Data Fusion

Instructor: Dr. Moshiri

Reyhane Vahedi 810101303

Spring 2024

Homework 5

Section 1:

(1)Stacking: Stacking is a way to ensemble multiple classifications or regression model, and bagging and boosting are in fact, expansions on the stacking method. The point of stacking is to explore a space of different models for the same problem. The idea is that we can attack a learning problem with different types of models which are capable to learn some part of the problem, but not the whole space of the problem. So, we can build multiple different learners and you use them to build an intermediate prediction, one prediction for each learned model. Then we add a new model which learns from the intermediate predictions the same target. This final model is said to be **stacked** on top of the others, hence the name. Thus, we might improve your overall performance, and often we end up with a model which is better than any individual intermediate model. So to sum it up, in this method various weak learners are ensembled in a parallel manner in such a way that by combining them with Meta learners, we can predict better predictions for the future. The general scheme of this algorithm is given in fig 1.

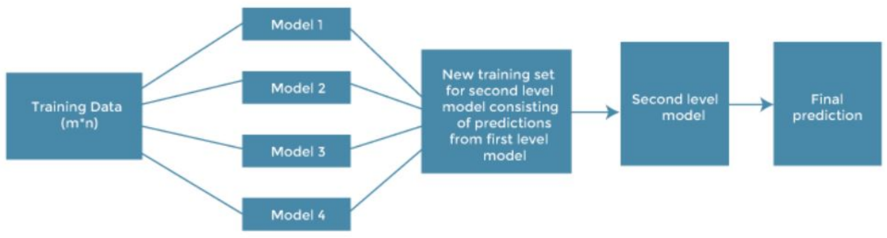

Training Data (m*n)
Model 1
Model 2
Model 3
Model 4
New training set for second level model consisting of predictions from first level model
Second level model
Final prediction

Figure 1: Stacking

(2)Bagging(Bootstrap aggregating): Bagging is another method of ensemble modeling, which is primarily used to solve supervised machine learning problems. It is generally completed in two steps as follows: a) Bootstrapping: It is a random sampling method that is used to derive samples from the data using the replacement procedure. In this method, first, random data samples are fed to the primary model, and then a base learning algorithm is run on the samples to complete the learning process. b) Aggregation: This is a step that involves the process of combining the output of all base models and, based on their output, predicting an aggregate result with greater accuracy and reduced variance. In summary, bagging involves training multiple models independently and in parallel on different random subsets of the data, where each model is trained on a bootstrap sample, which is a random subset of the original data (sampled with replacement). By exposing theses models to different parts of the dataset, bagging reduces variance and helps avoid over-fitting. The predictions from all these models are then combined through simple averaging to make the overall prediction.

(2)Boosting: Boosting is another ensemble method that enables each member to learn from the preceding member's mistakes and make better predictions for the future. Unlike the bagging method, in boosting, all base learners (weak) are arranged in a sequential format so that they can learn from the mistakes of their preceding learner. Hence, in this way, all weak learners get turned into strong learners and make a better predictive model with significantly improved performance. Additionally, bagging typically involves simple averaging of models, while boosting assigns weights based on accuracy.

To point out some of the differences between bagging and boosting, firstly as previously said, the first is parallel while the second is sequential. Additionally, bagging reduces variance while boosting reduces bias. Bagging can be used with unstable models like decision trees while boosting works better for stable models like linear regression. Both methods have their strengths and weaknesses. Bagging is simpler to run parallel while boosting can be more powerful and accurate. In practice, it helps to test both on a new problem to see which performs better. The differences between the bagging and boosting algorithms can be seen in fig 2.

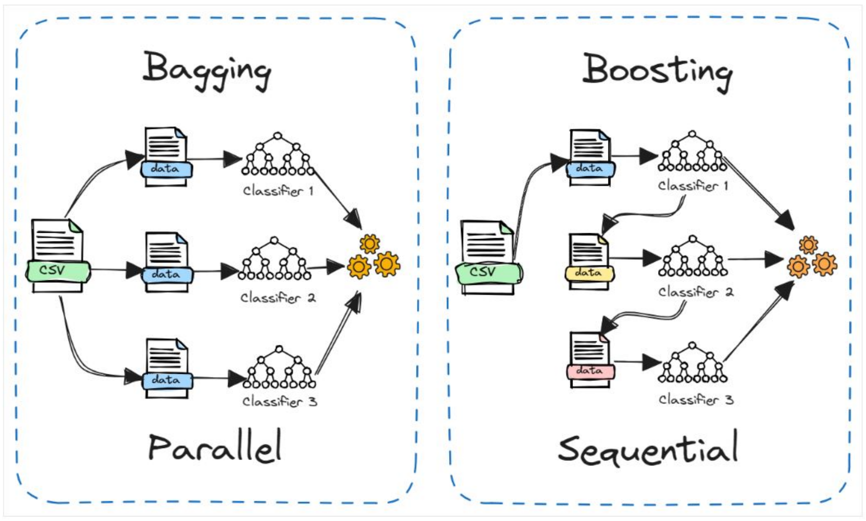
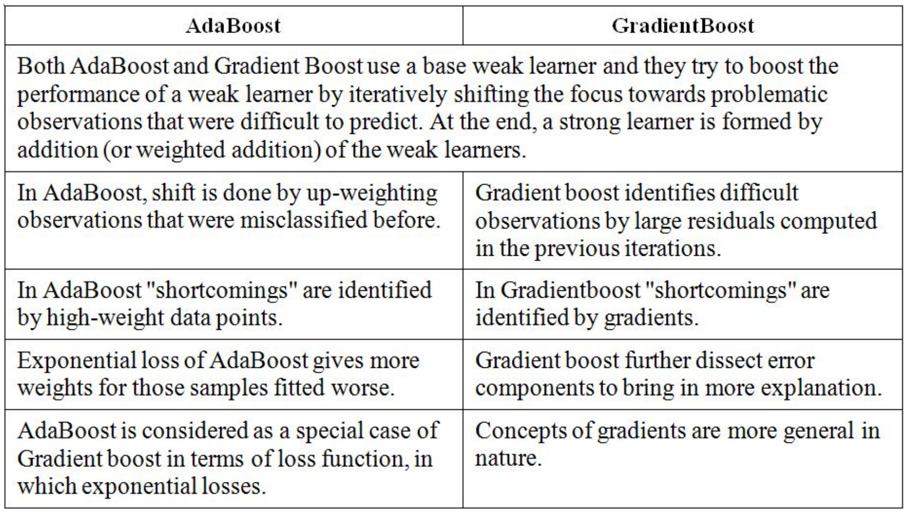


Figure 2: Differences of Bagging and Boosting

In comparison, Bagging is superior in reducing variance by averaging predictions from multiple models trained independently on bootstrapped subsets of the data, and it also makes the model more robust by reducing sensitivity to noise. However, it lacks model interpretability and relies on generally weak models with low prediction accuracy which could be disadvantageous. Boosting on the other hand, improves accuracy by creating a strong model by iteratively correcting errors made by weak models, and it also assigns higher weights ti misclassified instances which improves the handling of class imbalance. However, it is computationally more expensive than bagging and can be sensitive to outliers because of its weight assignment method. Lastly, stacking can improve prediction accuracy even further by combining predictions from diverse base models leading to an overall better performance. This model is also flexible and allows heterogeneous models to work together. However, it is a complex model which entails creating a meta-model and if not handled carefully, it can lead to data leakage. In summary, bagging focuses on variance reduction, boosting aims for accuracy improvement, and stacking leverages diverse models for better predictions.

(2)AdaBoost: AdaBoost or Adaptive Boosting, is an ensemble technique used in machine learning to improve the performance of weak learners. AdaBoost is the first successful boosting algorithm developed for **binary classification**. It assigns higher weights to incorrectly classified instances, focusing on areas where the base learner fails and often uses decision trees with only one level (decision stumps) as base learners. The main difference between boosting and AdaBoost, lies in their objective; The objective of AdaBoost is to minimize the exponential loss function by focusing on areas where the base learner (weak model) fails, while the objective of boosting is the principle of stagewise addition, or combining multiple weak learners into a strong final model. In summary, AdaBoost is a specific type of boosting that emphasizes correcting errors through weighted instances, making it a powerful tool for improving weak classifiers. AdaBoost is less sensitive to outliers, and its base classifiers are mostly decision trees, used for binary classification. The differences of Boosting (Gradient Boosting) and AdaBoost are summarized in fig 3.

References:
1) https://www.geeksforgeeks.org/stacking-in-machine-learning/
2)https://www.javatpoint.com/stacking-in-machine-learning
3) https://www.datacamp.com/tutorial/what-bagging-in-machine-learning-a-guide-with-examples
4)https://www.analyticsfordecisions.com/bagging-vs-boosting-vs-stacking/
5) https://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning/
6) https://dataheadhunters.com/academy/gradient-boosting-vs-adaboost-battle-of-the-algorithms/
7) (fig 3:) https://medium.com/fragile-by-design/first-dive-into-machine-learning-and-ai-competitive-da



AdaBoost	GradientBoost
Both AdaBoost and Gradient Boost use a base weak learner and they try to boost the performance of a weak learner by iteratively shifting the focus towards problematic observations that were difficult to predict. At the end, a strong learner is formed by addition (or weighted addition) of the weak learners.	
In AdaBoost, shift is done by up-weighting observations that were misclassified before.	Gradient boost identifies difficult observations by large residuals computed in the previous iterations.
In AdaBoost "shortcomings" are identified by high-weight data points.	In Gradientboost "shortcomings" are identified by gradients.
Exponential loss of AdaBoost gives more weights for those samples fitted worse.	Gradient boost further dissect error components to bring in more explanation.
AdaBoost is considered as a special case of Gradient boost in terms of loss function, in which exponential losses.	Concepts of gradients are more general in nature.

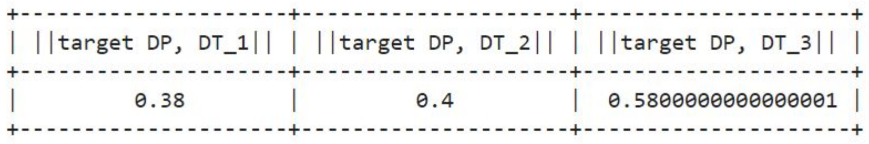
Figure 3: Differences of Boosting and AdaBoost

Section 2:

In general, it could be said that the squared Euclidean Distance between two matrices is their squared Frobenius Norm, or more generally the sum of the difference between their individual, corresponding, elements. This distance measure, for 3x3 matrices, can be formulated as below:
(1)

$$\text{Distance} = \sum_{i=1}^{3}\sum_{j=1}^{3}(A_{ij} - B_{ij})^2$$

While this measure can easily be calculated, a simple script was written for it using equation (1), and the resulting distances between the Decision Profile x, and DT_1, DT_2, and DT_3, can be seen in fig 4.



```
+---------------------+--------------------+---------------------+
| ||target DP, DT_1|| | ||target DP, DT_2|| | ||target DP, DT_3|| |
+---------------------+--------------------+---------------------+
|        0.38         |        0.4         | 0.5800000000000001 |
+---------------------+--------------------+---------------------+
```

Figure 4: Dist. between target DP and the three given DTs

It can be seen that the distance between the target DP and the first Decision Template is less than that of the two other DTs. So we can say that the given DT belongs to DT_1 or is in other words, considered **Novice**.
References:
1)https://math.stackexchange.com/questions/3331513/is-there-a-geometric-interpretation-about-the-euc

Section 3:

When speaking on methods of decision level data fusion, they can generally be divided into two categories regarding the characteristics of the base classifiers' outputs. What we have previously talked about in class, entails that we can use methods like majority voting, weighted majority voting and etc. to fuse decisions that are crisp values, however we did not discuss the methods used when the base classifiers result in probabilities and not crisp decisions. The first category of methods, is generally called Hard Voting since the values are crisp and the decisions of the base classifiers are final with no mentions of confidence. The second mentioned category which deals with probabilities, or in other words confidence scores, is called **Soft Voting**. In a more elaborate manner, in hard voting, each individual classifier in the ensemble predicts a class label for a given input, the final ensemble prediction is based on the majority vote (i.e., the most frequent predicted class label) among all classifiers. It also assumes that the classifiers' predictions are binary. However in soft voting, the probability distribution of predicted class labels from each classifier is considered. Each classifier provides a probability score for each class label, and the final prediction is based on the max of the sums of these predicted probabilities across all classifiers. Soft Voting takes the confidence levels into account as previously said, and it is better suited for well-calibrated classifiers, which can prove beneficial but hard to achieve, and is generally more complex than Hard Voting. Soft Voting can be broken down into three stages:

1) For each class, we collect the predicted probabilities from all base models or classifiers.

2) Then we calculate the average or weighted average of these probabilities to determine the final probability for each class.

3) Lastly we select the class with the highest probability as the final prediction.

Suppose we have N base models or classifiers denoted by $M_1, M_2, \ldots, M_n$, and we want to perform soft voting to combine their probabilistic outputs. Let's assume we have K classes denoted by $C_1, C_2, \ldots, C_k$.

For each base model, M_i provides the probability distribution P_{ij} for each class C_j, where i ranges from 1 to N and j ranges from 1 to K. The probability P_{ij} represents the likelihood of a sample belonging to class C_j according to model M_i. The soft voting classifier combines the probabilities from all base models to determine the final probability for each class. The soft voting formula is as follows:

(2)

$$P(C_j) = \frac{1}{N}\sum_{i=1}^{N} P_{ij}$$

Where $P(C_j)$ is the final probability for class C_j, N is the number of base models, and P_{ij} is the probability assigned to class C_j by model M_i.

References:

1) https://www.baeldung.com/cs/hard-vs-soft-voting-classifiers

2)https://rasbt.github.io/mlxtend/user_guide/classifier/EnsembleVoteClassifier/

3)https://medium.com/@awanurrahman.cse/understanding-soft-voting-and-hard-voting-a-comparative-analy

4)https://towardsdatascience.com/how-to-attain-a-deep-understanding-of-soft-and-hard-voting-in-ensem

Section 4: Simulation

a) After fusing the data according to the instructions of part (b), the fused dataset is made up of three series of positional and numerical measurements, or three separate measurement vectors per fusion of nine columns, resulting from three sensors. These new fused measurement vectors are high in variance and are of considerably larger lengths. So because of this variance and the size of data, algorithms such as NB, KNN, or Logistic Regression wouldn't prove sufficient. So, we will be using **Random Forest, SVM, and XGBoost**, as our binary classifiers. To elaborate more on our choices, we can summarize each of these classifiers' pros and cons regarding our problem statement. **Random Forest** is an ensemble method that combines multiple decision trees. It's robust against overfitting, handles high-dimensional data well, and can handle both numerical and categorical features. The randomness in feature selection and bootstrapping helps reduce variance which is beneficial since we are dealing with highly variant data. Random Forest can handle noisy and high-variance data effectively, it also captures complex interactions between features and provides feature importance scores. **XGBoost** is an efficient gradient boosting algorithm that combines weak learners to create a strong model. It handles high variance, overfitting, and missing data well. Again, this algorithm too can handle highly variant data efficiently which fits our problem well. **SVM** aims to find the optimal hyperplane that best separates classes. It works well with high-dimensional data and can handle non-linear relationships using kernel functions. Again, this algorithm can handle noisy and variant data very well, and it can also capture complex patterns even from a smaller number of features.

b) The basic idea of the Kalman filter is to use a model of the system being measured, and to update the model as new measurements become available. The filter works by making a prediction of the current state of the system based on the previous state estimate and the system model, and then combining this prediction with a new measurement to obtain an updated state estimate. Kalman Filter in its classic form, fuses a prior data with a measurement. However, for fusion purposes, it can be reimagined to fuse two measurements to estimate a new state. Using this theory, we could say that for N different measurements, we can fuse N-1 vectors to have the fusion of all measurements at hand. However, to employ this filter for fusion, with the equations given below, we need to suppose that the measurements are independent and have zro covariance, they are observed at the same time and have n time delay, and they also have a mean of zero. While the first two are just assumed to be true, for the third case, we normalize each vector so that it is a zero-mean vector bore fusing the data.

Then we can use the following equations to calculate the weight of each measurement instance in a sequential fusion. First, we can consider the system measurements to be of the following nature,

$$Y_1[n] = X[n] + V[n]$$

$$Y_2[n] = X[n] + W[n]$$

Figure 5: Assumed Measurement Equations

Then, by having in mind the previous assumptions, we can say that the fusion following the KF rules, would be as depicted in fig 6.

$$\hat{X} = \alpha Y_1 + (1-\alpha)Y_2$$

Figure 6: Estimation of State Using Fusion

Then for finding out the weights, or the value of α, we first calculate the MSE which is shown in fig 7.

$$\begin{aligned} E[(\hat{X}-X)^2] &= E[(\alpha Y_1 + (1-\alpha)Y_2 - X)^2] \\ &= E[(\alpha X + (1-\alpha)X + \alpha V + (1-\alpha)W - X)^2] \\ &= E[(\alpha V + (1-\alpha)W)^2] \\ &= \alpha^2\sigma_v^2 + (1-\alpha)^2\sigma_w^2. \end{aligned}$$

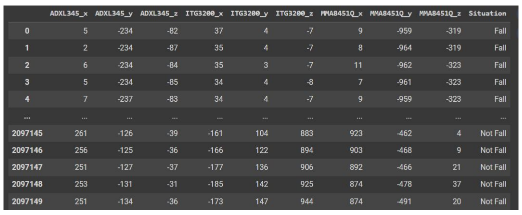
Figure 7: MSE calculation

Lastly, by taking the derivative of the previous equation in respect to α, we can minimize the value of MSE by choosing the following weights. These weights can be calculated just by having the variances of each measurement's noise, or in our problem, the variances of each vector.

To summarize, in this section, we first normalized the vectors, then fused them sequentially (meaning that we first fused two sensors and then fused the resulting measurements with the third sensor,) to reach a state in which we have three columns of data instead of nine. The initial and final results of the dataset are shown in figures 9 and 10 respectively.

$$\hat{X} = \frac{\sigma_w^2}{\sigma_v^2 + \sigma_w^2} Y_1 + \frac{\sigma_v^2}{\sigma_v^2 + \sigma_w^2} Y_2 .$$

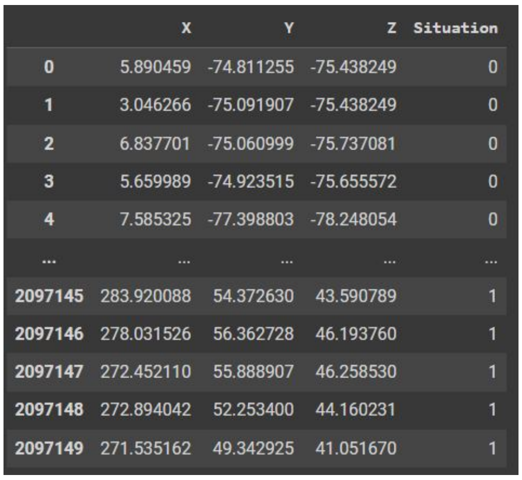
Figure 8: Final Weights



	ADXL345_x	ADXL345_y	ADXL345_z	ITG3200_x	ITG3200_y	ITG3200_z	MMA8451Q_x	MMA8451Q_y	MMA8451Q_z	Situation
0	5	-234	-82	37	4	-7	9	-959	-319	Fall
1	2	-234	-87	35	4	-7	8	-964	-319	Fall
2	6	-234	-84	35	3	-7	11	-962	-323	Fall
3	5	-234	-85	34	4	-8	7	-961	-323	Fall
4	7	-237	-83	34	4	-7	9	-959	-323	Fall
...	...	...	...	...	...	...	...	...	...	...
2097145	261	-126	-39	-161	104	883	923	-462	4	Not Fall
2097146	256	-125	-36	-166	122	894	903	-468	9	Not Fall
2097147	251	-127	-37	-177	136	906	892	-466	21	Not Fall
2097148	253	-131	-31	-185	142	925	874	-478	37	Not Fall
2097149	251	-134	-36	-173	147	944	874	-491	20	Not Fall

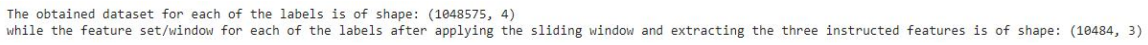
Figure 9: Initial State of the Dataset

c) In this part, we first define sliding windows over our dataset. As per the instructions, the sampling rate is 200Hz and the time period is 1 second, while each two sequential windows are 50 percent overlapping. All of this means that we can define every 200 samples as a window if it has 100 samples in common with the previous one and 100 samples in common with the next one. Since we aim to extract features from these windows, before separating and saving the windows, we first segregate the dataset into two parts: One containing all the data rows labeled as "Fall" and one containing all those labeled as "Not Fall". So finally when we slide the defined windows over the data frame, we will get two different sets of windows, one for each label, and each window containing 3 columns for the fused positions and one for the label. For the next step, we define three functions to obtain the asked features from the windows. The features are sum vector magnitude, angle between z-axis and vertical axis, and standard deviation magnitude. These three features were extracted from each window, which contains 3 values for 3 axes and a label, meaning that we have three features per window. The shape of the initial dataset separated by label, compared to the size of **each** feature set(the same size as the extracted windows), can be seen in fig 11.



	X	Y	Z	Situation
0	5.890459	-74.811255	-75.438249	0
1	3.046266	-75.091907	-75.438249	0
2	6.837701	-75.060999	-75.737081	0
3	5.659989	-74.923515	-75.655572	0
4	7.585325	-77.398803	-78.248054	0
...	...	...	...	...
2097145	283.920088	54.372630	43.590789	1
2097146	278.031526	56.362728	46.193760	1
2097147	272.452110	55.888907	46.258530	1
2097148	272.894042	52.253400	44.160231	1
2097149	271.535162	49.342925	41.051670	1

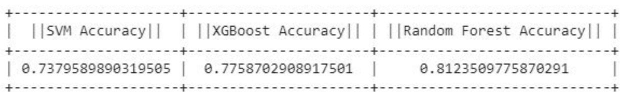
Figure 10: Final State of the Dataset



```
The obtained dataset for each of the labels is of shape: (1048575, 4)
while the feature set/window for each of the labels after applying the sliding window and extracting the three instructed features is of shape: (10484, 3)
```

Figure 11: Comparison between the sizes of the "Fall"/"Not Fall" datasets with the feature sets after window sliding

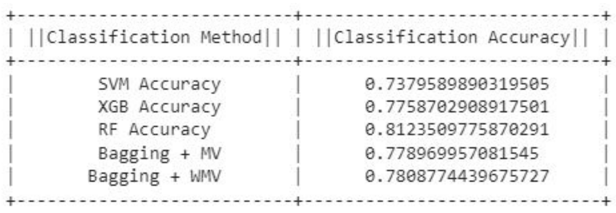
d) In this part, we first train each of the three previously chosen classifiers individually. For this purpose, we also concatenated the previously separated feature sets for the two labels into one complete feature set containing all three features for both labels. Then we apply a 80-20 test-train-split on this concatenated dataset. By using the classifiers and scoring functions of the *sklearn* library, we can get each classifiers accuracy. The report can be seen in fig 12.



```
+-------------------+---------------------+--------------------------+
|  ||SVM Accuracy||  | ||XGBoost Accuracy|| | ||Random Forest Accuracy|| |
+-------------------+---------------------+--------------------------+
| 0.7379589890319505 | 0.7758702908917501  |    0.8123509775870291    |
+-------------------+---------------------+--------------------------+
```

Figure 12: Accuracy of each individual classifier

Then we defined majority voting and weighted majority voting (**considering the weights of 0.45, 0.35, and 0.2 for the best to worst performing classifiers: RF, XGBoost, and SVM**) and applied bagging with each of these two voting techniques. The resulting accuracies of these ensemble methods in comparison to the use of single classifiers can be seen in fig 13.



```
+--------------------------+---------------------------+
| ||Classification Method|| | ||Classification Accuracy|| |
+--------------------------+---------------------------+
|       SVM Accuracy       |     0.7379589890319505     |
|       XGB Accuracy       |     0.7758702908917501     |
|       RF Accuracy        |     0.8123509775870291     |
|       Bagging + MV       |     0.778969957081545      |
|      Bagging + WMV       |     0.7808774439675727     |
+--------------------------+---------------------------+
```

Figure 13: comparison between the accuracy of single classifiers with the ensemble methods

e) While stacking can be effective, it can in some cases lead to potential overfitting. Some factors adding to this possibility are Information Leakage, Over-complexity of Base Models, Over-abundance of Base Models, insufficient Data, or Wrong Hyper-parameter Tuning. 1) Stacking involves training a meta-model on the predictions made by base models using out-of-sample data. If the base models have already seen the same data used for training the meta-model, there's a risk of Information Leakage.

2) Information Leakage occurs when the meta-model learns to exploit patterns that are specific to the training data used by the base models, leading to overfitting.

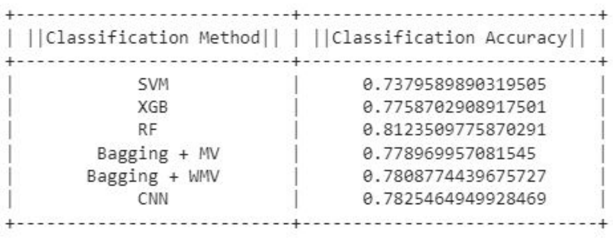
3) If the base models are overly complex or deep, they may capture noise or idiosyncrasies in the training data. The meta-model can then learn to amplify these artifacts, resulting in overfitting.

4) On another note, each base model contributes to the ensemble, but if there are too many base models included in the ensemble, hence an elevated diversity, the meta-model may struggle to generalize well and will be again, subject to overfitting.

5) Stacking requires a sufficiently large dataset to train both base models and the meta-model so if the dataset is small, overfitting will again become more likely.

6) Lastly, hyper-parameters of base models and the meta-model both need tuning. Overfitting can occur if these hyper-parameters are not optimized properly.

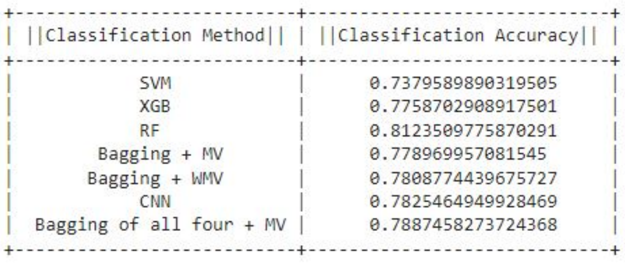
f) For this section a CNN using the MLPClassifier from $sklearn.neural_network$, with a hidden layer size of $(20, 20, 20)$, a *relu* activation function, and an ADAM solver has been trained in 500 iterations. The accuracy of this model in comparison to all previous models is given in fig 14.



```
+--------------------------+---------------------------+
| ||Classification Method|| | ||Classification Accuracy|| |
+--------------------------+---------------------------+
|           SVM            |    0.7379589890319505     |
|           XGB            |    0.7758702908917501     |
|            RF            |    0.8123509775870291     |
|       Bagging + MV       |    0.778969957081545      |
|      Bagging + WMV       |    0.7808774439675727     |
|           CNN            |    0.7825464949928469     |
+--------------------------+---------------------------+
```

Figure 14: Accuracy of each individual classifier compared to the accuracy of the CNN

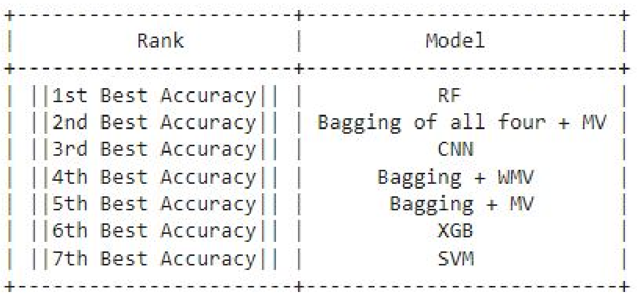
Then we applied the same bagging + Majority voting technique as before on the four trained classifiers, including the CNN. The results can be seen in fig 15.



```
+---------------------------+---------------------------+
| ||Classification Method|| | ||Classification Accuracy|| |
+---------------------------+---------------------------+
|            SVM            |    0.7379589890319505     |
|            XGB            |    0.7758702908917501     |
|             RF            |    0.8123509775870291     |
|        Bagging + MV       |    0.778969957081545      |
|       Bagging + WMV       |    0.7808774439675727     |
|            CNN            |    0.7825464949928469     |
|  Bagging of all four + MV |    0.7887458273724368     |
+---------------------------+---------------------------+
```

Figure 15: Accuracy of all tested models

Lastly to sum everything up, the results are listed in fig 16. We can see that the single RF classifier has the best performance out of all the single and ensemble models, which could be because of the characteristics of the dataset which math RF's criteria. Then Bagging four classifiers with Majority Voting, the single CNN, Bagging the initial three classifiers with Weighted Majority Voting, Bagging the initial three classifiers with Majority Voting, XGB, and finally SVM, have the best to worst accuracies.



```
+-----------------------+--------------------------+
|          Rank         |          Model           |
+-----------------------+--------------------------+
| ||1st Best Accuracy|| |            RF            |
| ||2nd Best Accuracy|| | Bagging of all four + MV |
| ||3rd Best Accuracy|| |           CNN            |
| ||4th Best Accuracy|| |      Bagging + WMV       |
| ||5th Best Accuracy|| |       Bagging + MV       |
| ||6th Best Accuracy|| |           XGB            |
| ||7th Best Accuracy|| |           SVM            |
+-----------------------+--------------------------+
```

Figure 16: Best to worst performing models

References:
1)https://dsp.stackexchange.com/questions/85434/kalman-filter-on-sensor-fusion
2) https://dsp.stackexchange.com/questions/44451/mmse-estimation-fusion-of-2-measurements
3) https://medium.com/@satya15july_11937/sensor-fusion-with-kalman-filter-c648d6ec2ec2
4) https://stats.stackexchange.com/questions/440686/is-this-kind-of-stacked-ensemble-method-prone-to-
5) https://rasbt.github.io/mlxtend/user_guide/classifier/StackingCVClassifier/
6) https://machinelearningmastery.com/stacking-ensemble-machine-learning-with-python/
7) https://towardsdatascience.com/8-simple-techniques-to-prevent-overfitting-4d443da2ef7d
8) https://stats.stackexchange.com/questions/217726/isnt-stacking-models-a-direct-approach-to-overfit
9) https://www.stat.cmu.edu/~ryantibs/papers/sensorfus.pdf