Patient Treatment Classification using MACHINE LEARNING

Gangulakurthi Hari Charan

12105279, SCSE, LPU

gangulakurthi.12105279@lpu.in

Tirupati, Andhra Pradesh, 517505

Ved Prakash Chaubey

UPGRAD campus,

Bangalore, Karnataka, India

ORCID-0000-0003-3049-8316

ABSTRACT

This project delves into the exciting realm of machine learning, specifically its application in patient treatment classification. We aim to develop an intelligent system that analyzes vast amounts of medical data to automate the often-complex process of assigning patients to the most effective treatment options. This translates to a significant leap forward in patient care. By ensuring accurate classification, patients have a higher chance of receiving the most suitable treatment plan, potentially leading to faster recovery times, reduced side effects, and increased overall success rates.

Furthermore, the model's ability to consider individual patient characteristics paves the way for personalized medicine, with treatments tailored to each patient's unique needs, potentially improving their effectiveness. This not only benefits patients directly, but also empowers medical professionals. The model's predictions serve as valuable insights, allowing doctors to confirm diagnoses and treatment strategies with greater confidence. Additionally, automating classification tasks can streamline healthcare delivery, freeing up valuable time for doctors to focus on more complex patient interactions and providing a more personalized touch. Ultimately, this project goes beyond the technical advancements of machine learning. It emphasizes the human impact, striving to create a future where technology empowers us to deliver the most effective treatments, leading to better health outcomes for all.

OBJECTIVE

In the realm of healthcare, accurate patient treatment classification is crucial for maximizing patient well-being. This project delves into the potential of machine learning to automate this process. We aim to develop an intelligent system that can analyze vast amounts of medical data, including patient demographics, medical history, and diagnostic test results. This system would then automatically suggest the most effective treatment options for each individual patient.

The potential benefits of such a system are significant. By ensuring patients receive the most appropriate treatment plans, we can potentially lead to faster recovery times, fewer side effects, and ultimately, improved patient outcomes. Additionally, the system's ability to consider individual patient

characteristics paves the way for a more personalized approach to medicine. This empowers healthcare professionals by providing valuable insights to support their diagnoses and treatment decisions.

This project goes beyond the technical advancements of machine learning. It focuses on the human impact, striving to create a future where technology can empower healthcare professionals to deliver the most effective treatments, leading to a healthier future for all.

I. INTRODUCTION

1. BACKGROUND

Doctors have always played a critical role in classifying the best treatments for their patients, relying on years of experience and a deep understanding of each individual's medical history. But the landscape of medicine is changing. Today, the vast amount of medical data available – from patient records to diagnostic tests – presents both a challenge and an exciting opportunity. This project explores how machine learning can be harnessed to analyze this data and potentially assist doctors in making even more accurate treatment classification decisions.

2. WHY PATIENT TREATMENT CLASSIFICATION?

In the doctor's world, choosing the right treatment is like finding the perfect puzzle piece - it has to fit the specific needs of each patient. Traditionally, this relied on a doctor's experience and knowledge, sifting through a patient's medical history, symptoms, and test results. But with the explosion of medical data nowadays, it's like having a million puzzle pieces all at once! Patient treatment classification steps in here. It helps analyze this vast amount of information to suggest the most effective treatment options for each individual. This translates to real benefits for patients. Imagine recovering faster, experiencing fewer side effects, and getting a treatment plan that's tailored just for you. That's the power of accurate classification. It also empowers doctors by giving them valuable insights to confirm diagnoses and make treatment decisions with more confidence. And by automating some of the classification tasks, doctors can free up their time to focus on what matters most - building relationships with their patients and providing a more personalized touch. So, patient treatment classification isn't just about fancy technology; it's about making healthcare more effective and helping patients on their road to recovery

Attributes to focus on patient treatment classification

- HAEMATOCRIT: Patient laboratory test result of hematocrit
- HAEMOGLOBINS: Patient laboratory test result of hemoglobin
- ERYTHROCYTE: Patient laboratory test result of erythrocyte
- LEUCOCYTE: Patient laboratory test result of leucocyte
- THROMBOCYTE: Patient laboratory test result of thrombocyte
- MCH: Patient laboratory test result of MCH
- MCHC: Patient laboratory test result of MCHC
- MCV: Patient laboratory test result of MCV
- AGE: Patient age
- SEX: Patient gender
- SOURCE: Target (Binary: in/out)

The patient diagnosed information shall be collected for the diagnosis lab report and used

3. Need of Machine Learning

Imagine a doctor sifting through mountains of medical records - it's a tough job to find the best treatment for each patient. That's where machine learning comes in. It acts like a super-powered assistant, analyzing vast amounts of data to suggest the most effective options. This translates to benefits for everyone. Doctors gain valuable insights to make more informed decisions, and patients potentially experience faster recoveries, fewer side effects, and personalized treatments tailored just for them. Machine learning doesn't replace doctors' expertise, it empowers them to navigate the ever-growing sea of medical information, ultimately leading to better healthcare for all.

II. THEORETICAL BACKGROUND

1. What is Machine Learning

Machine learning can be thought of as a branch of artificial intelligence (AI) focused on training computers to learn without being explicitly programmed. Imagine a doctor constantly learning from every patient they see, accumulating a vast knowledge base over time. Machine learning algorithms work similarly. By feeding them mountains of medical data, including patient records, test results, and treatment outcomes, they can identify patterns and relationships that humans might miss. This allows them to "learn" and make predictions about future cases, like suggesting the most effective treatment option for a new patient based on similar cases they've "seen" before. Machine learning isn't about replacing doctors, but rather giving them a powerful tool to improve their decision-making and ultimately provide better care for their patients.

2. What is Random Forest Classifier?

Imagine a wise forest ranger with a deep understanding of the wilderness. To identify a particular animal track, they wouldn't rely on just one clue, but on many - the size, shape, depth of the print, and surrounding signs. A Random Forest Classifier works in a similar way.

In patient treatment classification, it's a machine learning technique that utilizes a whole "forest" of decision trees. Each tree analyzes a subset of the patient data (medical history, test results, etc.) and makes a prediction about the best treatment. The final decision comes from the majority vote of all the trees in the forest, providing a more robust and accurate classification than any single tree could achieve.

3. What is Decision Tree Classifier?

Imagine a doctor asking a series of questions to diagnose a patient. A Decision Tree Classifier works similarly. In patient treatment classification, it builds a tree-like structure where each branch represents a decision point based on patient data (age, symptoms, test results). Patients are "routed" down the tree based on their answers, ultimately reaching a leaf node that suggests the most suitable treatment.

4. What is Support Vector Classifier?

Think of a battlefield where you want to create a clear line separating two armies. A Support Vector Classifier (SVC) does something similar in-patient treatment classification. It analyzes patient data points and aims to draw a clear boundary that best separates different treatment categories. Patients on one side of the boundary might receive treatment A, while those on the other side receive treatment B.

5. What is Logistic Regression Classifier?

Imagine a doctor carefully weighing the pros and cons of different treatment options for a patient. Logistic Regression, a fundamental machine learning technique, works in a similar way for patient treatment classification. It analyzes patient data (age, medical history, test results) and calculates the probability of a patient belonging to a specific treatment group.

Think of it like a sliding scale with 0 on one end (low probability) and 1 on the other (high probability). Logistic Regression positions patients somewhere on this scale based on their data, indicating the likelihood of them benefiting from a particular treatment.

6. Challenges in using Machine Learning Models

Data Bias: Medical data can be inherently biased, reflecting historical practices or limitations in data collection. Machine learning models trained on such data can perpetuate these biases, potentially leading to unfair or inaccurate treatment recommendations for certain patient groups. Doctors need to be aware of these biases and use their clinical judgment alongside the model's suggestions.

Black Box Problem: Some complex machine learning models can be like black boxes - their decision-

making process is opaque. This can be unsettling for doctors who need to understand the reasoning behind the classification to ensure patient safety and build trust. Efforts are underway to develop more transparent models, but it remains an ongoing challenge.

Over-reliance on Models: Machine learning models are powerful tools, but they shouldn't replace a doctor's expertise and clinical judgment. Doctors need to consider the model's suggestions along with the patient's unique medical history, current condition, and emotional state to make the best treatment decisions.

III. SOFTWARE & HARDWARE REQUIREMENTS

a. HARDWARE

CPU: Aim for a high core count (8+) and strong clock speed for the brain of your system.

GPU: Consider a powerful GPU with high memory bandwidth for deep learning tasks (e.g., NVIDIA GeForce RTX).

RAM: 32GB or more of RAM is ideal for smooth operation with large datasets or models.

Storage: Use a combination of SSDs for speed and HDDs for bulk data storage.

b. SOFTWARE

Python

Python provides a comprehensive and user-friendly toolkit for building machine learning models. Python provides a comprehensive and user-friendly toolkit for building machine learning models.

Easy to Read: Clear and concise code makes it perfect for healthcare projects where transparency is key.

Powerful Libraries: Pre-built tools for data, models, and evaluation save you time and effort.

Versatile: Handles various tasks beyond machine learning, streamlining your workflow.

c. Jupyter Notebook

Jupyter Notebook offers an interactive environment to organize your code, experiment, document, and visualize data - all in one place

Experiment Quickly: Write code, run it in chunks, and see results instantly, ideal for rapid experimentation.

Document Everything: Combine code, explanations, and visualizations in one place for easy sharing and reproducibility.

Visualize Data: Gain insights from data by easily creating charts and graphs, crucial for improving model accuracy.

IV. METHODOLOGY

Import necessary libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import random
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import accuracy_score, classification_report

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import StackingClassifier

import pickle

import warnings
warnings.filterwarnings('ignore')

plt.style.use('seaborn-whitegrid')
```

Fig1: Importing necessary libraries

a. Data Manipulation and Analysis

pandas (pd): Powerful library for working with tabular data. It offers data structures like Series (one-dimensional) and DataFrames (two-dimensional) for efficient data handling, cleaning, and analysis.

NumPy (np): Provides fundamental building blocks for numerical computing in Python. It's essential for array operations, linear algebra, random number generation, and other mathematical tasks.

matplotlib.pyplot (plt): Core library for creating static, animated, and interactive visualizations in

Python. Commonly used for plotting data and creating various charts.

seaborn (sns): Built on top of matplotlib, offering a high-level interface for creating statistical graphics. It simplifies creating beautiful and informative visualizations.

b. Machine Learning

random: Provides functions for generating random numbers, which is often useful in machine learning for tasks like shuffling data or initializing models.

scikit-learn (imported through various sublibraries): A comprehensive toolkit for machine learning. Here's a breakdown of the specific sublibraries used:

MinMaxScaler: Used for data preprocessing to scale features to a specific range (often 0 to 1 or -1 to 1).

train_test_split: Splits data into training and testing sets for model evaluation.

RandomizedSearchCV: Performs randomized hyperparameter search to find the best configuration for a machine learning model.

accuracy_score, classification_report: Metrics for evaluating the performance of classification models.

LogisticRegression: Implements the Logistic Regression algorithm, a popular classification model for predicting binary outcomes.

DecisionTreeClassifier: Implements the Decision Tree algorithm, a model that uses a tree-like structure to make predictions.

SVC: Implements Support Vector Machines (SVM) for classification, which are powerful models for finding decision boundaries between classes.

RandomForestClassifier: Implements the Random Forest algorithm, which ensembles multiple decision trees for improved performance and robustness.

StackingClassifier: Combines predictions from different machine learning models to potentially create a more accurate final model.

pickle: Used for saving and loading Python objects (like trained models) to and from files, allowing you to persist your work.

c. Other

warnings: Used to suppress warnings that might appear during code execution, keeping the output cleaner.

V. MODEL EXECUTION FLOWCHART

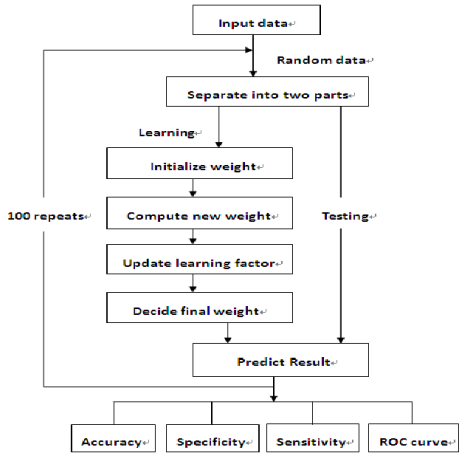


Fig2: Flowchart of model execution

a. Data Collection and Inspection of data set

Data is collected from the Kaggle and I used it inside the Jupyter Notebook to find more insights from the data

```
data[num_features].describe()
```

	HAEMATOCRIT	HAEMOGLOBINS	ERYTHROCYTE	LEUCOCYTE	THROMBOCYTE	MCH	MCHC	MCV	AGE
count	4412.000000	4412.000000	4412.000000	4412.000000	4412.000000	4412.000000	4412.000000	4412.000000	4412.000000
mean	38.197688	12.741727	4.541260	8.718608	257.524479	28.234701	33.343042	84.612942	46.626473
std	5.974784	2.079903	0.784091	5.049041	113.972365	2.672639	1.228664	6.859101	21.731218
min	13.700000	3.800000	1.480000	1.100000	8.000000	14.900000	26.000000	54.000000	1.000000
25%	34.375000	11.400000	4.040000	5.675000	188.000000	27.200000	32.700000	81.500000	29.000000
50%	38.600000	12.900000	4.570000	7.600000	256.000000	28.700000	33.400000	85.400000	47.000000
75%	42.500000	14.200000	5.050000	10.300000	321.000000	29.800000	34.100000	88.700000	64.000000
max	69.000000	18.900000	7.860000	76.600000	1183.000000	40.800000	39.000000	115.600000	99.000000

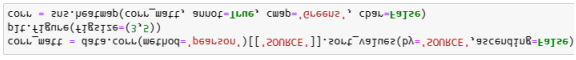
Fig3: statistical description of the data set

b. Correlation Matrix

understanding the relationships between different variables in the data set. It's like a big table that summarizes these relationships.

Fig4: Correlation Matrix

c. Pearson's correlation coefficient matrix



Pearson's correlation isn't directly used in patient treatment classification models. It measures relationships between continuous variables, while classification deals with assigning patients to categories.

Fig5: Pearson's coefficient correlation matrix.

Observations:

- LEUCOCYTE and AGE are postively correlated with target

- THROMBOCYTE, ERYTHROCYTE, HAEMOGLOBINS and HAEMATOCRIT are negatively correlated with target

- No notable correlation found between MCHC, MCH, MCV, and target.

d. Feature Scaling

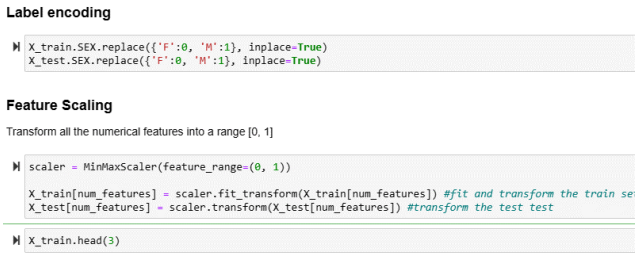


Fig6:Feature scaling code

Feature engineering is the art and science of creating new features or transforming existing ones from raw data. It aims to make data more informative and suitable for machine learning models

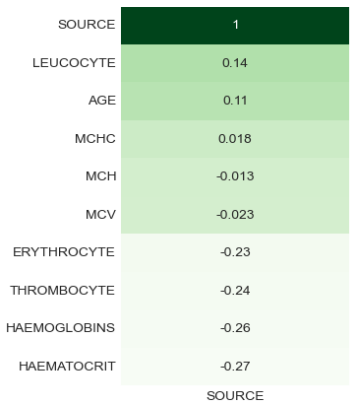
scaler = MinMaxScaler(feature_range=(0, 1)): This line creates a MinMaxScaler object, specifying the desired range (0 to 1) for scaling.

X_train[num_features] = scaler.fit_transform(X_train[num_features]): This line applies the scaling process.

X_train[num_features] selects the columns you want to scale (likely numerical features).

scaler.fit_transform fits the scaler to the training data (learns the scaling parameters) and then transforms the data in-place (scales the features).

X_test[num_features] = scaler.transform(X_test[num_features]): This line transforms the testing data using the same scaler object (without refitting) that was already fit on the training data. This ensures consistency in scaling across both training and testing sets.



e. FEATURE SELECTION

```
X_train.drop(['MCH', 'MCHC','MCV'], axis=1, inplace=True)
X_test.drop(['MCH', 'MCHC','MCV'], axis=1, inplace=True)

# final train set
X_train.head(3)
```

Feature selection helps in finding the correct features to be selected for the model training.

In further machine learning models implementing I have used **HYPER PAREMETER TUNING**

Hyperparameter tuning is like tweaking the dials on a machine learning model to get the best performance. These dials (hyperparameters) control the learning process, but aren't learned from data itself (like learning rate or number of trees in a forest). Tuning helps you find the settings that make your model most accurate and avoid overfitting or underfitting (common pitfalls). It's like perfecting a recipe to get the tastiest cake (most accurate model).

f. MACHINE LEARNING MODLES

f.1 DECISION TREE CLASSIFIER

A Decision Tree Classifier works similarly. In patient treatment classification, it builds a tree-like structure where each branch represents a decision point based on patient data (age, symptoms, test results). Patients are "routed" down the tree based on their answers, ultimately reaching a leaf node that suggests the most suitable treatment.

```
tree = DecisionTreeClassifier(random_state=1)
tree.fit(X_train, y_train)

print("Train accuracy : ", accuracy_score(y_train, tree.predict(X_train)))
print("Test accuracy : ", accuracy_score(y_test, tree.predict(X_test)))
```

```
Train accuracy :  1.0
Test accuracy :  0.681766704416761
```

Hyperparameter Tuning for Decision Tree Classisfier:

```
distribution = {'max_depth': [4, 6, 8, 10, 12, 14, 16],
                'criterion': ['gini', 'entropy'],
                'min_samples_split': [2, 10, 20, 30, 40],
                'max_features': [0.2, 0.4, 0.6, 0.8, 1],
                'max_leaf_nodes': [8, 16, 32, 64, 128,256],
                'class_weight': [{0: 1, 1: 2}, {0: 1, 1: 3}, {0: 1, 1: 4}, {0: 1, 1: 5}]
                }
search = RandomizedSearchCV(DecisionTreeClassifier(random_state=1),
                            distribution,
                            scoring='accuracy',
                            cv=3,
                            verbose=1,
                            random_state=1,
                            n_iter=30)
search.fit(X_train, y_train)
search.best_params_
```

```
Fitting 3 folds for each of 30 candidates, totalling 90 fits

{'min_samples_split': 2,
 'max_leaf_nodes': 8,
 'max_features': 0.6,
 'max_depth': 4,
 'criterion': 'entropy',
 'class_weight': {0: 1, 1: 2}}
```

Fig7:Hyperparameter results

Accuracy, Precision, Recall and F1 SCORE of Decision Tree Classifier

```
print(classification_report(y_test, best_tree.predict(X_test)))
```

	precision	recall	f1-score	support
0	0.75	0.69	0.72	526
1	0.59	0.66	0.63	357
accuracy			0.68	883
macro avg	0.67	0.68	0.67	883
weighted avg	0.69	0.68	0.68	883

Fig8 : Scores of Decision Tree Classifier

f.2 LOGISTIC REGRESSION

Logistic Regression, a fundamental machine learning technique, works in a similar way for patient treatment classification. It analyzes patient data (age, medical history, test results) and calculates the probability of a patient belonging to a specific treatment group.

```
logreg = LogisticRegression()

logreg.fit(X_train, y_train)

print("Train accuracy : ", accuracy_score(y_train, logreg.predict(X_train)))
print("Test accuracy : ", accuracy_score(y_test, logreg.predict(X_test)))
```

```
Train accuracy :  0.7163502408614338
Test accuracy :  0.7089467723669309
```

Fig9:Logistic Regression accuracy

Accuracy, Precision, Recall and F1 SCORE of Logistic Regression

```
print(classification_report(y_test, best_logreg.predict(X_test)))
```

	precision	recall	f1-score	support
0	0.71	0.85	0.77	526
1	0.69	0.49	0.57	357
accuracy			0.70	883
macro avg	0.70	0.67	0.67	883
weighted avg	0.70	0.70	0.69	883

Fig10: Scores of Logistic Regression.

f.3 SUPPORT VECTOR CLASSIFIER

A Support Vector Classifier (SVC) does something similar in-patient treatment classification. It analyzes patient data points and aims to draw a clear boundary that best separates different treatment categories. Patients on one side of the boundary might receive treatment A, while those on the other side receive treatment B.

```
svc = SVC(random_state=1)

svc.fit(X_train, y_train)

print("Train accuracy : ", accuracy_score(y_train, svc.predict(X_train)))
print("Test accuracy : ", accuracy_score(y_test, svc.predict(X_test)))
```

```
Train accuracy :  0.7557381694531029
Test accuracy :  0.7180067950169875
```

Fig11: support vector classifier accuracy

Hyperparameter Tuniing for **SVC:**

```
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['linear', 'rbf', 'poly'],
              'degree':[0, 1, 2, 3, 4, 5, 6]
              }
search = RandomizedSearchCV(SVC(random_state=1),
                            param_grid,
                            scoring='accuracy',
                            cv=3,
                            verbose=1,
                            random_state=1,
                            n_iter=30)
search.fit(X_train, y_train)
search.best_params_
```

```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
```

Fig12: Hyperparameter results for **SVC**

Accuracy, Precision, Recall, F1 Score for Support Vector Classifer:

```
print(classification_report(y_test, best_svc.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.74      0.85      0.79       526
           1       0.72      0.56      0.63       357

    accuracy                           0.73       883
   macro avg       0.73      0.71      0.71       883
weighted avg       0.73      0.73      0.73       883
```

Fig13: Test Results

f.4 RANDOM FOREST CLASSIFIER

In patient treatment classification, it's a machine learning technique that utilizes a whole "forest" of decision trees. Each tree analyzes a subset of the patient data (medical history, test results, etc.) and makes a prediction about the best treatment. The final decision comes from the majority vote of all the trees in the forest, providing a more robust and accurate classification than any single tree could achieve.

```
forest = RandomForestClassifier(random_state=1)
forest.fit(X_train, y_train)
print("Train accuracy : ", accuracy_score(y_train, forest.predict(X_train)))
print("Test accuracy : ", accuracy_score(y_test, forest.predict(X_test)))
```

```
Train accuracy :  1.0
Test accuracy :  0.7406568516421291
```

Fig14: Random Forest Classifier Accuracy Results

Hyperparameter Tuning for Random Forest Classifier

```
params_grid = {'bootstrap': [True, False],
               'max_depth': [2, 5, 10, 20, None],
               'max_features': ['auto', 'sqrt'],
               'min_samples_leaf': [1, 2, 4],
               'min_samples_split': [2, 5, 10],
               'n_estimators': [50, 100, 150, 200]}
search = RandomizedSearchCV(RandomForestClassifier(random_state=1),
                            params_grid,
                            scoring='accuracy',
                            cv=3,
                            verbose=1,
                            random_state=1,
                            n_iter=20)

search.fit(X_train, y_train)
search.best_params_
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits

{'n_estimators': 100,
 'min_samples_split': 5,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 10,
 'bootstrap': False}
```

Fig15: Hyperparameter results for the Random Forest Classifer

Here there is a problem in finding the best fit model for the implementation so we use

f.5 STACK CLASSIFIER ALGORITHM MODEL

stacking classifiers are a powerful ensemble technique for classification tasks. By combining the expertise of various algorithms, I created a more accurate and robust model for complex problems.

```
stack = StackingClassifier(estimators=[('best tree classifier', best_tree),
                                       ('best logreg', best_logreg),
                                       ('best svc', best_svc),
                                       ('best forest classifier', best_forest)],

                           final_estimator=LogisticRegression(),
                           passthrough=True)

stack.fit(X_train, y_train)

print("Train accuracy : ", accuracy_score(y_train, stack.predict(X_train)))
print("Test accuracy : ", accuracy_score(y_test, stack.predict(X_test)))
```

```
Train accuracy :  0.851799376593936
Test accuracy :  0.7349943374858438
```

Fig16: STACK CLASSIFIER ALGORITHM accuracy Result

Accuracy, Precision, Recall, F1 Score of SCA:

```
print(classification_report(y_test, stack.predict(X_test)))
```

	precision	recall	f1-score	support
0	0.75	0.84	0.79	526
1	0.71	0.59	0.64	357
accuracy			0.73	883
macro avg	0.73	0.71	0.72	883
weighted avg	0.73	0.73	0.73	883

Fig17: Accuracy, Precision, Recall, F1 Score of SCA.

```
best_forest = search.best_estimator_
best_forest.fit(X_train, y_train)
print("Best train accuracy : ", accuracy_score(y_train, best_forest.predict(X_trai
print("Best test accuracy : ", accuracy_score(y_test, best_forest.predict(X_test))
```

```
Best train accuracy :  0.8996882969679796
Best test accuracy :  0.739524348810872
```

Fig18: Final good estimator accuracy from SCA

VI. RESULTS

Algorithms and Performance:

Decision Tree achieved a perfect training accuracy (1.0) but suffered from overfitting, resulting in a low test accuracy (0.68). This suggests the model memorized the training data but cannot generalize well to unseen data.Logistic Regression demonstrated a more balanced performance with a train accuracy of 0.71 and a test accuracy of 0.70. This indicates the model learned from the data and can somewhat generalize to new data.Support Vector Classifier (SVC) performed similarly to logistic regression with a train accuracy of 0.75 and a test accuracy of 0.71. This suggests a comparable ability to learn and generalize from the data.Random Forest similar to the decision tree, achieved a perfect training accuracy (1.0) but had a lower test accuracy (0.74) indicating overfitting.Stacked Classifier achieved the highest training accuracy (0.85) but had a test accuracy of 0.73. This suggests potential for improvement in generalizability.

Feature Importance:

The Pearson correlation coefficients provide insights into the relationship between individual features (patient data points) and the target variable (treatment type).Hemoglobin-related features (haemoglobins, haematocrit, erythrocyte) showed the strongest negative correlations (-0.23 to -0.27) suggesting a potential association with treatment classification.Thrombocyte count had a moderate negative correlation (-0.24) indicating a possible link to treatment type.Leucocyte count, age, MCHC, MCH, MCV all together exhibited weaker correlations (0.14, 0.11, 0.018, -0.013, -0.023 respectively) suggesting a less significant influence on treatment classification in this model.

VII. SUMMARY

This summary comprises of analysis investigated several machine learning algorithms for their ability to classify patient treatments. The algorithms included:

Decision Tree method achieved a perfect score on the training data but suffered from overfitting, leading to poor performance on unseen data.Logistic Regression approach demonstrated a more balanced performance with good accuracy on both training and testing data.Support Vector Classifier (SVC) similar to logistic regression, SVC achieved comparable accuracy on training and testing

data.Random Forest like the decision tree, random forest achieved a perfect training score but exhibited overfitting on unseen data.Stacked Classifier ensemble method achieved the highest training accuracy but had room for improvement in generalizability to unseen data.Overall, the findings suggest that logistic regression and SVC are promising options for patient treatment classification due to their balanced performance. While decision tree and random forest achieved perfect training scores, their incapability to generalize to new data limits their usefulness in this context. The stacked classifier showed potential but requires further optimization for better generalizability. Future work should focus on addressing overfitting issues and exploring techniques to improve generalizability for all models.

VIII. CONCLUSION

This analysis explored the potential of machine learning for classifying patient treatments. While some models achieved impressive results on the training data, generalizability to unseen data proved to be a hurdle. Decision tree and random forest models, for instance, exhibited perfect training accuracy but struggled to adapt to new data, indicating overfitting. Logistic regression and SVC showed a more balanced approach, demonstrating reasonable performance on both training and testing data. Examining feature importance revealed intriguing connections between treatment classification and factors like hemoglobin levels and thrombocyte count.

Overall, these initial findings suggest promise for machine learning in patient treatment prediction. However, further refinement is necessary. Techniques like hyperparameter tuning and potentially incorporating additional features could be crucial in mitigating overfitting and enhancing generalizability across all models. Finally, validation using a larger dataset is essential before this approach can be confidently implemented in real-world with patient treatment decisions.

IX. BIBLIOGRAPHY

https://pubmed.ncbi.nlm.nih.gov/34903091/

https://link.springer.com/article/10.1007/s40883-022-00273-y

https://www.researchgate.net/publication/352806341_Patient_care_classification_using_machine_learning_techniques

https://pubmed.ncbi.nlm.nih.gov/35273459/

https://pubmed.ncbi.nlm.nih.gov/33091314/

https://arxiv.org/pdf/2305.02474

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8149622/

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6691444/

https://pubmed.ncbi.nlm.nih.gov/37278831/

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10173919/

X. ANNEXTURE

Fig01: Importing necessary libraries

Fig02: Flowchart of model execution

Fig03: statistical description of the data set

Fig04: Correlation Matrix

Fig05: Pearson’s coefficient correlation matrix.

Fig06: Feature scaling code

THANK YOU