SOFTWARE ENGINEERING INTERN AT PLURAL TECHNOLOGY Pvt Ltd.

An Internship Report

DR. VISHWANATH KARAD

MIT WORLD PEACE UNIVERSITY

Submitted by,

Aditya Mehta (1032180599)

Rajat Belgundi(1032180203)

Sumanth Vullamparthi(1032180622)

Yash Kulkarni(1032180209)

Under the supervision of

Mr. Deenanath Yadav

(Project Manager)

and

Prof. Trupti Baraskar

School of Computer Science Engineering

MIT World Peace University

Kothrud, Pune - 411038

ACKNOWLEDGEMENT

I take this opportunity to express my sincere gratitude towards Mr. Sunil Savaram and Mr.Nitin Baraskar, Plural Technology Pvt Ltd, Pune for taking me as an intern and allowing me to complete my Internship at Plural Technology Pvt Ltd, Pune. I would like to express my gratitude towards Mr. Amit and Ms Shubhangi for providing great guidance on Aras Innovator Software in the PLM domain. And also to Mr. Ganesh Motekar, Plural Technology Pvt Ltd, Pune for imparting knowledge on Jira Software. Thanks to HR coordinator Mrs. Sheetal Sadhu, Plural Technology Pvt Ltd, Pune for making us feel comfortable to express and solve our issues. I am grateful to Mr. Deenanath Yadav and Mr. Ganesh Motekar, Plural Technology Pvt Ltd, Pune for guiding and helping me throughout my project. It is not without them that I could have been able to complete my internship at the innovative environment of Plural Technology Pvt Ltd, Pune. I would like to express my deep gratitude for their support and help extended during the entire period of Internship. I would like to use this platform to express my heartfelt gratitude to the HOS, Prof. Dr. Bedekar and College Supervisor Prof. Trupti Bhaskar for the kind support and inspiration which has immensely strengthened my confidence during the period of internship. Finally, I would like to extend my gratitude towards my team of fellow Interns at Plural Technology Pvt Ltd, Pune who together with me made it possible to complete the projects and research in our respective fields.

ABSTRACT

Machine Learning is the trend in recent years. A variety of organizations, industries, companies have started adopting ML in their activities to gain a competitive edge in the market. Machine Learning is being used almost everywhere and the results are quite satisfying. Machine Learning in PLM(Product Lifecycle Management) is a concept which is in the initial stages, hence, this is a great opportunity to do so. Many manufacturing industries, automobile industries and other product based organizations have been collecting data regarding their product like: product details, product history, product usage, environment compatibility (product) and so on. The amount of data available is huge and the current trend points to the fact that not making efficient use of the data is a sign of losing the edge over others in business. PLM solutions when combined with ML can achieve the greatest of heights, thus, making it an attractive opportunity for organizations to adopt ML in PLM. There are a variety of cloud services which make MLOps quite easy to comprehend and implement. With the use of these facilities, ML can be integrated into PLM solutions, thus, beneficial for both companies and clients. One of the key benefits of integrating ML into PLM solutions is reduction of lead time of products This project gives a good idea of how product details like the dimensions of the product as data can be used and few manual task can be automated, thus, leading to reduction in time required, which eventually translates to financial benefit ("Time is money"). This project has a web application so that getting predictions for users can be both easy and interactive. The message conveyed via means of this project that organizations can adopt and use ML in their work easily as a result of these Cloud Services. The use case implemented here tries to depict how the implementation in local Python can be taken to the cloud via AWS and how MLOps can easily be adopted by companies.

Keywords/Abbreviations

- ML : Machine Learning
- PLM : Product Lifecycle Management
- KNN : K Nearest Neighbours
- SVM : Support Vector Machine
- XGBoost : Extreme Gradient Boosting
- AWS : Amazon Web Services
- MLOps
- Docker Container
- AWS ECR : Amazon Web Services Elastic Container Registry
- AWS S3 : S3 Bucket for storage
- AWS Lambda
- AWS API Gateway
- AWS SageMaker

TABLE OF FIGURES:-

TABLE OF CONTENTS:-

Chapter 1 INTRODUCTION

1.1 Background

1.1.1 Machine learning in PLM:

Machine learning the new “buzzword” is helping all industries to leverage their resources and perform more efficiently than before.
Using machine learning techniques in Product Life Cycle management will greatly help industries to predict future product behaviour and with this develop better products and services. The long term hope is to achieve:

a) Increased speed and accuracy by which product developing companies can create new knowledge about their own products and markets based on actual data collected from production and end-users.
b) Maximize the value of each product individually by using every product’s production and usage history to identify and predict quality problems in the whole product lifecycle.

1.1.2 Classification of components:

Complex products are composed of components which need to be classified beforehand. The classification of components helps the manufacturing team to get a good idea of the requirement.
Classification requirements may vary from one product to another.
Complex products like vehicles have many components and a lot of data about the components which is ever Increasing which is why automating the classification of components is necessary.
Our concept relies on the automated classification of product data using supervised machine learning techniques.
Similarity based component retrieval by using classification techniques is the use case depicted in this project.

1.1.3 AWS (Amazon Web Services)

Amazon Web Services(AWS) is a subsidiary of Amazon which provides on-demand cloud computing platforms and APIs to individuals, companies, and governments, on a metered pay-as-you-go basis. AWS offers a wide variety of cloud based services which

helps companies to take their workflow to the cloud with ease. MLOps which seemed a daunting task has become quite simple for companies to implement with AWS.
The use case implemented here makes use of AWS services like S3, Lambda, SageMaker,API Gateway, ECR.

1.2 Motivation

- Machine learning in PLM is not much explored and with the capabilities ML has shown, involving ML in PLM is itself fascinating and challenging.
- Assigning components to the appropriate classes is difficult to do manually especially for product orders in bulk. Hence, Machine Learning can be used to classify components into their proper classes.
- To design a web app wherein, the manufacturers can enter the component specifications and can receive the class of the component in return.

1.3 Scope

This project aims to use product dimensional data and classify the component into its appropriate class.
The classification task is performed by using supervised Machine Learning techniques.

CHAPTER 2 Literature Review

Modern machine learning methods have the potential to supply industrial product lifecycle management (PLM) with automated classification of product components. Classification of product components helps organizations reduce the lead time for their product.
In the competitive world, organizations have realized the significance of reuse of product and parts data. Effective and efficient reuse of product data is becoming crucial for product innovation, which is essential for long term success.

Essentially, the inability to search existing parts quickly leads to increased parts cost, bloated parts inventory, and higher product performance risk, while worsening the search problem further due to the sheer number of parts that need to be searched.
By use of several classes, all components can be differentiated according to their properties. Due to partitioning, retrieving components can be supported by a given classification.
In addition to finding existing components, classification can be helpful to support integrated product development processes (PDPs) during and after the design phase by linking different storage systems and methodologies.

Machine learning in parts classification helps industries to reuse product data and quickly get the components (along with their class) for a specific order. This leads to reduction of about 50% to 60% of time taken for parts classification, which translates into proportionate savings in cost.
The idea of similarity based component retrieval (for ex. spare parts) is the use case explored in this project.
Manually classifying parts is not feasible in the current times for bulk orders. Also, the fact that computers can interpret a large number of attributes faster than humans can transform parts classification into a Machine Learning problem. The ML approach to this problem has technical as well as financial benefits. Hence, the use of Machine Learning proves beneficial for parts classification.

Used Tools and Technologies for Implementation

2.1.1 Platforms

1. AWS SageMaker (Jupyter Notebook instance, endpoints)
2. AWS S3 Storage
3. AWS ECR(Repository) & Docker Image
4. AWS Lambda and API Gateway
5. Google Colab (Local Python Implementation)
6. GitHub

2.1.2 Programming Languages

1. Python
2. React JS

2.1.3. Libraries and packages

1. Scikit-Learn
2. Pandas
3. Numpy
4. Matplotlib
5. Seaborn
6. Boto3
7. Sagemaker

2.1.4. Algorithms

1. XGBoost
2. Logistic Regression
3. K Nearest Neighbors (KNN)
4. SVM (Support Vector Machine Classifier

CHAPTER 3 METHODOLOGY

3.1 Data Information

Source of data: Data collected by referred research paper via GitHub.
Dataset information:

The dataset in this use case consists of data of brake pads for disc brakes of a car. The brake pads either belong to the front or rear axle (vorderachse or hinterachse).The data contains dimensional attributes like length, width and thickness (in millimetres) of brake pads.

The entire dataset comprises 4001 entries and 5 columns, some of which are null (NaN). On removal of NaN values from the target feature (classOfPart) we have a dataset consisting of 1440 entries and 5 columns.

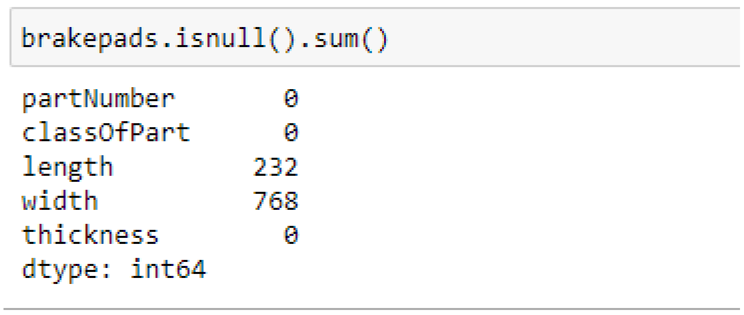
Attributes:
1. Part number (unique identification)
2. Length
3. Width
4. Thickness
5. Class Of Part (target feature)

3.2 Data Preprocessing:

We have dropped entries having more than one missing value (threshold = 4 non NA values).

This results in 1275 samples of brake pads.

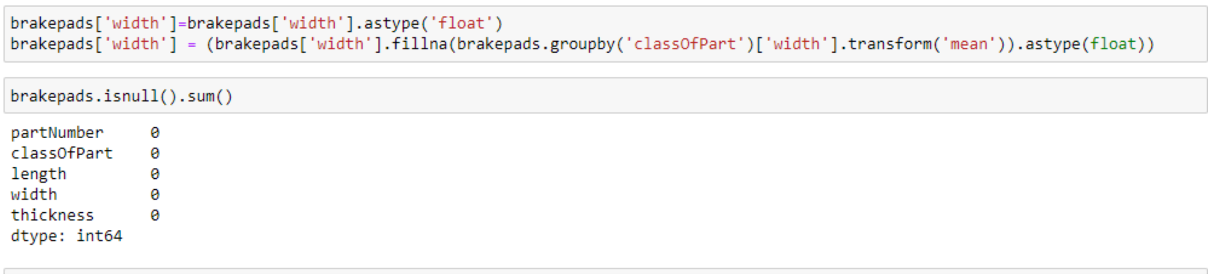
Checking missing values in the attributes:



```
brakepads.isnull().sum()
```

```
partNumber       0
classOfPart      0
length         232
width          768
thickness        0
dtype: int64
```

In order to handle missing values in the length and width features we have replaced nan values with column wise mean of length and width with respect to the classOfPart (brake pad class).



```
brakepads['width']=brakepads['width'].astype('float')
brakepads['width'] = (brakepads['width'].fillna(brakepads.groupby('classOfPart')['width'].transform('mean')).astype(float))
```

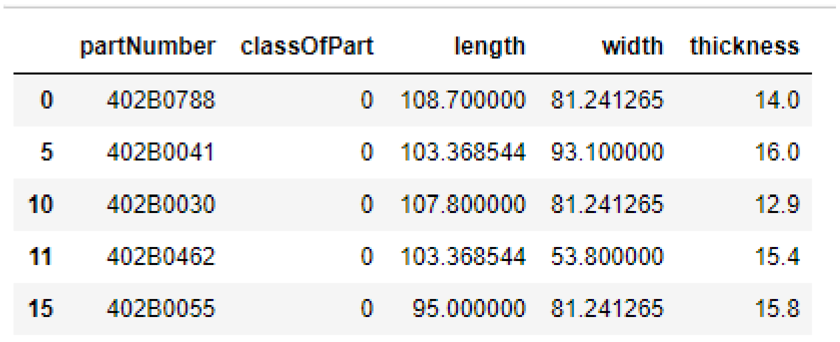
```
brakepads.isnull().sum()
```

```
partNumber     0
classOfPart    0
length         0
width          0
thickness      0
dtype: int64
```

Thus, we have filled in the missing values.

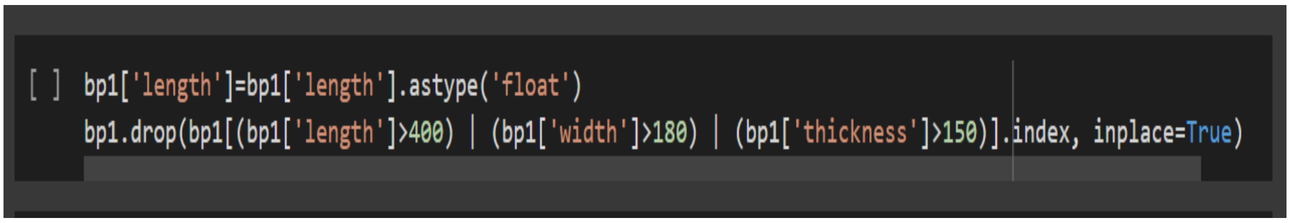
We also replace the categorical classes in the target feature with 0 and 1 as we have only two classes of brake pads.

This is the final dataset which we will work with.



	partNumber	classOfPart	length	width	thickness
0	402B0788	0	108.700000	81.241265	14.0
5	402B0041	0	103.368544	93.100000	16.0
10	402B0030	0	107.800000	81.241265	12.9
11	402B0462	0	103.368544	53.800000	15.4
15	402B0055	0	95.000000	81.241265	15.8

There were a few outliers in the data set which we removed as follows:



```
[ ]  bp1['length']=bp1['length'].astype('float')
     bp1.drop(bp1[(bp1['length']>400) | (bp1['width']>180) | (bp1['thickness']>150)].index, inplace=True)
```

3.3 Data Visualization

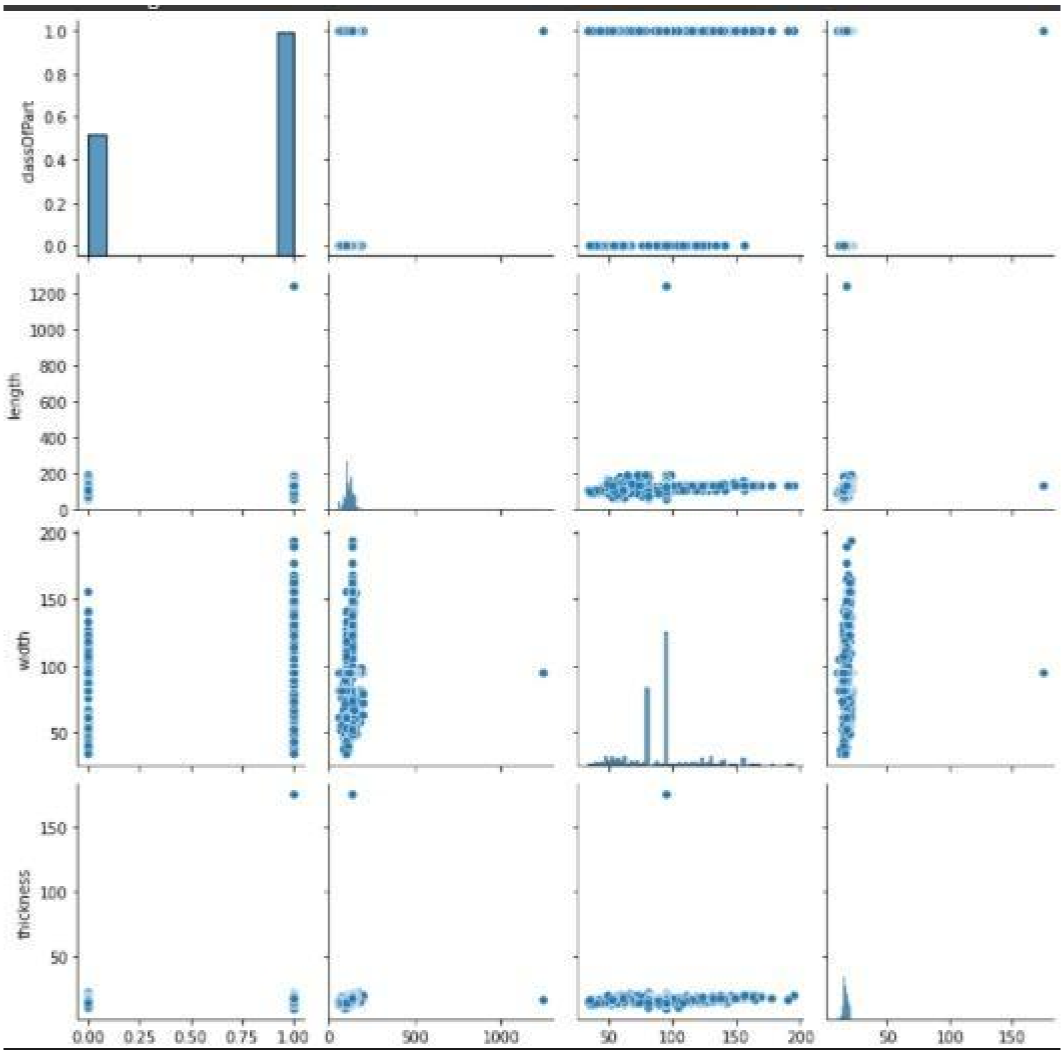


Fig 1: Overall data visualization

Clusters of data based on length and width:

Used Elbow Method to determine appropriate clusters

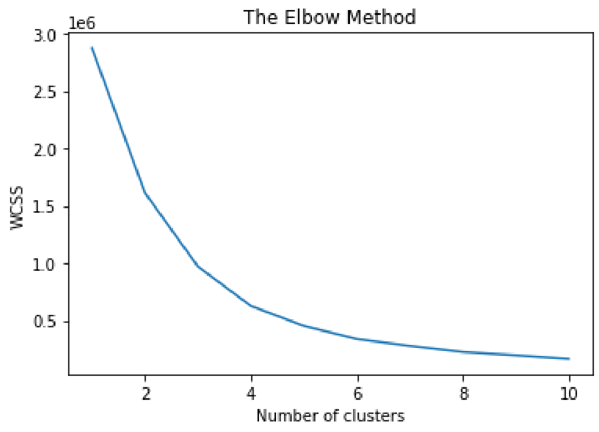


Fig 2: Elbow Method(width-length)

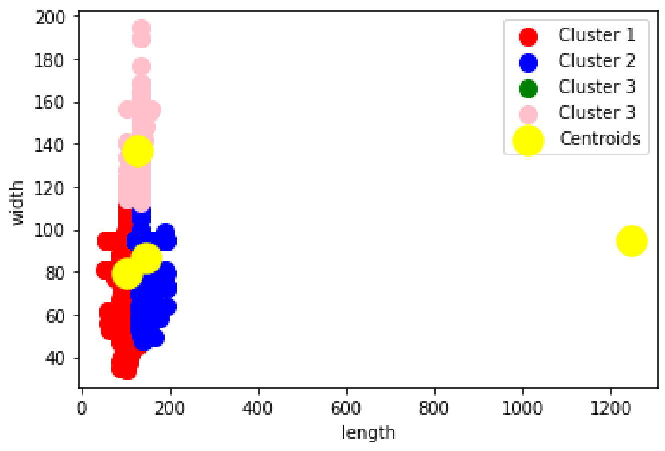


Fig 3: Clustering result

Clusters of length-thickness

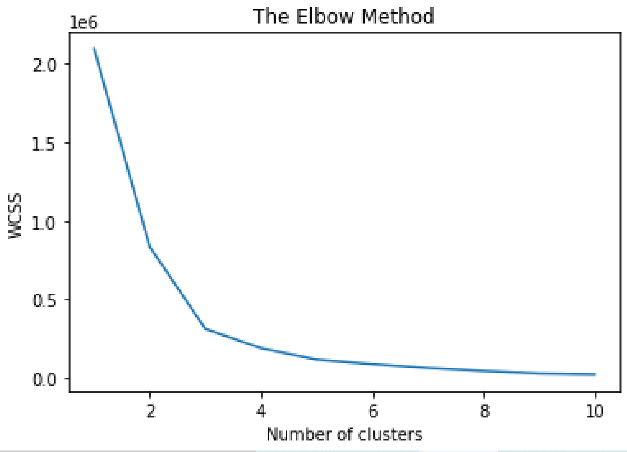


Fig 4: Elbow Method(thickness-length)

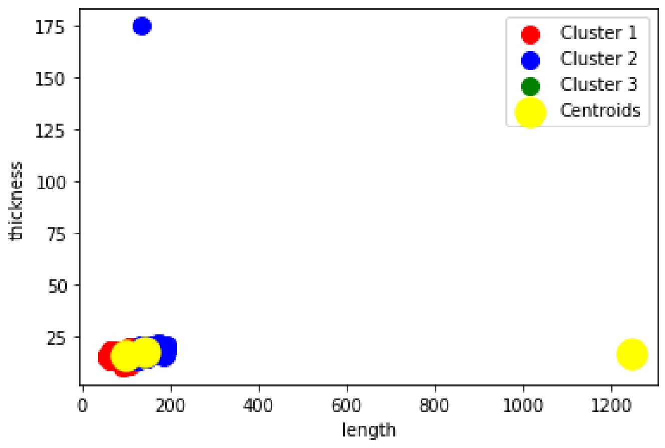


Fig 5: Clustering result

Clusters of width thickness:

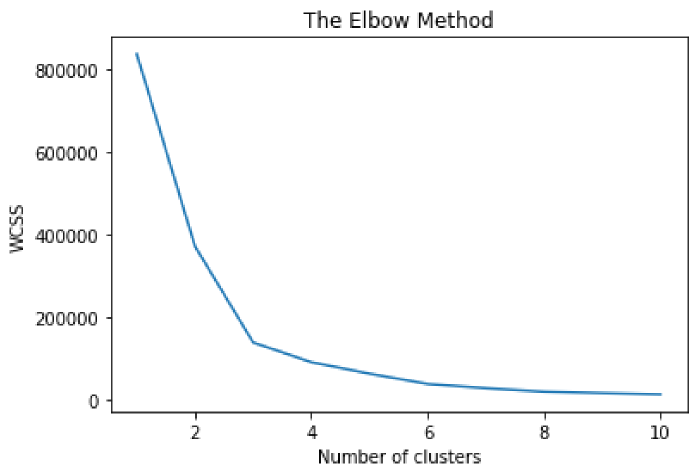


Fig 6: Elbow Method(thickness-width)

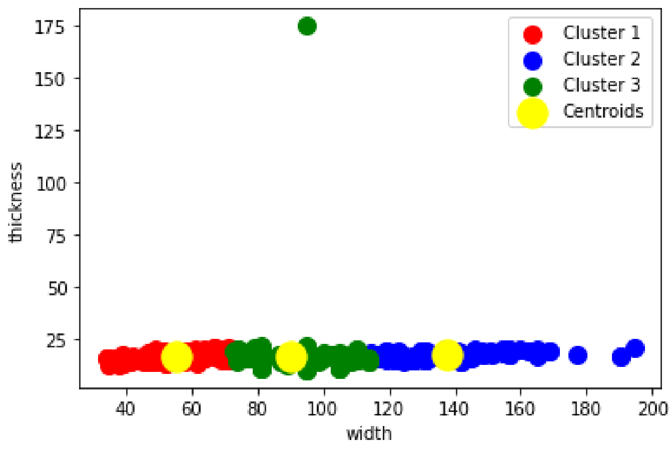


Fig 7: Clustering result

Observation :a] Based on width the data points have a maximum thickness of 25mm(outliers effect not shown here)

Removing Outliers:

As we can spot a few outliers from the clusters obtained above, we have to remove them as they hamper model performance.

Thus, after removing outliers our data looks as follows:

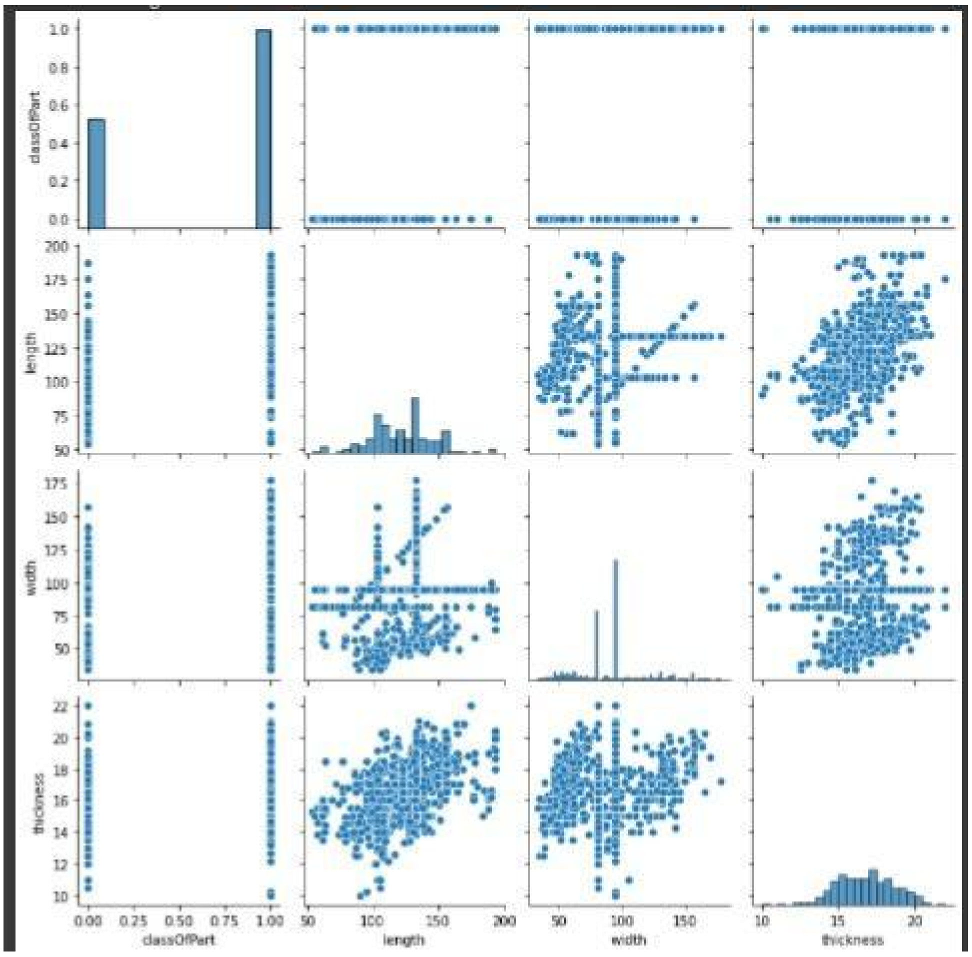


Fig 8 : Overall data (without outliers)

Target feature based on length and width:

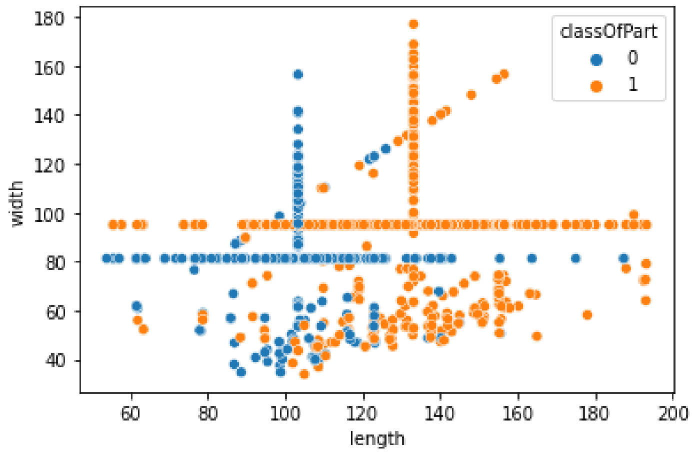


Fig 9 : width vs length

Observation: a] Majority of the data points belonging to class 1(front axle) have length > 120mm

Target feature based on length and thickness:

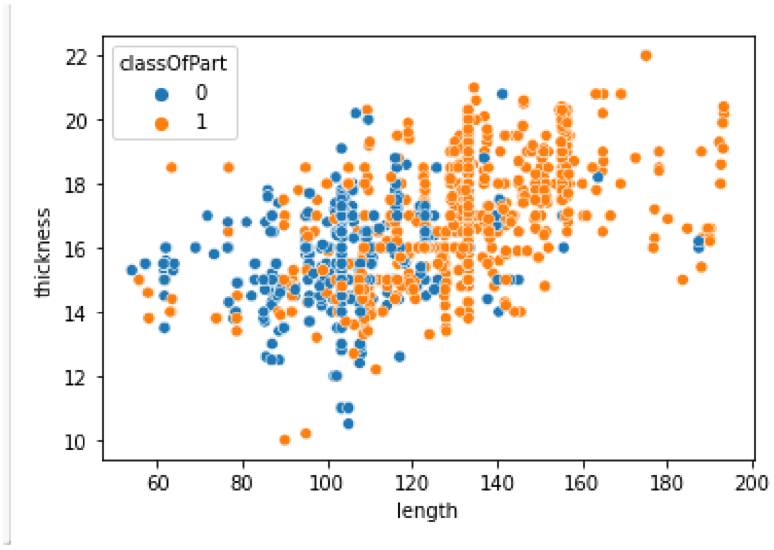


Fig 10: thickness vs length

Observation : a] Based on this plot, front axle(class 1) has a slightly greater thickness than rear axle (class 0)

Target feature based on width and thickness:

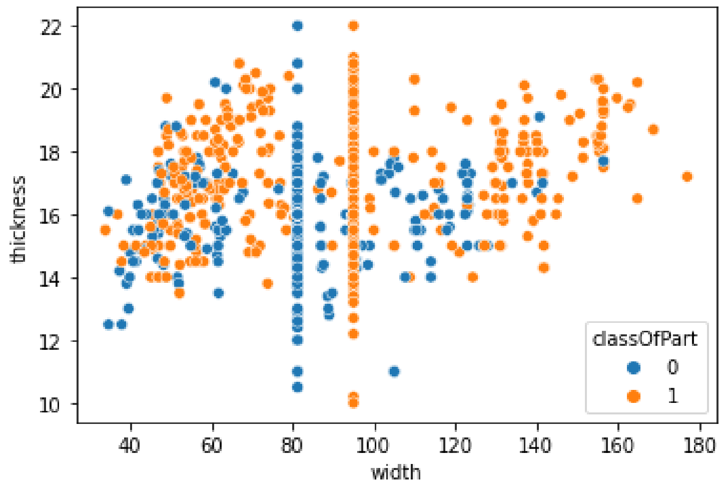


Fig 11: thickness vs width

Observation :a] Very few samples of rear axle(class 0) having width > 125 mm are present

3.4 Machine Learning Algorithms

3.4.1 Logistic Regression :

Logistic Regression is a core supervised learning technique used to solve classification problems.

Logistic Regression has been built up from Linear Regression.

This can be used to solve binary classification problems (0 & 1), as well as multi-class classification problems.

In Logistic Regression, the sigmoid function is applied to the output of Linear Regression which makes sure that the output of the algorithm is the probability that the event occurs (always between 0 & 1). By setting the appropriate threshold (default=0.5) the algorithm can predict the classes 0 or 1.

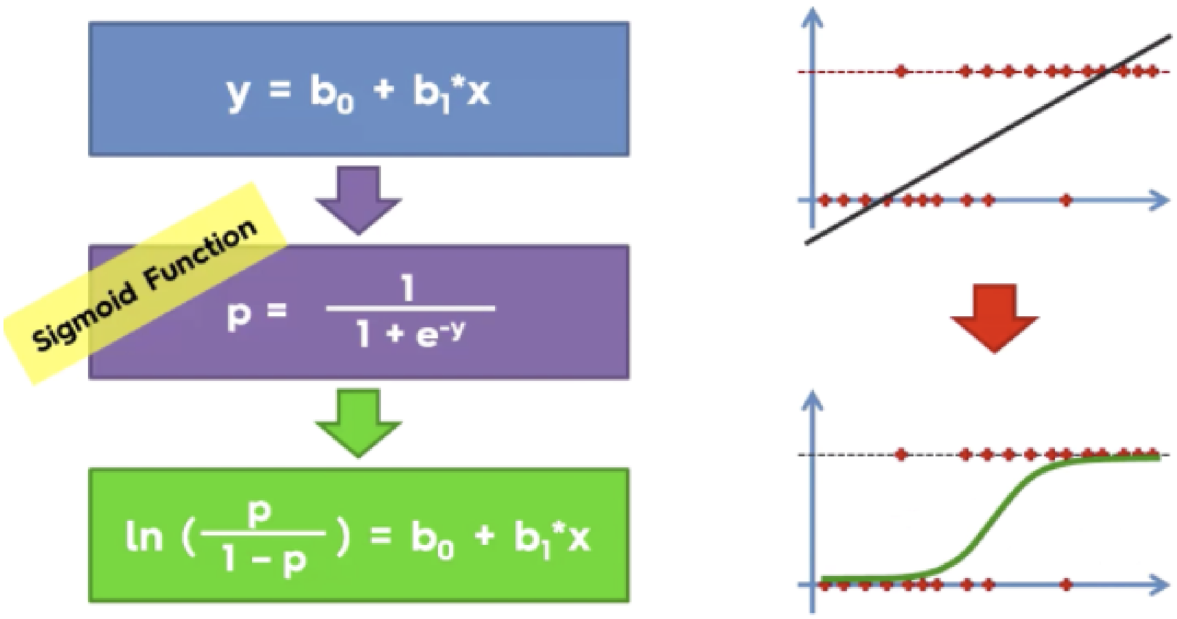

$y = b_0 + b_1{}^*x$
Sigmoid Function
$p = \frac{1}{1 + e^{-y}}$
$\ln\left(\frac{p}{1-p}\right) = b_0 + b_1{}^*x$

Fig 12: Logistic Regression

Logistic Regression is a very good algorithm to start with for a classification task.

Python Implementation: scikit-learn’s linear model package has Logistic Regression module (sklearn.linear_model.LogisticRegression)

3.4.2 K Nearest Neighbors(KNN):

K Nearest Neighbors (popularly KNN) is another supervised learning technique used for classification use cases.

KNN assumes that similar things are closer to each other.

KNN Algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean/Manhattan distance of **K** number of neighbors**.**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean/Manhattan distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbors is maximum.
- **Step-6:** Our model is ready.

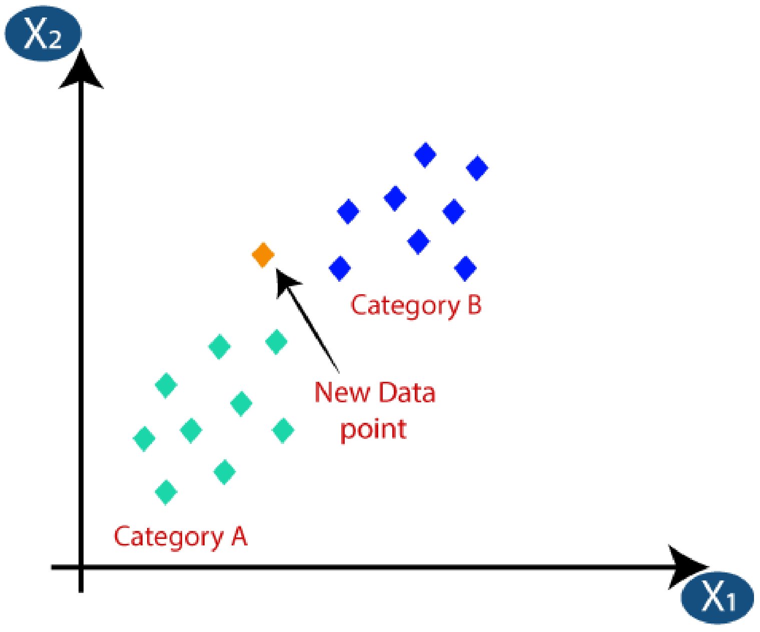


Fig 13 : KNN(1)

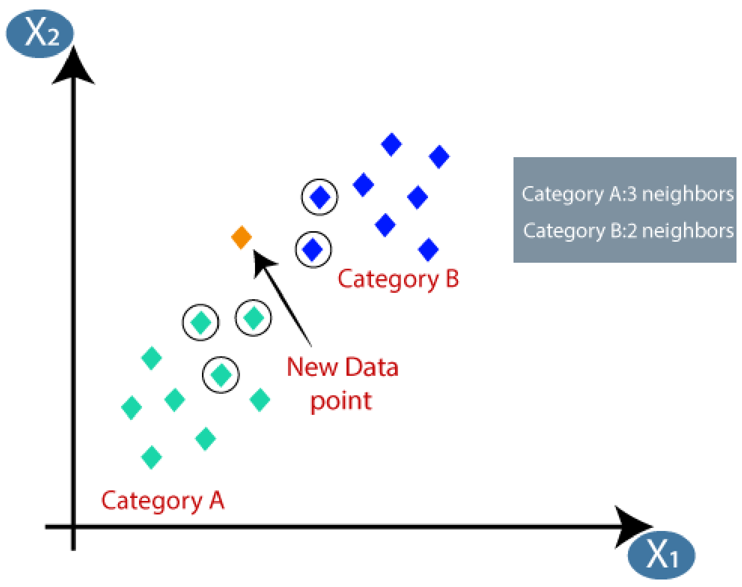


Fig 14: KNN(2)

KNN is sensitive to outliers and hence, its classification boundaries can change based on the presence of outliers.

Python Implementation : scikit-learn's neighbors package has KNeighborsClassifier(sklearn.neighbors.KNeighborsClassifier).

3.4.3 SVM(Support Vector Machine):

SVM is a supervised machine learning algorithm which is capable of performing both regression and classification tasks. Mostly it is used for classification tasks. SVM model works by plotting our data points in an n dimensional space (n is number of features).The classification task is performed by finding the appropriate hyper-plane for our data (the one which separates our data really well).

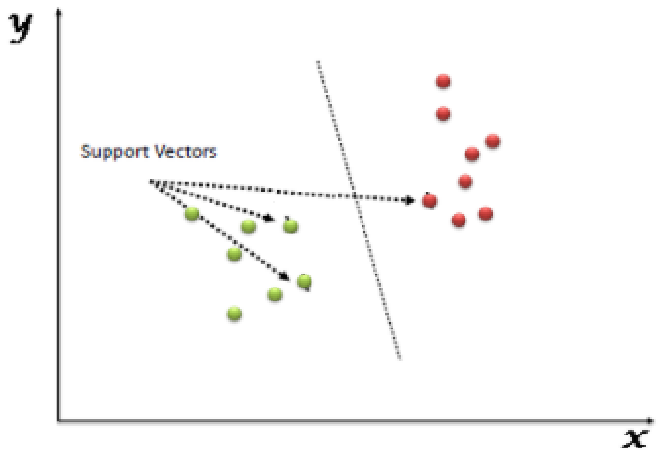


Fig 15 SVM

The hyper-plane (straight line in above 2D graph) is formed with the help of points known as support vectors. The support vectors are the points which are the nearest to the possible hyper-planes (distinguishing lines). The best fit hyper-plane is the one which maximises the distance between nearest points of either class (Red and Green). This maximum distance is referred to as margin.
SVM has a good misclassified points handling capability. As the best hyper-plane is formed by maximum margin rule, the outliers will not have any effect on the classification task.
Also, the maximum margin rule helps in avoiding as much as misclassification as possible as it maximises the distance between the nearest points of either class. This makes SVM a very popular algorithm.
SVM has a special trick called the "Kernel Trick" which helps SVM to classify points which are not linearly separable (it uses complex data transformations).
Outliers can affect SVM as they may lead to heavy misclassification. Hence, the data must be pre-processed well.
Python Implementation : scikit-learn's svm package has SVC module(sklearn.svm.SVC)

3.4.4 XGBoost

XGBoost is the latest sensation among ML algorithms. XGBoost stands for Extreme Gradient Boosting. As the name suggests, boosting means improving or making better than before. XGBoost is an ensemble learning technique, which means it makes use of many base learners/weak learners and improves their performance via boosting and uses them collectively to get the best possible prediction/output. This algorithm is a sequential ensemble boosting technique. XGBoost uses a tree learner known as decision tree as the base learner.

These base learners are built sequentially and each time a new learner is built it uses the results of previous learner as a base result and improves it (by adding its own findings). This makes sure that by the time we finish our process we have a result which is way better than the result which we would have got by using the base learner. The best part about this algorithm is that none of the base learners look to change anything in the previous base learner. All of them just learn from the mistake made previously and they improve it and set a new benchmark. The algorithm can be related to a scenario like: If one is stuck in traffic, he/she will look at the traffic ahead and learn from it and then manoeuvre their vehicle accordingly or take a different route at the first opportunity they get.

XGBoost is quite scalable as the base learners (trees) are not quite deep and are built sequentially using parallelized implementation. The algorithm is a Greedy algorithm, hence, we must mention a stopping criteria for it.

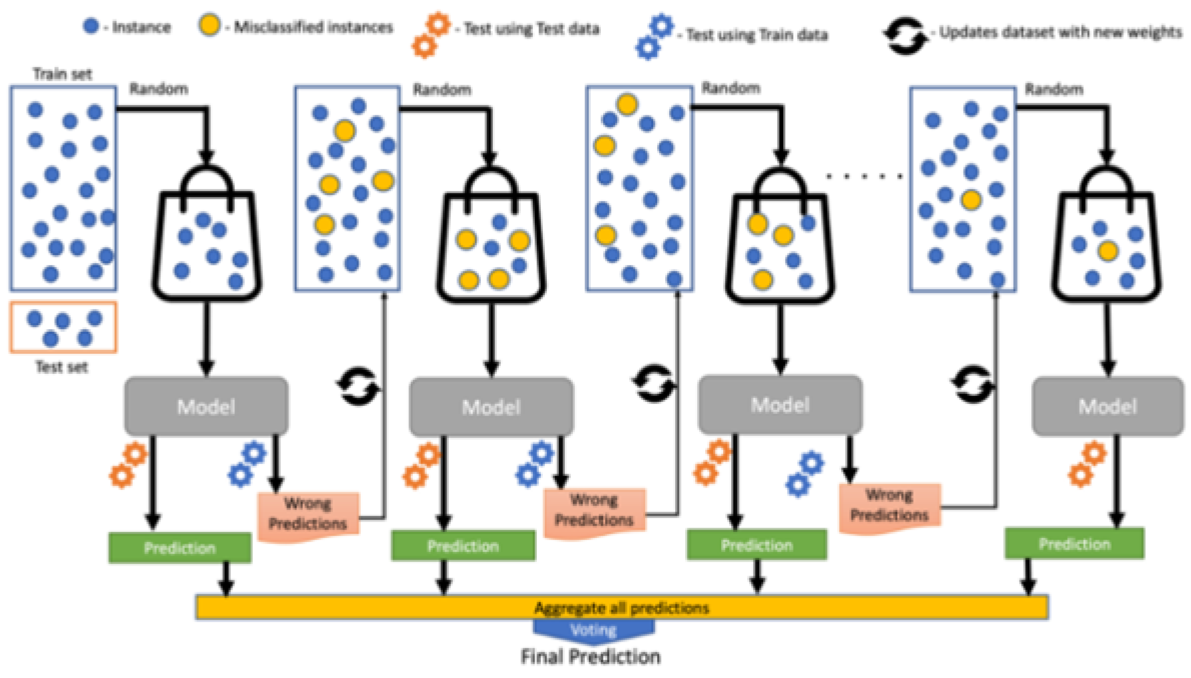
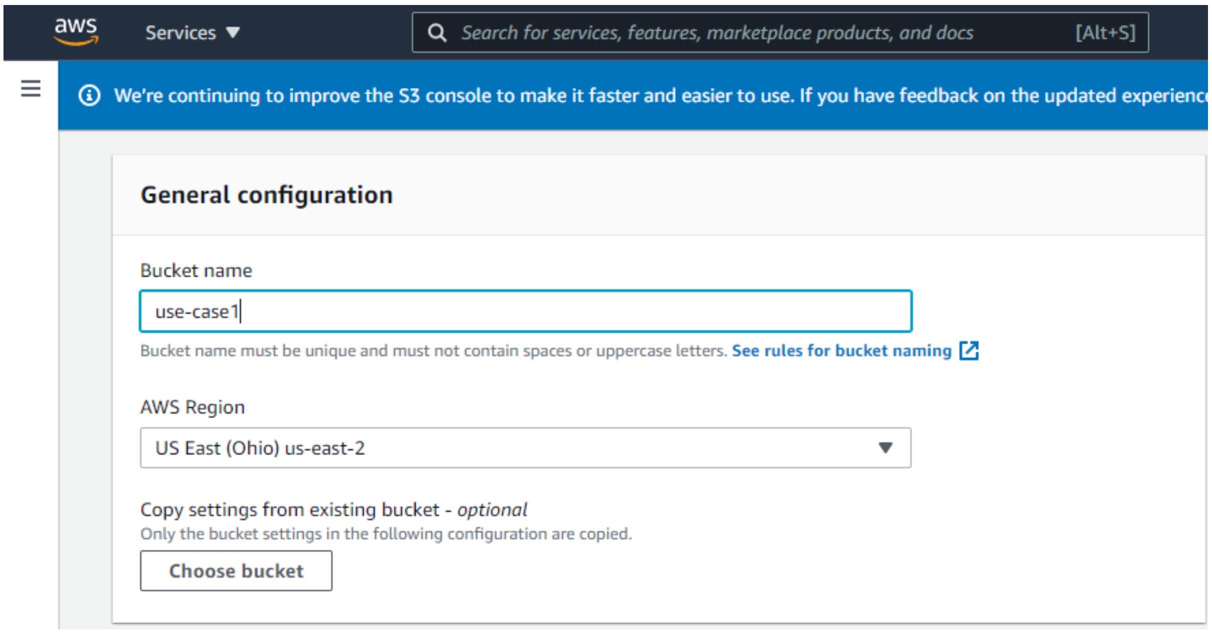


Fig 16: Boosting algorithm

3.5 Implementation in AWS SageMaker

3.5.1. Initial Setup (AWS Sagemaker)

1. Create a Jupyter Notebook instance in Sagemaker.
2. Create S3 Bucket with an appropriate region and maintain the same region throughout the project on AWS.


aws
Services ▼
Search for services, features, marketplace products, and docs
[Alt+S]

We're continuing to improve the S3 console to make it faster and easier to use. If you have feedback on the updated experienc

General configuration

Bucket name

use-case1

Bucket name must be unique and must not contain spaces or uppercase letters. See rules for bucket naming

AWS Region

US East (Ohio) us-east-2

Copy settings from existing bucket - *optional*

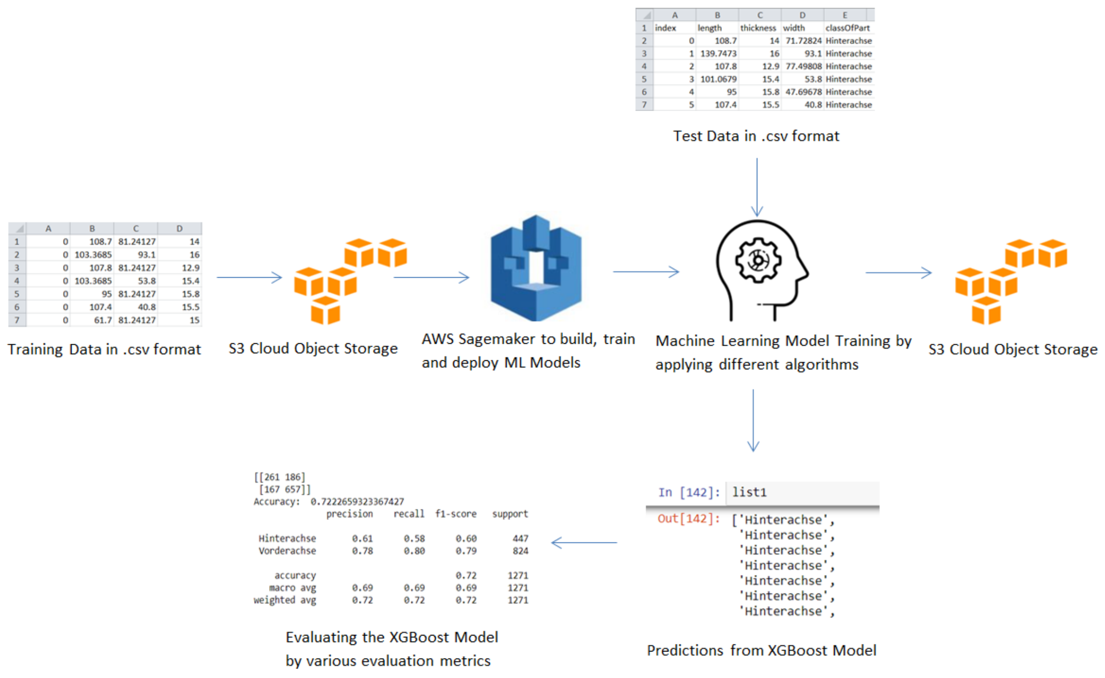
Only the bucket settings in the following configuration are copied.

Fig 17 : Creation of S3 Bucket

3. Remove header and index from the dataset and then upload to S3 bucket.
4. In the notebook instance create code cells and import necessary libraries (numpy, pandas, sagemaker, boto3 etc.).
5. Set up an execution role using the inbuilt get_execution_role() function.
6. Read the training data from S3 bucket and store it into a data frame using pandas.
7. Create an output file to store the trained model.
8. In order to start using the algorithm and fit it into the data, establish a session in SageMaker.

3.5.1.1 Architecture Diagram

Machine Learning Workflow


Test Data in .csv format
Training Data in .csv format
S3 Cloud Object Storage
AWS Sagemaker to build, train and deploy ML Models
Machine Learning Model Training by applying different algorithms
S3 Cloud Object Storage
Evaluating the XGBoost Model by various evaluation metrics
Predictions from XGBoost Model

	A	B	C	D	E
1	index	length	thickness	width	classOfPart
2	0	108.7	14	71.72824	Hinterachse
3	1	139.7473	16	93.1	Hinterachse
4	2	107.8	12.9	77.49808	Hinterachse
5	3	101.0679	15.4	53.8	Hinterachse
6	4	95	15.8	47.69678	Hinterachse
7	5	107.4	15.5	40.8	Hinterachse

	A	B	C	D
1	0	108.7	81.24127	14
2	0	103.3685	93.1	16
3	0	107.8	81.24127	12.9
4	0	103.3685	53.8	15.4
5	0	95	81.24127	15.8
6	0	107.4	40.8	15.5
7	0	61.7	81.24127	15

```
In [142]: list1

Out[142]: ['Hinterachse',
           'Hinterachse',
           'Hinterachse',
           'Hinterachse',
           'Hinterachse',
           'Hinterachse',
           'Hinterachse',
```

```
[[261 186]
 [167 657]]
Accuracy:  0.7222659323367427
              precision    recall  f1-score   support

 Hinterachse       0.61      0.58      0.60       447
 Vorderachse       0.78      0.80      0.79       824

    accuracy                           0.72      1271
   macro avg       0.69      0.69      0.69      1271
weighted avg       0.72      0.72      0.72      1271
```

Fig 18 : ML workflow in this use case

3.5.2 Implementation of inbuilt algorithms

We have implemented 3 inbuilt algorithms in SageMaker namely XGBoost, Linear Learner (Logistic Regression), KNN (K Nearest Neighbors)

1. Retrieve image of algorithms XGBoost, Linear Learner, KNN from AWS container.
2. Build the model and configure the training job.
3. Specify the training data location and fit our models onto the data.

3.5.3 Implementation of own algorithms

AWS SageMaker has limited inbuilt algorithms. However, if we wish to use a particular algorithm from scikit-learn then we need to package our algorithm into a container and then deploy on SageMaker.

The container which we are going to use is a Docker container.

Docker: A tool to package our applications and algorithms into a virtual container which can be run on cloud platforms.

Container: A container is a standard unit of software that packages up code and all its dependencies so the application can run quickly & reliably from one computing environment to another.

Docker container image: It is a lightweight standalone, executable package of software that includes everything needed to run an application code, system tools, libraries and settings.

Docker container structure

```
|-- Dockerfile

|-- build_and_push.sh

|-- local_test

`-- svm

    |-- nginx.conf

    |-- predictor.py

    |-- serve

    |-- train

    `-- wsgi.py
```

- **`Dockerfile`** describes how to build your Docker container image.
- **`build_and_push.sh`** is a script that uses the Dockerfile to build container images and then push it to ECR.
- **`svm`** is the directory which contains the files that will be installed in the container.
- **`local_test`** is a directory that shows how to test the new container on any computer that can run Docker, including an Amazon SageMaker notebook instance. Using this method, you can quickly iterate using small datasets to eliminate any structural bugs before you use the container with Amazon SageMaker.

Svm folder contents:

- **`nginx.conf`** is the configuration file for the nginx front-end. Generally, you should be able to take this file as-is.
- **`predictor.py`** is the program that actually implements the Flask web server and the decision tree predictions for this app. You'll want to customize the actual prediction parts to your application. Since this algorithm is simple, we do all the processing here in this file, but you may choose to have separate files for implementing your custom logic.
- **`serve`** is the program started when the container is started for hosting. It simply launches the gunicorn server which runs multiple instances of the Flask app defined in `predictor.py`. You should be able to take this file as-is.
- **`train`** is the program that is invoked when the container is run for training. You will modify this program to implement your training algorithm.
- **`wsgi.py`** is a small wrapper used to invoke the Flask app. You should be able to take this file as-is.

3.5.3.1 Architecture diagram

Docker with ECR

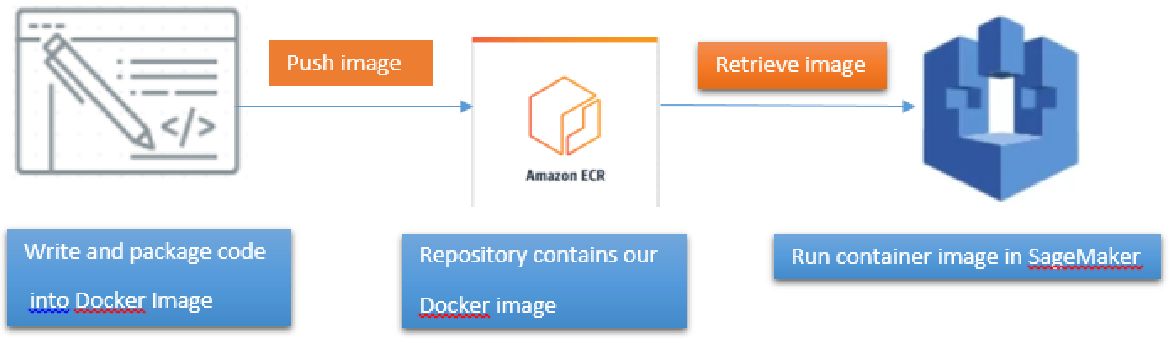

Push image
Amazon ECR
Retrieve image
Write and package code into Docker Image
Repository contains our Docker image
Run container image in SageMaker

Fig19:- Docker with ECR

3.5.4. Integration with lambda function

AWS Lambda is a serverless compute service that lets you run code without provisioning or managing servers, creating workload-aware cluster scaling logic, maintaining event integrations, or managing runtimes. With Lambda, you can run code for virtually any type of application or backend service - all with zero administration. We used python language to implement the lambda function. Lambda function makes it easy for us to keep the integration logic separate from the model. The lambda function gets some features as input and it transforms the data into a format that can be fed to the machine learning model endpoint and gets the result or the predicted label from the end-point.

In our case the lambda function connects to 4 endpoints :

a. XGBoost
b. KNN
c. LinearLearner
d. SVM

The steps that were followed to implement Lambda function for our endpoints are:-

· Prerequisites required to use lambda function by IAM user:-

Admin must assign following permissions to IAM user:-

1. Permission to access Lambda function created by admin

2. While creating Lambda Function, Admin must assign ROLE to required Lambda Function having following 2 policies:-

a. AWSLambdaBasicExecutionRole (AWS defined policy)

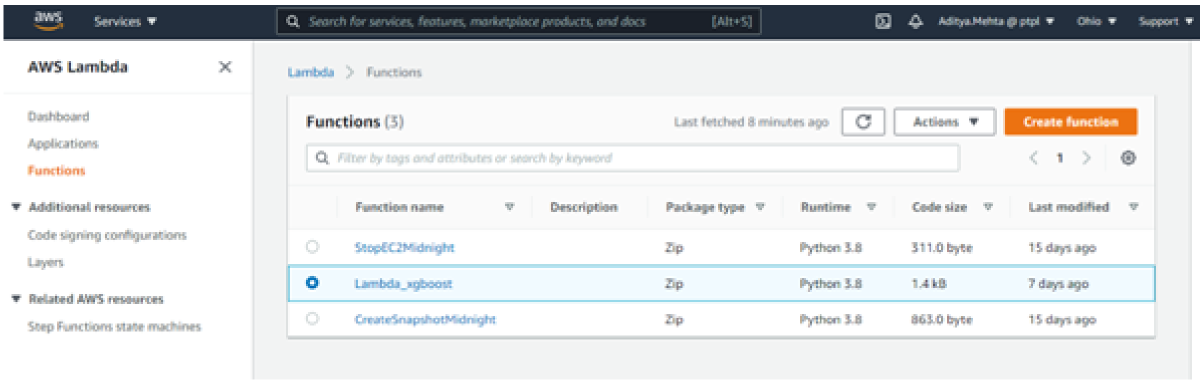
b. SagemakerInvokeEndpoint (Admin defined policy)

○ In this policy, Admin has to select SAGEMAKER as service; READ as action level and grant read only access to ‘All Resources’

Steps to execute Lambda Function:-

1) Search for Lambda Function in AWS management console dashboard

2) Access the Lambda function that admin created.

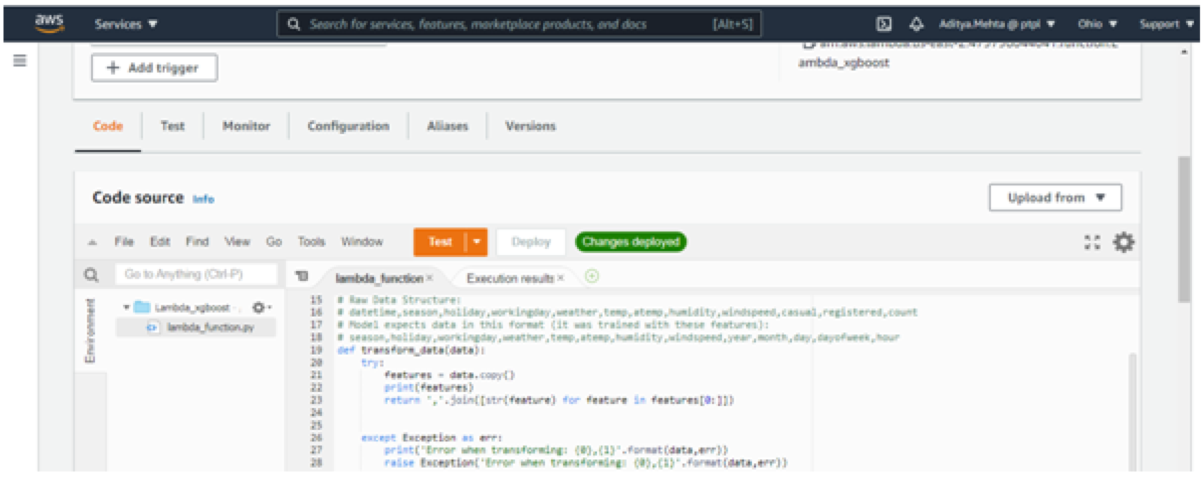


	Function name	Description	Package type	Runtime	Code size	Last modified
○	StopEC2Midnight		Zip	Python 3.8	311.0 byte	15 days ago
●	Lambda_xgboost		Zip	Python 3.8	1.4 kB	7 days ago
○	CreateSnapshotMidnight		Zip	Python 3.8	863.0 byte	15 days ago

Fig20:- Accessing Lambda Function

3) In workspace, enter the code under the code section that includes data transformations required to be done on inputs before feeding into model and on output received from model before displaying it to the user. Also creation of Lambda handler function is mandatory in code because it acts as an entry point.

4) Once code is created, deploy it. Then test it for the specified input (considering which code is written)



```
# Raw Data Structure:
# datetime,season,holiday,workingday,weather,temp,atemp,humidity,windspeed,casual,registered,count
# Model expects data in this format (it was trained with these features):
# season,holiday,workingday,weather,temp,atemp,humidity,windspeed,year,month,day,dayofweek,hour
def transform_data(data):
    try:
        features = data.copy()
        print(features)
        return ",".join([str(feature) for feature in features[0:]])


    except Exception as err:
        print('Error when transforming: {0},{1}'.format(data,err))
        raise Exception('Error when transforming: {0},{1}'.format(data,err))
```

Fig21:- Deployed Lambda Function

5) Before testing code, make sure required endpoints are active.

6) Test the code against the inputs given in the dataset.

7) Once tested in the correct way, Lambda function is successfully implemented.

3.5.5 API Gateway integration

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. APIs act as the "front door" for applications to access data, business logic, or functionality from your backend services. Using API gateway gives us the freedom to use any programming language on the client side. In our case we used JS(React framework). The steps we followed were:

a. Create a Restful API (parts-classification-api)

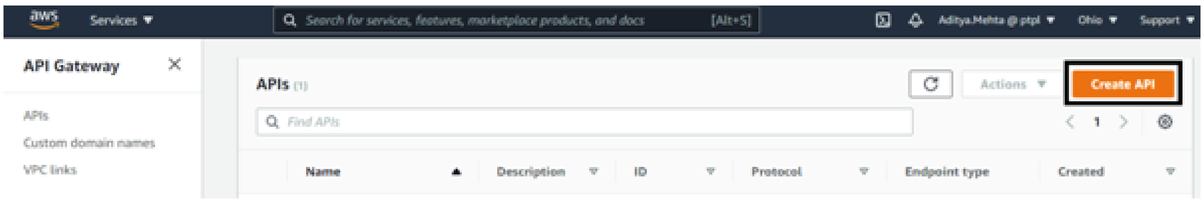
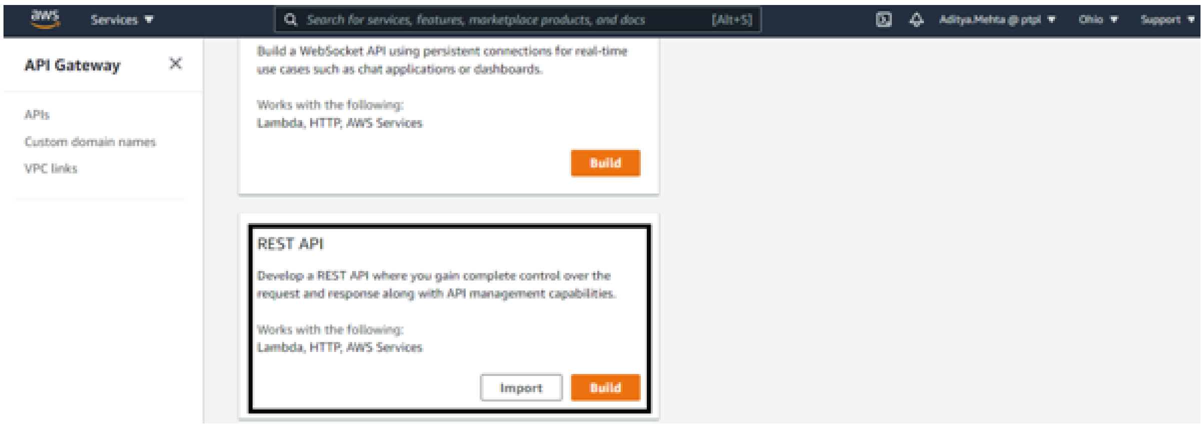


Fig22:- Creation of API



Build a WebSocket API using persistent connections for real-time use cases such as chat applications or dashboards.

Works with the following:
Lambda, HTTP, AWS Services

REST API

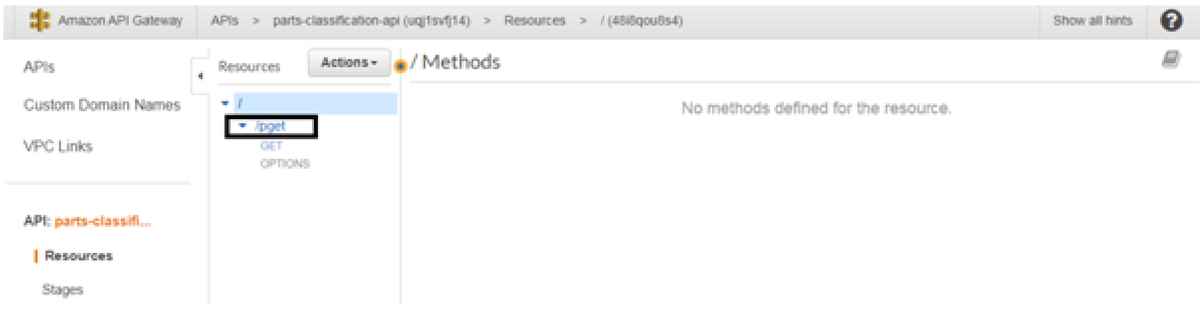
Develop a REST API where you gain complete control over the request and response along with API management capabilities.

Works with the following:
Lambda, HTTP, AWS Services

Fig23:- Selection of REST API

b. Assign roles to our API - communication with lambda function (Done by admin)

c. Create a resource and then a method inside the resource. We created a resource named '/pget' and created a GET method inside it.



No methods defined for the resource.

Fig24:- /pget

d. The GET method gets 4 params which are UserOption, length, width, and thickness. These 4 params are extracted from the request and are sent to the Lambda function as input.

e. We also need to ‘enable CORS’ which helps us to make requests from the specified domains. The Enable CORS creates an “OPTIONS” method which is a mock integrated method. Its only work is to let the client know if it is allowed to make a request to this API.
f. Deploy the API on a stage (“live stage”)

3.5.5.1. Architecture Diagram

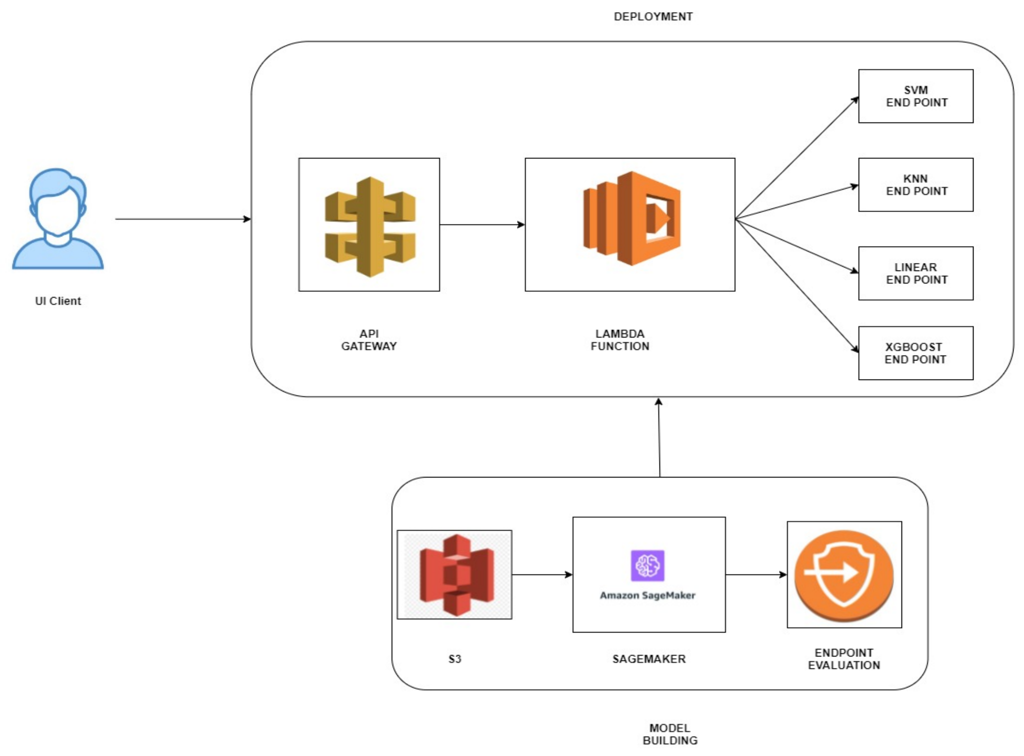
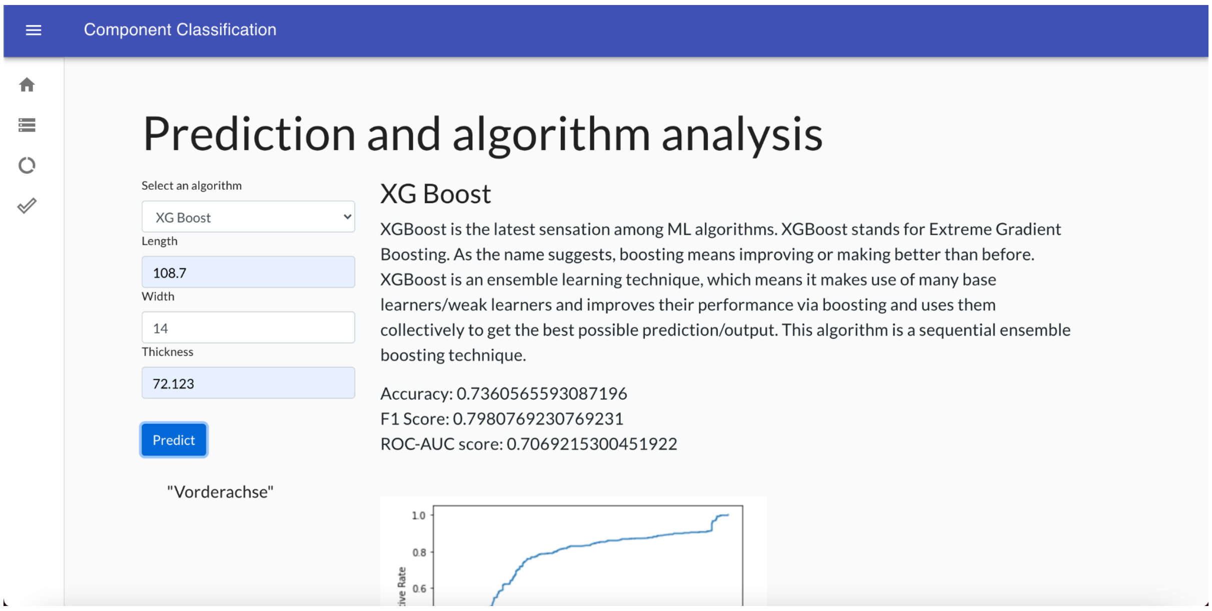


Fig25:- Deployment and Model Building Architecture Diagram

3.6 User Interface Screenshots

3.6.1 XG Boost



Component Classification

Prediction and algorithm analysis

Select an algorithm: XG Boost

Length: 108.7

Width: 14

Thickness: 72.123

Predict

"Vorderachse"

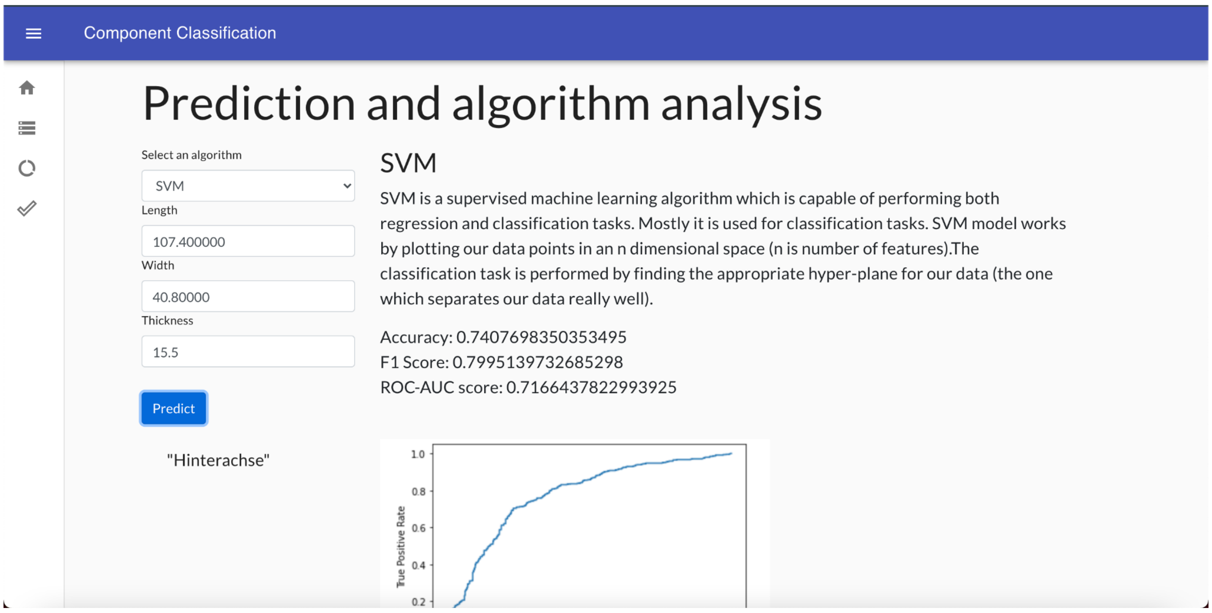
XG Boost

XGBoost is the latest sensation among ML algorithms. XGBoost stands for Extreme Gradient Boosting. As the name suggests, boosting means improving or making better than before. XGBoost is an ensemble learning technique, which means it makes use of many base learners/weak learners and improves their performance via boosting and uses them collectively to get the best possible prediction/output. This algorithm is a sequential ensemble boosting technique.

Accuracy: 0.7360565593087196
F1 Score: 0.7980769230769231
ROC-AUC score: 0.7069215300451922

Fig26:- UI-XGBoost

3.6.2 SVM



Component Classification

Prediction and algorithm analysis

Select an algorithm: SVM

Length: 107.400000

Width: 40.80000

Thickness: 15.5

Predict

"Hinterachse"

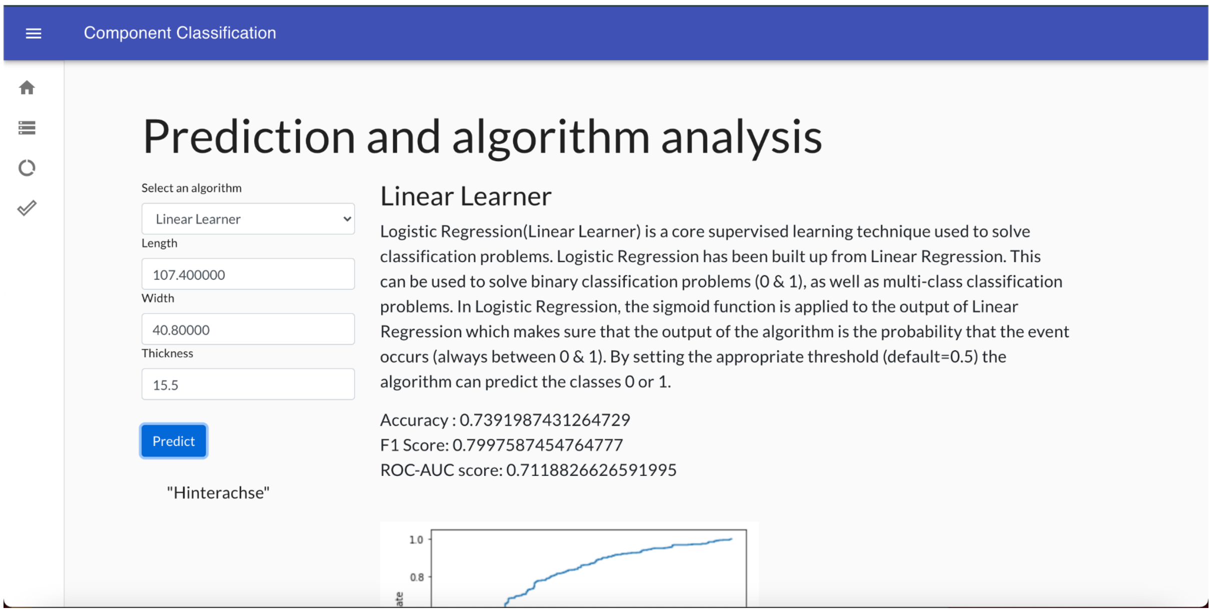
SVM

SVM is a supervised machine learning algorithm which is capable of performing both regression and classification tasks. Mostly it is used for classification tasks. SVM model works by plotting our data points in an n dimensional space (n is number of features).The classification task is performed by finding the appropriate hyper-plane for our data (the one which separates our data really well).

Accuracy: 0.7407698350353495
F1 Score: 0.7995139732685298
ROC-AUC score: 0.7166437822993925

Fig27:- UI-SVM

3.6.3 Linear Learner



Prediction and algorithm analysis

"Hinterachse"

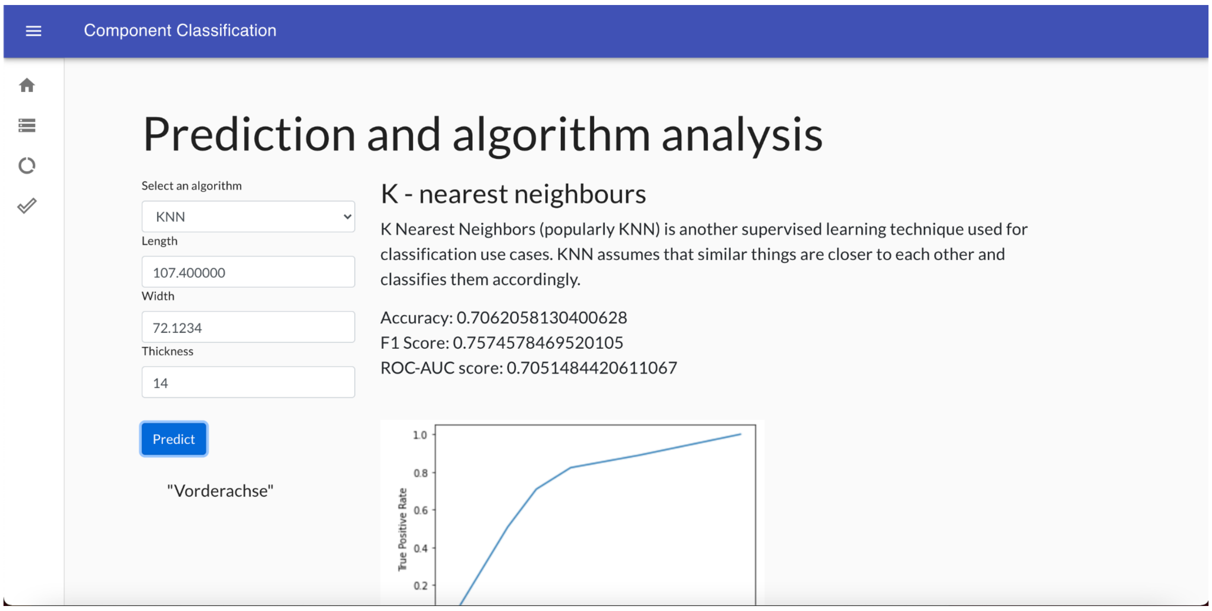
Linear Learner

Logistic Regression(Linear Learner) is a core supervised learning technique used to solve classification problems. Logistic Regression has been built up from Linear Regression. This can be used to solve binary classification problems (0 & 1), as well as multi-class classification problems. In Logistic Regression, the sigmoid function is applied to the output of Linear Regression which makes sure that the output of the algorithm is the probability that the event occurs (always between 0 & 1). By setting the appropriate threshold (default=0.5) the algorithm can predict the classes 0 or 1.

Accuracy : 0.7391987431264729
F1 Score: 0.7997587454764777
ROC-AUC score: 0.7118826626591995

Fig28:- UI-Linear Learner

3.6.4 KNN



Prediction and algorithm analysis

"Vorderachse"

K - nearest neighbours

K Nearest Neighbors (popularly KNN) is another supervised learning technique used for classification use cases. KNN assumes that similar things are closer to each other and classifies them accordingly.

Accuracy: 0.7062058130400628
F1 Score: 0.7574578469520105
ROC-AUC score: 0.7051484420611067

Fig29:- UI-K-nearest neighbours

CHAPTER 4 Results

4.1 Evaluation Metrics:

You build a model, get feedback from metrics, make improvements and continue until you achieve a desirable model performance. Evaluation metrics explain the performance of a model. An important aspect of evaluation metrics is their capability to discriminate among model results.

4.1.1. Confusion Matrix:

The confusion matrix visualizes the accuracy of a classifier by comparing the actual and predicted classes. The binary confusion matrix is composed of squares:

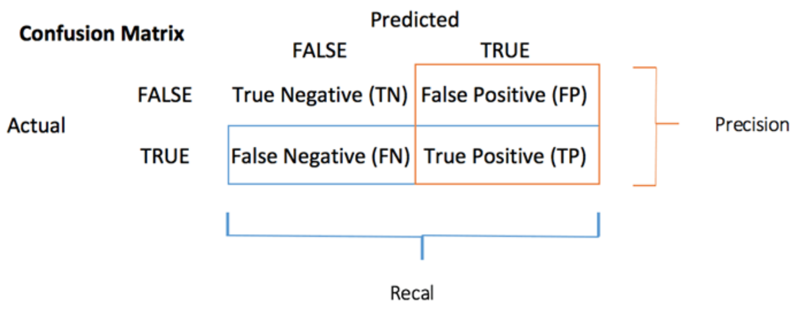


Fig30:- Confusion Matrix

TP: True Positive: Predicted values correctly predicted as actual positive

FP: Predicted values incorrectly predicted an actual positive. i.e., Negative values predicted as positive

FN: False Negative: Positive values predicted as negative

TN: True Negative: Predicted values correctly predicted as an actual negative

4.1.2. Accuracy score:

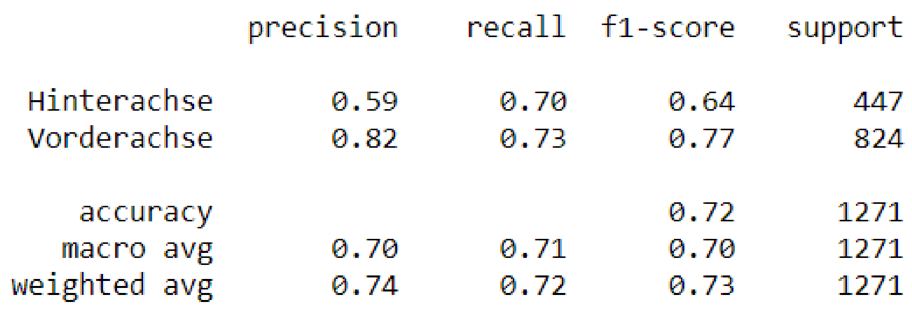
Accuracy is the ratio of correct predictions to the total number of correct predictions.

You can compute the accuracy test from the confusion matrix:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

4.1.3. Classification Report:

A Classification report is used to measure the quality of predictions from a classification algorithm. How many predictions are True and how many are False. More specifically, True Positives, False Positives, True negatives and False Negatives are used to predict the metrics of a classification report as shown below.



```
              precision    recall  f1-score   support

 Hinterachse       0.59      0.70      0.64       447
 Vorderachse       0.82      0.73      0.77       824

    accuracy                           0.72      1271
   macro avg       0.70      0.71      0.70      1271
weighted avg       0.74      0.72      0.73      1271
```

Precision – Accuracy of positive predictions.

Precision = TP/(TP + FP)

Recall – Recall is the ability of a classifier to find all positive instances. Fraction of positives that were correctly identified.

Recall = TP/(TP+FN)

F1 score – percent of positive predictions that were correct

The F1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0.

$$\boxed{\text{F1 Score} = 2*(\text{Recall} * \text{Precision}) / (\text{Recall} + \text{Precision})}$$

4.1.4. ROC AUC score

The *Receiver Operator Characteristic (ROC)* curve is an evaluation metric for binary classification problems. It is a probability curve that plots the TPR(True Positive Rate) against FPR(False Positive Rate) at various threshold values and essentially separates the 'signal' from the 'noise'. The *Area Under the Curve (AUC)* is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.

Higher the area under the curve better the model performance.

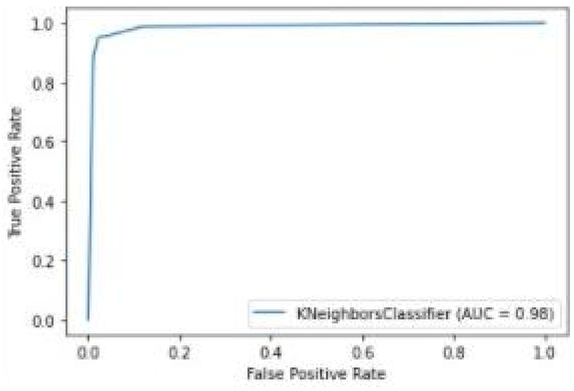


Fig31:-ROC AUC score

4.2 Test Results

1. Local Python:

Evaluation metrics comparative table:

ML models	**Evaluation Metrics Used**		
	Accuracy score	F1 score	ROC-AUC score
XGBoost	73.60	79.80	70.69
KNN	70.62	75.74	70.51
Logistic Regression	73.91	79.99	71.18
SVM	74.07	79.95	71.66

Fig32:- Evaluation Metrics

Confusion matrix and ROC-AUC curves of the 4 algorithms(XGBoost,KNN,Logistic Regression,SVM)

1. XGBoost

   ROC-AUC curve

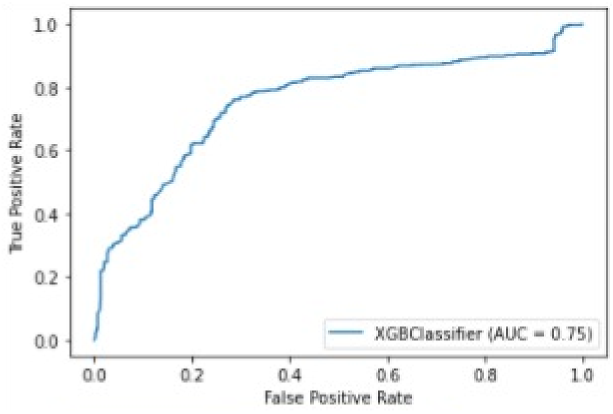


Fig33:- XGBoost-ROC-AUC score

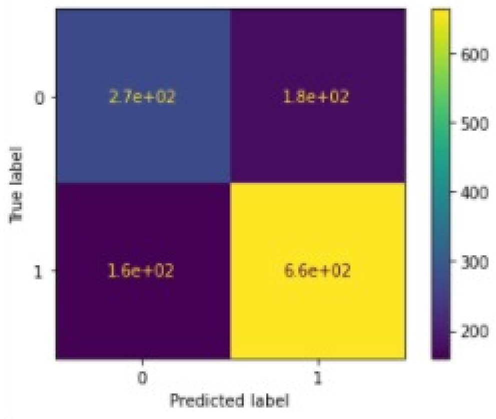


Fig34:- Confusion Matrix (XGBClassifier)

2. KNN

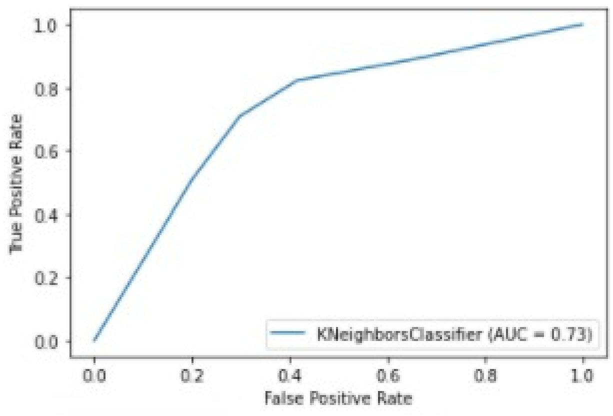


Fig35:- KNN-ROC-AUC curve

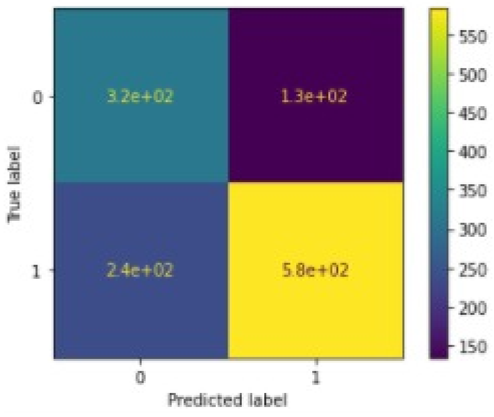


Fig36:- Confusion Matrix (KNeighbourClassifier)

3. Logistic Regression

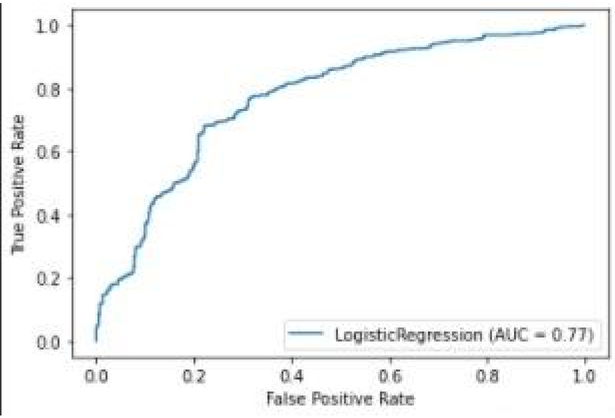


Fig37:- LogisticRegression-ROC-AUC curve

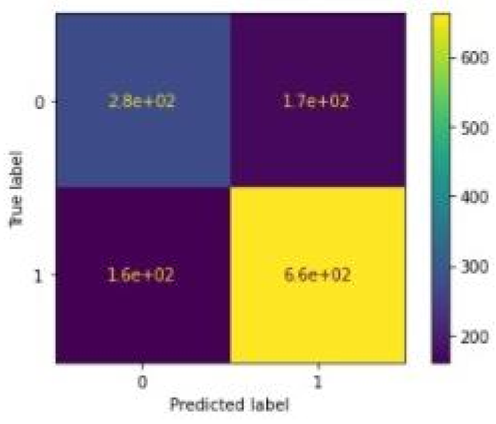


Fig38:- Confusion Matrix (Logistic Regression)

4. SVM(Support Vector Machine)

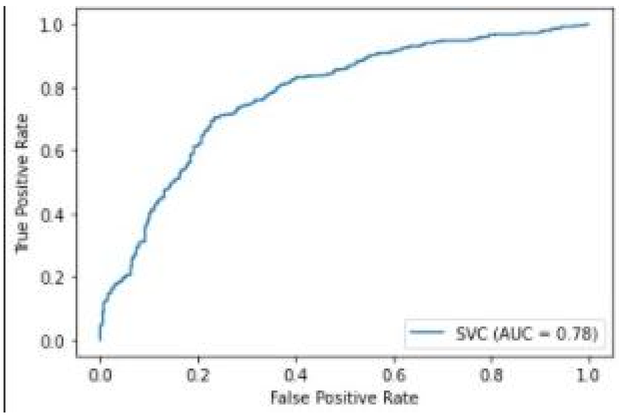


Fig39:- SVM-ROC-AUC curve

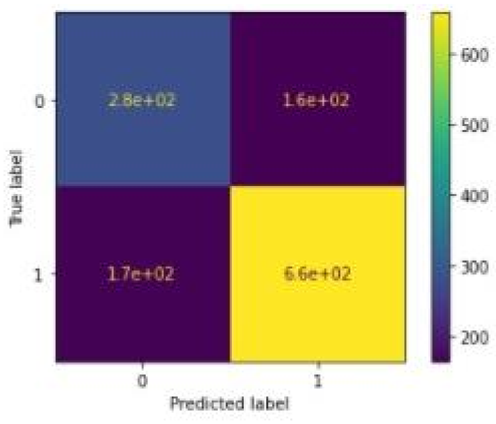


Fig40:- Confusion Matrix(SVC)

4.3 Observations:

1. Local Python
    - SVM classifier has achieved the highest accuracy of 74.07 on unseen data.

- Logistic Regression classifier has achieved the highest F1 score of 79.99 on unseen data.
- SVM classifier has achieved the highest ROC-AUC score of 71.18 on unseen data.

2. AWS SageMaker
    - KNN has achieved the highest accuracy of 72.77 on unseen data

Chapter 5 Conclusion

The project also portrays how industries in mechanical, chemical, automotive and other non IT sectors can also implement Machine Learning in their business to increase revenue. Cloud services like AWS Sagemaker have made it very convenient where the users don’t have to worry about deployment, upscaling etc. They have inbuilt algorithms which mostly meet the needs of 80% use cases. The user just needs to have a clear idea on the use case and should have a large amount of data related to the use case.

The project has successfully implemented 4 supervised machine learning algorithms on the given dataset. The dataset, even though small, was a good medium to observe and test how the machine learning algorithms classify the data. The presence of a high number of null values lead to dropping of rows having more than one null values. The dataset had a few points which were having extremely high values of dimensions which is not seen often. Such data points are outliers which were removed by setting limits for the dimensions.

The result of clustering on the dataset indicates that there is a possibility of having more than 2 clusters which does seem right considering the results of the ML algorithms(not having very high accuracy). The decision boundary separating the 2 classes of axles is not very easy to decide as the data points are intermixed.

The models trained on this data have shown a good performance on test data obtained by splitting data into train and test sets. When the trained models were tested out on new and unseen data they performed moderately well. These results are shown with the help of evaluation metrics in the results table. The ROC-AUC curves aid in interpreting the performance of the models.

Chapter 6 Suggestions for future work

The dataset used for the project here has dimensional data of the product(namely length,width, thickness) which is very useful for classification of components. The dataset here is not very large as a result of which the models have performed moderately well. The machine learning models need data as their input of good quality and fairly large quantity as well. The models can learn more when they have good quality and a good amount of data which helps them in performing better.

In order to get better results we can add some more data about axles of brake pads. More data would help to avoid overfitting and also help models learn more about the data.

One more suggestion would be to make use of hyperparameter tuning for our algorithms(a basic attempt at tuning the models has been done in the project).

Machine learning for parts classification has a great impact on reducing lead times of the product and the scope of ML in PLM is not limited. We are just getting started with ML in PLM. ML in PLM can lead to identification of a variety of use cases which will then make it possible to introduce more businesses and industries to the benefits of machine learning.

Chapter 7 Bibliography

[1] towards-automated-classification-of-product-data-based-on-machine-learning by S. Schleibaum , S. Kehl , P. Stiefel and J. P. Müller

[2] machine-learning-driven-parts-classification by CIM Data

[3] oreilly-ml-ops by Mark Treveil and the Dataiku Team

[4] AWS Certified Machine Learning Specialty (MLS-C01) by Chandra Lingam

[5] GitHub - SorenSc/ClassificationOfProductData (link for referred GitHub code)

[6] AWS Documentation for XGBoost
https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost.html

[7] AWS Documentation for KNN

https://docs.aws.amazon.com/elasticsearch-service/latest/developerguide/knn.html

[8] AWS Documentation for Linear Learner

https://docs.aws.amazon.com/machine-learning/latest/dg/learning-algorithm.html

[9] AWS bring your own container

https://docs.aws.amazon.com/sagemaker/latest/dg/docker-containers-notebooks.html

[10] Use Scikit Learn with Amazon SageMaker

https://docs.aws.amazon.com/sagemaker/latest/dg/sklearn.html