea by comparing their performance using metrics like accuracy, precision, recall, F1, and AUC. In addition, we will look at the impact of class imbalance, by training the models on both balanced and imbalanced datasets, before evaluating them on the unbalanced dataset, reflecting real-world conditions.

The report is structured as follows: Section 2 presents the theoretical background, Section 3 outlines the methods used, Section 4 gives the results, and Section 5 discusses the results. The code for this project can be found at: https://github.com/TrondVQ/FYS_STK_P3.

2 Theory

2.1 Decision Trees and Random Forests

Decision trees are widely used in machine learning and are suitable for both regression and classification tasks. In classification tasks, like in this project, these models predict categorical outcomes by assigning observations to the class that appears most frequently among training data within a defined region [4].

A classification tree begins at a root node, where the input data is introduced. The dataset is then divided progressively at decision nodes, each of which applies a specific condition or test to separate the data into smaller, more homogeneous subsets. This process continues until the tree reaches terminal nodes, known as leaf nodes, where final class predictions are made. By using this structured approach, classification trees can identify complex relationships in data.

However, decision trees have limitations. They are often less accurate than alternative classification techniques and can be highly sensitive to small changes in the input data, resulting in significant variability in the structure of the tree [4]. These drawbacks highlight the need for more robust methods.

2.1.1 Random Forests

Random forests were developed to address the shortcomings of decision trees, particularly their tendency to overfit and lack of robustness. A random forest is an ensemble model consisting of multiple decision trees. Each tree makes an independent prediction, and the final classification is determined by majority voting across all trees [5].

Random forests improve decision trees in several ways, particularly by introducing randomness and aggregation to reduce variance and enhance stability. However, their performance depends on several critical hyperparameters:

- Number of Estimators: This defines how many trees are included in the forest. Increasing this number typically reduces variance and improves performance, although gains diminish as the number of trees grows.
- Maximum Depth: This controls the depth of each tree, limiting the number of splits allowed. Proper tuning of maximum depth prevents overfitting by constraining the model's complexity.

- Maximum Features: This parameter specifies the maximum number of features considered at each split. By randomly selecting features for each tree, random forests introduce diversity among the trees, reducing the risk of overfitting.
- Minimum Samples per Leaf: This sets a lower limit on the number of samples required at each leaf node. Enforcing this constraint ensures that leaf nodes are not overly specific to training data, thereby improving generalization.
- Minimum Samples to Split: This defines the minimum number of samples required to perform a split at a decision node, regulating how the trees grow during training.

2.2 FeedForward Neural Network

This subsection is taken from ”Project 2 - Classification and Regression: From Linear and Logistic Regression to Neural Networks” found at the GitHub repository https://github.com/TrondVQ/FYS_STK_P2.
The FeedForward Neural Network (FFNN) consists of an input layer, one or more hidden layers, and an output layer. It uses backpropagation to train the model by adjusting its parameters to make the most accurate predictions. An example of a typical FFNN architecture is shown in Figure 2.1.

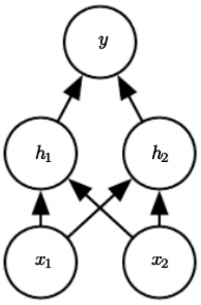

Figure 2.1: Feedforward Neural Network from Goodfellow et al.[6]. x_1 and x_2 are the nodes of the input layer, h_1 and h_2 are the nodes of the hidden layer, and y is the output layer.

The network takes a design matrix **X** as input, which is then processed by the hidden layers. In each hidden layer l, the input is multiplied by a weight matrix $\mathbf{W}_l$ and added to a bias term $\mathbf{b}_l$, resulting in:

$$z_l = XW_l + b_l \tag{1}$$

Before passing to the next layer, $\mathbf{z}_l$ is transformed using an activation function to introduce non-linearity. The activated value is given by a = f($\mathbf{z}_l$). For the output layer (when l = L), $\mathbf{z}_L$ is also processed through an activation function to generate the final output.

2.2.1 Activation functions

As discussed earlier, selecting an activation function for the hidden layers is crucial, and the choice is often determined through experimentation. The activation function in the output layer, however, depends on the type of output desired. Here, we consider three common activation functions [6].

The sigmoid function:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2}$$

The sigmoid function maps input $\mathbf{z}_l$ to a value between 0 and 1, making it suitable for classification tasks. However, if the output saturates near 0 or 1, the derivative approaches zero, leading to ”vanishing gradients” where the weights and biases stop updating effectively, slowing down the learning process.

The ReLU function:

$$\text{ReLU(x)} = \max(0,x) \tag{3}$$

ReLU addresses the vanishing gradient issue by allowing gradients to pass through for positive values, thus improving training speed. However, it can suffer from the ”dying ReLU” problem, where neurons become inactive if too many input values are negative, causing gradients to be zero and halting learning for those neurons.

The Leaky ReLU function:

$$\text{The Leaky ReLU function: Leaky ReLU(x)} = \begin{cases} x & \text{if x} > 0 \\ \delta x & \text{otherwise} \end{cases} \tag{4}$$

Leaky ReLU modifies the ReLU function by introducing a small slope δ (commonly set to 0.01) for negative input values, preventing the gradients from being zero. This helps mitigate the ”dying ReLU” problem by allowing small updates for negative inputs, ensuring that learning continues even if many values are negative.

2.2.2 Backpropagation

Backpropagation is used to optimize the weights $\mathbf{W}$ and biases $\mathbf{b}$ in the hidden layers and the output layer of a neural network. It begins with random initialization of these parameters, and then iteratively adjusts them to minimize the cost function. The process starts at the output layer L and propagates backward through each hidden layer l, updating the parameters based on the error from the subsequent layer l + 1. To optimize the parameters, the weights and biases are updated using the gradient of the cost function with respect to these parameters, scaled by the learning rate. The error for the output layer L is computed by differentiating the cost function C with respect to the output z_L [7]:

$$\delta^L = \frac{\partial C}{\partial z} = \frac{\partial a}{\partial z}\frac{\partial C}{\partial a} = f^{'}(z^L)\frac{\partial C}{\partial a}, \tag{5}$$

Here, f'(z^L) is the derivative of the activation function for the output layer, and $\frac{\partial C}{\partial a}$ represents the derivative of the cost function with respect to the output activations. For the hidden layer l, the error is calculated using the error of the subsequent layer l + 1 and the weights W^{i+1}:

$$\delta^l = \delta^{l+1}(W^{l+1})^T \frac{\partial a^l}{\partial z^l} = \delta^{l+1}(W^{l+1})^T f'(z^l) \tag{6}$$

This expression propagates the error backward through the network, taking into account the derivative of the activation function for each layer. The optimal weights and biases are found using gradient descent, which updates the parameters iteratively. The gradients for the weights in layer l are given by:

$$\nabla W^l = (a^{l-1})^T \delta^l \tag{7}$$

Where a^{l-1} is the activation from the previous layer. For the first layer, we will have the design matrix as **a**. The gradient for updating the bias in layer l is:

$$\nabla b^l = \sum_{i=1}^{n_{\text{inputs}}} \delta_i^l \tag{8}$$

This process is repeated iteratively, updating the weights and biases until the changes in the parameters are small enough, indicating that an acceptable convergence threshold has been reached.

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) build upon the principles of feedforward neural networks, where neurons compute weighted sums of inputs followed by activation functions. Like standard neural networks, CNNs are trained using backpropagation and gradient descent, and employ loss functions (e.g., Softmax) to optimize performance [8].

The primary difference between CNNs and standard FFNNs lies in how they process input data. A fundamental characteristic of FFNNs is that they perform affine transformations on input data, which means that input data, regardless of its original form (e.g., images, sound clips, unordered collections of features), must be converted into a flattened vector format before any transformation is applied [9]. This flattening process unfortunately loses any spatial information present in the original data, which can be critical for accurate interpretation, particularly in the context of images. Furthermore, analyzing an image with a fully connected FFNN would require connecting each pixel individually to input neurons, leading to an impractical large number of parameters.

In contrast, CNNs are specifically designed for grid-like data structures, such as images. They keep the original spatial dimensions - width, height, and depth - of the input data. Instead of connecting every neuron in one layer to all neurons in the next, CNNs use localized connections. Each neuron in a layer is connected only to a small, specific region (called ”patches”) of the previous layer [9]. This localized connectivity is achieved through a process called convolution, where a sliding window (also known as a kernel or filter) scans the input data. Each kernel computes feature maps by identifying spatial patterns, such as edges or textures, in the input image [10]. In the case of time-series and signal data, like in the SHHS1 dataset, a one-dimensional CNN is used. This type of CNN uses the same mechanisms to detect localized temporal patterns, where instead of using the dimensions, width, height and depth, it uses one, the time axis in our case.

The underlying mathematics of CNNs is based on the convolution operation, which is a core concept in functional analysis. Convolution combines two functions to produce a new function that describes how one function modifies the other [8]. For a one-dimensional case, convolution is expressed as [8]:

$$y(t) = \int x(a)w(t-a)da \tag{9}$$

Here, x(a) is the input, and w(t-a) is the kernel or weight function. This integral can again be rewritten more compactly:

$$y(t) = (x * w)(t) \tag{10}$$

By using the convolution operation, CNNs can effectively retain the spatial information in the input data, allowing the network to recognize and capture important patterns and features crucial for tasks like image classification, which would be impractical with traditional fully connected neural networks.

3 Methods

The code we developed can be found in the Github repository https://github.com/TrondVQ/FYS_STK_P3. See README for information on how to reproduce the results in the report. ChatGPT (December 2024 version, model GPT4o) by OpenAI was used to aid the implementation of the code, helping with error messages and implementation guidance [11]. It is noted in the code where ChatGPT was used to generate/aid the code.

3.1 Dataset - Sleep Heart Health Study

The Sleep Heart Health Study is a multi-center cohort study conducted by the National Heart Lung & Blood Institute of the US examining the cardiovascular and other consequences of sleep-disordered breathing. The study investigates whether sleep-related breathing is associated with an increased risk of coronary heart disease, stroke, all cause mortality, and hypertension [3].

The study contains five datasets, but as our focus is sleep apnea classification, we focus on the datasets with polysomnogram data, which are the shhs1 and shhs2 datasets.

- The SHHS1 dataset contains polysomnogram data from ”the baseline and first follow-up visits, collected on 6,441 individuals between 1995 and 1998” [12].
- The SHHS2 dataset which contains polysomnogram data from a follow-up visit, with data from 3,295 of the participants in the period January 2001–June 2003

The data was collected through a EEG-based polysomnography at home using the Compumedics Sleep Watch polysomnograph. 12 channels were collected: Oximetry, Heart Rate, Chest wall and abdomen movement, Nasal/oral airflow, Body position, Electroencephalogram (EEG) (2 central; one for redundancy in case of failure/loss), Electrooculogram (EOG) (bilateral) Electromyogram - chin (EMG), Electrocardiogram (ECG) [13].

Each polysomnography recording consists of a signal file in European Data Format (.EDF) and two versions of event scoring and epoch staging annotations in XML format: NSSR and Profusion [14].

- EDF data – Signal data files in the European Data Format
- Annotation data(XML) – Contains event annotations in XML format.

3.2 Preprocessing of the SHHS1 dataset

In this project, we will use only the SHHS1 dataset as our primary data source because it provides a comprehensive baseline and avoids the additional complexity that comes with processing multiple datasets. Furthermore, the annotation version we chose was the NSRR XML files instead of the Profusion, due to their simplicity. In addition, as the annotation data only includes the channels SaO2, EMG, NEW AIR/AIRFLOW, and ABDO RES, we will extract only these channels from the EDF data. Furthermore, as we only look at sleep apnea classification, we chose to not process the sleep stages annotations also included in the annotation data.

3.2.1 Different sampling rates

The EDF signal data contains several channels with different sampling rates; for example, the SaO2 channel has a sampling rate of 1 Hz, while the EMG channel has a sampling rate of 125 Hz. Consequently, the number of data points differs across the channels. To use these channels together for machine learning, we need to resample the data so that all channels have the same sampling rate. We have chosen to downsample all channels to 1 Hz, as this is the lowest sampling rate and SaO2 is the most relevant channel for sleep apnea classification. In addition, while conducted on a different dataset, work by Wang et al. [1] has shown that even considerably downsampled signals (down to 1 Hz) can retain essential characteristics for accurate apnea prediction.

```python
#  Resamples all signals to a target sampling rate.
# Asked ChatGPT for an initial version of this. Has since tuned it.
def resample_signals(signals, sampling_rates, target_rate=1):
    resampled_signals = {}
    for channel, signal in signals.items():
        original_rate = sampling_rates[channel]
        target_length = int(len(signal) * (target_rate / original_rate))
        resampled_signals[channel] = resample(signal, target_length)
    return resampled_signals

```

3.2.2 Combining EDF and Annotation data

To prepare the SHHS1 dataset for machine learning, we combine the EDF signal data with the corresponding XML annotation files to create a labeled dataset. We adopt a binary labeling approach, grouping both apnea and hypopnea events under a single ”apnea” label. This decision aligns with the Apnea-Hypopnea Index (AHI), which is commonly used to classify sleep apnea without distinguishing between these two event types [15].

$$\text{AHI} = \frac{\text{Total number of Apneas and Hypopneas}}{\text{Total Sleep Time (hours)}} \tag{11}$$

The EDF data is segmented into 30-second windows with a 15-second overlap. Although Li et al. which have conducted similar research found the best results using a 30-second moving window with a 29-second overlap for similar research, we opted for a 15-second overlap due to computational constraints [16]. Labels are assigned based on the events identified in the annotations:

- Positive Label - A window is labeled as positive if apnea or hypopnea events within it last at least 10 seconds [15].
- Negative Label - Otherwise, the window is labeled as negative.

Only the EDF channels specified in the annotation data (SaO_2, EMG, NEW AIR/AIRFLOW, and ABDO RES) are extracted for analysis.

```python
#The whole segment_and_label_edf_data function is shown in Github. This section shows the labeling process in combining the EDF data and XML annotations
...
label = 0  #0 = no apnea/hypopnea
 for _, event in xml_annotations_df.iterrows():
            #if annontation is annotated as hypopnea/apnea
            if event["event_concept"] in apnea_related_events:
                event_start = event["start"]
                event_end = event["start"] + event["duration"]

                # If there is 10 seconds overlap between the annontation and the window, label the segment as apnea/hypopnea
                overlap_start = max(segment_start, event_start)
                overlap_end = min(segment_end, event_end)
                overlap_duration = overlap_end - overlap_start

                if overlap_duration >= 10:
                    label = 1
                    break
...
```

3.2.3 Imbalance of classes

The SHHS1 dataset is inherently unbalanced, as the majority of EDF data will correspond to the negative class (normal breathing). This imbalance is a challenge as the machine learning models may become biased towards predicting the majority class and thereby have a poor performance detecting apnea events. To address this problem, we tested common methods like Random Oversampling, Random Undersampling and SMOTE on the baseline models to assess which one performed the best. In conclusion, SMOTE proved to have the best performance on the baseline models, which therefore was our balancing method when producing a balanced dataset to tune the models on. The tests on the baseline models can be found in Appendix I.

3.2.4 Standardization of data

For Neural Network algorithms, specifically FFNN and CNN, we standardized the data using manual calculations. Neural networks are sensitive to the scale of input features because they rely on weighted sums and distance-based calculations. Standardization ensures that each feature contributes equally to the model's learning process. It was later discovered that scikit-learn's Standardscaler was compatible with the dataset after changing the preprocessing process, but re-training the models would have been too time-consuming, to make this change.

In contrast, algorithms based on decision trees, such as Random Forests, do not require feature standardization because they split data based on feature thresholds and are thus insensitive to the scale of input features[17]. However, to ensure a fair comparison across all models, we applied standardization to the data used for Random Forests as well. This standardization does not negatively impact the Random Forest algorithm's performance.

```python
#standardization
mean = X_train.mean(axis=0)
std = X_train.std(axis=0)

X_train = (X_train - mean) / std
X_test = (X_test - mean) / std
```

3.3 Implementation of Machine learning algorithms in Python

For each algorithm, we created two models: one trained on the unbalanced dataset and another trained on the balanced dataset. Both models were subsequently evaluated on the unbalanced test set. This approach allows us to assess the impact of balancing the training data before the model encounters an unbalanced ”real-world scenario” in the test set.

3.3.1 Random forest

We implemented the Random Forest classifier using the Scikit-Learn library.

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(random_state=seed)
```

3.3.2 Feed-forward Neural Network

The Feed-Forward Neural Network was implemented using the Scikit-learn library, MLPClassifier. The reason for not using the custom FFNN from Project 2 is due to its incompatibility with RandomSearch, requiring several changes.

```
from sklearn.neural_network import MLPClassifier
nn =MLPClassifier(random_state=seed)
```

3.3.3 Convolutional Neural Network

For the Convolutional Neural Network, we utilized TensorFlow’s Keras API. Here we used a CNN architecture by Morten Hjorth-Jensen as a starting point[8], where we have adjusted it to the dataset, which is a 1D time-series data, used for binary classification, and added L2 regularization like with FFNN. Here we use binary_crossentropy and AUC for loss optimization as we have an imbalanced dataset and binary classification.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense,
    Input
from tensorflow.keras.regularizers import l2

def create_cnn(time_steps, num_features):
    model = Sequential()
    model.add(Conv1D(filters=32, kernel_size=3, activation='relu',
     ↪  input_shape=(time_steps, num_features),
     ↪  kernel_regularizer=l2(0.0001)))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Conv1D(filters=64, kernel_size=3, activation='relu'),
     ↪  kernel_regularizer=l2(0.0001))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Conv1D(filters=64, kernel_size=3, activation='relu'),
     ↪  kernel_regularizer=l2(0.0001))
    model.add(MaxPooling1D(pool_size=2))
    #  Last fully connected layer
    model.add(Flatten())
    # Binary classification  so one output and sigmoid
    model.add(Dense(units=1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy',
     ↪  metrics=['AUC'])
    return model
```

3.3.4 Hyperparameter tuning

For hyperparameter tuning of the different algorithms, we chose to use Random Search from the Scikit-learn library. Random Search is a more computationally efficient method compared to for example Grid Search, which exhaustively evaluates all possible combinations of hyperparameters. Instead, Random Search evaluates a predefined number of randomly selected combinations. This is approach is beneficial in models like Random Forests, FFNN, and CNN, which all have large hyperparameter spaces. For both FFNN and CNN, we included L2 regularization for both to ensure a fair comparison. Random forest inherently performs regularization as it is an ensemble of decision trees, providing the average.

Due to the imbalance in the dataset, we used *ROC_AUC* metric as the scoring metric for the random search optimization. s. Additionally, we employed StratifiedKFold for cross-validation instead of standard cross-validation to ensure that each fold maintains the same class distribution as the original dataset. Standard cross-validation might result in folds with uneven class distributions.

An example of its implementation through FFNN. Implementation is the same for CNN and Random forest:

```
from sklearn.model_selection import RandomizedSearchCV, StratifiedKFold
```

```
#The params grid to test. The example below is for FFNN
param_grid = {
    'hidden_layer_sizes': [(50,), (100,), (150,), (200,), (50, 50), (100,
     ↪  100)],
    'activation': ['relu', 'logistic', 'tanh'],
    'learning_rate_init': uniform(0.0001, 0.01),
     ...
}
random_search_smote = RandomizedSearchCV(
    estimator= FFNN_model
    param_distributions=param_grid,
    n_iter=100,
    scoring='roc_auc',  # AUC for imbalanced data
    cv=StratifiedKFold(n_splits=3, shuffle=True, random_state = seed),
    verbose=2,
    random_state = seed,
    n_jobs=-1
)

random_search.fit(X_train, y_train)

print("Best Hyperparameters for unbalanced dataset:",
 ↪  random_search_unbalanced.best_params_)
FFNN_best_trained_model random_search_unbalanced.best_estimator_
```

3.3.5 Evaluation method for the models

For the evaluation, we chose to use several metrics in addition to accuracy due to the class imbalance in the SHHS1 dataset. The accuracy metric only measures the correctly classified instances out of the total. In an imbalanced dataset, where, for example, the ratio is 90:10, a model may achieve 90% accuracy by solely predicting the majority class. To address this, we implemented an evaluation function using `classification_report` from scikit-learn. This function includes the evaluation metrics precision, recall, and F1-score for each class. Additionally, we included the ROC AUC metric.

- Precision - Measures the percentage of correctly predicted positive cases out of the predicted positive cases for a class

- Recall - Measures the percentage of actual positive cases in the dataset that were correctly identified by the model.
- F1 - ”The weighted harmonic mean of precision and recall. The closer to 1, the better the model.”[18]. This metric accounts for both false positives and false negatives.
- ROC AUC - The Area Under the Receiver Operating Characteristic curve is a metric that evaluates ”the performance of a binary classification model at various classification thresholds. It is commonly used in machine learning to assess the ability of a model to distinguish between two classes[19].

In the evaluation function, we applied a threshold of 0.5 to the predicted probabilities to convert them into binary predictions. This threshold was chosen as it is the standard default; however, it may not be optimal, especially in cases of class imbalance. Further tuning could improve the trade-off between precision and recall. The ROC AUC metric does not require a predefined threshold, as it measures overall model performance across all thresholds and is useful in the initial phase of evaluation. In contrast, the metrics provided by the `classification_report` are specific to the chosen threshold.

Below is the implementation of the evaluation function:

```python
def evaluate_model(model,  X_test, y_test, model_type = "rf", dataset_balance = "balanced" ):
    if(model_type == "rf"):      #if random_forest
        y_pred_probs = model.predict_proba(X_test)[:, 1]
    else:
        y_pred_probs = model.predict(X_test)
    #threshold for binary prediction
    y_pred = (y_pred_probs > 0.5).astype(int)

    print(classification_report(y_test, y_pred,  target_names=["Normal", "Apnea/Hypopnea"]))

    r_auc_score = ROC_auc_score(y_test, y_pred_probs)
    print(f"ROC AUC Score: {r_auc_score:.4f}")

```

4 Results

In this section, we analyze the performance of the Random Forest, FFNN, and CNN models trained on both the unbalanced and balanced training sets. We report performance metrics such as accuracy, precision, recall, F1-score, and ROC AUC, and we compare baseline (default parameters) performance with performance after hyperparameter tuning. Here we only show the baseline performance of the unbalanced dataset and balancing using SMOTE. Full outputs can be found in Appendix I. In addition, the grids used for hyperparameter tuning can be found in Appendix II.

4.1 Random Forest

We first implemented Random forest using the Scikit-learn library. We ran the algorithm without any hyperparameter tuning to assess its baseline performance on the dataset, shown in table 4.1. The hyperparameters we will explore later are the hyperparameters: number of estimators, max depth, *min_samples_split*, *min_samples_leaf*. The default values for these are: *n_estimators=100, max_depth=None, min_samples_split=2, min_samples_leaf=1* [20].

Subsequently, we applied Random Search from the Scikit-learn library to optimize the hyperparameters. The best hyperparameters identified for the two scenarios were:

- Balanced dataset using SMOTE: {'n_estimators': 500, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 0.5, 'max_depth': 50}
- Unbalanced dataset: {'n_estimators': 300, 'min_samples_split': 10, 'min_samples_leaf': 8, 'max_features': 0.5, 'max_depth': 20}

Dataset	Class	Precision	Recall	F1	Accuracy	ROC AUC
Unbalanced (Default)	Normal	0.81	0.94	0.87	0.78	0.7416
	Apnea/Hypopnea	0.55	0.25	0.35		
Unbalanced (Hypertuned)	Normal	0.80	0.97	0.88	0.79	0.7565
	Apnea/Hypopnea	0.62	0.19	0.29		
Balanced (Default)	Normal	0.84	0.80	0.82	0.73	0.7373
	Apnea/Hypopnea	0.42	0.50	0.46		
Balanced (Hypertuned)	Normal	0.84	0.80	0.82	0.73	0.7397
	Apnea/Hypopnea	0.42	0.50	0.46		

Table 4.1: Baseline and hypertuned classification performance of the Random Forest model for Unbalanced and Balanced (SMOTE) datasets.

For the unbalanced dataset with default parameters, the *Normal* class achieved high recall (0.94) and F1-score (0.87), while the *Apnea/Hypopnea* class showed lower recall (0.25) and F1-score (0.35). The overall accuracy was 0.78, with an ROC AUC score of 0.7416. After hyperparameter tuning on the unbalanced dataset, the *Normal* class maintained high recall (0.97) and F1-score (0.88), whereas the *Apnea/Hypopnea* class had improved precision (0.62) but lower recall (0.19) and F1-score (0.29). The overall accuracy increased slightly to 0.79, with an ROC AUC score of 0.7565.

For the balanced dataset with default parameters, the *Normal* class had balanced precision (0.84) and recall (0.80), resulting in an F1-score of 0.82. The *Apnea/Hypopnea* class achieved a precision of 0.42, recall of 0.50, and F1-score of 0.46. The overall accuracy was 0.73, with an ROC AUC score of 0.7373. After hyperparameter tuning on the balanced dataset, the overall accuracy and ROC AUC score showed slight improvements to 0.7397, while class-level performance metrics remained consistent with the default case.

4.2 Feed-Forward Neural Network (FFNN)

We then moved on the FFNN where we applied the same methodology as with Random forest, with showing baseline results and results after hyperparameter tuning. The default hyperparameters for the MLPClassifier we included in our search were *hidden˙layer˙sizes=(100,), activation=’relu’, alpha=0.0001, batch˙size=’auto’, learning˙rate˙init=0.001, max˙iter=200, hidden˙layer˙sizes = 100*[20].

The best hyperparameters identified after Random search for the two datasets were:

- Balanced dataset using SMOTE: ’activation’: ’logistic’, ’alpha’: np.float64(0.00014094462751221385), ’batch_size’: 128, ’hidden_layer_sizes’: (100, 100), ’learning_rate_init’: np.float64(0.008152416875301138), ’max_iter’: 421
- Unbalanced dataset: ’activation’: ’logistic’, ’alpha’: np.float64(0.00014094462751221385), ’batch_size’: 128, ’hidden_layer_sizes’: (100, 100), ’learning_rate_init’: np.float64(0.008152416875301138), ’max_iter’: 421

The table 4.2 presents the baseline and tuned performance on the test set for FFNN:

Dataset	Class	Precision	Recall	F1	Accuracy	ROC AUC
Unbalanced (Default)	Normal	0.77	1.00	0.87	0.77	0.6334
	Apnea/Hypopnea	0.69	0.01	0.03		
Unbalanced (Hypertuned)	Normal	0.79	0.97	0.87	0.78	0.5505
	Apnea/Hypopnea	0.60	0.13	0.21		
Balanced (Default)	Normal	0.85	0.41	0.55	0.49	0.6321
	Apnea/Hypopnea	0.27	0.75	0.40		
Balanced (Hypertuned)	Normal	0.86	0.57	0.69	0.60	0.6335
	Apnea/Hypopnea	0.32	0.69	0.44		

Table 4.2: Baseline and hypertuned classification performance of the Feed-forward neural network model for Unbalanced and Balanced (SMOTE) datasets.

For the unbalanced dataset with default parameters, the “Normal” class achieved high Precision (0.77), Recall (1.00), and F1-Score (0.87), while the “Apnea/Hypopnea” class showed lower Precision (0.69), Recall (0.01), and F1-Score (0.03). The overall accuracy was 0.77, with an ROC AUC score of 0.6334.

After hyperparameter tuning on the unbalanced dataset, the “Normal” class maintained high Precision (0.79), Recall (0.97), and F1-Score (0.87). The “Apnea/Hypopnea” class showed Precision (0.60) but had a Recall of 0.13 and F1-Score of 0.21. The overall accuracy increased slightly to 0.78, with an ROC AUC score of 0.5505.

For the balanced dataset with default parameters, the “Normal” class had balanced Precision (0.85) and Recall (0.41), resulting in an F1-Score of 0.55. The “Apnea/Hypopnea” class achieved a Precision of 0.27, Recall of 0.75, and F1-Score of 0.40. The overall accuracy was 0.49, with an ROC AUC score of 0.6321.

After hyperparameter tuning on the balanced dataset, the “Normal” class improved to Precision (0.86), Recall (0.57), and F1-Score (0.69). The “Apnea/Hypopnea” class achieved a Precision of 0.32, Recall of 0.69, and F1-Score of 0.44. The overall accuracy increased to 0.60, with an ROC AUC score of 0.6335.

4.3 Convolutional Neural Network (CNN)

Like Random Forest and FFNN, we applied the same methodology to the CNN, here the hyperparameter grid was smaller due to long processing time.

The default hyperparameters for the MLPClassifier we included in our search were *’alpha’:*

np.float64(0.0001), 'batch_size': 32, 'epochs': 20, 'filters1': 32, 'filters2': 64, 'kernel_size': 3, 'pool_size': 2.

The best hyperparameters identified after Random search for the two datasets were:

- Balanced dataset using SMOTE: 'alpha': np.float64(0.0017886930756498543), 'batch_size': 512, 'epochs': 41, 'filters1': 128, 'filters2': 128, 'kernel_size': 3, 'pool_size': 2
- Unbalanced dataset: 'alpha': np.float64(0.0017886930756498543), 'batch_size': 512, 'epochs': 41, 'filters1': 128, 'filters2': 128, 'kernel_size': 3, 'pool_size': 2

Table 4.3 presents the baseline and tuned performance on the test set for CNN:

Dataset	Class	Precision	Recall	F1	Accuracy	ROC AUC
Unbalanced (Default)	Normal	0.77	1.00	0.87	0.77	0.5356
	Apnea/Hypopnea	0.00	0.00	0.00		
Unbalanced (Hypertuned)	Normal	0.77	1.00	0.87	0.77	0.5369
	Apnea/Hypopnea	0.65	0.01	0.02		
Balanced (Default)	Normal	0.77	0.84	0.80	0.69	0.5223
	Apnea/Hypopnea	0.24	0.17	0.20		
Balanced (Hypertuned)	Normal	0.78	0.59	0.68	0.56	0.5250
	Apnea/Hypopnea	0.24	0.44	0.31		

Table 4.3: Baseline and hypertuned classification performance of the Convolutional neural network model for Unbalanced and Balanced (SMOTE) datasets.

For the unbalanced dataset with default parameters, the "Normal" class achieved high Precision (0.77), Recall (1.00), and F1-Score (0.87), while the "Apnea/Hypopnea" class showed 0.00 for Precision, Recall and F1-Score. The overall accuracy was 0.77, with an ROC AUC score of 0.5356.

After hyperparameter tuning on the unbalanced dataset, the "Normal" class maintained high Precision (0.77), Recall (1.00), and F1-Score (0.87). The "Apnea/Hypopnea" class showed improved Precision (0.65), Recall of 0.01 and F1-Score of 0.02. The overall accuracy was 0.77, with an improved ROC AUC score of 0.5369.

For the balanced dataset with default parameters, the "Normal" class had balanced Precision (0.77) and Recall (0.84), resulting in an F1-Score of 0.80. The "Apnea/Hypopnea" class achieved a Precision of 0.24, Recall of 0.17, and F1-Score of 0.20. The overall accuracy was 0.69, with an ROC AUC score of 0.5223.

After hyperparameter tuning on the balanced dataset, the "Normal" class improved to Precision (0.78), Recall (0.59), and F1-Score (0.68). The "Apnea/Hypopnea" class achieved a Precision of 0.24, Recall of 0.44, and F1-Score of 0.31. The overall accuracy was 0.56, with an improved ROC AUC score of 0.5250.

4.4 Model Comparison

Table 4.4 summarizes the performance obtained by each model after hyperparameter tuning on the test set.

Model and dataset	Class	Precision	Recall	F1	Accuracy	ROC AUC
Random forest (Unbalanced)	Normal	0.80	0.97	0.88	0.79	0.7565
	Apnea/Hypopnea	0.62	0.19	0.29		
FFNN (Unbalanced)	Normal	0.79	0.97	0.87	0.78	0.5505
	Apnea/Hypopnea	0.60	0.13	0.21		
CNN (Unbalanced)	Normal	0.77	1.00	0.87	0.77	0.5369
	Apnea/Hypopnea	0.65	0.01	0.02		
Random forest (Balanced)	Normal	0.84	0.80	0.82	0.73	0.7397
	Apnea/Hypopnea	0.42	0.50	0.46		
FFNN (Balanced)	Normal	0.86	0.57	0.69	0.60	0.6335
	Apnea/Hypopnea	0.32	0.69	0.44		
CNN (Balanced)	Normal	0.78	0.59	0.68	0.56	0.5250
	Apnea/Hypopnea	0.24	0.44	0.31		

Table 4.4: Comparison of the best hyperparameter-tuned results across all models on the balanced and unbalanced dataset.

5 Discussion

In this section, we evaluate the performance of Random Forest, FFNN and CNN on the SHHS1 dataset. We assess the strengths and weaknesses of the models, and relate the results to existing literature.

5.1 Models performance on the SHHS1 dataset

5.1.1 Random forest

On the unbalanced dataset, the Random Forest using default parameters achieved an accuracy of 0.78, with a precision of 0.81, recall of 0.94, and F1-score of 0.87 for the majority class (*Normal breathing*). However, the performance on the minority class (*Apnea/Hypopnea*) was significantly lower, with a precision of 0.55, recall of 0.25, and F1-score of 0.35. These results show that while Random Forest effectively identifies *Normal* cases, it struggles to detect *Apnea/Hypopnea* events in the unbalanced dataset. This behavior is expected, as the model becomes biased toward the majority class due to the inherent class imbalance.

On the balanced dataset, Random Forest using default parameters demonstrated significant improvements in detecting the minority class. The recall for *Apnea/Hypopnea* improved from 0.25 to 0.50, and the F1-score increased from 0.35 to 0.46. However, the balancing also came at the cost of precision for the minority class dropping from 0.55 to 0.42 and slightly lower recall for the majority class decreasing from 0.94 to 0.80. These changes reflect the effects of the SMOTE balancing algorithm, which increased the model's sensitivity to the minority class but slightly compromised its performance on the majority class. This trade-off shows the challenges in optimizing performance across imbalanced datasets.

Another observation is that hyperparameter tuning showed minimal improvements for the Random Forest algorithm, both on the balanced and unbalanced datasets. This limited impact could be due to the chosen hyperparameter search space or constraints within the dataset itself. Further exploration of the hyperparameter space or alternative data preprocessing strategies might show better results. Additionally, this may indicate that the Random Forest algorithm has limitations in addressing the specific characteristics of this dataset and problem.

5.1.2 Feed Forward Neural Network

On the unbalanced dataset, the FFNN with default parameters struggled to classify the minority class (Apnea/Hypopnea), demonstrating extremely low recall (0.01) and F1-score (0.03), although it achieved relatively high precision for the minority class (0.69). This suggests that while the model is occasionally able to correctly identify minority cases, it overwhelmingly favors the majority class (Normal), resulting in a lack of sensitivity to minority instances. Similar to the Random Forest model, the FFNN effectively identified Normal cases but performed poorly at detecting Apnea/Hypopnea events, highlighting the challenges of imbalanced datasets.

On the balanced dataset, the FFNN exhibited significant improvements in detecting the minority class, both pre- and post-hyperparameter tuning. Recall increased dramatically from 0.01/0.13 to 0.75/0.69, and F1-score rose from 0.03/0.21 to 0.40/0.44. These results indicate that balancing the dataset effectively enhanced the model's ability to recognize Apnea/Hypopnea events. However, this improvement came at the expense of performance on the majority class (Normal), with significantly reduced recall and F1-scores for this class. This trade-off is indicative of the balancing algorithm's influence, where enhancing minority class detection can inadvertently reduce sensitivity to the majority class.

The effects of hyperparameter tuning on the FFNN model varied across datasets. For the unbalanced dataset, tuning yielded modest improvements in the minority class detection, with recall increasing from 0.01 to 0.13 and F1-score rising from 0.03 to 0.21. On the balanced dataset, however, tuning resulted in only marginal gains and some performance trade-offs, indicating limited effectiveness. This limited impact could also be due to the chosen hyperparameter search space or constraints within the dataset itself. Further exploration of the hyperparameter space or alternative data preprocessing strategies might show better results.

5.1.3 Convolutional neural network

The CNN model demonstrated limited effectiveness on both the balanced and unbalanced datasets. While its performance on the Normal class was comparable to other models, it struggled significantly in detecting Apnea/Hypopnea cases, particularly on the unbalanced dataset, where it failed to detect any instances. This is as expected, however, no predictions do indicate that CNN struggled to learn meaningful representations of the minority class.

On the balanced dataset, the CNN showed improved detection of the minority class, but its overall performance remained below expectations, even after hyperparameter tuning. Although tuning provided some enhancements, the gains were insufficient to make CNN a strong contender for this task. Studies such as Wang et al. have shown that CNNs can achieve excellent performance in time-series classification tasks, particularly for medical data [1]. Likewise, as with Random forest and FFNN, this limited impact could also be due to the chosen hyperparameter search space or constraints within the dataset itself. Further exploration of the hyperparameter space or alternative data preprocessing strategies might show better results.

5.2 Challenges and constraints

Random Forest performed the best out of the models in handling both the balanced and unbalanced dataset (table 4.4), particularly with higher recall and F1-scores for the minority class (Apnea/Hypopnea). However, despite efforts in balancing and optimization, the overall model performance remained modest, particularly for detecting the minority class. These results suggest that while neural networks hold promise, their effectiveness depends heavily on factors such as dataset size, architectural design, and extensive hyperparameter tuning. While Wang et al. found great result with CNNs for classifying sleep apnea using time series data [1], in this study, limited access to high-performance computing resources constrained our ability to fully explore deeper architectures and more exhaustive optimization strategies. With our resources, the hyperparameter tuning process for CNN took 4 hours for just one dataset, which often had to be done several times due to bug fixes and changes. As a result, tree-based models like Random Forest provided more consistent results under the given constraints.

Additionally, working with the SHHS1 dataset presented challenges, including its imbalanced nature and the need for domain-specific preprocessing techniques, like the preprocessing of EDF signal preprocessing the annotation data. Moreover, limited familiarity with sleep study data, computational restrictions, combined with the scope of this project, restricted our ability to preprocessing the data and the models' performance on it. We extend our gratitude to Marta Quemada Lopez, a Doctoral Research Fellow from the Institute of Informatics at the University of Oslo, for her guidance on preprocessing and labeling the SHHS1 dataset.

6 Conclusion

This study evaluated the performance of three machine learning models—Random Forest, Feedforward Neural Networks (FFNN), and Convolutional Neural Networks (CNN)—for classifying sleep apnea events using the SHHS1 dataset. Our findings underscore the critical importance of addressing class imbalance and optimizing model hyperparameters to improve classification performance, especially for the minority class.

The use of SMOTE to balance the dataset substantially improved minority class metrics across all models. Balancing the dataset before training consistently enhanced recall and F1-scores across all models, highlighting the value of preprocessing when addressing class imbalance. Among the tested models, Random Forest showed the best performance for detecting apnea/hypopnea, achieving an F1-score of 0.46 on the balanced dataset. While CNNs have shown great performance for time-series classification [1], their performance in this study was limited. This limitation is likely due to computational constraints, insufficient hyperparameter tuning, and the inherent complexity of the dataset. This constraint also applies to both Random Forest and FFNN. In addition, working with the SHHS1 dataset presented challenges, including its imbalanced nature and the need for properly tested preprocessing techniques.

To address these challenges and improve results, future work should focus on:

- Exploring advanced architectures, such as hybrid CNN-LSTM models, to better capture spatial and temporal patterns in sleep apnea classification.
- Conducting more comprehensive hyperparameter tuning and leveraging advanced class imbalance handling techniques, such as adaptive resampling or class-weighted loss functions.
- Investigating preprocessing parameters in greater depth, such as binary thresholds, downsampling strategies, sliding window sizes, and window overlaps, which were only partially assessed due to the scope of this project.

Bibliography

1. Wang E, Koprinska I and Jeffries B. Sleep Apnea Prediction Using Deep Learning. IEEE Journal of Biomedical and Health Informatics 2023; 27:5644–54. DOI: 10.1109/JBHI.2023.3305980
2. Apple. Sleep$_{A}pnea_{N}otifications_{O}n_{A}pple_{W}atch_{S}eptember_{2}$024. Estimating Breathing Disturbances and Sleep Apnea Risk from Apple Watch 2024 Sep
3. NSRR. About the Study. Available from: https://sleepdata.org/datasets/shhs/pages/full-description.md
4. James G, Witten D, Hastie T and Tibshirani R. An Introduction to Statistical Learning: With applications in R. p. 311-316, 319-321. New York: Springer, 2017
5. Raschka S, Liu Y and Mirjalili V. Machine Learning with PyTorch and Scikit-Learn. p. 95-98. Birmingham: Packt Publishing, 2022
6. Goodfellow I, Bengio Y and Courville A. Deep Learning. p. 170-171. MIT Press, 2016
7. Hjorth-Jensen M. Week 42 Constructing a Neural Network code with examples. 2024. Available from: https://github.com/CompPhysics/MachineLearning/blob/master/doc/LectureNotes/week42.ipynb
8. Hjorth-Jensen M. Week 44, Convolutional Neural Networks (CNN). 2024. Available from: https://github.com/CompPhysics/MachineLearning/blob/master/doc/LectureNotes/week44.ipynb
9. Hjorth-Jensen M. Week 45, Convolutional Neural Networks (CCNs) and Recurrent Neural Networks (RNNs). 2024. Available from: https://github.com/CompPhysics/MachineLearning/blob/master/doc/LectureNotes/week45.ipynb
10. Amini A. MIT 6.S191 (2020): Convolutional Neural Networks. At 10:35 in video. 2020. Available from: https://www.youtube.com/watch?v=iaSUYvmCekI
11. OpenAI. ChatGPT: An AI Language Model (December 2024 version). Retrieved from https://www.openai.com. https://www.openai.com. 2024
12. NSRR. SHHS dataset introduction. Available from: https://sleepdata.org/datasets/shhs/pages/04-dataset-introduction.md
13. NSRR. SHHS PROTOCOL. p. 25-26. Available from: https://sleepdata.org/datasets/shhs/files/m/browser/documentation/SHHS1_Protocol.pdf?inline=1
14. NSRR. SHHS dataset Polysomnography introduction. Available from: https://sleepdata.org/datasets/shhs/pages/05-polysomnography-introduction.md

15. Medicine JH. Obstructive sleep apnea. 2024 Jul. Available from: https://www.hopkinsmedicine.org/health/conditions-and-diseases/obstructive-sleep-apnea

16. Li MH, Yadollahi A and Taati B. Noncontact Vision-Based Cardiopulmonary Monitoring in Different Sleeping Positions. IEEE Journal of Biomedical and Health Informatics 2017; 21:1367–75. DOI: 10.1109/JBHI.2016.2567298

17. Arya N. Does the Random Forest Algorithm Need Normalization? 2022 Jul. Available from: https://www.kdnuggets.com/2022/07/random-forest-algorithm-need-normalization.html

18. Bobbitt Z. How to Interpret the Classification Report in sklearn (With Example). 2022 May. Available from: https://www.statology.org/sklearn-classification-report/

19. GeeksforGeeks. AUC ROC Curve in Machine Learning. GeeksforGeeks 2020 Nov. Available from: https://www.geeksforgeeks.org/auc-roc-curve/

20. developers scikit-learn. RandomForestClassifier. 2024. Available from: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Appendix

I Appendix

A Random forest full output

A.1 Random forest baseline output including random oversampling and undersampling

```
Evaluation for Balanced_oversampling dataset and model rf:
Classification Report:
                precision    recall  f1-score   support

        Normal       0.82      0.89      0.86     23916
Apnea/Hypopnea       0.49      0.36      0.41      7074

      accuracy                           0.77     30990
     macro avg       0.66      0.62      0.63     30990
  weighted avg       0.75      0.77      0.75     30990


Evaluation for Balanced_Undersampling dataset and model rf:
Classification Report:
                precision    recall  f1-score   support

        Normal       0.87      0.66      0.75     23916
Apnea/Hypopnea       0.37      0.68      0.48      7074

      accuracy                           0.67     30990
     macro avg       0.62      0.67      0.62     30990
  weighted avg       0.76      0.67      0.69     30990

Evaluation for Unbalanced dataset and model rf:
Classification Report:
                precision    recall  f1-score   support

        Normal       0.81      0.94      0.87     23916
Apnea/Hypopnea       0.55      0.25      0.35      7074

```

```
      accuracy                           0.78     30990
     macro avg       0.68      0.60      0.61     30990
  weighted avg       0.75      0.78      0.75     30990

ROC AUC Score: 0.7416


Evaluation for Balanced_smote dataset and model rf:
Classification Report:
                precision    recall  f1-score   support

        Normal       0.84      0.80      0.82     23916
Apnea/Hypopnea       0.42      0.50      0.46      7074

      accuracy                           0.73     30990
     macro avg       0.63      0.65      0.64     30990
  weighted avg       0.75      0.73      0.74     30990


```

A.2 Random forest tuned output

```

Best Hyperparameters for balanced dataset using SMOTE: {'n_estimators': 500,
 ↪  'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 0.5,
 ↪  'max_depth': 50}

Evaluation for Hypertuned_smote dataset and model rf:
Classification Report:
                precision    recall  f1-score   support

        Normal       0.84      0.80      0.82     23916
Apnea/Hypopnea       0.42      0.50      0.46      7074

      accuracy                           0.73     30990
     macro avg       0.63      0.65      0.64     30990
  weighted avg       0.75      0.73      0.74     30990

```

```
ROC AUC Score: 0.7397



Best Hyperparameters for unbalanced dataset: {'n_estimators': 300,
 ↪  'min_samples_split': 10, 'min_samples_leaf': 8,
 'max_features': 0.5, 'max_depth': 20}

Evaluation for Hypertuned_Unbalanced dataset and model rf:
Classification Report:
                precision    recall  f1-score   support

        Normal       0.80      0.97      0.88     23916
Apnea/Hypopnea       0.62      0.19      0.29      7074

      accuracy                           0.79     30990
     macro avg       0.71      0.58      0.58     30990
  weighted avg       0.76      0.79      0.74     30990

ROC AUC Score: 0.7565

```

B FFNN full output

B.1 FFNN baseline output including random oversampling and undersampling

```
Evaluation for Balanced - OS dataset and model ffnn:
Classification Report:
                precision    recall  f1-score   support

        Normal       0.85      0.50      0.63     23916
Apnea/Hypopnea       0.29      0.69      0.41      7074

      accuracy                           0.54     30990
     macro avg       0.57      0.59      0.52     30990
  weighted avg       0.72      0.54      0.58     30990

ROC AUC Score: 0.6342
```

```
Evaluation for Balanced - US dataset and model ffnn:
Classification Report:
                precision    recall  f1-score   support

        Normal       0.83      0.56      0.67     23916
Apnea/Hypopnea       0.29      0.61      0.39      7074

      accuracy                           0.57     30990
     macro avg       0.56      0.59      0.53     30990
  weighted avg       0.71      0.57      0.61     30990

ROC AUC Score: 0.6268
Evaluation for Balanced - SMOTE dataset and model ffnn:
Classification Report:
                precision    recall  f1-score   support

        Normal       0.85      0.41      0.55     23916
Apnea/Hypopnea       0.27      0.75      0.40      7074

      accuracy                           0.49     30990
     macro avg       0.56      0.58      0.48     30990
  weighted avg       0.72      0.49      0.52     30990

ROC AUC Score: 0.6321
Evaluation for Unbalanced dataset and model ffnn:
Classification Report:
                precision    recall  f1-score   support

        Normal       0.77      1.00      0.87     23916
Apnea/Hypopnea       0.69      0.01      0.03      7074

      accuracy                           0.77     30990
     macro avg       0.73      0.51      0.45     30990
  weighted avg       0.76      0.77      0.68     30990

ROC AUC Score: 0.6334
```

B.2 FFNN tuned output

```
Best Hyperparameters for balanced dataset using SMOTE: {'activation': 'logistic', 'alpha': np.float64(0.00014094462751221385), 'batch_size': 128, 'hidden_layer_sizes': (100, 100), 'learning_rate_init': np.float64(0.008152416875301138), 'max_iter': 421}

Classification Report:
                precision    recall  f1-score   support

        Normal       0.86      0.57      0.69     23916
Apnea/Hypopnea       0.32      0.69      0.44      7074

      accuracy                           0.60     30990
     macro avg       0.59      0.63      0.57     30990
  weighted avg       0.74      0.60      0.63     30990

ROC AUC Score: 0.6335


Best Hyperparameters for unbalanced dataset:
Best Hyperparameters for unbalanced dataset: {'activation': 'logistic', 'alpha': np.float64(0.00014094462751221385), 'batch_size': 128, 'hidden_layer_sizes': (100,
100), 'learning_rate_init': np.float64(0.008152416875301138), 'max_iter': 421}
Evaluation for Unbalanced dataset and model ffnn:
Classification Report:
                precision    recall  f1-score   support

        Normal       0.79      0.97      0.87     23916
Apnea/Hypopnea       0.60      0.13      0.21      7074

      accuracy                           0.78     30990
     macro avg       0.69      0.55      0.54     30990
  weighted avg       . Also tes0.75      0.78      0.72     30990

ROC AUC Score: 0.5505
```

C CNN full output

C.1 CNN baseline output including random oversampling and undersampling

```
 _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/tvq/Library/Python/3⌋
 ↪  .9/lib/python/site-packages/sklearn/metrics/_classification.py:1531:
 ↪  UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in
 ↪  labels with no predicted samples. Use `zero_division` parameter to control
 ↪  this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
                precision    recall  f1-score   support

        Normal       0.77      1.00      0.87     23896
Apnea/Hypopnea       0.00      0.00      0.00      7065

      accuracy                           0.77     30961
     macro avg       0.39      0.50      0.44     30961
  weighted avg       0.60      0.77      0.67     30961


ROC AUC Score: 0.5356
Evaluation for Balanced - OS dataset and model CNN:
Classification Report:
                precision    recall  f1-score   support

        Normal       0.77      0.79      0.78     23896
Apnea/Hypopnea       0.23      0.21      0.22      7065

      accuracy                           0.66     30961
     macro avg       0.50      0.50      0.50     30961
  weighted avg       0.65      0.66      0.66     30961


ROC AUC Score: 0.5009
Evaluation for Balanced - US dataset and model CNN:

Classification Report:
                precision    recall  f1-score   support

        Normal       0.77      0.76      0.77     23896
Apnea/Hypopnea       0.23      0.25      0.24      7065

      accuracy                           0.64     30961
     macro avg       0.50      0.50      0.50     30961
  weighted avg       0.65      0.64      0.65     30961


ROC AUC Score: 0.5025
Evaluation for Balanced - SMOTE dataset and model CNN:

Classification Report:
                precision    recall  f1-score   support

        Normal       0.77      0.84      0.80     23896
Apnea/Hypopnea       0.24      0.17      0.20      7065

      accuracy                           0.69     30961
     macro avg       0.51      0.51      0.50     30961
  weighted avg       0.65      0.69      0.67     30961

ROC AUC Score: 0.5223

```

C.2 CNN tuned output

```


Best Hyperparameters for balanced dataset using SMOTE: {'alpha':
 ↪  np.float64(0.0017886930756498543), 'batch_size': 512, 'epochs': 41,
 ↪  'filters1': 128, 'filters2': 128, 'kernel_size': 3, 'pool_size': 2}

Classification Report:
                precision    recall  f1-score   support

        Normal       0.78      0.59      0.68     23896
Apnea/Hypopnea       0.24      0.44      0.31      7065
```

```

      accuracy                           0.56     30961
     macro avg       0.51      0.52      0.49     30961
  weighted avg       0.66      0.56      0.59     30961

ROC AUC Score: 0.5250



Best Hyperparameters for unbalanced dataset: {'alpha': np.float64(0.0017886930756498543), 'batch_size': 512, 'epochs': 41, 'filters1': 128, 'filters2': 128, 'kernel_size': 3, 'pool_size': 2}

Classification Report:
                precision    recall  f1-score   support

        Normal       0.77      1.00      0.87     23896
Apnea/Hypopnea       0.00      0.00      0.00      7065

      accuracy                           0.77     30961
     macro avg       0.39      0.50      0.44     30961
  weighted avg       0.60      0.77      0.67     30961

ROC AUC Score: 0.5369

```

II Appendix

A Random search grids

A.1 Random forest

```
param_grid = {
    'n_estimators': [50, 100, 200, 300, 500],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 4, 8],
    'max_features': ['sqrt', 'log2', 0.2, 0.5],
}
```

A.2 FeedForward Neural Network

```
param_grid = {
    'hidden_layer_sizes': [(50,), (100,), (150,), (200,), (50, 50), (100, 100)],
    'activation': ['relu', 'logistic', 'tanh'],
    'learning_rate_init': uniform(0.0001, 0.01),
    'alpha': uniform(0.0001, 0.01),
    'max_iter': randint(300, 1000), # we know from the baseline that it needs more than 200 to converge
    'batch_size': [16, 32, 64, 128]
}
```

A.3 Convolutional Neural Network

```
param_grid = {
    'filters1': [32, 64, 128],
    'filters2': [64, 128],
    'kernel_size': [3, 5, 7],
    'pool_size': [2],
    'epochs': randint(30, 50), #time constrains
    'alpha': uniform(0.0001, 0.01),
    'batch_size': [128, 256, 512] #batch size 16 took too long about 20 min per. Also based on FFNN, larger batch sized performed better
}
```