University of Illinois at Urbana Champaign Varun Kasbekar

Hydraulic accumulator Sensor Prediction
REPORT

Target Condition

- Hydraulic accumulator / bar

Classification Algorithms

- Support Vector Machine

- Random Forest

- Gradient Boosted Machine

The first step in any data science project is preprocessing the data. I have started this particular project with Data preprocessing. I have extracted data from text files which contained Sensor data. I have stored this data into respective dataframes.

Then I move to the Data Cleaning part in which I start with checking for missing values in each dataframe.

Missing Value Detection

Here I have checked for missing values in each dataframe. I have got negative results indicating that there are no missing values in any of the dataframes. I have created a custom fucnction which takes the dataframe as the input and computes or checks for missing values inside it.

The function prints out whether there are any missing values in the dataframe or not.

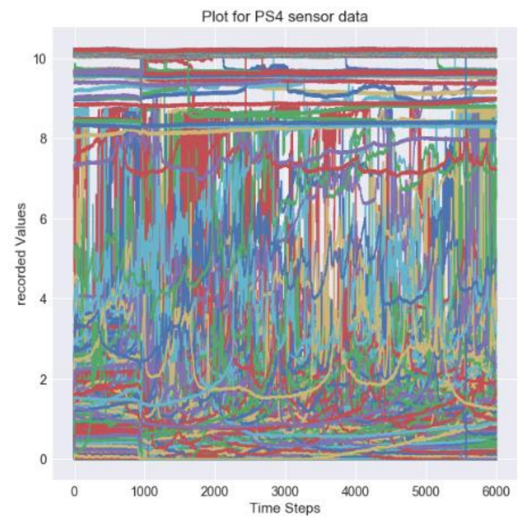
Then I move ahead to analyzing the dataframes in detail. For this purpose, I plot each dataframe separately. This will help me analyze the data closely and also help me to understand presence of Outliers.

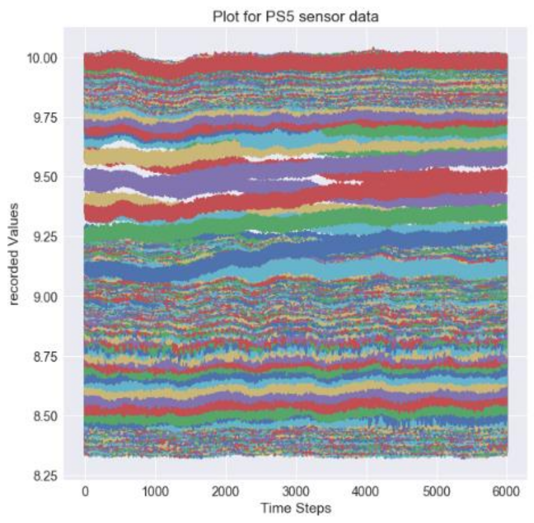
Visualizing the Data for detecting outliers

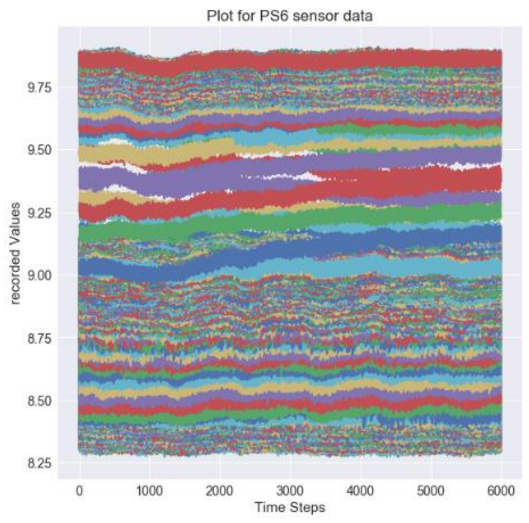
Here I am plotting line graphs of every variable. These graphs show the fluctuation of the variable value over time (60, 600 or 6000 time steps depending on the frequency), for 2205 instances (therefore 2205 lines).

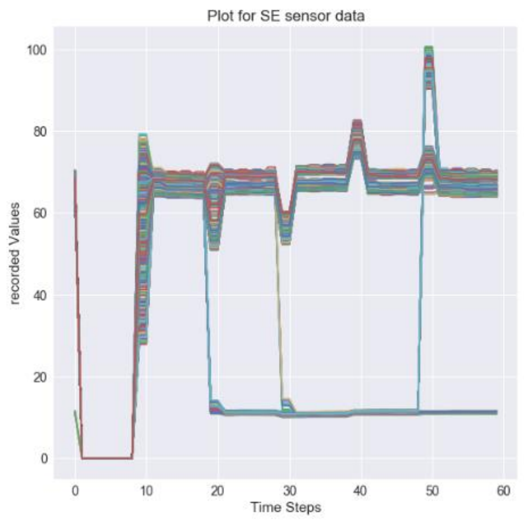
The x-axis represent the time steps, or the number of measurements, and y-axis represents the value of the measurement.

The purpose of this is to see the data and determine whether any individual values seem to be grossly out of range or not conforming to any trend, making them outliers. This also helps to decide what features may be useful in building the classification model.









Screenshots of 4 plots from the code. Rest of the plots can be found in the Jupyter Notebook.

The next part is feature extraction and engineering. Generally features are extracted for optimizing Machine Learning Models. I will use the features to train my dataset .

Feature engineering

Now I define a function to calculate the features that I am going to feed into my model. From the visualizations above, it can be seen that the spread and trend of value fluctuations seem to differ and probably have a bearing on the output class.

I am therefore extracting features that represent this:

- Standard deviation captures the spread of the data
- Skewness is a measure of the asymmetry of the distribution
- Kurtosis is a measure of whether the data is heavy-tailed or light-tailed with respect to a normal distribution, and captures the frequency of extreme deviations from the center of the distribution

along with mean, median, maximum, minimum and periodicity.

Even here I have written a custom function that will extract specified features and store them in a numpy array.

I have extracted 8 features for 17 sensors respectively.

Further I proceed with making the data ready for training in my Machine Learning model. I aggregate the feature data for all the 17 sensors.

Aggregating extracted feature data

Here I am combining the features extracted from all the different variables into one feature matrix, by stacking them horizontally. I use np.hstack( ) to horizontally stack the feature data for all the sensors.

Data Dimensionality

Here I am checking the data dimensionality.

It has 2205 rows, representing 2205 instances or samples. It has 136 columns, representing 17x8 features, as there are 8 features for each of the 17 training variables.

I have used the shape( ) function to display the dimensionality of the aggregated data.

```
(2205, 136)
```

The variable y denotes my target condition which I am trying to predict.

i.e. Hydraulic accumulator / bar

Here I start Machine Learning where I split the data into training and testing samples.

Train-test split

Here I am splitting my feature and class data into train and test samples, so that I have a held-out test set for evaluation. I am specifying a random state for reproducibility of results, and will be doing so for all my classifiers.

I will be splitting the data 80/20 into train and test, as there is a good amount of data (2000~ samples), but not too much or too little.

Standardizing the Data

Then I am standardizing the data in the following manner.
I am standardizing the feature data for both train and test samples. Many machine learning algorithms assume that the input data is standardized, i.e. of 0 mean and unit variance.

Training and testing

As mentioned above
I will be using the following three classification algorithms in my analysis:

- Support Vector Machine
- Random Forest Classifer
- Gradient Boosted Machine

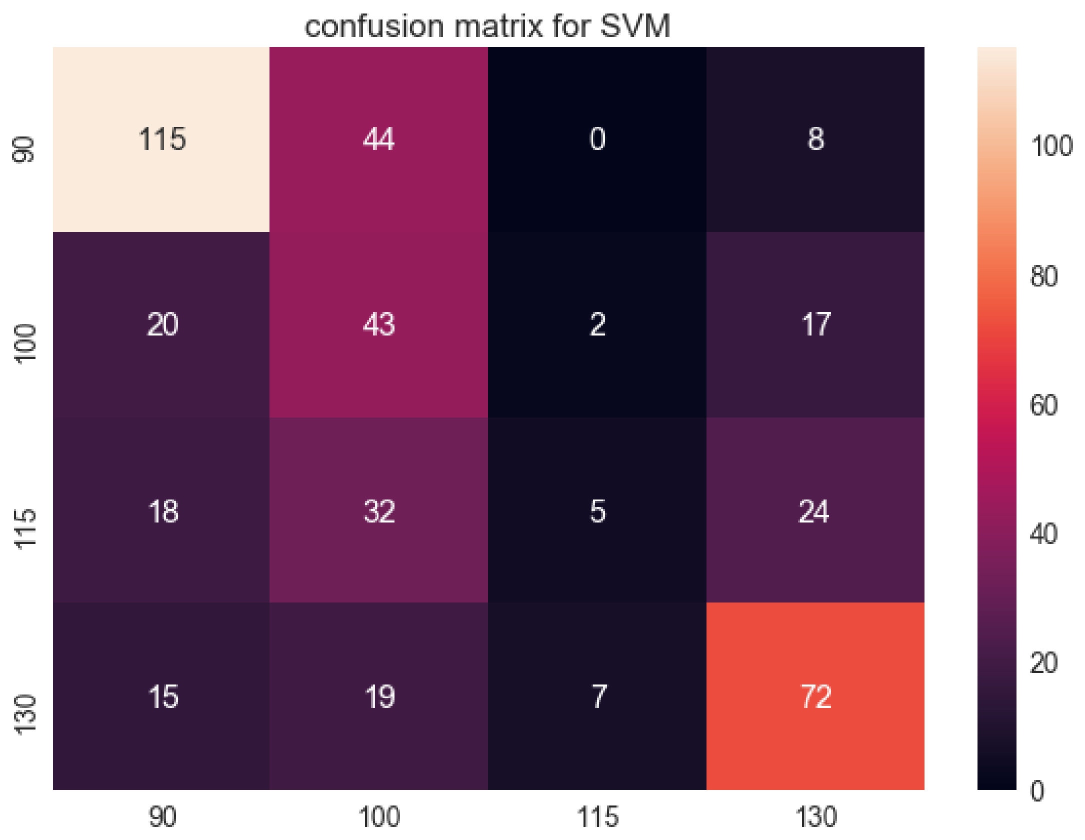
Support Vector Machine

The Support Vector Machine is a fast classifier that maps training examples into a vector space, such that the examples of the separate categories are divided by a gap that is as wide as possible - the computational complexity is relatively low as compared to other classifiers, but it can still perform quite efficiently, especially with parameter tuning and cross-validation. It tends to perform better than classifiers like Random Forests with limited data, but in the presence of adequate data, Random Forests and Gradient Boosted Trees outperform SVMs.

For the purpose of this analysis, I will be starting with a basic linear Support Vector Machine classifier as it is, in order to obtain a baseline against which to compare my other results.

I am calculating the output metrics of accuracy, precision, recall and F1 score and plotting a confusion matrix to see how the classifier performed.

Confusion matrix for Support Vector Machine



Random Forest Classifier

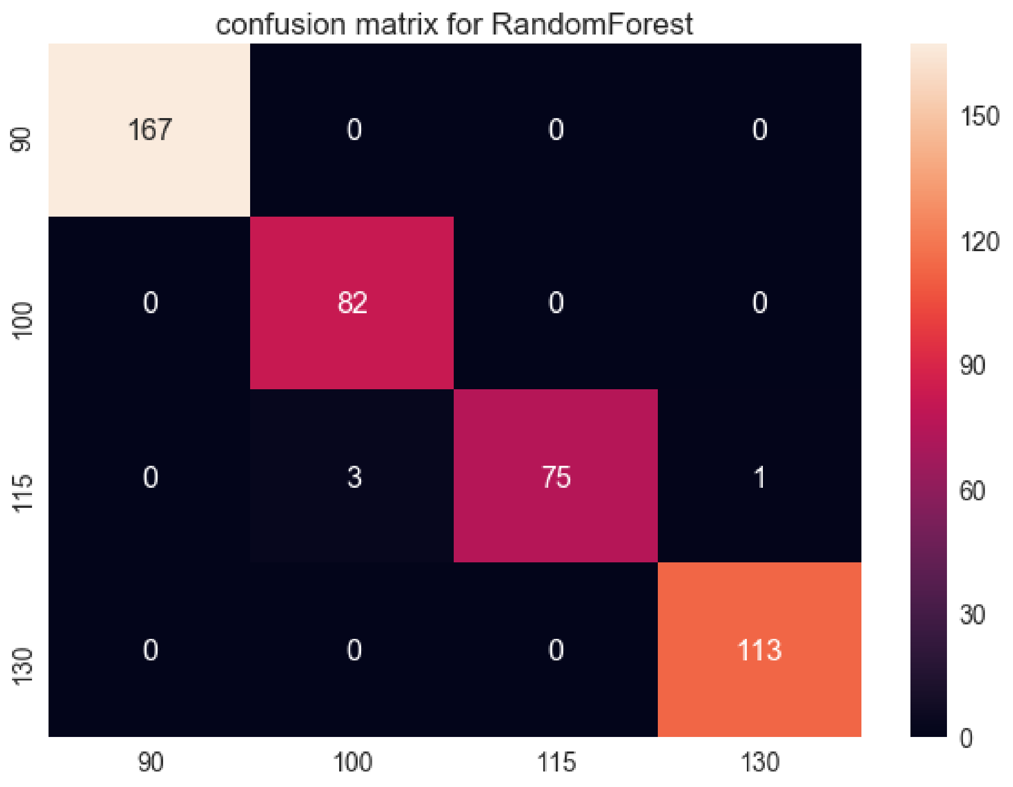
The random forest algorithm is highly scalable to any number of dimensions.

It is an improvement on regular decision trees and it constructs multiple trees through random sampling, and then outputs the mode of the classes of the individual trees, therefore not running the risk of overfitting, which is common in decision trees that get too complex.
The randomness helps to make the model more robust. It is generally seen to be very effective.

In order to determine the best combination of parameters, I am using Grid Search along with Cross-Validation with a predefined parameter grid in order to find the best model.

This algorithm has given me the most accurate results in comparison to the other two classification algorithms.

Confusion matrix for Random Forests

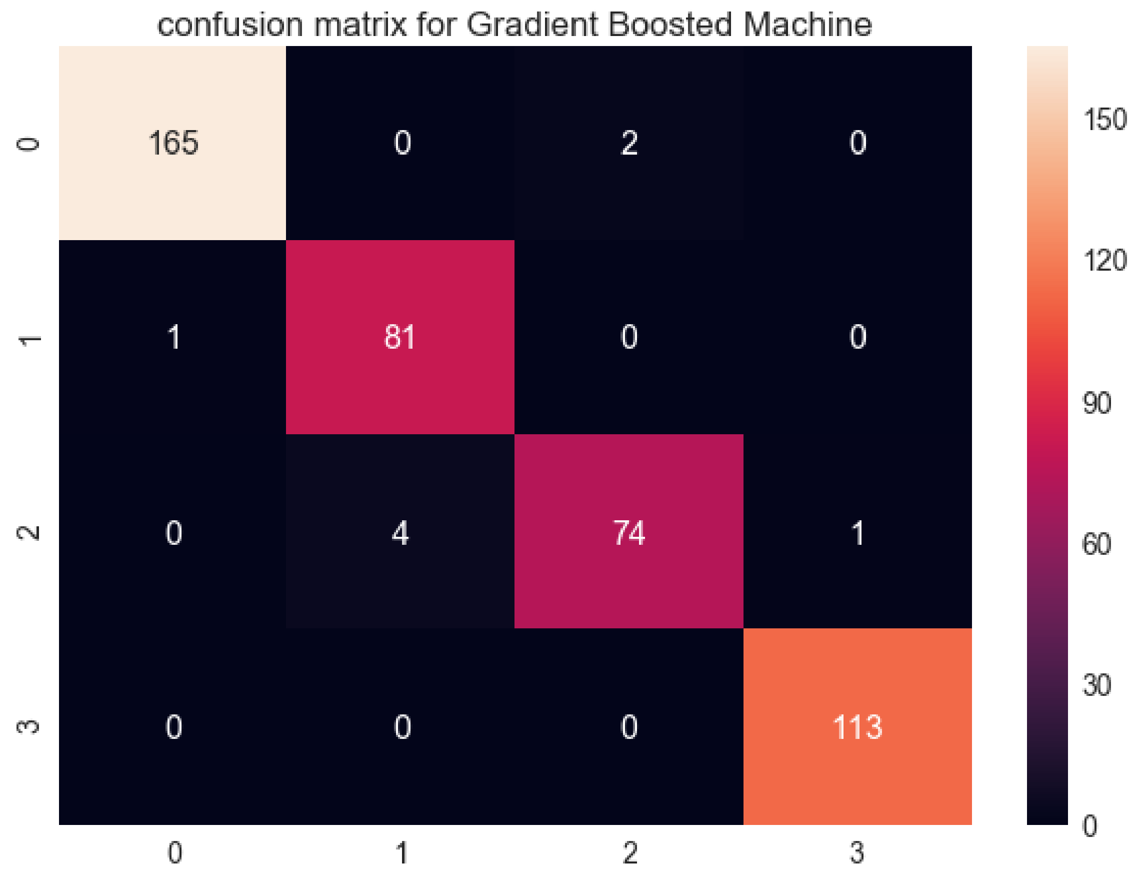


Gradient Boosting Classifier

Gradient Boosting is often effective when a large amount of training data is available. In this method Gradient Boosted Trees are built sequentially, with each new tree helping to correct errors made by previously trained trees, and misclassified instances being given higher priority in training.

Benchmark results tend to show that Gradient Boosted Trees perform better than Random Forest classifiers.

Confusion matrix for Random Forests

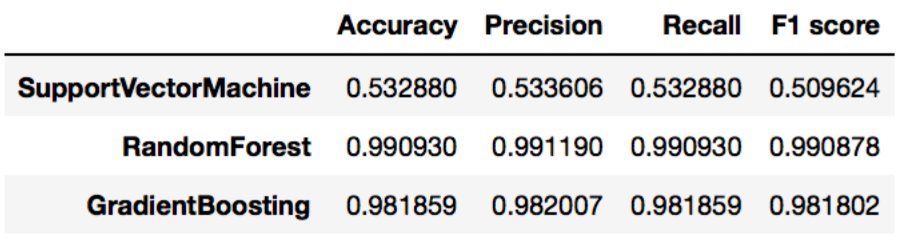


Result Comparison

The linear Support Vector Machine classifier gave an accuracy of 53.3% and an F1 score of 51.0%.

The tuned Random Forest model gave the best results, with an accuracy of 99.1% and F1 score of 99.1%.

The Gradient Boosting classifier performed almost as well as the Random Forest model, with accuracy of 98.2% and F1 score of 98.2%.



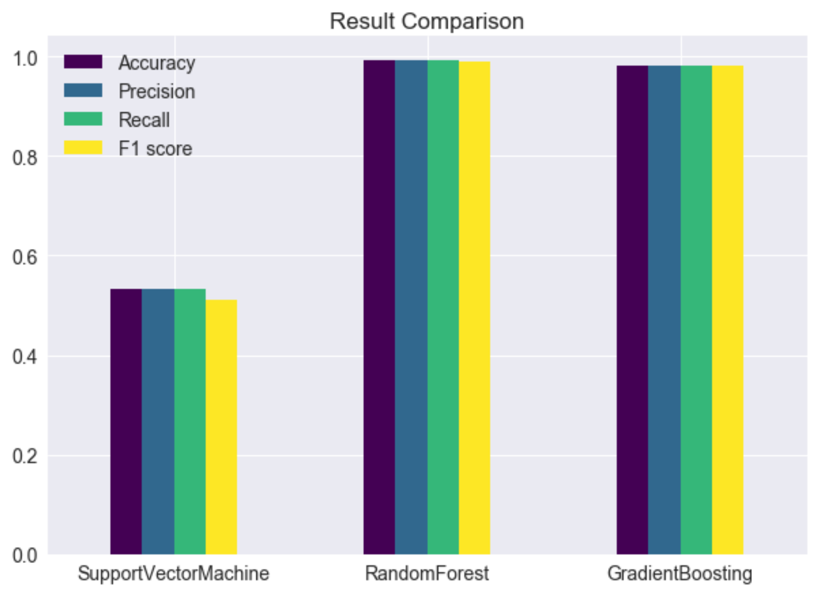
	Accuracy	Precision	Recall	F1 score
SupportVectorMachine	0.532880	0.533606	0.532880	0.509624
RandomForest	0.990930	0.991190	0.990930	0.990878
GradientBoosting	0.981859	0.982007	0.981859	0.981802

Results Visualization

I am storing all the precision, recall, f1 score and accuracy values for all the three classification algorithms in a results dataframe.

I am finally plotting the results dataframe using barplot to visualize these values in a more descriptive manner.

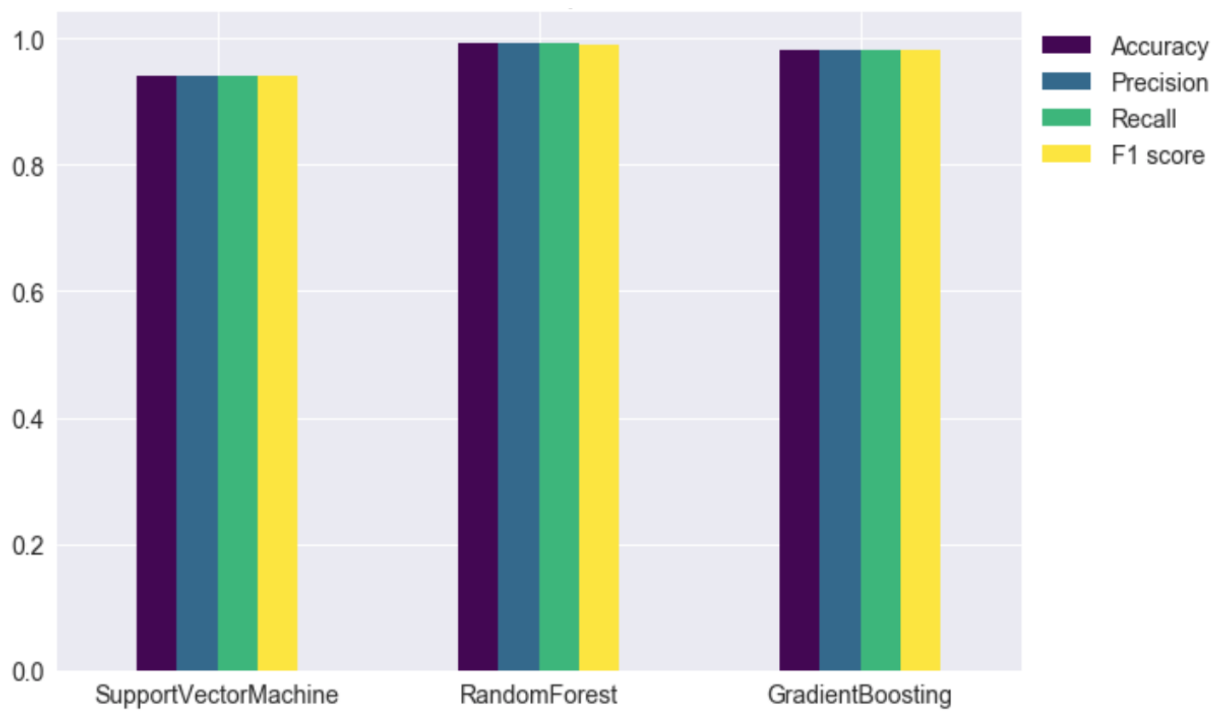
It is clearly evident that SVM does not perform well enough for our data whereas random Forests perform the best and Gradient Boosting also performs equally well.



Recommendation for SVC using Scaled Data

After the train-test split, I scaled the train and test feature data and stored it. A common recommendation made to improve performance on a Support Vector Machine is to use normalized feature vectors, in order to ensure that larger magnitude values of one variable do not dominate another. GBM and RF do not require this as they are tree classification algorithms that attempt to partition and rank features, rather than assigning coefficients or weights.

When using scaled features, the performance of the SVC is significantly higher, with accuracy and F1 score of 0.941, although it is still slightly lower than GBM and RF.


SupportVectorMachine
RandomForest
GradientBoosting
Accuracy
Precision
Recall
F1 score