Final Project - Report

Dalton Murray

College of Business & I.T., Lawrence Technological University

INT 7623, Data Science for Business

Dr. Kamel Rushaidat

April 28, 2024

**Final Project - Report**

**Introduction**

As the world progresses there is an extremely high growth rate in new businesses that are creating more advanced and more complex machinery. This results in an overall better usage of technology, higher efficiency, and higher effectiveness of the utilization of machinery. Unfortunately, because of this, it is often difficult to know the limitations of new machinery, and to remember the limitations of old machinery, and to know when it might fail. It is possible, however, to not need to know these limitations and test them and record this data. With this recorded data, you can then import it into a machine learning algorithm, such as KNN, to then be able to predict if the machinery with the set specifications, such as torque, rotational speed, and temperature will likely result in a failure or not.

This project, specifically, will use a dataset of 10,000 records with the columns UDI, which is just a incremental id, product id which is the product id of the specific machine, the type of machine, the air temperature in kelvin, the process temperature in kelvin, the rotational speed in rpm, the torque in nm, the tool wear per minute, the target fail/no fail, and the specific failure type if there was one.

**Initialize**

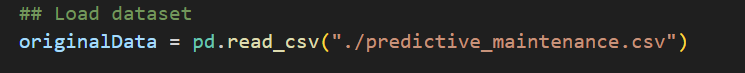
In the initialization step I determine all of the modules/libraries which are needed for the project, and then import them. I also loaded the dataset into a dataframe using pd.read\_csv.

Importing modules:

**A screen shot of a computer

Description automatically generated**

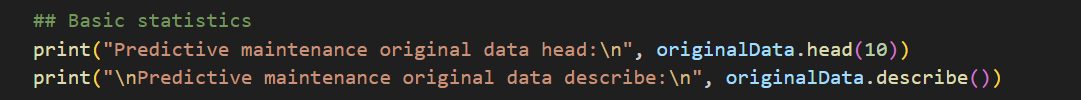
Loading dataset:



**Data insights**

Using the below data and graphs I am able to determine quite a few different things such as a rough visualization/estimate of where a machinery will fail at. Specifically, the pair plot is extremely useful in doing this task. We are able to see that it is more likely than not if a machinery is around the edges of the specific graphs that it is more likely to fail rather than a machinery closer to the middle such as looking at torque. We are able to determine fairly clear relationships with these graphs and determine where a machinery will fail.

Basic statistics:

****

Output:

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

Histograms:

A screen shot of a computer program

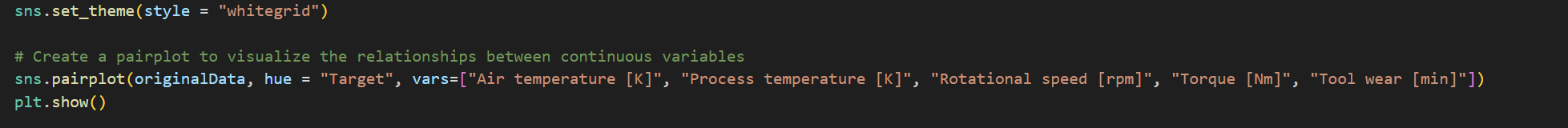
Description automatically generated

Output:

A screenshot of a computer screen

Description automatically generated

Pairplot:



Output:

A screenshot of a computer screen

Description automatically generated

Categorical:

A black background with colorful text

Description automatically generated

Output:

A screenshot of a computer

Description automatically generated

Heatmap:

A screen shot of a computer code

Description automatically generated

Output:

A screenshot of a computer

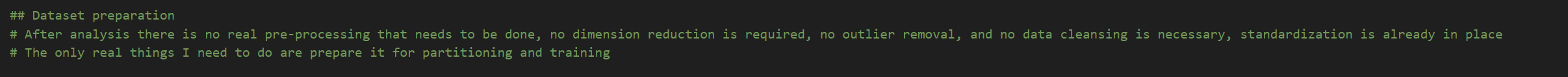
Description automatically generated

**Data preparation**

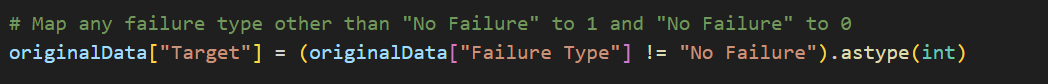
In this step I determined that I do not need to modify any actual data to prepare it for KNN classification.

After this, since my provided dataset I found has many classes for failure type, such as overstrain, power failure, and tool wear failure, I needed to change this so it is only two classes. I determined that anything that isn’t considered a failure is simply considered a failure to be the best action. I did this by changing the original data dataframe and mapping anything other than No Failure to then be a failure. After this, I defined what the numeric features are, such as air temperature, process temperature, rotational speed, torque, and tool wear, and the categorical features as type which is 1 or 0/failure, not failure. Continuing, in the pre-processor, I applied transformers to numerical and categorical features, the StandardScaler and the OneHotEncoder transformers. I then dropped any other data as it isn’t necessary. I then properly applied these transformers to the data using fit\_transform, setting the proper axis, and dropping data now no longer necessary as it has been transformed. Continuing further, I extracted feature names using .named\_transformers\_[“Cat”].get\_feature\_names\_out and making this transformedFeatureNames, I then turned this into a list and added the numeric features to it and set this to newFeatureNames. Finally, I converted this back into dataframes.

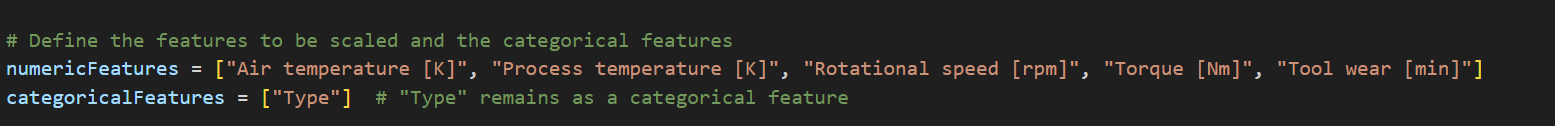
Start/no real changes to original dataset needed:

****

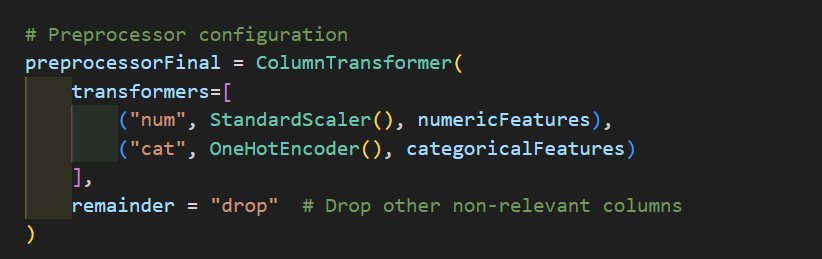
Failure/classes mapping:



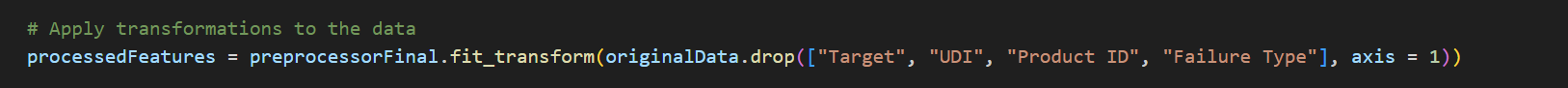
Define numeric features and categorical features:



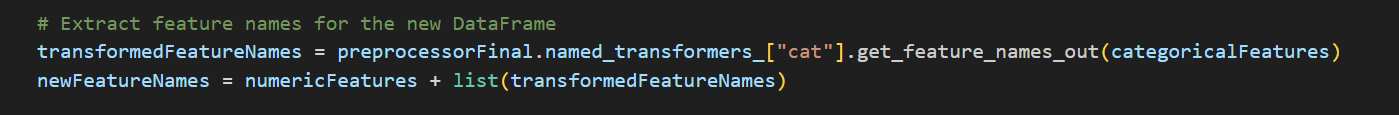
Preprocessing/transformers:



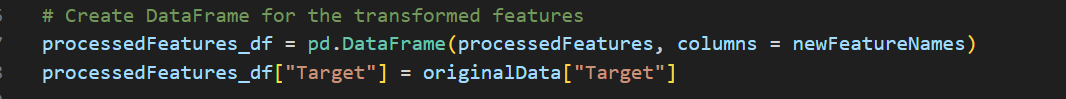
Apply transformations:



Extracting feature names:



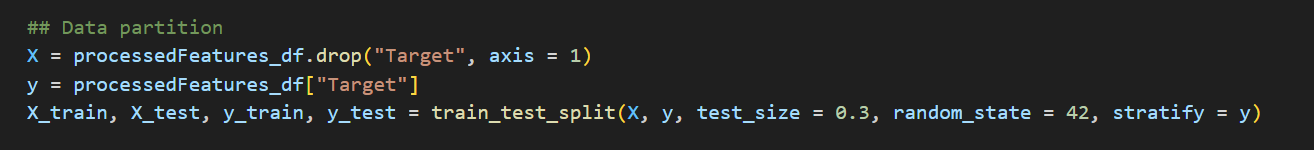
Creating dataframe from extracted features:



**Data partitioning**

My data partitioning step was very simple, I determined that it would be best to do a 70/30 split for training and testing. I made an x and y dataframe with the specific features I wanted for both, and then used train\_test\_split to randomly and properly split the data using the test\_size 0.3 to split it into 30% test.

Splitting data 70% training 30% testing/validation:

****

**Values of K**

When thinking about the values of K, I took more of a range/spread approach. I started off by using 1, I then jumped to 3, then 5, then 7, then 10, then 15. I felt that these numbers would be best because it gave me a wide range of numbers, I knew that the ideal model of nearest neighbors wouldn’t be too large, which I felt 15 was starting to get there, or too small, and I felt 1 was too small, however, I included them anyways because it would allow me to prove this.

Setting a range of values to K to determine which 3 are the top ones:

**A black background with white numbers

Description automatically generated**

**Training/Testing:**

I took a more simple approach in my training and testing phase. Since everything is already available to me, I simply made a for loop to then train and test each of rhtme using KNeigborsClassifier, and fit, which would properly train and test everything for me. This does it based on both X\_train, and y\_train, which has been set previously in my data partitioning stage to include a training data and test data split. As a result of my usage of libraries, I didn’t have to make algorithms to calculate distances, finding the nearest neighbors, or making predictions. However, I will still explain them.

Calculate the distance: Calculating the distance between the data point that needs to be classified, or predicted, and all of the other data points in the data set is the first step. This is so that we can determine how close a new data point is to the existing points. There are a few different distance measurements which can be used for this such as Euclidean, Manhattan, or Hammering distance.

Find the nearest neighbors: Finding the nearest neighbors is the second step. Once you have calculated the distances you can then sort these distances, and then it identifies the k nearest data points, with k being our defined parameter, and the closest neighbor. If you have too small of a k value it can make the model too sensitive to noise, and with too large of a k you include points that are too far away from k which makes the other points of k have less influence to a point that is closer.

Making predictions: In making predictions you have classification tasks, and regression tasks. Classification tasks uses KNN to assign the new data points to the class most common among its k nearest neighbors, each neighbor will typically get an equal vote, however, these votes can be weighted to give closer points a higher influence. In regression tasks, the prediction is the average or median of the values of its k nearest neighbors, and can be a basic average or weighted on distance.

Comparing accuracy of all models:

K = 1: 0.967

K = 3: 0.974

K = 5: 0.971

K = 7: 0.971

K = 10: 0.970

K = 15: 0.969

With these results, we can see why having a large spread of K values is important for us to determine which is the most accurate. The most accurate is k = 3, and the least accurate is k = 1.

**A computer code on a black background

Description automatically generated**

**Evaluation**

Formatting outputs for evaluation:

**A screen shot of a computer program

Description automatically generated**

Output:

A screenshot of a computer error

Description automatically generated

A screen shot of a computer error

Description automatically generated

A computer screen with white text

Description automatically generated

A computer screen shot of a error

Description automatically generated

A computer screen shot of a error

Description automatically generated

A computer screen shot of a error

Description automatically generated

Comparing results, precision:

K = 1: 0.532

K = 3: 0.745

K = 5: 0.737

K = 7: 0.800

K = 10: 0.818

K = 15: 0.789

The highest precision is k = 10, and the lowest precision is k = 1.

Comparing results, recall:

K = 1: 0.394

K = 3: 0.365

K = 5: 0.269

K = 7: 0.231

K = 10: 0.173

K = 15: 0.144

The highest recall is k = 1, and the lowest recall is k = 15.

Comparing results, misclassification error:

K = 1: 0.033

K = 3: 0.026

K = 5: 0.029

K = 7: 0.029

K = 10: 0.030

K = 15: 0.031

Comparing results, confusion matrix:

K = 1:

Predicted: No Predicted: Yes

Actual: No 2860 36

Actual: Yes 63 41

K = 3:

Predicted: No Predicted: Yes

Actual: No 2883 13

Actual: Yes 66 38

K = 5:

Predicted: No Predicted: Yes

Actual: No 2886 10

Actual: Yes 76 28

K = 7:

Predicted: No Predicted: Yes

Actual: No 2890 6

Actual: Yes 80 24

K = 10:

Predicted: No Predicted: Yes

Actual: No 2892 4

Actual: Yes 86 18

K = 15:

Predicted: No Predicted: Yes

Actual: No 2892 4

Actual: Yes 89 15

**Best model**

There are best models for different aspects of this test. If precision is the most important factor, then k = 10 is the best. If recall is the most important, then k = 1 is the best. If the overall error rate is the most important, then k = 3 is the best.

Overall, I believe k = 10 is the best. It is the most balanced choice of the top 3, it minimizes false positives the most, has a reasonable error rate, has a higher precision but a lower recall rate.

**Conclusion**

It is possible for a better model to be made, however, using k = 10 is a well-balanced model that would provide great benefits. Using different algorithms and fine-tuning it could result in even better results.

**References**

Band, A. (2020, May 23). How to find the optimal value of K in KNN?. Towards Data Science. <https://towardsdatascience.com/how-to-find-the-optimal-value-of-k-in-knn-35d936e554eb>

Bansal, S. (2021, November 6). Machine Predictive Maintenance Classification. Kaggle. <https://www.kaggle.com/datasets/shivamb/machine-predictive-maintenance-classification>

Grus, J. (2019). Data Science from Scratch: First Principles with Python (2nd ed.).

IBM. (n.d.). What is the k-nearest neighbors algorithm?. <https://www.ibm.com/topics/knn>

Srivastava, T. (2018, March 25). A Complete Guide to K-Nearest Neighbors (Updated 2024). <https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/>

Yashasvic, C. (2023, December 14). How to Find The Optimal Value of K in KNN. GeeksforGeeks. <https://www.geeksforgeeks.org/how-to-find-the-optimal-value-of-k-in-knn/>

I pledge that on all academic work that I submit, I will neither give nor receive unauthorized aid, nor will I present another person's work as my own.

Dalton Murray