Machine Learning Project Report

Course title and number: Machine Learning, 10204350

Prepared by: Saja Abdulazeez, 20110060

Contents

I. Introduction:

Describe the problem you are addressing and why is it important?

The goal of this paper is to predict the number of human occupancy (number of people in a room) utilizing data collected from non-intrusive IoT sensors that can aid in optimizing energy consumption in an organization or any given space where an occupancy pattern is established.

This can automate lighting, heating and even air conditioning levels based on human occupancy, resulting in energy conservation especially for large organizations and enhancing the comfort of those in the space. If for example employees are in a rush to leave the office, they might forget the AC or the lights on through the weekend. In the long run this can save an organization cost and could be considered more eco-conscious.

Describe the dataset's source, collection method, attributes, size, and domain [1].

The data was collected through non-intrusive sensors including CO2, illumination (light), temperature, sound, and passive infrared (motion) sensors; previous studies utilized intrusive sensors such as cameras, Wi-Fi or wearable sensors that raised some privacy concerns.

- These non-intrusive sensors were setup in a 6m x 4.6m lab containing 4 office desks where a temperature, light, and sound sensor were setup at each desk (4 sensors each S1-S4), 1 CO2 sensor was set in the middle of the room (S5), and 2 passive infrared (PIR S6-S7) sensors were set up at the ceiling for maximised readings.
- The readings from the sensors were transmitted every 30 seconds to a master node (M) which append the received data from all the sensors into a file with the corresponding timestamp; this was applicable due to the Zigbee module which is a wireless communication protocol [2] developed with IEEE standards for communication between IoT devices and networks.
- The collected data transmissions weren’t 100% synchronized and had quite noticeable time variations (in seconds) so the time stamps were merged based on a common vector.
- A new feature named CO2 slope was derived through CO2 level sensor, because the readings took a while to stabilise additionally, it was a good indicator of human occupancy. CO2 slope was derived through calculating the slope of the line between 25 points through a linear regression model.

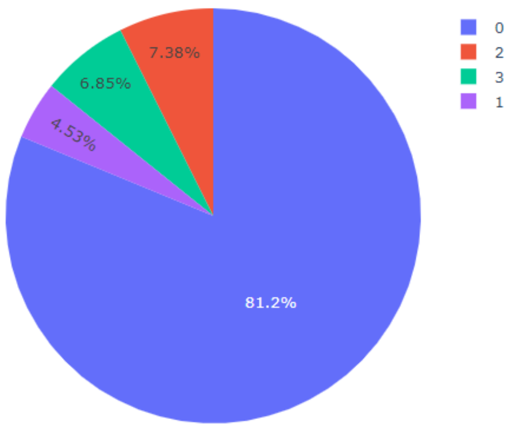
- Accurate ground truth of room occupancy count was collected manually through the employees who signed in / out when leaving or entering the lab.
- The final dataset contains 10,000 data points with 16 features relating to each sensor.

Describe the learning problem you are trying to solve.

Utilizing supervised machine learning classifiers and statistical techniques, can we predict the human occupancy level in a given space from time-series data collected through non-intrusive IoT sensors?

How did you prepare training and test data before implementing machine learning models?

Firstly, I checked the distribution of the labels, room occupancy count, the target variable. I noticed a great variation between labels 0, 1, 2 ,3, a clear imbalance so I chose to randomly under sample to the minority class '1' and to synthetically over sampled to the majority class '0' using the imbalanced learn library.



- The reason for this is so I can test the waters with a base model and see how differently they would perform; I ended up utilizing both since under sampling resulted in a low count, I would use that sample in GridSearchCV and for the over sampled data with SMOTE I would train 65% – test 35% split for model evaluation.

Secondly, I dropped the time and date columns as I felt they won't be of value to the model predictions, the repetition of the dates could confuse the models and take them more into account than the other 'more important' features.

From observing the data, I've noticed significant variation across the different sensors. To ensure fair results I employed the standard scaler to **scale the X features**. Additionally, it's worth mentioning that the sensor readings are in different units, temperature in degrees Celsius, light in LUX, sound in 0.0V, CO2 in 5ppm and motion was either 0 or 1. Typically, in multiclass classification there isn't a need to scale the response (y).

II. Methods:

Explain why the provided models are appropriate to solve this problem.

The provided models fall under supervised classification models and are especially useful for multiclass classification as per our problem, the machine learning models utilized include:

- **Support Vector Machines** are a good fit for complex, non-linearly separable data, and those with high dimensional spaces (features > samples) in small to medium datasets. The dataset provided is of medium size with around 32,912 instances (after SMOTE) and 16 features. However, SMVs are quite sensitive to the scale of the input features, hence scaling the data beforehand is essential.
    - From a holistic view, it aims to find the best hyperplane in the feature space that maximize the margin– creating the widest possible separation between different classes– essentially a well-defined decision boundary. Instances on the hyperplane (decision boundary) are called **support vectors.**
    - The larger the margin the more flexible the decision boundary – a more generalized model on new instances, the smaller the margin the stricter the penalty resulting in more misclassified points and a tendency to overfit. This is controlled by the regularization parameter C 'cost', the higher the C the stricter the misclassification penalty [3].
    - The interesting thing about SVMs is the **kernel trick**, where if the given data is not linearly separable, instead of manually transforming the data through mathematical operations (like squaring or logs) or even adding new features; the kernel trick can "*project the data to another dimension*" in a hyperparameter 'kernel' either by polynomial, RBF, sigmoid or linear functions.
    - SVMs are widely utilized across various classification use cases due to their ability to provide flexibility in defining decision boundaries, especially on complex patterns in the data. This allows SVMs to generalize well to new, unseen data.
- **Random Forests** are an ensemble learning technique that combines multiple decision tree classifiers with bagging – a resampling method that iteratively draws samples from the original data with replacement – this way each tree is trained on a slightly diverse subsample in parallel (independent of each other). This randomness prevents strong features from skewing the predictions solving the issue of overfitting decision trees [4]. The result is a majority vote, each tree's prediction is accounted for; the class (label) with the most votes is assigned to the prediction.

- They are suitable for real-life problems as they are less sensitive to outliers and noise in the data, they also provide a measure of feature importance when interpreting the model for feature selection [5]. This aids us in feature engineering – selecting the appropriate features for the best predictions.

Boosting is an ensemble method that utilizes a series of weak classifiers – often decision trees as a base model – trained sequentially (one after the other) to create a strong classifier. Unlike bagging there is no resampling of the data however, the sequential learning allows for a *'slow learning approach'* [6] where the next weak classifier is fit on the residuals of its predecessor rather than the Y labels as the dependant variable. The two boosting classifiers are used are mentioned below:

- **Gradient Boost (GB)** is a boosting classifier that utilizes gradient decent as it aims to fit the next classifier to the residual errors ($Y_{\text{original}} – Y_{\text{predicted}}$) made by its predecessor. This model has many hyperparameters such as learning rate, subsample, number of trees, depth of trees and others which requires precise hyperparameter tuning when finding the best fit for your data [7]. A disadvantage with GB is its tendency to overfit, unlike random forests, these week classifiers are learning sequentially – repeatedly training it on the same dataset may lead to the model memorizing specific instances within the data.
- **Extreme Gradient Boost** (XGBoost): a relatively new ML model released in 2014 that is very similar to gradient boost except it introduces training in parallel since gradient boost would take slower computing times and tended to overfit – XGB utilizes high level of optimization with regularization parameters for both L1 and L2 norms and the ability to label encode data [8] – converting categorical data to numerical. Containing many unique features, this open-source library is very flexible with a minimum of 12 hyperparameters [9] it encompasses, the process of tuning alone would be time-intensive even with grid search CV but the trade-off with result is worth it – depending on the use case.

Demonstrate how you will test the machine learning application using a range of test data and explain each stage of this activity (Apply k-fold cross-validation).

- To get the best hyperparameters according to the data across these ML models I have **applied Grid Search** cross validation with specific hyperparameter grids for each model; this cross-validation technique searches through the given parameter grid across defined K-folds, each fold the model is trained on a different combination of hyperparams and is evaluated based on a performance metric of my choosing which I set to accuracy score. Grid Search CV returns the best hyperparameters based on the highest mean accuracy score.
    - In the 'cv' parameter in grid search I chose the stratified k-fold method that I initialized with 10 folds for the SVM linear and RBF however, the random forest, gradient boost and XGBoost had many hyperparameters to go through so I opted for 5 folds.
- **Stratified K-fold** is a cross validation approach that partitions the data into *K* subsamples, the model is trained *K* times with each iteration a single different subsample is tested on. This way I ensure that I went through each sample at least once for a more homogenous generalization to the data to evaluate the learning method [10].
- This entire process can be considered **internal cross validation** [10]; both hyperparameter tuning and cross validation are used on the training data to evaluate the models in an automated manner rather than manually or explicitly coded through nested for loops.
- This entire process was done utilizing the randomly under sampled data as I believe provided a comprehensive selection of the whole data. It is worth noting that applying this approach on **1468** instances is far more convenient and time efficient than on **26329** instances. Additionally, I chose to print the wall and CPU times '%time' just to observe the difference across models.
- After the best hyperparameters were returned from the grid search:
    - I would initialize each model with the corresponding hyperparameters then fit the models on the oversamples training X's and Y's using '.fit'
    - Predict with the X_test and save the predictions in a variable 'predictions_model'

- With this variable I can get the values for the evaluation metrics and plot confusion matrices to get a deeper understanding on **how** well the model predicted.

Explain in detail the machine learning algorithms you are using to address this problem.

1. Support Vector Machine Classifier – RBF [11]:

Given the decision function of an SVM:

$$f(x(t)) = \sum_{i=1}^{M} \alpha_i^* y_i K(x_i^*, x(t)) + b^* \qquad (1)$$

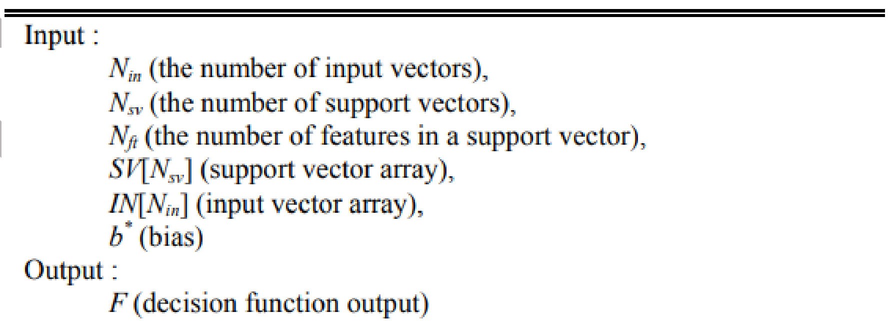
- x_i^* the i^{th} vector in M support vectors.
- y_i its corresponding class label.
- $x(t)$ tth input frame vector
- b^* optimization bias
- α^* Lagrange multiplier – for constrained optimization problems such as SVM to find optimal hyperplane [18].

And the radial basis function as a kernel RBF:

$$K(x_i^*, x(t)) = \exp(-\gamma \left\| x_i^* - x(t) \right\|^2) \qquad (2)$$

- Where γ is a hyperparameter controlling the 'shape' of decision boundary for non-linearly separable data.

Pseudocode for the decision function (1):



```
Input :
        $N_{in}$ (the number of input vectors),
        $N_{sv}$ (the number of support vectors),
        $N_{ft}$ (the number of features in a support vector),
        $SV[N_{sv}]$ (support vector array),
        $IN[N_{in}]$ (input vector array),
        $b^*$ (bias)
Output :
        $F$ (decision function output)
```

```
for $i \leftarrow 1$ to $N_{in}$ by 1 do
    $F = 0$
    for $j \leftarrow 1$ to $N_{sv}$ by 1 do
        $dist = 0$
        for $k \leftarrow 1$ to $N_{ft}$ by 1 do
            $dist += (SV[j].feature[k] - IN[i].feature[k])^2$
        end
        $\kappa = \exp(-\gamma \times dist)$
        $F += SV[j].\alpha^* \times \kappa$
    end
    $F = F + b^*$
end
```

Fig. 1. Pseudocode for SVM-based classifier.

1. Outer for loop iterates through input vectors *(i)* in the array IN[N_{in}] until N_{in}
2. Initialize $F = 0$ which will hold the value of the decision function output
3. Second loop iterates through support vectors *(j)* in array SV[N_{sv}] until N_{sv}
4. Set $dist = 0$ which corresponds to the value $||x_i^* - x(t)||^2$ in equation (2)
5. Inner loop iterates through each feature (inputs and support vectors) until N_{ft}
6. Compute the square distance between the k^{th} feature of *SV[j]* and *IN[i]* and accumulate in *dist*
7. Ends innermost loop iterating over k features
8. κ corresponds to the kernel value its function and translates $\mathbf{exp}(-\boldsymbol{\gamma}||\boldsymbol{x_i^*} - \boldsymbol{x}(\boldsymbol{t})||^2)$ into **exp(-$\boldsymbol{\gamma}$ ×*dist*)**
9. Decision function output updated by adding the product of alpha α^* of the support vector *j* multiplied by the kernel value
10. End, stops loop iterating over support vectors j
11. Adds bias term $b^*$to the function output
12. End, stops loop iterating over input vectors *i*

To sum up, this pseudocode explains how an SVM evaluates its decision function for each input vector considering the support vectors (vectors that lie on the decision boundary) and based on the resulting output of the decision function F; the prediction is compared with a predefined threshold to determine the class of the input which in this case is binary classification.

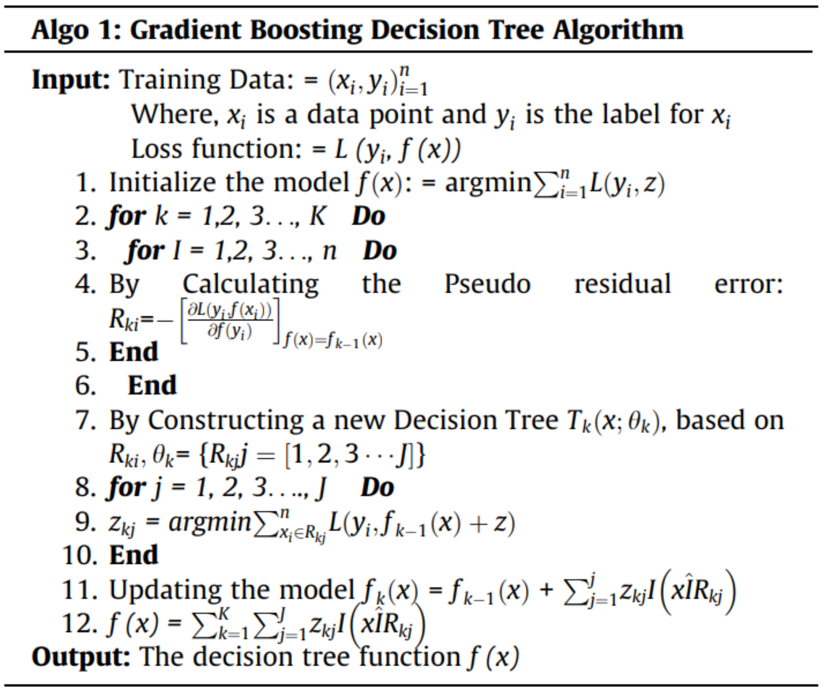
2. Random Forests [12]:

```
Pseudo-code for Random Forest:
Step 1: To Generate Forest
    For j in number of trees
      For i in number of nodes
        Randomly select n features from the dataset.
        for each feature in  random feature set
          calculate information gain.
        End for
        Create a node for the feature with highest information gain.
      End for
    End for
Step 2: Finding the class label
    For each class_label
      Average the probabilities obtained from all trees in the forest
    End for
    Assign the class_label with highest average to the instance.
```

This ensemble technique is divided into two main steps: **generate a forest** and **find the corresponding class label**. For step 1 iterating through *(j)* collection of decision trees and *(i)* number of nodes; a random sample of *(n)* features is drawn and for each feature in the dataset, information gain is computed $(Gain = Entropy_{parent} - Entropy_{children})$ for each split, a node is created for the feature with the highest information gain.

In step 2 assigning the corresponding label for an instance is determined through averaging the probabilities (likelihood of an instance to belong to a class) across all trees generated in the forest in step 1. Finally, the class label with the highest probability is assigned to the instance.

3. Gradient Boost [13]:



```
Algo 1: Gradient Boosting Decision Tree Algorithm
Input: Training Data: = $(x_i, y_i)_{i=1}^{n}$
        Where, $x_i$ is a data point and $y_i$ is the label for $x_i$
        Loss function: = $L\ (y_i, f\ (x))$
  1. Initialize the model $f(x)$: = argmin$\sum_{i=1}^{n} L(y_i, z)$
  2. for $k$ = 1,2, 3. . ., $K$   Do
  3.   for $I$ = 1,2, 3. . ., $n$   Do
  4. By Calculating the Pseudo residual error:
     $R_{ki} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(y_i)}\right]_{f(x)=f_{k-1}(x)}$
  5. End
  6.   End
  7. By Constructing a new Decision Tree $T_k(x; \theta_k)$, based on
     $R_{ki}, \theta_k = \{R_{kj}j = [1, 2, 3 \cdots J]\}$
  8. for $j$ = 1, 2, 3. . .., $J$   Do
  9. $z_{kj}$ = $argmin\sum_{x_i \in R_{kj}}^{n} L(y_i, f_{k-1}(x) + z)$
 10. End
 11. Updating the model $f_k(x)$ = $f_{k-1}(x)$ + $\sum_{j=1}^{j} z_{kj} I\left(x\hat{I}R_{kj}\right)$
 12. $f\ (x)$ = $\sum_{k=1}^{K}\sum_{j=1}^{J} z_{kj} I\left(x\hat{I}R_{kj}\right)$
Output: The decision tree function $f\ (x)$
```

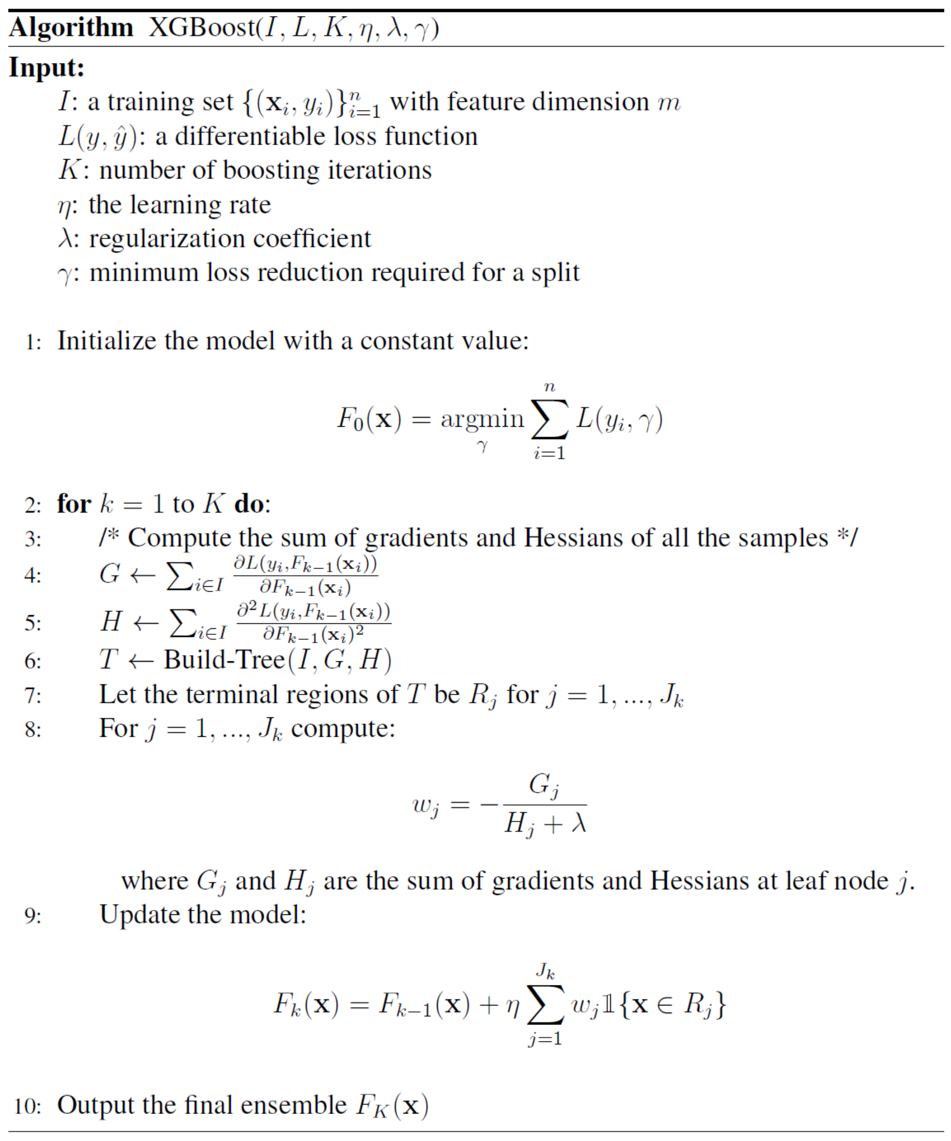
This boosting classifier based on decision trees takes the inputs: labeled training data ranging for *n* number of samples: $\{(x_1, y_1) \ldots (x_n, y_n)\}$ and the loss function **$L(y_i, f(x))$** of negative gradient used to calculate the **pseudo residuals** (following with steps 1-12)

1. The model f(x) is initialized as the argument that minimizes the sum of all the loss functions computed over all points
2. For each boosting iteration k from 1 to K
3. For each data point *i* to *n*
4. Calculate the pseudo residual error R_{ki} using the loss function
5. Ends iteration through data points
6. Ends iteration through boosting iterations
7. Based on the calculated pseudo residuals R_{ki} construct a new decision tree $T_k(x; \theta_k)$ is constructed on x as the predictor to calculate the risk minimization parameter θ_k where $\theta_k = \{R_{ki}j = [1, 2, 3, \ldots, J]\}$
   - The decision tree logically models relations among predictor variables, splitting input space x into J sections of R_1, R_2, …., R_J ; each output will be computed as Z_J for the corresponding region R_J [14]
8. For each J region
9. Compute z_{kj} that minimizes loss function L + z
10. Ends iteration through J regions

11. The model f$_k$(x) is updated + Z function which maps each input I to its corresponding region
12. The final function is given

Each iteration focuses on improving the model's performance by errors made of the previous model, a sequential learning technique. Gradient boosting decision trees are constructed based on pseudo residuals of the previous tree hence the k-1 in step 4 and 9. Final output is a model, an ensemble of these trees; prediction for any boosting classifier is the sum of the weighted predictions across all the decision trees (weak classifiers) in the ensemble[19].

4. XGBoost [14]:



```
Algorithm XGBoost($I, L, K, \eta, \lambda, \gamma$)
Input:
    $I$: a training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ with feature dimension $m$
    $L(y, \hat{y})$: a differentiable loss function
    $K$: number of boosting iterations
    $\eta$: the learning rate
    $\lambda$: regularization coefficient
    $\gamma$: minimum loss reduction required for a split

1: Initialize the model with a constant value:
       $F_0(\mathbf{x}) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$
2: for $k = 1$ to $K$ do:
3:     /* Compute the sum of gradients and Hessians of all the samples */
4:     $G \leftarrow \sum_{i \in I} \frac{\partial L(y_i, F_{k-1}(\mathbf{x}_i))}{\partial F_{k-1}(\mathbf{x}_i)}$
5:     $H \leftarrow \sum_{i \in I} \frac{\partial^2 L(y_i, F_{k-1}(\mathbf{x}_i))}{\partial F_{k-1}(\mathbf{x}_i)^2}$
6:     $T \leftarrow$ Build-Tree$(I, G, H)$
7:     Let the terminal regions of $T$ be $R_j$ for $j = 1, ..., J_k$
8:     For $j = 1, ..., J_k$ compute:
           $w_j = -\frac{G_j}{H_j + \lambda}$
       where $G_j$ and $H_j$ are the sum of gradients and Hessians at leaf node $j$.
9:     Update the model:
           $F_k(\mathbf{x}) = F_{k-1}(\mathbf{x}) + \eta \sum_{j=1}^{J_k} w_j \mathbb{1}\{\mathbf{x} \in R_j\}$
10: Output the final ensemble $F_K(\mathbf{x})$
```

XGBoost, very similarly to gradient boost utilizing decision trees instead of Entropy for information gain, here a similarity score is calculated for each node with the following equation [15]:

$$\textit{Similarity Score} = \frac{\textit{Gradient}^2}{\textit{Hessian} + \lambda}$$

Where Hessian is the number of residuals, Gradient^2 squared sum of residuals, Lambda is the regularization parameter.

1. Firstly, we initialize the model as the argument that minimizes the sum of the loss functions computed over all points as a constant
2. For each boosting iteration k from 1 to K
3. Compute the sum of negative gradients (residuals) denoted as G for all samples
4. Compute the sum of hessians denoted as H for all samples
5. Build a tree T on the training set I, gradient G, Hessian H
6. As mentioned in GB, input space is split in regions of R_j here denoted as ‘terminal regions of the tree T.
7. Iterating through nodes j to J_k compute the similarity score for each region at each leaf node
8. Model is updated as seen is step 9
9. Repeat for number of estimators
10. Final output is the final ensemble classifier denoted by $F_K(x)$

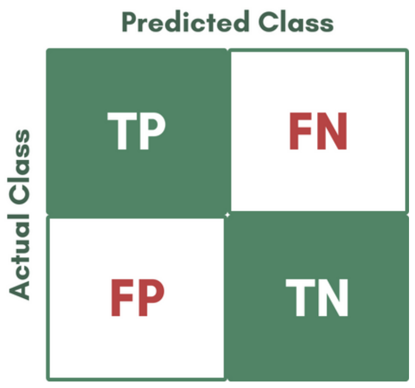
III. Evaluation:

What performance measures did you use to evaluate the effectiveness of your models?

I utilized mainly accuracy and f1 score and precision. I also plotted confusion matrices and for the tree-based methods feature importance to really see what features contribute the most in the predictions which are further explored in this section.

Why did you use these metrics?

- **Confusion Matrix**: this wouldn't be considered a performance metric; however, majority of performance metrics are based on what the confusion matrix presents. It visualizes how well the classification models predicted the true labels for each class. Whereas accuracy would give a holistic view on the performance, the confusion matrix delves into the true positives (TP), true negatives (TN), false positives (FP) and false negatives (FP) of the predictions; a rule of thumb is to minimize the false positive – type 1 error, actual class was false (0) but prediction was true (1) – and false negatives – type 2 error, actual was true (1) but prediction was false (0) –regardless of the use case [10].



- **Accuracy**: this metric can provide an overall measure of correctly classified predictions from all predictions. This metric was used by the original research paper to determine which models performed best however, accuracy overlooks class imbalance[10], on the outside it could return a high accuracy, but we must consider other metrics to justify the result.

$$Accuracy = \frac{Correct\ Predictions}{All\ Prediction} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision**: determines the correctly classified predictions from all classified predictions in the same class, in other words *"how many instances classified in class 1 were actually class 1"* just like the equation below [16].

$$Precision_{class\ 1} = \frac{True\ Positive_{class\ 1}}{True\ Positive_{class\ 1} + False\ Positives_{class\ 1}}$$

In Sklearn, a parameter 'average' must be defined when working with multiclass classification to the averaged precision across all classes, it can be set to 'micro'. 'macro' or 'weighted'. I opted for 'micro' which averages the totalled true positives over the sum of all true positives and false positives initially assigning equal weights to each instance whereas ‘macro’ assigns equal weight to each class [16]

As per our specified labels:

$$Precision_{micro} = \frac{TP_0 + TP_1 + TP_2 + TP_3}{TP_0 + FN_0 + TP_1 + FN_1 + TP_2 + FN_2 + TP_3 + FN_3}$$

The same has been applied to the F1 score and Recall.

- **F1 score** is the harmonic mean between the precision and *recall – how good the model was at predicting real true event* – through it we can get an idea, especially for imbalanced / real-life datasets of how well the model identifies relevant instances while minimizing the false positives FP and false negatives FN in other words, the trade-off between the precision and recall.

Evaluate how, based on the performance measures, you were able to enhance the model.

I was quite suspicious of my models performing well from the start however accuracy alone is not a good enough measure as all models were rounding on 99 – 100% looking at the confusion matrices clarified which models did the best and which models had higher FP – FN rates. Without the grid search, the iterative way would’ve been to train-validation-test split the data. Training occurs on the model and its through validation returning the scores, hyperparameter tuning occurs and once the model is giving optimal results or no change has been seen, these hyperparams are set and testing can take place. Given the fact that I utilized grid search with k-fold validation, the optimal hyperparams have already been distinguished so the next steps were to train and test the model and the results will be explored in the following section.

IV. Results and Discussion:

Discuss the reliability of your results and whether they are balanced, overfitting, or underfitting.

I believe my results are balanced however, with some careful considerations and trying out different techniques for predictions, my final choice was to oversample my data with SMOTE since Random Over sampling generates more instances from within the data, that repetition of instances skewed my results and all the models resulted in a 1.0 metrics with many misclassified points observed in the confusion matrix.

Trying out testing on the randomly under sampled data I believe gave the most realistic results however the instances were very few, 1468 training and 368 for testing. A massive drop from 8104 training instances – regular, not resampled data – I could risk losing a lot of information resulting in under representation of some critical patterns in the data resulting in a model that is too simple and doesn't generalize well – underfitting.

My final choice was to train– test split my data into 65% - 35% to ensure unbiased results and quite frankly the instances almost tripled from the SMOTE, so I believe it was a fair choice to have more instances to be tested on. A validation split was considered however, k-fold validation has already been applied and I didn't want the models to memorize the data in result in an overfit model.

A little observation – the data was quite repetitive, what I mean by that is there were 4 different columns of temperature who's values I would argue skewed around the same variance I analysed this through the df.describe(). In the research paper they used PCA and heterogonous fusion of the data which could have aided in more realistic results.

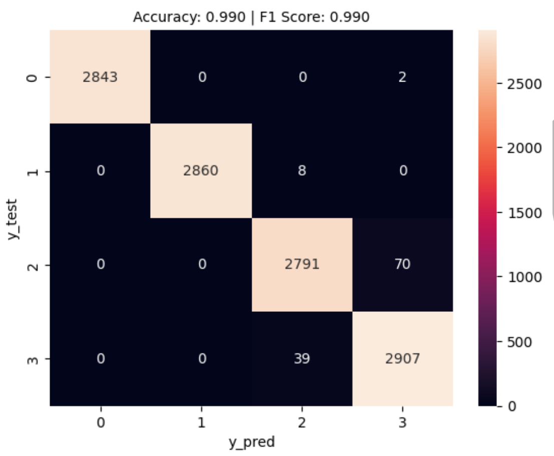
Analyse the result of the applications to determine the effectiveness of the algorithms.

Model	Test Accuracy	Test Micro-F1
Decision Tree - Base	0.998	0.998
SVM – Linear	0.989	0.989
SVM – RBF	0.996	0.996
Random Forests	0.999	0.999
Gradient Boost	1.0	1.0
XGBoost	1.0	1.0

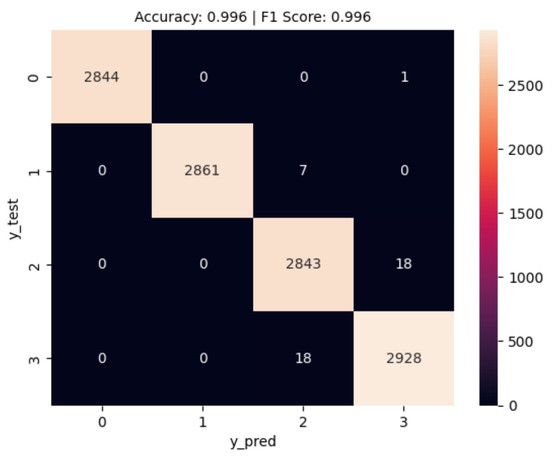
Figure 1: Results of each ML model rounded to 3 decimal places as per the original research paper.

Overall, we can say that all classifiers resulted in high accuracies and F1 scores and preformed quite well on the given dataset the tree-based classifiers gave perfect results, almost makes me question overfitting however, from seeing the confusion matrices, they also showed the least amount of FP and FN rates which means these models minimize both types of errors, a good balance between the precision and recall. In our use case I would prefer minimizing both as FP would indicate predicting human occupancy when actually the space is empty, and FN would indicate no occupancy when actually the space has occupants.

1. SVM

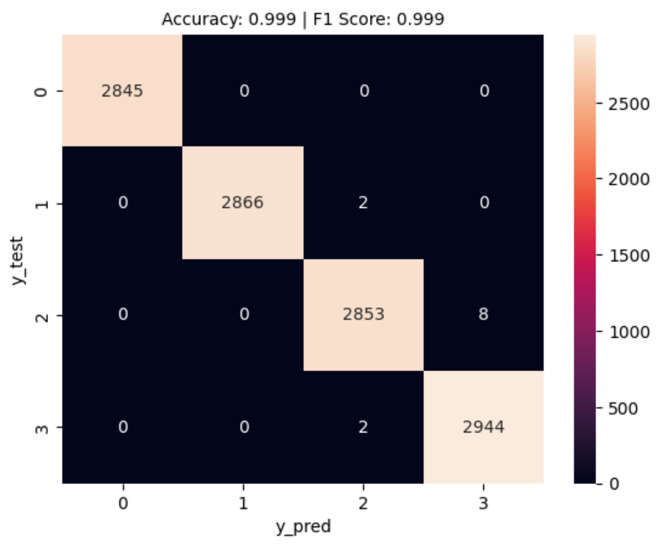


Linear: with an accuracy of 0.99 and a similar F1 score, this model preformed quite well with a hyper parameter of C=1. A closer at the confusion matrix, the model incorrectly predicted 39 instances as the class 2 whereas its true class is 3. Also misclassified 70 instances of class 2 as class 3, 8 instances of class 1 as class 2 and 2 instances of class 0 as class 3.



RBF: as the confusion matrix depicts, accuracy and F1 score of 0.996 with hyperparameters C = 10, gamma = 0.01, there are also similarly misclassified points as SVM Linear but are much more subtle, 18 compared to 70 of class 2 predicted as class 3 and similarly 18 compared to 39 of class 3 predicted as class 2.

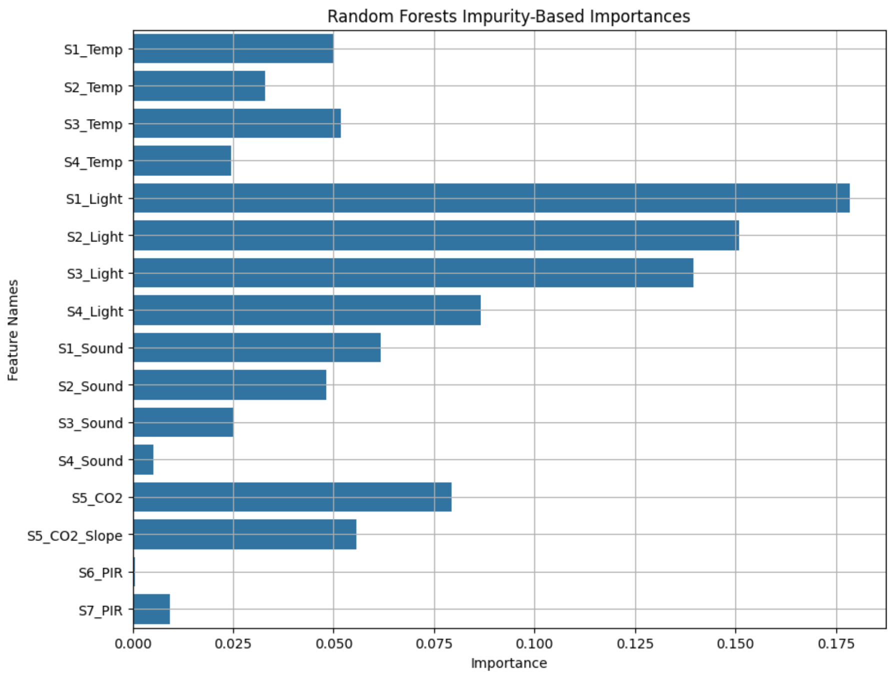
2. Random Forests



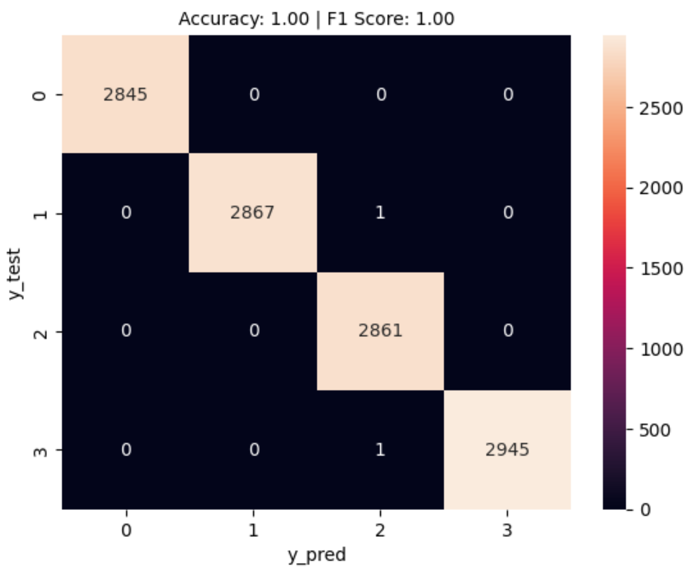
Significant improvement with the random forests with hyperparameters max_depth=10, max_features = 'sqrt', min_samples_leaf=1, min_samples_split=2, n_estimators=100. Seeing as only 2 misclassified points in the FN and 10 in the FP, I'm picking up on some similar patterns between classes 2 and 3.

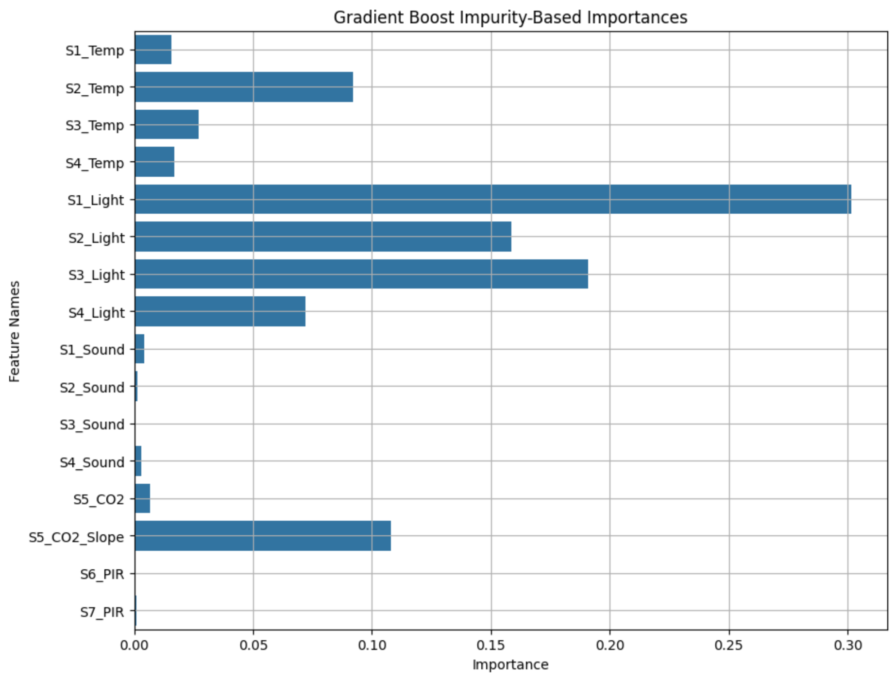
the Sklearn feature importance by default is mean decrease in impurity – assessing how extensively the feature has been used across all trees in the forest and is calculated as the normalized total reduction of the Gini impurity for tree-based classifiers [17].

In other words, how much each feature contributed to the overall classification performance of our model. Referring to the random forest's importance, it is easily distinguishable that the light features have the most effect on the model. The research did mention discarding that feature as they believe any person could leave the light on when they exit, making the models prone to false positives. Keeping light aside, a good indicator of human occupancy were CO2 (aprox 0.080) and CO2_slope (0.055) which is logical as more people occupy the room more CO2 is produced which was also mentioned in the paper as *'CO2 data seemed to give an excellent indication for the number of occupants in the room'*[1] the runner ups would be temperature (0.051 S3_Temp) and sound (aprox 0.06 S2_Sound), surprisingly the motion sensors had little to no effect probably because the only movement is leaving and entering whereas the rest of the time is spent sat at the desk.



3. Gradient boost I would argue gave the best results in both performance metric and minimizing FP and FN, however we'll see in the next section some of its drawbacks.

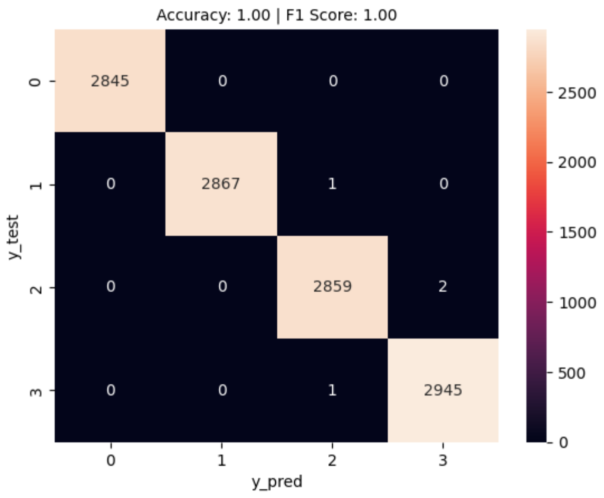


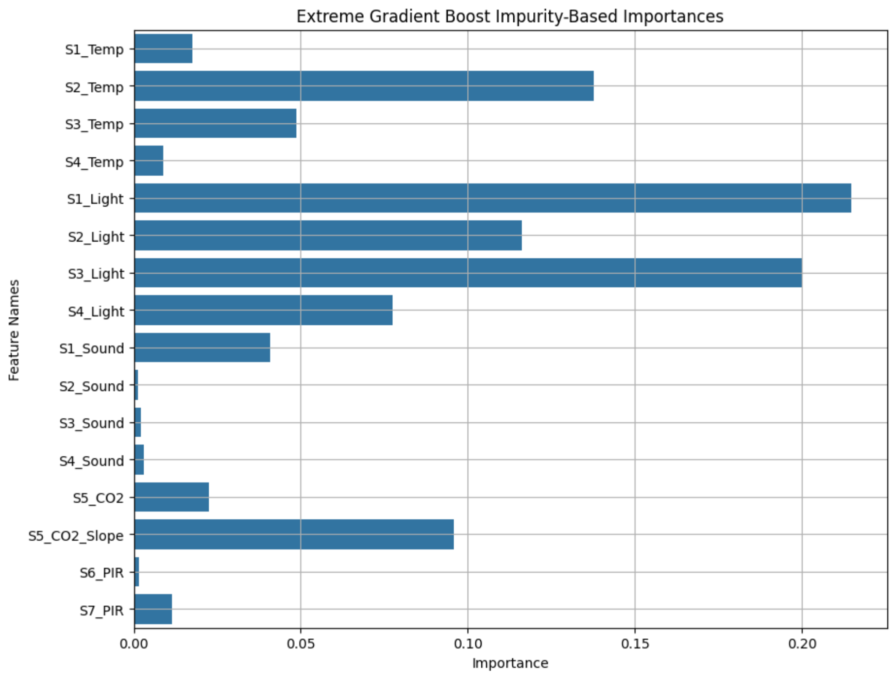


Similarly, the gradient booster also had significant effect from light (0.30), but disregarding that, CO2 slope showed a higher contribution (0.12) than that of the random forest, and a similar importance of the temperature features however, sound and motion were quite insignificant.

4. XGBoost

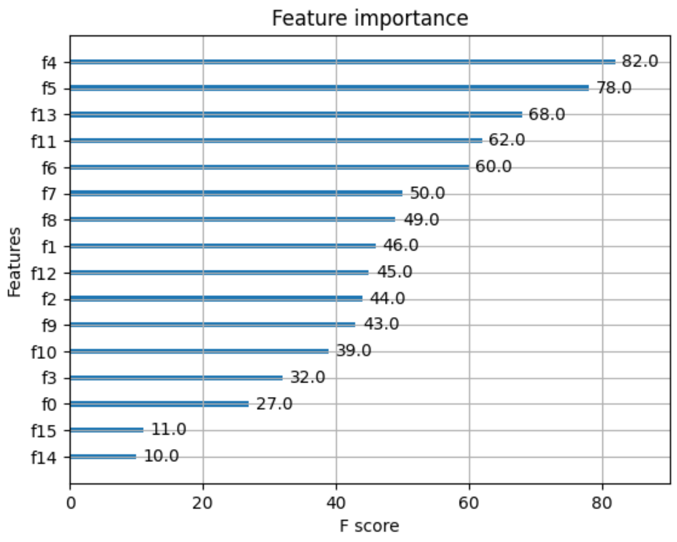
This boosting classifier also resulted in 100% accuracy and F1 simultaneously minimizing FP and FN





Feature importance in XGB showed a higher significance in S2_Temp compared to the previous two ensemble models (0.13). CO2 slope of (0.9) higher than random forest, similarly a high contribution of the light measures.

XGB library also offers feature importance based on the similarity score with S1_Light sitting at 82.0, S2_Light at 78.0, CO2_slope at 68.0 and surprisingly S4_Sound at 62.0; quite different results than the mean impurity base feature importance.



Draw conclusions regarding the strengths and weaknesses of the different algorithms.

Now because the models gave quite similar results, I chose to investigate other aspects to compare the models and really visualize the trade-off between computation time and accuracy, consider the table below:

Model	CPU time Training	Wall time Training	GridSearch-Kfold CV Wall time	Total misclassified points / 11520
SVM – Linear	1.86s	2.86s	11.3s	119
SVM – RBF	674ms	675ms	28.9s	44
Random Forests	3.67s	3.67s	18.4s	12
Gradient Boost	30.9s	31.1s	5min 27s	2
XGBoost	937ms	502ms	CPU time: 49.3s Wall time: 28.2s	4

CPU time refers to the actual processing time used by the CPU to execute the code and wall time, or real time is actual time that elapses in our time.

What I mainly wanted to see here is the difference between the gradient boost and its extreme version XGB. To analyse the table, SVM linear, SVM rbf and random forests had the shortest training times, gradient boost had the longest training times and the longest GridSearch CPU and wall times but in return the least number of misclassified points, this proves the ‘slow learning’ technique. XGBoost performs very well close to that of regular gradient boost minus the long CPU and wall times. So, in the end it depends on the context of the problem whether to prioritize performance accuracy or computation time.

Identify further enhancements which can be done in the future? Discuss any limitations and future improvements of your project.

When conducting EDA, I noticed the dates where during the Christmas-new year's holiday so as I plotted the room occupancy count across the days, out of 7, 4 had completely zero occupancy count which raised a couple of questions. 'Can those days be dropped and not considered in the ML modelling?', 'but do those days influence the ML model so it learns what the patterns are for 0 occupancy? which is practically majority of the time'. So, I decided to take the safe route and not drop the 4 days of holiday.

I would like to take into account the feature importance perhaps I could do feature selection; also according to the research paper and my insights on feature importance there was a skewness towards light they explained the potential reason behind this could be employees would switch on the lights but forget them on when they leave the room, whatever the reason might be, it could leave the model susceptible to false positives.

I would also consider modelling / computing the AUC ROC curves and scores to compare the false positive and true positive rates across the ML classifiers.

When experimenting different pipelines there was a trade-off between getting 'realistic' results and high accuracy, my personal POV; I'm not a huge fan of resampling data unless it's fundamentally required; powerful ensemble learning techniques such as XGBoost can handle class imbalances, perhaps training and testing the regular data with these ensemble techniques could result in more reliable results.

V. References

[1] Singh, A.P., Jain, V., Chaudhari, S., Kraemer, F.A., Werner, S. and Garg, V. (2018). Machine Learning-Based Occupancy Estimation Using Multivariate Sensor Nodes. *2018 IEEE Globecom Workshops (GC Wkshps),* [online] pp.1–6. doi:https://doi.org/10.1109/GLOCOMW.2018.8644432.

[2] Rosencrance, L. (2017). *What Is Zigbee? - Definition from WhatIs.com*. [online] IoT Agenda. Available at: https://www.techtarget.com/iotagenda/definition/ZigBee [Accessed 10 Jan. 2024].

[3] Aurélien Géron (2019). *Hands-on Machine Learning with Scikit-Learn and TensorFlow : concepts, tools, and Techniques to Build Intelligent Systems*. Second ed. Sebastopol, Ca: O'reilly Media, pp.461–484.

[4] Aurélien Géron (2019). *Hands-on Machine Learning with Scikit-Learn and TensorFlow : concepts, tools, and Techniques to Build Intelligent Systems*. Second ed. Sebastopol, Ca: O'reilly Media, pp.548–558.

[5] R, S.E. (2021). *Understand Random Forest Algorithms with Examples (Updated 2023)*. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/#Working_of_Random_Forest_Algorithm [Accessed 18 Jan. 2024].

[6] Gareth Michael James, Witten, D., Hastie, T.J. and Tibshirani, R. (2013). *An Introduction to Statistical Learning : with Applications in R*. New York: Springer, pp.358–359.

[7] PANCHOTIA, R. (2021). *Introduction to Gradient Boosting Classification*. [online] Medium. Available at: https://medium.com/analytics-vidhya/introduction-to-gradient-boosting-classification-da4e81f54d3.

[8] guest_blog (2018). *XGBoost: Introduction to XGBoost Algorithm in Machine Learning*. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/?utm_source=reading_list&utm_medium=https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/.

[9] xgboost.readthedocs.io. (n.d.). *XGBoost Parameters — xgboost 1.5.2 documentation*. [online] Available at: https://xgboost.readthedocs.io/en/stable/parameter.html.

[10] Notion Lecture notes – 'Model Assessment' Week 6 – Nov 19th, 2023

[11] Lim, C., Lee, S.-R. and Chang, J.-H. (2012). Efficient Implementation of an SVM-based speech/music Classifier by Enhancing Temporal Locality in Support Vector References. *IEEE Transactions on Consumer Electronics*, [online] 58(3), p.899. doi:https://doi.org/10.1109/TCE.2012.6311334.

[12] Devi, R.G. and Sumanjani, P. (2015). *Improved Classification Techniques by Combining KNN and Random Forest with Naive Bayesian Classifier*. [online] IEEE Xplore. doi:https://doi.org/10.1109/ICETECH.2015.7274997.

[13] Hasan Zulfiqar, Yuan, S.-S., Huang, Q.-L., Sun, Z., Dao, F.-Y., Yu, X. and Lin, H. (2021). Identification of Cyclin Protein Using Gradient Boost Decision Tree Algorithm. *Computational and Structural Biotechnology Journal*, 19, pp.4125–4126. doi:https://doi.org/10.1016/j.csbj.2021.07.013.

[14] Yehoshua, D.R. (2023). *XGBoost: the Definitive Guide (Part 1)*. [online] Medium. Available at: https://towardsdatascience.com/xgboost-the-definitive-guide-part-1-cc24d2dcd87a.

[15] Khan, M. (2021). *Explain the Step by Step Implementation of XGBoost Algorithm.* [online] Captain Sheikh Imtiaz and Sons. Available at: https://www.csias.in/explain-the-step-by-step-implementation-of-xgboost-algorithm/

[16] Evidently AI Team (n.d.). *Accuracy, precision, and Recall in multi-class Classification*. [online] www.evidentlyai.com. Available at: https://www.evidentlyai.com/classification-metrics/multi-class-metrics.

[17] scikit-learn developers (2022). *Feature importance — Scikit-learn course*. [online] inria.github.io. Available at: https://inria.github.io/scikit-learn-mooc/python_scripts/dev_features_importance.html.

[18] Berwick, R. (n.d.). *An Idiot's guide to Support vector machines (SVMs) R. Berwick, Village Idiot SVMs: A New Generation of Learning Algorithms*. [online] Massachusetts Institute of Technology . Available at: https://web.mit.edu/6.034/wwwbob/svm.pdf.

[19] Notion Lecture notes – ' Ensemble Methods' Week 10 – December 17th, 2023