Analysis and Results

Ashish Sharma

1 Approach towards the Solution

The problem set its scope on the identification and ranking of important features present in the data, according to their importance/predictive power, in a machine learning model. Apart from that, it is required to substantiate this ranking of feature importance. The argument points out that this problem is a classification problem, which deals with the feature importance present in the data. The features importances present in the data can be studied by using various machine learning algorithms.

2 Selection of a Model: XGBoost

The standard models Random Forest or Decision-Tree are considered to be the first approach to solve various challenges, i.e. fitting and evaluation of data, classification and regression [3]. These models lead to satisfactory results but sometimes fail to achieve the results [2]. There are various alternatives to these standard models, which can be used to fit and evaluate the data, rather than moving on to more complex Deep Learning models.

To name, a few alternative models are XGBoost [4], AdaBoost [13], and many more [7]. An algorithm is considered for the analysis based on prerequisites, the model, which is more closely aligned with the assumptions and kind of data it can handle, help us to narrow down our search to the best suitable method.

I chose XGBoost [12] over the other available machine learning algorithms with various merits, which are also the reason that XGBoost performs very well and produce reasonable results, as follows [4, 6]

- XGBoost is an Extreme Gradient Boosting (Gradient, Stochastic and Regularized) machine learning algorithm that is highly efficient to use.
- XGBoost improves the computation speed and model performance over other models and fits data faster.
- It can handle missing data in the dataset.
- The in-build regularization model formalism to avoid overfitting plays a crucial role in the better performance of XGBoost.
- The parallelization/scalability of the tree-construct and depth-first approach increase the computation speed.

- It works very well on structured data, small data, big data and complicated data.

The boosting trees in XGBoost, in comparison to the decision tree, follow an iterative approach. That means it follows an ensemble technique in which many models are combined to perform the final one. Further, the boosting trains models in succession rather than in isolation of one another such that each model is trained to correct the errors made by previous ones that help against overfitting. The models are added sequentially until no further improvements can be achieved.

3 Exploring Artificial Dataset

In this section, I will dig out the properties of provided data. First, I will check if there are any Null values, missing or unknown data type present in the data. Checking out the data properties will help to identify the main features of the data before analysing it. The benefits are as follows

- Concentrate on the main features which help in building the model.
- Get rid of features that are irrelevant and will not contribute to the model performance.

The code used for this analysis is in the python file **'task_solution.py'** or in the jupyterlab notebook file **'main_task_code.ipynb'** provided alongside this file. First, I started by loading the sensors data into Pandas [15] data frame and then exploring the available fields.

```
import pandas as pd

## """ Read the data from the provided task_file """
task_file = ('./task_data.csv')
with open(task_file,mode='rb') as f:
    dataset = pd.read_csv(f)

print(dataset.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   sample index   400 non-null    object
 1   class_label    400 non-null    float64
 2   sensor0        400 non-null    float64
 3   sensor1       400 non-null    float64
 4   sensor2       400 non-null    float64
 5   sensor3       400 non-null    float64
 6   sensor4       400 non-null    float64
 7   sensor5       400 non-null    float64
 8   sensor6       400 non-null    float64
 9   sensor7       400 non-null    float64
 10  sensor8       400 non-null    float64
 11  sensor9       400 non-null    float64
dtypes: float64(11), object(1)
memory usage: 37.6+ KB
```

```
print(dataset.describe())
```

```
       class_label     sensor0     sensor1     sensor2     sensor3
count   400.000000  400.000000  400.000000  400.000000  400.000000
mean      0.000000    0.523661    0.509223    0.481238    0.509752
std       1.001252    0.268194    0.276878    0.287584    0.297712
min      -1.000000    0.007775    0.003865    0.004473    0.001466
25%      -1.000000    0.299792    0.283004    0.235544    0.262697
50%       0.000000    0.534906    0.507583    0.460241    0.510066
75%       1.000000    0.751887    0.727843    0.734937    0.768975
max       1.000000    0.999476    0.998680    0.992963    0.995119

          sensor4     sensor5     sensor6     sensor7     sensor8     sensor9
count  400.000000  400.000000  400.000000  400.000000  400.000000  400.000000
mean     0.497875    0.501065    0.490480    0.482372    0.482822    0.541933
std      0.288208    0.287634    0.289954    0.282714    0.296180    0.272490
min      0.000250    0.000425    0.000173    0.003322    0.003165    0.000452
25%      0.249369    0.269430    0.226687    0.242848    0.213626    0.321264
50%      0.497842    0.497108    0.477341    0.463438    0.462251    0.578389
75%      0.743401    0.738854    0.735304    0.732483    0.740542    0.768990
max      0.999412    0.997367    0.997141    0.998230    0.996098    0.999465
```

The data seemed to be continuous. Even if there are missing or Nan Values, XGBoost can handle them. Next, we will try to find out if there are any correlations between the features of the data or not using Pandas [15] corr() and seaborn [17] heatmap functions.

```
corr_matrix = dataset.corr()

## Heatmap of the correlation matrix
sns.heatmap(corr_matrix, cmap='Blues', annot=True)
plt.savefig('./Correlation_Matrix of Sensors.png',dpi=300)
plt.show()
plt.close()
```

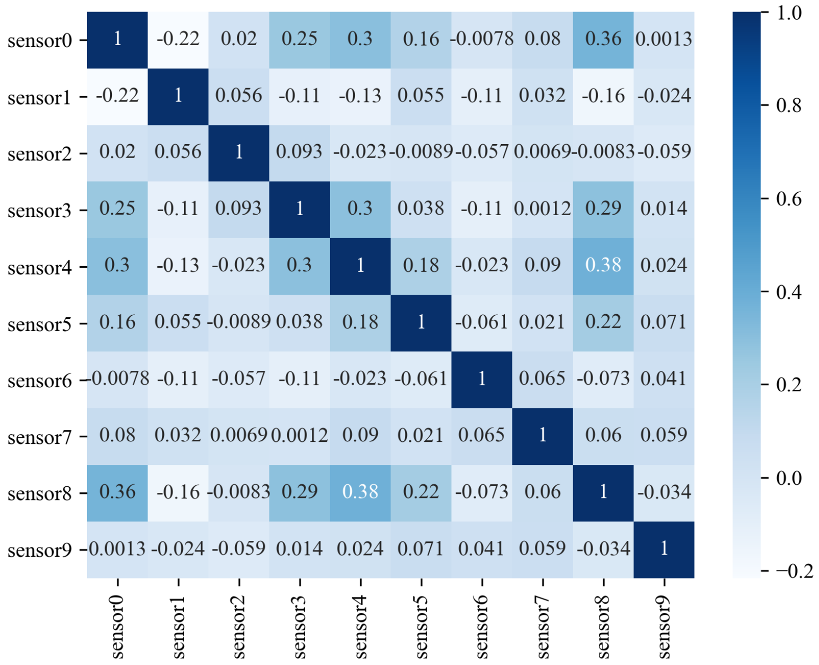

Figure 1: The plot shows the cross-correlation between the various features of data.

From the correlation matrix plot in Figure 1, we can conclude that there are no correlations present between the various features of the data. Since we have explored the data, the next step would be to search for an important feature. In the next section, I will explain a model I chose earlier for this purpose.

3.1 Building the Model

Throughout this section, I will use XGBoost to predict which feature is most important in the provided data. To the next step, after loading and exploring the data, it is apparent to divide the data into training features and labels **check code provided either in task_solution.py or main_task_code.ipynb**.

```
## Split the data into training features and labels.
## Extracting Labels from the data
Y = dataset['class_label']

## Extracting Features from the data
dataset.drop(['sample index','class_label'],axis=1,inplace=True)
X = dataset
```

To classify the most important feature present in the provided data, first, I divided the data into training and test dataset by using sklearn [16] train_test_split() function. Here, I decided to go with the 20% of test data. It is obvious to go with a small test dataset such that we can have a large training dataset to train the model. The other reason to choose only 20% was the small size of the data.

```
import sklearn
from sklearn.model_selection import train_test_split

## Split data into train and test dataset
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
```

The training and test dataset are ready for further analysis. I imported XGBClassifier() from XGBoost [5] and pass it with some parameters. Finally, I fit the model to training features and labels. Now, we are all set to make predictions about the important features in the given data.

```
import XGBoost as xgb
import sklearn
from sklearn import model_selection

## Defining the model to rank the sensors according to their importance/
## predictive power with respect to the class labels of the samples
## Testing the Accuracy and Efficiency of model

## Instantiate model
model = xgb.XGBClassifier(n_estimators=500, learning_rate=0.05,
        eval_metric='logloss')

## Fitting model using training data set
model.fit(X_train, y_train, early_stopping_rounds=50,eval_set=[(X_train,
          y_train), (X_test, y_test)])
```

Now, I have fitted my model to predict the important features present in the data and would like to know/visualize which feature turns out to be the most important. For this purpose, XGBoost provides another constructive built-in feature importance plot module named as plot_importance() [5] which plots the score of each feature ordered by their importance and corresponding F-score as shown in Figure 2. This could be beneficial while selecting features with one or more model under consideration.

```
from xgboost import plot_importance
import matplotlib.pyplot as plt

## Visualizing the importance of each feature/sensor in the data set
## with their F-values.
plot_importance(model,importance_type='gain')
```

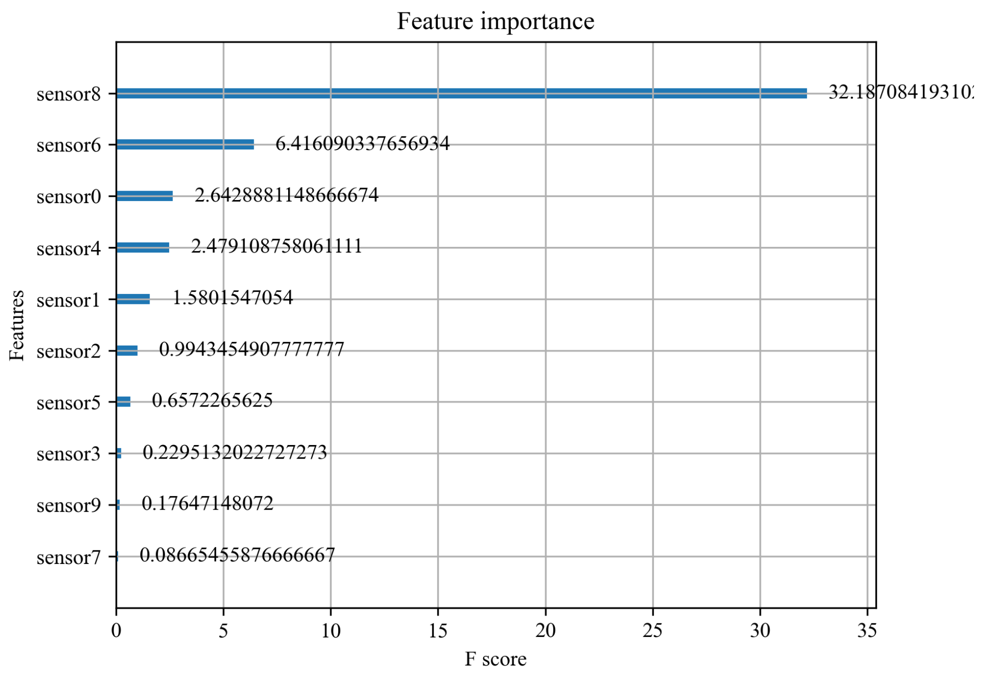

Figure 2: Plot showing the relationship between features in the given data with their importance and F-score.

```
plt.show()
plt.close()
```

```
Note: Results may vary.
```

It is crucial to choose the right importance_type, which shows how importance is being calculated. Either it can lead to different results from what we get from the model [8].

Finally, I have the important features and their corresponding importance values, and I ranked them based on their importance. Later I saved them into a CSV data file.

```
import numpy as np

## Ranking of sensors based on their importance value in a reverse order
rank_sensors = sorted(zip(model.feature_importances_, X.columns), reverse=True)
for label in rank_sensors:
    print('For {} Predicitve Score is {}'.format(label[1], label[0]))
```

After knowing about the most important feature that my model predicted, I would like to investigate how effective my model was during the entire process. For this, I evaluate the mean absolute error in the prediction and the accuracy of my model using the sklearn [16] in-built features.

```
from sklearn.metrics import accuracy_score, mean_absolute_error

## Evaluate the predictions for Test dataset
predictions = model.predict(X_test)

## Mean Absolute error
print("Mean Absolute Error : " + str(mean_absolute_error(predictions, y_test)))

## Model Accuracy i.e how correct is the classifier?
accuracy = accuracy_score(y_test, predictions)
print('Accuracy of Model is : %.2f%%'%(accuracy*100.0))

## Output
Mean Absolute Error : 0.025
Accuracy of Model is : 98.75%

Note: Results may vary.
```

Making predictions with the model of my choice and using accuracy as my measure, I have achieved 98.75% accuracy.

Now we have everything in our hand, let's try to visualize the output, i.e. plotting the sensors and their importance in reverse order. For this purpose, I used seaborn [17] and matplotlib [14] Python libraries.

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.rcParams["font.family"] = "Times New Roman"
import matplotlib.font_manager as font_manager

## Defining a function to plot the features vs importance score
def plot_features(features_list, importance_list, model_name, palette=None):
    sns.set()
    sns.set_style("whitegrid")
    plt.figure(figsize=(8, 4))
    sns.barplot(features_list, np.array(importance_list), palette=palette)
    plt.xticks(sensors_list, sensors)
    plt.ylabel('Importance Score')
    plt.title(model_name)
    # plt.tight_layout()
    plt.savefig('./'+model_name+'_Sensors_vs_Importance_value_plot',dpi=300)
    plt.show()
    plt.close()
```

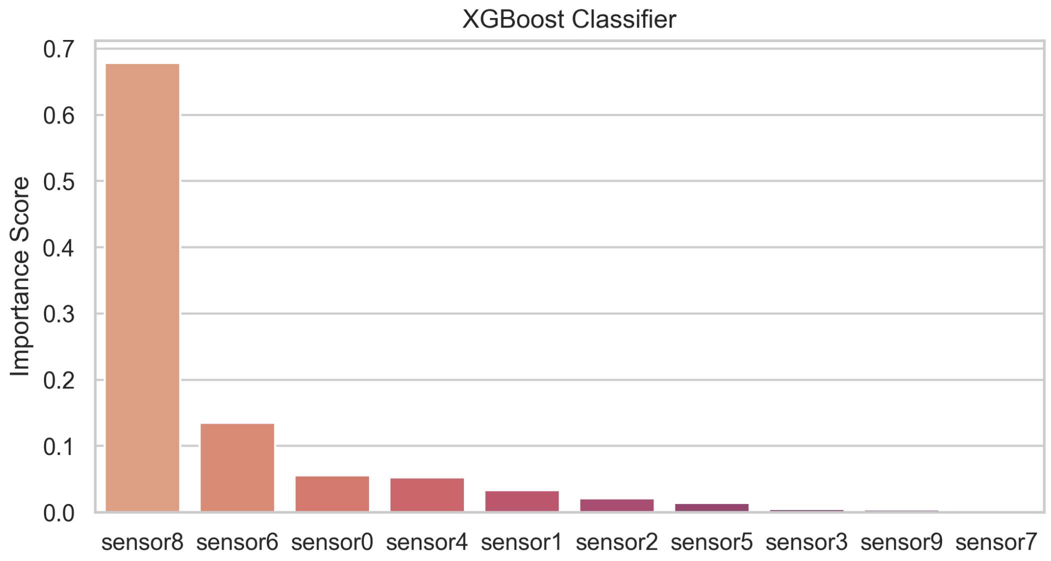

Figure 3: Bar plot showing the sensors on x-axis and their importance values on y-axis. From the plot it is easy to conclude that sensor8 is the most important feature in the provided data.

```
## Plot features with respect to their importance value
plot_features(sensors_list, scores, 'XGBoost Classifier', palette='flare')
```

From the Figure 3, we can conclude that sensor8, followed by sensor6, comes out to be the most important feature present in the provided data.

3.2 Weaknesses of Model

There is no doubt that XGBoost works very effectively to solve the problem. But few points can cause the model to give inaccurate results. Here I will mention these points that one needs to take care of while building the XGBoost Model.

- Feature importance can lead to misleading results due to blind use. Therefore, it is crucial to be aware of and understand feature results in detail [8, 9].
- The poor choice of parameters used to build the XGBoost model can lead to overfitting and thus to the overall inefficiency of the model [1, 10, 11].

3.3 Scalability of Model

The scalability of a model can be achieved by using the sklearn SelectFromModel() class [16] which takes a model and transform the dataset into a subset with selected features [9]. The SelectFromModel() class takes a pre-trained model and then use a threshold to decide between the features. The threshold value is used on the SelectFromModel() instance to select the same values of features on the training and test dataset.

I wrapped the model in SelectFromModel() instance which then used to select features on the training dataset and then train the model on the selected subset of features. After this, the model is trained on the test dataset. This concept can be used to test multiple thresholds for selecting the features by feature importance.

```
import xgboost as xgb
import sklearn
from sklearn import model_selection
from sklearn.metrics import accuracy_score, mean_absolute_error
from sklearn.feature_selection import SelectFromModel

## Scalability with respect to numbers of features
## Fitting model using importance as threshold

thresholds = np.sort(model.feature_importances_)
for threshold in thresholds:
    ## Selecting feature based on threshold value
    select_feature = SelectFromModel(model, threshold = threshold, prefit = True)
    select_X_train = select_feature.transform(X_train)
    ## Train the model
    select_feature_model = xgb.XGBClassifier(eval_metric='logloss')
    select_feature_model.fit(select_X_train, y_train)
    ## Evaluating the model
    select_X_test = select_feature.transform(X_test)
    feature_prediction = select_feature_model.predict(select_X_test)
    feature_prediction = [round(value) for value in feature_prediction]
    ## Accuracy
    feature_accuracy = accuracy_score(y_test, feature_prediction)
    print('Threshold = %.5f, n=%d, Accuracy : %.2f%%' %
     (threshold,select_X_train.shape[1],feature_accuracy*100.0))
```

```
Threshold = 0.00183, n=10, Accuracy : 98.75%
Threshold = 0.00372, n=9, Accuracy : 98.75%
Threshold = 0.00484, n=8, Accuracy : 98.75%
Threshold = 0.01385, n=7, Accuracy : 98.75%
Threshold = 0.02096, n=6, Accuracy : 98.75%
Threshold = 0.03330, n=5, Accuracy : 98.75%
Threshold = 0.05225, n=4, Accuracy : 98.75%
Threshold = 0.05570, n=3, Accuracy : 98.75%
Threshold = 0.13522, n=2, Accuracy : 98.75%
Threshold = 0.67834, n=1, Accuracy : 87.50%
```

4 Alternative Methods

As stated earlier, there are various models which can be used to study the given problem. But each model has its own merits and demerits. Here I will point out a few of such machine learning algorithms with their properties **check code provided either in task_solution.py or main_task_code.ipynb**.

4.1 Random Forest

: Random Forest [3] is the most popular machine learning algorithm which provides relatively good accuracy and robustness.

The Random Forest model tends to show weakness when the dataset has two or more correlated features [2]. On interpreting the data, this scenario can lead to an incorrect conclusion that is dominating features can end up with low scores, and the method can be biased towards variable with many categories. The other which can impose challenge is the collinearity and inflation towards the continuous values.

```
## https://scikit-learn.org/stable/modules
random_forest = RandomForestClassifier()
random_forest.fit(X_train, y_train)

## Extract Feature Importance Value and Rank them in a reverse order
rank_rf = sorted(zip(random_forest.feature_importances_, X.columns), reverse=True)
for label in rank_rf:
    print('For RandomForest {} Predictive Score is {}'.format(label[1], label[0]))

## Separating features and their importance values
scores_rf = list(zip(*rank_rf))[0]
sensors_rf = list(zip(*rank_rf))[1]
sensors_rf_list = np.arange(len(sensors_rf))

## Plot features with respect to their importance value
plot_features(sensors_rf_list, scores_rf, 'RandomForest Classifier', palette='crest')

## Evaluate the predictions
rf_predictions = random_forest.predict(X_test)

## Mean Absolute error
print("Mean Absolute Error : " + str(mean_absolute_error(rf_predictions, y_test)))

## Model Accuracy i.e how correct is the classifier?
rf_accuracy = accuracy_score(y_test, rf_predictions)
print('Accuracy of Random Forest Model is : %.2f%%'%(rf_accuracy*100.0))
```

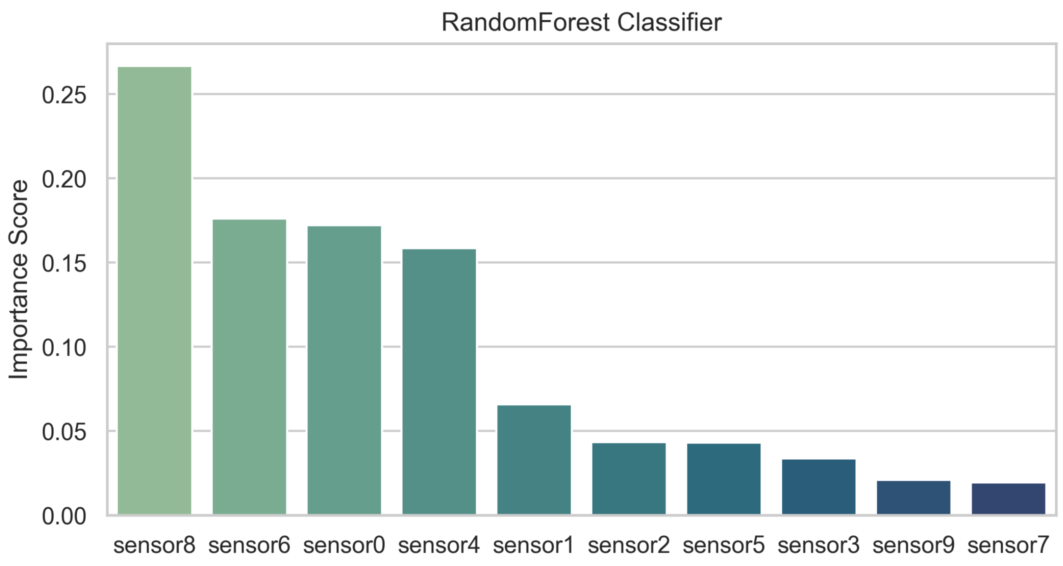

Figure 4: Importance feature plot with Random Forest Classifier model. Features are on the x-axis and their importance values on the y-axis. From the bar plot, it is easy to conclude that sensor8 comes out to be the most important feature in the provided data.

```
Mean Absolute Error : 0.0
Accuracy of Random Forest Model is : 100.00%
```

4.2 AdaBoost

AdaBoost[13], like any other machine learning model, has its advantages and disadvantages. It aims at combining several weak learners to form a single strong learner. AdaBoost is fast, simple and easier to use with minimum tweaking of parameters. AdaBoost can also be used beyond binary classification, such as with text or numeric data.

Contrary to this, to disadvantages, boosting techniques learns progressively, thus requiring quality data. AdaBoost is sensitive to noisy data, and outliers, if not eliminated, can lead to inaccurate results.

```
## https://scikit-learn.org/stable/modules
adaboost = AdaBoostClassifier()
adaboost.fit(X_train,y_train)

## Extract Feature Importance Value and Rank them in a reverse order
rank_ada = sorted(zip(adaboost.feature_importances_, X.columns), reverse=True)
for label in rank_ada:
    print('For AdaBoost {} Predictive Score is {}'.format(label[1], label[0]))
```

```
## Separating features and their importance values
scores_ada = list(zip(*rank_ada))[0]
sensors_ada = list(zip(*rank_ada))[1]
sensors_ada_list = np.arange(len(sensors_ada))

## Plot features with respect to their importance value
plot_features(sensors_ada_list, scores_ada, 'AdaBoost Classifier'
              , palette='dark:salmon_r')

## Estimating the performance of AdaBooster Classifier
ada_prediction = adaboost.predict(X_test)

## Mean Absolute error
print("Mean Absolute Error : " + str(mean_absolute_error(ada_prediction, y_test)))

## Model Accuracy i.e how correct is the classifier?
ada_accuracy = accuracy_score(y_test, ada_prediction)
print('Accuracy of AdaBoost Model is : %.2f%%' % (ada_accuracy*100.0))
```

```
Mean Absolute Error : 0.025
Accuracy of AdaBoost Model is : 98.75%
```

5 Conclusion

In this analysis, I showed how to dig out the feature importance in a machine learning model, which is not limited to just one specific model. Understanding the results is a crucial step for any business, therefore one requires to have a deep knowledge of concepts to understand which variables are the most important for the model and how they can impact its overall efficiency. Consequently, one can lead to further improvements into the model, and at last, to a better understanding of business.

References

[1] Xgboost : Optimal Parameters, editor = Machine Learning Mastry, year = 2019, howpublished = https://towardsdatascience.com/selecting-optimal-parameters-for-xgboost-model-training-c7cd9ed5e45e#:~:text=classification%20error%20plot%20shows%20a,the%20next%2010%20iterations)%20works., urldate = 2021-05-30.

[2] Limits of Random Forest Algorithm. https://towardsdatascience.com/a-limitation-of-random-forest-regression-db8ed7419e9f, 2019.

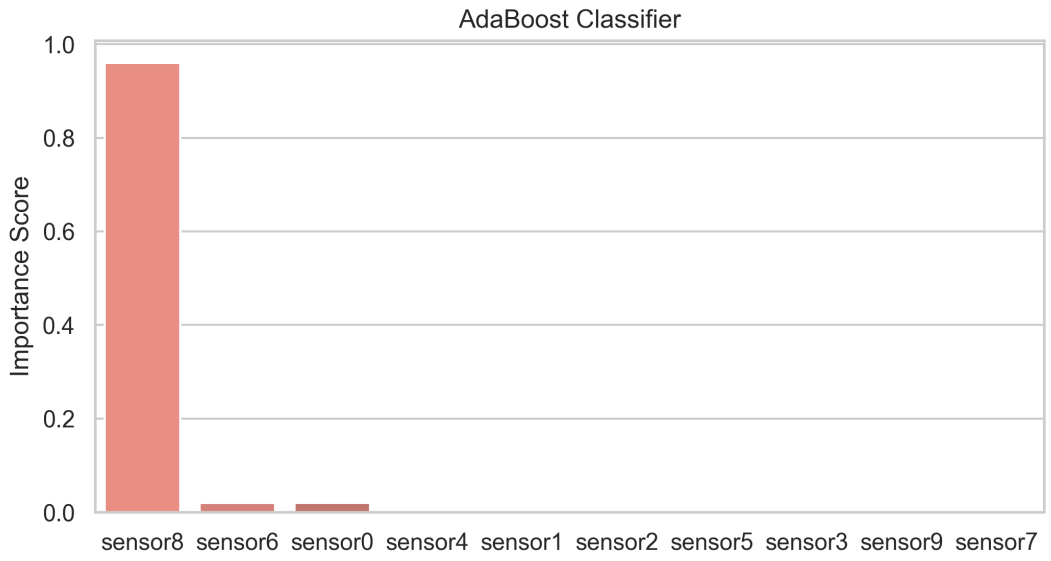

Figure 5: Importance feature bar plot with the AdaBoost Classifier model. Features are on the x-axis and their importance values on the y-axis. From the bar plot, it is easier to conclude that sensor8 is the most crucial feature in the provided data.

[3] Random Forest Algorithm. https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf308a9b76, 2019.

[4] Xgboost : A beginner's guide. https://towardsdatascience.com/a-beginners-guide-to-xgboost-87f5d4c30ed7, 2019.

[5] Xgboost Python Package. https://xgboost.readthedocs.io/en/latest/python/python_api.html#module-xgboost.sklearn, 2019.

[6] Xgboost Algorithm. https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d, 2019.

[7] Machine Learning Algorithms. https://medium.com/analytics-vidhya/a-guide-to-underrated-machine-learning-algorithms-alternatives-to-decision-tree-and ~:text=However%2C%20the%20alternatives%20I'm,easy%2Dto%2Duse%20approach%3A&text=a)%20XGBoost%20stands%20for%20Extreme,efficient%20and%20flexible%20to%20use., 2020.

[8] Xgboost : Feature Importance. https://towardsdatascience.com/be-careful-when-interpreting-your-features-importance-in-xgboost-6e16132588e7, 2020.

[9] Xgboost : Feature importance and feature selection. https://machinelearningmastery.com/feature-importance-and-feature-selection-with-xgboost-in-python/, 2020.

[10] Xgboost : Gradient Boosting. https://machinelearningmastery.com/tune-learning-rate-for-gradient-boosting-with-xgboost-in-python/, 2020.

[11] Xgboost : Overfitting. https://machinelearningmastery.com/avoid-overfitting-by-early-stopping-with-xgboost-in-python/, 2020.

[12] Tianqi Chen and Carlos Guestrin. Xgboost. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug 2016. doi: 10.1145/2939672.2939785. URL http://dx.doi.org/10.1145/2939672.2939785.

[13] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *ICML*, 1996.

[14] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

[15] The pandas development team. pandas-dev/pandas: Pandas, February 2020. URL https://doi.org/10.5281/zenodo.3509134.

[16] F. Pedregosa, G. Varoquaux, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[17] Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021. doi: 10.21105/joss.03021. URL https://doi.org/10.21105/joss.03021.