# MACHINE LEARNING WEB APPLICATION WITH FLASK

**A Project Report**

*Submitted in the partial fulfillment for the award of the degree of*

**BACHELOR OF ENGINEERING**

**IN**

COMPUTER SCIENCE WITH SPECIALIZATION IN

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

**Submitted by:**

UJJWAL (20BCS6849)

PRABHJOT SINGH (20BCS6895)

DIYA GOEL (20BCS6887)

KUMAR SAURAV (20BCS6856)

***Under the Supervision of:***

**Prof. Aadi Pratap Singh**

**CHANDIGARH UNIVERSITY, GHARUAN, MOHALI - 140413,**

**PUNJAB**

**May, 2024**

# <u>CERTIFICATE</u>

This is to certify that the project report entitled "**MACHINE LEARNING MODEL WEB APPLICATION WITH FLASK"** in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science Engineering of Chandigarh University, Punjab is a record of bonafide work carried out under my guidance and supervision.

| | |
|---|---|
| Program Leader | Project Guide |
| Dr. U. Hariharan | Prof. Aadi Pratap Singh |
| Program leader of CSE-AIT | Professor of CSE-AIT Assistant |
| (AUTONOMOUS) | (AUTONOMOUS) |

# ACKNOWLEDGEMENT

An endeavor over a long period can be the advice and support of many well-wishers. We take this opportunity to express our gratitude and appreciation to all of them. We owe our tributes to Dr. U. Hariharan, Program Leader, AIT- CSE for his valuable support and guidance during the period of project implementation. We wish to express our sincere thanks and gratitude to our project guide Aadi Pratap Singh, Assistant Professor, CSE-AIT, for stimulating discussions, analyzing problems associated with our project work, and guiding us throughout the project. Project meetings were highly informative. We express our sincere thanks for the encouragement, untiring guidance, and the confidence she had shown in us. We also thank the Head of the department Mr. Aman Kaushik and supporting management for providing resources when required. We are immensely indebted for their valuable guidance throughout our project. We also thank all the staff members of the department for their valuable advice would like to thank our parents, friends, and classmates for their encouragement throughout our project period. Last but not least, we thank everyone for supporting us in completing this project successfully.

| | |
|---|---|
| Diya Goel | 20BCS6887 |
| Prabhjot Singh | 20BCS6895 |
| Ujjwal | 20BCS6849 |
| Kumar Saurav | 20BCS6856 |

# DECLARATION

We, Diya Goel, Prabhjot, Ujjwal, and Kumar Saurav, students of 8th semester 4th-year B.E(HONS) with specialization in Artificial Intelligenceand Machine Learning, from Chandigarh University, hereby declare that the project work entitled "**MACHINE LEARNING MODEL WEB APPLICATION WITH FLASK**" is carried out by us and submitted in partial fulfillment of the requirements for the award of Bachelor of Technology in Computer Science Engineering, under Chandigarh University during the academic year 2020-2024 and has not been submitted to any other university for the award of any kind of degree.

| **Name** | **UID** | **Mobile Number** |
|---|---|---|
| Prabhjot Singh | 20BCS6895 | 9464874594 |
| Diya Goel | 20BCS6887 | 9541018686 |
| Ujjwal | 20BCS6849 | 6204814723 |
| Kumar Saurav | 20BCS6856 | 9546542733 |

## ABSTRACT

With the development in the science field, more and more research and developments have been made in the past few years. Accurate tools are needed to study and analyze long research papers accurately and in less time. So we have created a **Machine Learning Model Web Application With Flask** that can accurately classify the data written in research papers into different categories for quick analysis used for sequential data classification. The web application is made using Machine learning and applying NLP methods to it with the model trained using sequential neural networking. The application takes the data from the research paper and classifies the text inside it. The dataset that we are using for this purpose is named as PUBMED 200K RCT which consists of around 200,000 abstracts of randomized controlled trials, totaling 2.3 million sentences. The dataset is quite large compared to other datasets for similar purposes. The model we have created is accurately trained and its computation time is also less which is a great advantage of this model. The project culminates in the development of a user-friendly Flask web application, enabling users to interact with the trained model. Deployment is executed on a server or cloud platform for broader accessibility. In addition to accurately classifying research paper data into different categories for quick analysis, our Machine Learning Model Web Application With Flask offers a user-friendly interface, enhancing usability for researchers and professionals. By leveraging advanced NLP methods and a well-trained sequential neural network model, our application efficiently processes the vast amount of data contained in the PUBMED 200K RCT dataset. Documentation provides a detailed account of the methodology, model architecture, and user instructions, ensuring transparency and ease of use. The project's timeline is structured to accommodate each phase, from data pre-processing to deployment, with regular checkpoints for assessment and refinement. This systematic approach ensures a thorough exploration of the dataset, model development, and deployment, ultimately contributing to advancements in the field of medical text analysis.

# TABLE OF CONTENTS

# LIST OF FIGURES

# 1. INTRODUCTION

In the realm of human-computer interaction, there exists a compelling need for more natural and intuitive interfaces that can seamlessly bridge the gap between users and technology. In a world where new research and technical advancements are made at each step, it becomes of utmost importance that research articles written on them are accurately and thoroughly analyzed. So the primary objective of the model created by us is to categorize sentences, facilitating a deeper understanding of medical research findings. The project encompasses comprehensive data preprocessing, model development, evaluation, and deployment stages. Leveraging state-of-the-art Natural Language Processing (NLP) techniques, the project addresses the challenge of analyzing vast amounts of textual data in the biomedical field. There's a genuine need for more user-friendly interfaces, where users can upload their data and perform sequential task classification. Despite the promise of delivering accuracy, adaptability, and speed, the existing systems have faced many challenges. Our model focuses on and will deal with these problems by using appropriate training and preprocessing of the dataset. The dataset used is quite new and advanced comparable to other datasets used for the same purpose. The dataset consists of approximately 200,000 abstracts of randomized controlled trials, totaling 2.3 million sentences. Each sentence of each abstract is labeled with their role in the abstract using one of the following classes: background, objective, method, result, or conclusion. The purpose of releasing this dataset is twofold. First, the majority of datasets for sequential short-text classification (i.e., classification of short texts that appear in sequences) are small: we hope that releasing a new large dataset will help develop more accurate algorithms for this task. Second, from an application perspective, researchers need better tools to efficiently skim through the literature. Automatically classifying each sentence in an abstract would help researchers read abstracts more efficiently, especially in fields where abstracts may be long. By leveraging advanced NLP methods and a well-trained sequential neural network model, our application efficiently processes the vast amount of data contained in the PUBMED 200K RCT dataset. Moreover, the deployment of the application on a server or cloud platform ensures broader accessibility, facilitating widespread adoption and utilization by individuals and organizations within the scientific community.

## 1.1 Factors leading to development and advancements in machine learning models for text classification

**1.1.1 Increased Availability of Data**: The proliferation of digital content, including text documents, articles, and research papers, has provided a vast amount of labelled and unlabelled data for training machine learning models. This abundance of data allows for more robust training and validation of text classification models.

**1.1.2 Advancements in Natural Language Processing (NLP)**: Continuous advancements in NLP techniques, such as word embeddings, transformers, and attention mechanisms, have significantly improved the performance of text classification models. These techniques enable models to better understand and process the semantic and contextual information present in text data.

**1.1.3 Development of Deep Learning Architectures**: Deep learning architectures, such as recurrent neural networks (RNNs), convolutional neural networks (CNNs), and transformer models (e.g., BERT, GPT), have revolutionized text classification tasks. These architectures can capture complex patterns and dependencies within text data, leading to higher accuracy and efficiency in classification tasks.

**1.1.4 Availability of Pre-trained Models**: The availability of pre-trained language models, such as BERT, GPT, and XLNet, has accelerated the development of text classification models. Researchers and practitioners can leverage these pre-trained models as feature extractors or fine-tune them on domain-specific data, significantly reducing the time and resources required for model training.

**1.1.5 Advancements in Hardware and Computing Resources**: The availability of powerful hardware (e.g., GPUs, TPUs) and cloud computing resources has facilitated the training of large-scale text classification models. These resources enable researchers to experiment with complex architectures and larger datasets, leading to improved model performance and scalability.

## 1.2 Some already existing text classification models

**1.2.1 Naive Bayes Classifier**: Naive Bayes is a simple probabilistic classifier based on Bayes' theorem with the assumption of independence between features. It is commonly used for text classification tasks due to its simplicity and efficiency, especially for tasks with a large number of features.

**1.2.2 Support Vector Machines (SVM)**: SVM is a powerful supervised learning algorithm used for classification tasks. It works by finding the hyperplane that best separates the data into different classes. SVM has been successfully applied to text classification tasks, especially when dealing with high-dimensional feature spaces.

**1.2.3 Logistic Regression**: Logistic Regression is a linear classification algorithm that models the probability of a binary outcome. Despite its name, logistic regression is commonly used for binary classification tasks, including text classification. It is simple, interpretable, and often serves as a baseline model for more complex algorithms.

**1.2.4 Random Forest Classifier**: Random Forest is an ensemble learning method that combines multiple decision trees to make predictions. It is robust to overfitting and can handle high-dimensional data well. Random Forest has been applied to text classification tasks, especially when dealing with large datasets and complex feature spaces.

**1.2.5 Gradient Boosting Machines (GBM)**: GBM is another ensemble learning technique that builds a series of weak learners (usually decision trees) sequentially, with each subsequent learner correcting the errors of the previous one. GBM has been shown to achieve excellent performance in text classification tasks, particularly when combined with feature engineering techniques.

**1.2.6 Convolutional Neural Networks (CNNs)**: CNNs are deep learning models commonly used for image classification, but they have also been adapted for text classification tasks. CNNs can automatically learn hierarchical representations of text data by applying convolutional filters over word embeddings or character sequences.

**1.2.7 Recurrent Neural Networks (RNNs)**: RNNs are a type of neural network architecture designed to process sequential data, such as text. They are capable of capturing temporal dependencies and have been widely used for text classification tasks, including sentiment analysis, document classification, and spam detection.

## 1.3 <u>How Text Classification Models Work</u>

- Data collection involves gathering a labeled dataset of text documents and their corresponding categories. Preprocessing cleans and normalizes the text data by removing punctuation, converting to lowercase, and tokenizing.

- Feature extraction converts the preprocessed text into numerical representations, such as Bag-of-Words, TF-IDF, or word embeddings, to make it understandable for machine learning algorithms.

- Model training uses supervised learning techniques to train a classification model on the labeled dataset, employing algorithms like Naive Bayes, SVM, logistic regression, or deep learning models.

- Cross-validation techniques, such as k-fold cross-validation, help assess model generalization by splitting the dataset into multiple subsets for training and validation.

- Hyperparameter tuning optimizes model performance by adjusting parameters like learning rate, regularization strength, or network architecture.

- Transfer learning leverages pre-trained models on large text corpora to initialize or fine-tune models for specific text classification tasks, reducing the need for extensive training data.

- Ensemble methods combine predictions from multiple base models to improve overall classification accuracy, using techniques like bagging, boosting, or stacking.

- Model evaluation assesses the performance of the trained model on a separate validation dataset, using metrics like accuracy, precision, recall, and F1-score. Optimization involves fine-tuning the model by adjusting hyperparameters or exploring different architectures.

- Inference applies the trained model to classify new, unseen text data into relevant categories or classes, predicting the probability or label of each class based on the input text.

## 1.4 Applications of Text Classification Models

Text classification models have a wide range of applications across various domains due to their ability to automatically analyze and categorize textual data. Some common applications of text classification models include:

- **Spam Detection:** Text classification models can be used to classify emails, messages, or comments as spam or non-spam, helping to filter out unwanted or malicious content.

- **Sentiment Analysis:** Text classification models can classify text documents, reviews, or social media posts as positive, negative, or neutral, allowing businesses to gauge customer sentiment towards their products or services.

- **Topic Categorization:** Text classification models can categorize documents or articles into predefined topics or categories, facilitating content organization, retrieval, and recommendation.

- **Language Identification:** Text classification models can identify the language of a given text document, helping in language detection and localization tasks.

- **Intent Recognition:** Text classification models can classify user queries or commands into different intents, enabling chatbots and virtual assistants to understand user requests and provide relevant responses.

- **Document Classification:** Text classification models can classify legal documents, medical records, or news articles into different categories or classes, aiding in document management and information retrieval tasks.

- **Toxic Comment Detection:** Text classification models can identify toxic or abusive comments in online forums, social media platforms, or discussion boards, helping to maintain a safe and respectful online environment.

- **Fake News Detection:** Text classification models can distinguish between credible and fake news articles by classifying them based on their authenticity, source credibility, and factual accuracy.

## 1.5 <u>About Dataset Used</u>

PubMed 200k RCT is new dataset based on PubMed for sequential sentence classification. The dataset consists of approximately 200,000 abstracts of randomized controlled trials, totaling 2.3 million sentences. Each sentence of each abstract is labeled with their role in the abstract using one of the following classes: background, objective, method, result, or conclusion. The purpose of releasing this dataset is twofold. First, the majority of datasets for sequential short-text classification (i.e., classification of short texts that appear in sequences) are small: we hope that releasing a new large dataset will help develop more accurate algorithms for this task. Second, from an application perspective, researchers need better tools to efficiently skim through the literature. Automatically classifying each sentence in an abstract would help researchers read abstracts more efficiently, especially in fields where abstracts may be long, such as the medical field. PubMed 20k is a subset of PubMed 200k. I.e., any abstract present in PubMed 20k is also present in PubMed 200k.

Randomized controlled trials (RCTs) are a gold standard method in medical research for evaluating the effectiveness and safety of interventions or treatments. In an RCT, participants are randomly assigned to either an experimental group that receives the intervention being studied or a control group that receives either a placebo or standard treatment. By randomly assigning participants, RCTs aim to minimize bias and provide reliable evidence for the efficacy of interventions.

The "PubMed 200k RCT" dataset is a valuable resource for researchers, practitioners, and developers in the biomedical and healthcare fields. It provides access to a large and diverse collection of abstracts from RCTs covering a wide range of medical topics and interventions.

Researchers can use this dataset for various purposes, including:

- Natural Language Processing (NLP): Researchers can apply NLP techniques to analyze the text of RCT abstracts, extract key information, and identify patterns or trends in medical research.

- Clinical Decision Support: Healthcare professionals can leverage the findings from RCT abstracts to inform clinical decision-making and evidence-based practice. By accessing and analyzing abstracts from RCTs, clinicians can stay updated on the latest research findings and guidelines in their field.

## 1.6 PROBLEM STATEMENT:

In the realm of medical research and healthcare, the volume of textual data, such as research papers, clinical notes, and patient records, continues to grow exponentially. However, the manual analysis and categorization of this vast amount of textual information pose significant challenges for researchers, clinicians, and healthcare practitioners. Without efficient and accurate methods for classifying medical text, there exists a pressing need for automated solutions that can categorize and organize textual data according to relevant domains and topics.

Furthermore, the absence of robust text classification models tailored specifically for medical literature hampers various aspects of medical research and healthcare delivery. The lack of automated classification tools results in inefficiencies in literature review processes, hindering the timely synthesis and dissemination of critical medical knowledge. Additionally, without automated text classification, healthcare practitioners struggle to access and leverage relevant information from vast repositories of medical literature, impeding evidence-based decision-making and patient care.

Moreover, traditional keyword-based search methods are often insufficient for navigating the complexities of medical text, leading to information overload and suboptimal retrieval of relevant research findings. As a consequence, researchers and clinicians face challenges in identifying and accessing pertinent literature, limiting their ability to stay abreast of the latest developments in their respective fields.

Therefore, there is a clear imperative to develop and deploy robust machine learning-based text classification models tailored specifically for medical literature. Such models have the potential to streamline literature review processes, facilitate evidence-based decision-making, and enhance knowledge discovery in medical research and healthcare delivery. By automating the classification and organization of medical text, these models can alleviate the burden on researchers and clinicians, enabling them to focus their time and efforts on higher-value tasks such as data analysis, interpretation, and innovation in medical research and patient care.

In response to these challenges, the field of natural language processing (NLP) has seen a surge in research and innovation aimed at developing advanced text classification techniques tailored specifically for medical literature. These techniques leverage state-of-the-art machine learning algorithms, such as deep learning models like recurrent neural networks (RNNs) and transformers, to extract meaningful insights from vast repositories of medical text. By training these models on annotated datasets comprising diverse medical documents, researchers can effectively teach them to recognize and categorize text according to relevant domains, topics, and subtopics.

Furthermore, the integration of domain-specific knowledge and ontologies into text classification models holds promise for enhancing their accuracy and interpretability in the medical domain. By incorporating biomedical ontologies, such as the Unified Medical Language System (UMLS) and Medical Subject Headings (MeSH), into the training and inference processes, researchers can provide models with a deeper understanding of medical concepts and relationships. This enables the models to not only classify text accurately but also to provide meaningful context and connections between different pieces of medical literature.

As these advanced text classification models continue to evolve and improve, they have the potential to revolutionize various aspects of medical research and healthcare delivery. From facilitating more efficient literature review processes to empowering healthcare practitioners with timely access to relevant information, these models hold the key to unlocking valuable insights from the ever-growing corpus of medical text. By harnessing the power of machine learning and NLP, we can pave the way for a future where automated text classification becomes an indispensable tool in the arsenal of researchers, clinicians, and healthcare organizations striving to advance medical knowledge and improve patient outcomes.

## 1.7 SOME PREVIOUSLY DESIGNED MODELS:

**Multinomial Naive Bayes Classifier-**

Creating a machine learning text classifier model using the Multinomial Naive Bayes classifier involves several steps. First, the text data needs to be preprocessed, which may include steps such as tokenization, removing stop words, and stemming or lemmatization. Then, the data is typically split into training and testing sets to evaluate the performance of the model.

Next, the Multinomial Naive Bayes classifier is trained on the training data. This classifier is well-suited for text classification tasks because it works well with features that represent word counts or frequencies. It assumes that the features are independent of each other, given the class label, which is a reasonable assumption for text data.

During training, the Multinomial Naive Bayes classifier learns the probability distribution of words or features for each class label in the training data. This information is used to classify new or unseen text data by calculating the probability that a given document belongs to each class label, based on the observed word frequencies.

Once the model is trained, it can be evaluated on the testing data to assess its performance in terms of metrics such as accuracy, precision, recall, and F1-score. These metrics provide insights into how well the model generalizes to new, unseen text data.

Overall, creating a text classifier model using the Multinomial Naive Bayes classifier involves preprocessing the data, training the classifier on the training data, and evaluating its performance on the testing data. With its simplicity and effectiveness, the Multinomial Naive Bayes classifier is a popular choice for text classification tasks, including sentiment analysis, spam detection, and topic classification.

## Logistic Regression-

Creating a machine learning text classifier model using logistic regression involves several key steps. Initially, the text data is preprocessed, which may involve tasks like tokenization, removing stop words, and converting words into numerical representations. This processed data is then divided into training and testing sets to train and evaluate the model's performance.

Next, the logistic regression model is trained on the training data. In text classification tasks, logistic regression is often used as a binary classifier, where it predicts the probability that a given document belongs to a particular class (e.g., spam or non-spam). The model learns the relationship between the features (word frequencies or other text representations) and the target labels (class labels) in the training data.

During training, logistic regression optimizes the parameters of the model to maximize the likelihood of observing the training data given the model parameters. This optimization process typically involves techniques like gradient descent or other optimization algorithms.

Once the model is trained, it can be evaluated on the testing data to assess its performance. Metrics such as accuracy, precision, recall, and F1-score are commonly used to measure the model's effectiveness in correctly classifying text documents.

Overall, creating a text classifier model using logistic regression involves preprocessing the data, training the model on the training data, and evaluating its performance on the testing data. Logistic regression is a widely used and interpretable method for text classification tasks, making it a popular choice for applications like sentiment analysis, spam detection, and topic classification.

## Support Vector Machine-

Creating a machine learning text classifier model using Support Vector Machine (SVM) involves several essential steps. Initially, the text data undergoes preprocessing, which includes tasks such as tokenization, removing stop words, and converting text into numerical representations. This processed data is then divided into training and testing sets to train and evaluate the model's performance.

Next, the SVM model is trained on the training data. SVM is a powerful algorithm for binary classification tasks, including text classification. It works by finding the optimal hyperplane that separates the data points into different classes while maximizing the margin between them. In text classification, SVM learns to classify documents based on the features (word frequencies or other text representations) and their corresponding class labels in the training data.

During training, SVM optimizes the hyperplane parameters to achieve the best separation between the classes. This optimization process typically involves solving a convex optimization problem using techniques like gradient descent or other optimization algorithms.

Once the SVM model is trained, it can be evaluated on the testing data to assess its performance. Metrics such as accuracy, precision, recall, and F1-score are commonly used to measure the model's effectiveness in correctly classifying text documents.

Overall, creating a text classifier model using SVM involves preprocessing the data, training the model on the training data, and evaluating its performance on the testing data. SVM is known for its effectiveness in handling high-dimensional data and has been successfully applied to various text classification tasks, including sentiment analysis, spam detection, and topic classification.

**Random Forest Classifier-**

Creating a machine learning text classifier model using the Random Forest classifier entails several crucial steps. Initially, the text data is preprocessed, involving tasks such as tokenization, removing stop words, and transforming text into numerical representations. Subsequently, this processed data is divided into training and testing sets to facilitate training and evaluate the model's performance.

Next, the Random Forest classifier is trained on the training data. Random Forest is a versatile ensemble learning algorithm that constructs multiple decision trees during training. Each decision tree is trained on a random subset of the features and a random subset of the training data. During training, each decision tree independently learns to classify text documents based on the features (word frequencies or other text representations) and their corresponding class labels in the training data.

Once the Random Forest classifier is trained, it can be evaluated on the testing data to gauge its performance. Metrics like accuracy, precision, recall, and F1-score are commonly used to assess the model's effectiveness in accurately classifying text documents.

Overall, creating a text classifier model using Random Forest involves preprocessing the data, training the model on the training data, and evaluating its performance on the testing data. Random Forest is known for its robustness, scalability, and ability to handle high-dimensional data, making it a popular choice for text classification tasks, including sentiment analysis, spam detection, and topic classification.

**Neural Networks and NLP-**

Creating a machine learning text classifier model using neural networks and applying Natural Language Processing (NLP) offers several advantages over previously designed models. Initially, the text data is preprocessed, including tasks such as tokenization, removing stop words, and transforming text into numerical representations. These representations capture the semantic meaning and context of the text, enabling the model to learn complex patterns and relationships in the data.

Neural networks, particularly deep learning models like Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs), are well-suited for text classification tasks due to their ability to capture intricate features and hierarchies within the data. For instance, CNNs excel at extracting local features from text, while RNNs are adept at capturing sequential dependencies. By leveraging these neural network architectures, the model can learn representations of text data that are more nuanced and informative, leading to improved classification performance.

*Benefits-*

Neural networks can automatically learn relevant features from the raw text data, eliminating the need for manual feature engineering. This allows the model to adapt to the unique characteristics of the data and discover hidden patterns that may not be apparent to human observers. Furthermore, neural networks can handle large-scale datasets efficiently, making them suitable for tasks involving vast amounts of text data, such as sentiment analysis, document classification, and topic modeling.

Overall, creating a text classifier model using neural networks and applying NLP techniques offers enhanced performance, flexibility, and scalability compared to previously designed models. By leveraging the power of deep learning and NLP, the model can effectively analyze and classify text data, leading to more accurate predictions and actionable insights.

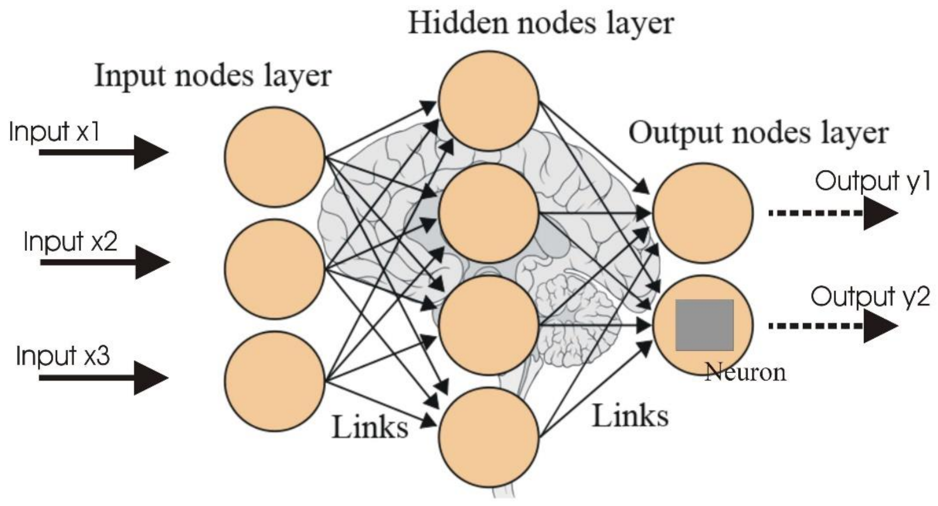


*Fig-1: Working of neural networks*

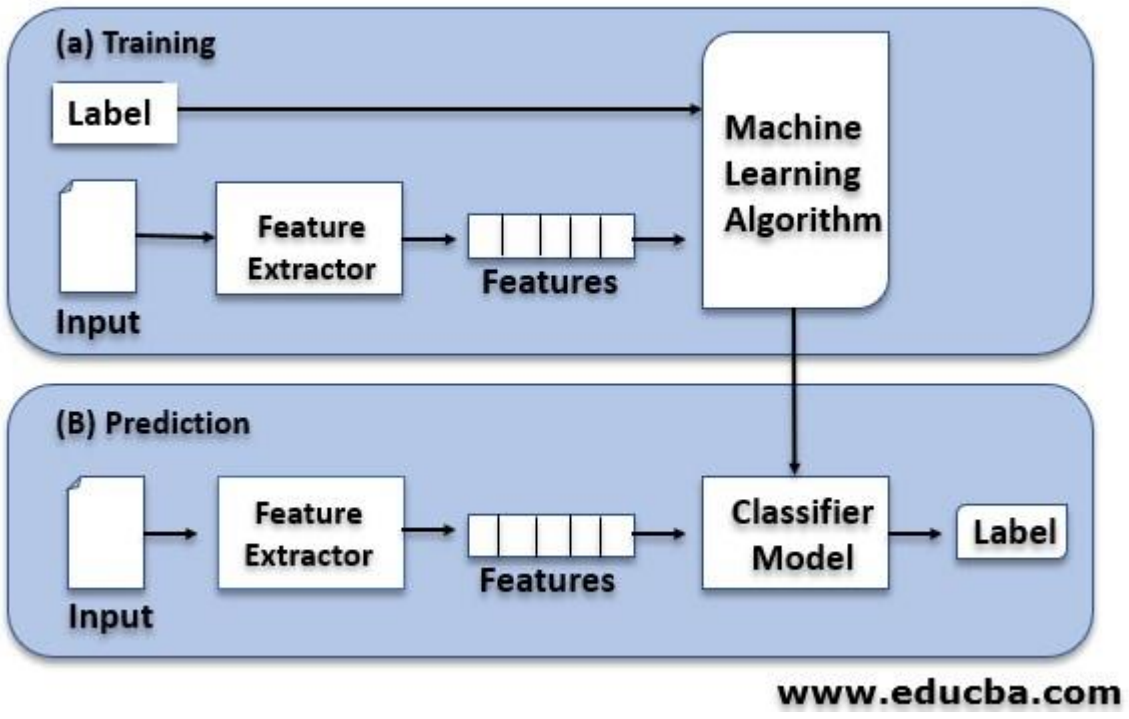


*Fig-2: Showing the working of model*

## 1.8 IMPORTANT DEFINITIONS:

## Machine Learning and Artificial Intelligence

The term artificial intelligence is frequently applied to the intellectual process characteristic of humans, such as reasoning ability, discovering new ways, or learning from previous experiences. The term is also applicable to any machine that exhibits characteristics of a human mind such as learning and problem-solving.

ML is a subset of AI that allows software applications to run the programs efficiently and accurately without being explicitly programmed to perform it. ML algorithms use previous data as input to predict new values as output.

Many of today's progressing companies like Google, Uber, Netflix, and Facebook, make Machine learning a central part of their operations. Older machine learning is categorized as to how an algorithm learns to be more accurate in predicting data.

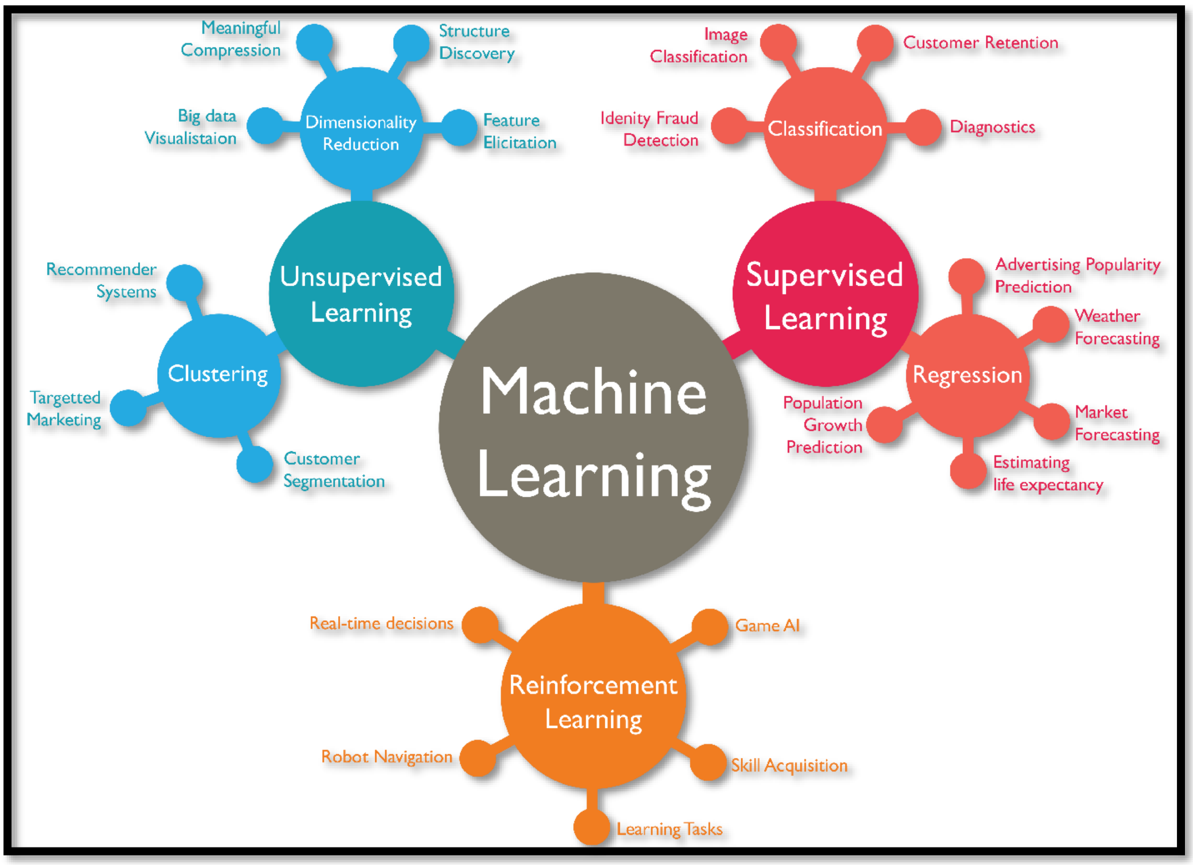


Fig-3: *Machine learning*

There are four basic approaches:

- Supervised machine learning algorithm: It is the type of ML8* algo* in which machines are trained using labeled data and on the basis of that output is predicted. Once the training process gets completed, the model is tested on basis of the test dataset, and the output is predicted.

- Unsupervised machine learning algorithm: It is the type of ML algo in which unlabeled data is present and the model itself finds the hidden patterns and sequences from the given data. Data is grouped according to similarities and represented data in compressed format.

- Semi-supervised machine learning algorithm: This approach combines elements of both supervised and unsupervised learning. It involves training the model using a combination of labeled and unlabeled data. The labeled data helps provide some guidance to the model, while the unlabeled data allows the model to discover additional patterns and relationships.

- Reinforcement learning algorithm: In this approach, an agent learns to interact with an environment and take actions based on trial and error. The agent receives feedback in the form of rewards or penalties, which helps it learn the optimal actions to maximize the reward over time. Reinforcement learning is often used in scenarios where the optimal solution is not known in advance.

Machine learning algorithms have revolutionized industries by enabling companies to extract valuable insights from vast amounts of data. They have been applied in various domains, such as image and speech recognition, natural language processing, recommendation systems, fraud detection, and autonomous vehicles.

The advancements in machine learning have paved the way for more sophisticated techniques, such as deep learning, which involves training neural networks with multiple layers to learn complex patterns and representations. Deep learning has achieved remarkable success in tasks such as image classification, speech recognition, and natural language processing.

As the field of machine learning continues to evolve, researchers and practitioners are exploring new algorithms, improving existing ones, and addressing challenges such as interpretability, fairness, and ethics. Additionally, the integration of machine learning with other fields, such as computer vision, robotics, and healthcare, presents new opportunities for innovation and development.

In conclusion, machine learning is a subset of artificial intelligence that enables software applications to learn and make predictions without explicit programming. With different approaches like supervised, unsupervised, semi-supervised, and reinforcement learning, machine learning algorithms have become essential tools for companies across various industries. The ongoing advancements in machine learning offer promising avenues for solving complex problems, making informed decisions, and driving innovation in the years to come.

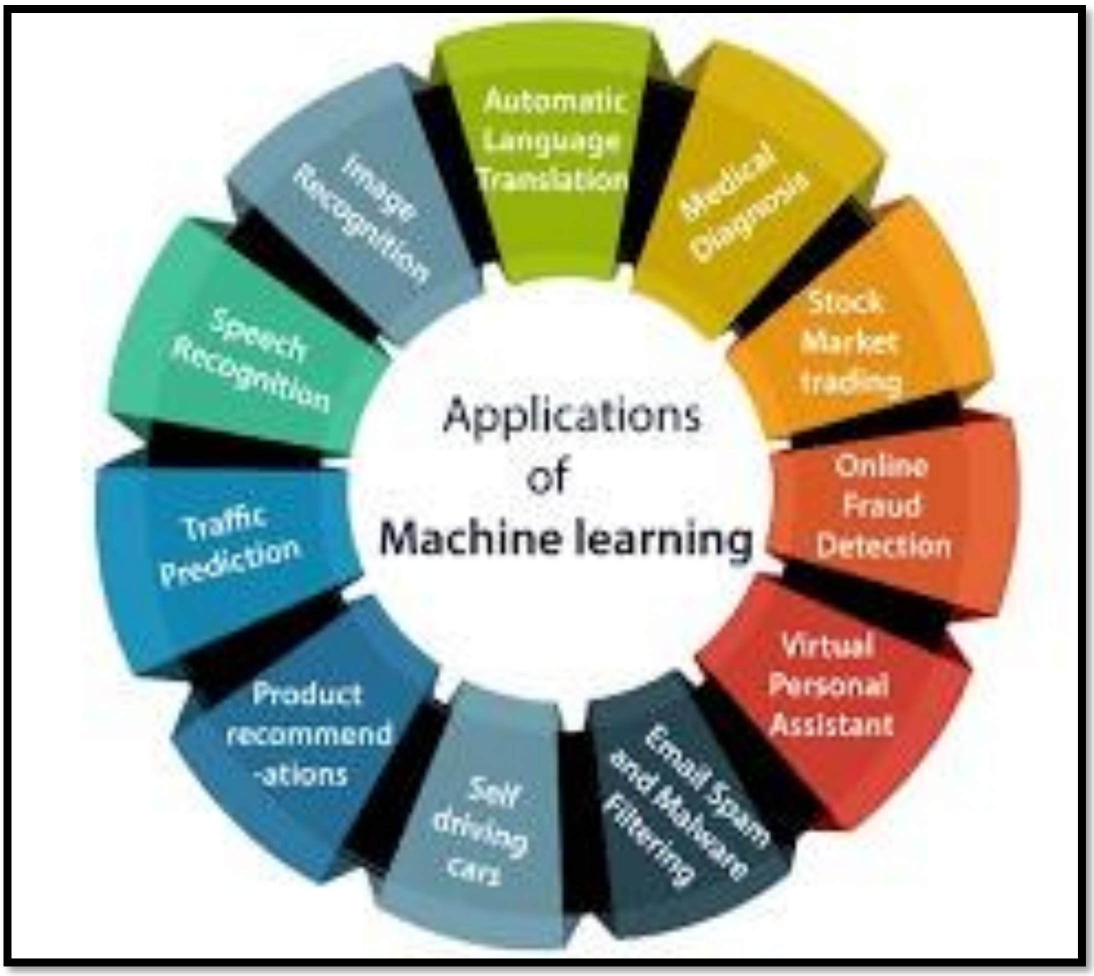

Applications of Machine learning
Automatic Language Translation
Medical Diagnosis
Stock Market trading
Online Fraud Detection
Virtual Personal Assistant
Email Spam and Malware Filtering
Self driving cars
Product recommend -ations
Traffic Prediction
Speech Recognition
Image Recognition

Fig-4: *Machine learning applications*

## DEEP LEARNING:

Deep Learning is an area of machine learning and artificial intelligence that focuses on training deep neural networks, which have numerous layers, to accomplish tasks such asimage and audio recognition, natural language processing, and decision making. Deep learning methods use a combination of non-linear transformations to model high-level abstractions and patterns in data.

Deep learning has the following key characteristics:

- Multiple Layers: Deep neural networks include multiple layers that allow them to learn complex data representations.
- Feature Learning: Deep learning systems learn useful features or representations from raw data automatically, removing the need for manual feature engineering.
- Hierarchical Learning: Deep learning models learn hierarchical representations, with each layer capturing multiple degrees of abstraction ranging from simple to complicated.
- Neural Networks: Deep learning models are commonly built on artificial neural networks, which are inspired by the structure and function of the human brain.
- Large Datasets: Deep learning frequently necessitates the use of large datasets for training in order to acquire correct and generalizable patterns.

Deep learning has demonstrated great effectiveness in a variety of applications, including image and video analysis, speech recognition, natural language understanding, recommendation systems, and autonomous driving. It has transformed the field of machine learning and contributed significantly to important advances in artificial intelligence.

## PYTHON

Nowadays, Python has emerged as one of the most sought-after programming languages in the software development industry. Its popularity can be attributed to a multitude of factors that make it a preferred choice for various projects and applications.

Python is an interpreted language, which means that it does not require a separate compilation step. This makes the development process faster and more flexible, allowingfor quick prototyping and iterative development. The interactive nature of Python facilitates an interactive coding environment, enabling developers to experiment, test, anddebug code more efficiently.

Python's object-oriented programming (OOP) paradigm further enhances its usability andcode organization. OOP allows for the creation of reusable and modular code, promotingbetter code maintenance and scalability. By leveraging the principles of encapsulation, inheritance, and polymorphism, developers can build complex software systems with ease.

One of the key advantages of Python is its vast ecosystem of inbuilt libraries and frameworks. These libraries provide a wide range of functionalities that can be readily incorporated into projects, saving developers significant time and effort. For instance, inthe context of the traffic monitoring project, Python's OpenCV library offers extensive capabilities for image and video processing, making it an ideal choice for handling mediamaterials. Additionally, the availability of deep learning frameworks such as TensorFlow and Pytorch allows for the implementation of complex machine learning models, such asYOLO and ResNet, enabling advanced object detection and classification.
The version of Python used for the project is an important consideration. The choice of Python 3.9 was made based on compatibility and functionality requirements. Certain libraries and frameworks may have specific version dependencies, and Python 3.9 ensuresthat these dependencies are met. Moreover, Python 3.9 introduces new features and enhancements, improving the overall development experience and performance of the system.

In summary, Python's popularity in the software development industry can be attributed to its interpreted nature, support for object-oriented programming, extensive library ecosystem, and compatibility with deep learning frameworks. By leveraging these features, developers can create efficient, scalable, and sophisticated systems like the trafficmonitoring project, streamlining the development process and delivering high-quality results.

## PYCHARM:

An IDE consists of an editor and a compiler that we use to write and compile programs. It has a combination of features required for developing software. An IDE consists of aneditor and a compiler that we use to write and compile programs. It has a combination offeatures required for developing software**.** It supports two versions: v2.x and v3.x. The PyCharm specifications that we have used in our project is a 64bit OS system and the version is 3.8.

An integrated development environment (IDE) is a software tool that combines an editorand a compiler or interpreter to facilitate software development. It provides a comprehensive set of features and functionalities required for writing, debugging, and compiling programs.

In the context of Python development, PyCharm is a popular IDE that offers a rich set oftools and features tailored specifically for Python programming. PyCharm supports bothPython 2.x and Python 3.x versions, allowing developers to work with the version that suits their project requirements.

For our project, we have utilized PyCharm as the IDE of choice. PyCharm provides an intuitive and user-friendly interface, enabling efficient coding and development. Its powerful editor includes features such as syntax highlighting, code completion, code navigation, and refactoring tools, which enhance productivity and code quality.

Furthermore, PyCharm offers seamless integration with the Python interpreter and supports various frameworks and libraries commonly used in Python development. It provides built-in support for version control systems, debugging tools, and automated testing frameworks, making it easier to manage and test the codebase.

The specifications for PyCharm in our project include a 64-bit operating system (OS) environment. This ensures compatibility with modern hardware and allows the IDE to take advantage of the available resources for optimal performance. The specific version used in our project is PyCharm 3.8, which corresponds to the 2020.3 release. This versionincludes enhancements, bug fixes, and updates that improve the overall development experience and stability.

By utilizing PyCharm as the IDE for our project, we benefit from its robust features, streamlined workflow, and compatibility with Python 3.8. This combination allows us to develop and manage our code effectively, ensuring efficient software development and a smooth programming experience.

## FLASK

Flask is a versatile and lightweight web framework for building web applications using Python. It provides developers with the necessary tools, libraries, and patterns to create web applications quickly and efficiently. One of Flask's key features is its simplicity and ease of use, making it an excellent choice for both beginners and experienced developers alike. With Flask, developers can define routes to handle different URL endpoints, render templates to generate dynamic HTML content, and manage user sessions and authentication. Flask follows the WSGI (Web Server Gateway Interface) specification, allowing seamless integration with web servers like Apache or Nginx. Moreover, Flask is highly extensible, with a vibrant ecosystem of third-party extensions that add functionality such as database integration, authentication, and API development. Its minimalist design philosophy and flexibility make Flask a popular choice for building a wide range of web applications, from simple static websites to complex dynamic platforms. Additionally, Flask's well-documented nature and active community support make it an attractive option for developers seeking a reliable and efficient framework for their web development projects.

Furthermore, Flask's lightweight nature makes it highly performant, ensuring that web applications built with Flask can handle high volumes of traffic efficiently. Its modular architecture also enables easy integration with other Python libraries and frameworks, empowering developers to leverage the vast ecosystem of tools available within the Python ecosystem.

Additionally, Flask fosters a culture of community collaboration and contribution, with a vibrant ecosystem of extensions and plugins developed by enthusiasts and experts alike. These extensions cover a wide range of functionalities, including database integration, authentication, authorization, caching, and more, allowing developers to extend the capabilities of their Flask applications with ease.

Overall, Flask's simplicity, flexibility, and performance make it a preferred choice for developers looking to build robust and scalable web applications. Whether you're a seasoned developer or just starting out, Flask provides an intuitive and powerful framework for bringing your web development ideas to life.

## JUPYTER NOTEBOOK

Jupyter Notebook is a powerful and versatile tool that revolutionizes the way researchers, data scientists, and educators interact with code and data. It provides an interactive computing environment where users can write and execute code, visualize data, and share insights—all within a single, browser-based interface. One of the key features of Jupyter Notebook is its support for multiple programming languages, including Python, R, Julia, and Scala, making it accessible to a wide range of users across different domains. With Jupyter Notebook, users can create documents containing live code, equations, visualizations, and narrative text, enabling them to conduct exploratory data analysis, prototyping machine learning models, and documenting research findings in a seamless and integrated manner.

Moreover, Jupyter Notebook fosters a culture of collaboration and reproducibility by allowing users to share their notebooks with colleagues, collaborators, and the broader community. This promotes transparency and open science, as researchers can easily share their code, methodologies, and results with others, facilitating knowledge exchange and collaboration. Additionally, Jupyter Notebook's support for rich media, such as images, videos, and interactive widgets, enhances the presentation of results and insights, making it easier for users to communicate their findings effectively.

Furthermore, Jupyter Notebook is highly extensible, with a rich ecosystem of extensions, plugins, and integrations that enable users to customize and extend its functionality to suit their specific needs. Whether you're conducting data analysis, developing machine learning models, teaching a class, or sharing research findings, Jupyter Notebook provides a flexible and intuitive platform for working with code and data. Its versatility, ease of use, and collaborative features make it an indispensable tool for anyone working with code and data in a variety of domains.

## KAGGLE

Kaggle is an invaluable platform that serves as a hub for data science enthusiasts, machine learning practitioners, and researchers alike. It provides a diverse range of datasets, competitions, and resources that empower users to explore, analyze, and apply data-driven solutions to real-world problems. One of Kaggle's standout features is its extensive collection of datasets spanning various domains, from healthcare and finance to natural language processing and computer vision. These datasets serve as a treasure trove of valuable information, enabling users to conduct exploratory data analysis, develop predictive models, and derive meaningful insights.

Moreover, Kaggle hosts a plethora of machine learning competitions where participants can test their skills, collaborate with peers, and compete for prizes. These competitions cover a wide range of topics and challenges, ranging from predicting customer churn and detecting fraudulent activities to classifying images and translating languages. Participating in Kaggle competitions provides an opportunity for individuals to sharpen their data science skills, learn from others, and gain practical experience in solving real-world problems.

Additionally, Kaggle offers a wealth of educational resources, including tutorials, courses, and kernels (Jupyter Notebooks), that cater to users of all skill levels. These resources cover various topics in data science, machine learning, and artificial intelligence, providing valuable learning opportunities for beginners and experienced practitioners alike. Furthermore, Kaggle fosters a vibrant and supportive community where users can ask questions, share knowledge, and collaborate on projects, creating a dynamic ecosystem of learning and innovation.

Furthermore, Kaggle provides access to Google Cloud Platform (GCP) resources, enabling users to leverage scalable computing power for running machine learning experiments, training models, and deploying solutions. This integration with GCP enhances Kaggle's capabilities and scalability, allowing users to tackle large-scale data projects and complex computational tasks with ease.

In summary, Kaggle serves as a hub for data science enthusiasts and professionals, offering a wealth of datasets, competitions, educational resources, and community support. It empowers users to explore, learn, collaborate, and innovate in the exciting field of data science, driving advancements and breakthroughs in various domains.

## **GITHUB**

GitHub is a web-based platform used for version control and collaboration on software development projects. It offers a range of features to facilitate the development process, including version control, code hosting, issue tracking, and project management tools. GitHub uses the Git version control system, which allows developers to track changes to their code over time, collaborate with other developers, and maintain a history of their project's development.

One of GitHub's key features is its repository hosting service, which allows developers to store their code in repositories (repos) and manage it efficiently. Repositories can be public, allowing anyone to view and contribute to the code, or private, restricting access to only authorized collaborators. GitHub provides a user-friendly interface for managing repositories, including features like branching, merging, and pull requests, which streamline the development workflow.

GitHub also offers robust collaboration tools, such as issue tracking and pull requests. Developers can use issues to report bugs, request features, or discuss ideas, while pull requests enable developers to propose changes to the codebase and collaborate on those changes before merging them into the main codebase.

Furthermore, GitHub provides integration with various third-party tools and services, such as continuous integration (CI) platforms, code review tools, and project management software, allowing developers to customize their workflow and streamline their development process.

Overall, GitHub is a powerful platform that plays a central role in modern software development, enabling developers to collaborate effectively, manage their codebase efficiently, and build high-quality software products.

## 2. OBJECTIVES

The objectives of a research paper text classification model may vary depending on the specific goals and requirements of the research project. However, some common objectives include:

1. **Automated Document Organization**: To develop a text classification model that automatically categorizes research papers into predefined topics, fields, or domains, aiding in document organization and retrieval.

2. **Topic Detection and Analysis**: To identify and analyze emerging trends, themes, or patterns within research papers, enabling researchers to gain insights into the current state of research in a particular field.

3. **Literature Review Assistance**: To assist researchers in conducting literature reviews by automatically categorizing and summarizing relevant research papers, saving time and effort in identifying key literature.

4. **Keyword Extraction and Highlighting**: To extract important keywords or phrases from research papers, allowing researchers to quickly identify and focus on the most relevant information within documents.

5. **Plagiarism Detection**: To develop a text classification model capable of identifying instances of plagiarism or academic misconduct within research papers, helping to maintain academic integrity and ethical standards.

6. **Prediction of Paper Impact**: To predict the potential impact or citation count of research papers based on their content and characteristics, aiding researchers and publishers in evaluating the significance of publications.

7. **Cross-Domain Research Exploration**: To enable cross-domain research exploration by classifying research papers into interdisciplinary categories or by identifying connections and overlaps between different fields of study.

8. **Quality Assessment and Filtering**: To develop a text classification model that assesses the quality and reliability of research papers based on various criteria, such as methodology, credibility of sources, and experimental results.

9. **Customization and Adaptation**: To create a flexible and adaptable text

classification model that can be customized and fine-tuned for specific research domains, languages, or document types.

10. **Integration with Research Tools**: To integrate the text classification model with existing research tools, databases, or platforms, providing researchers with seamless access to advanced document analysis and organization capabilities.

These objectives aim to address the challenges and requirements of research paper management, analysis, and exploration, ultimately enhancing the efficiency, effectiveness, and impact of research endeavors in various disciplines.

# 3. LITERATURE REVIEW

## 3.1 EXISTING METHODS

In the existing landscape of natural language processing (NLP) research, the analysis of medical research abstracts has emerged as a crucial area of focus. Prior to the introduction of the PubMed 200k RCT dataset, researchers faced challenges due to the lack of large-scale datasets tailored specifically for sequential sentence classification in medical texts. While some smaller datasets existed, they were either not publicly available, focused on non-RCT abstracts, or limited in size, hindering the development of accurate algorithms for sequential sentence classification tasks.

One of the most notable challenges in the existing system was the absence of structured abstracts in a significant portion of published RCTs. Unstructured abstracts made it difficult for researchers to quickly locate relevant information, slowing down tasks such as literature review and evidence-based medicine.

## 3.2 PROPOSED SYSTEM

The introduction of the PubMed 200k RCT dataset presents a novel solution to the limitations of the existing system. This dataset, consisting of approximately 200,000 abstracts of randomized controlled trials (RCTs) and totaling 2.3 million sentences, addresses the need for large-scale datasets in sequential sentence classification tasks within the medical domain.

The proposed method involves the meticulous labeling of each sentence in the dataset with one of five classes: background, objective, method, result, or conclusion. This labeling scheme provides researchers with a structured framework for analyzing and categorizing information within medical abstracts.

Furthermore, the dataset is split into training, validation, and test sets, enabling researchers to develop and evaluate algorithms for sequential sentence classification effectively. By providing a comprehensive resource with a sizable volume of meticulously labeled data, the proposed system aims to facilitate the development of accurate algorithms for tasks such as evidence-based medicine, literature review, and information retrieval in the medical domain.

In addition to the challenges mentioned, existing methods in medical text classification often relied on handcrafted features or shallow machine learning models, which may not capture the complex relationships and nuances present in medical texts. This limitation hindered the development of highly accurate and robust classifiers, particularly for tasks requiring sequential sentence classification.

Moreover, the lack of standardized evaluation metrics and benchmarks posed challenges for comparing the performance of different classification algorithms. Without a common benchmark dataset and evaluation framework, it was challenging to assess the generalizability and effectiveness of proposed methods across different domains and datasets.

To address these limitations, recent advancements in deep learning and NLP have shown promising results in medical text classification tasks. Deep learning models, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have demonstrated superior performance in capturing intricate patterns and dependencies in medical texts, leading to more accurate and robust classifiers.

Additionally, the emergence of transfer learning techniques, such as pre-trained language models like BERT (Bidirectional Encoder Representations from Transformers), has further improved the performance of medical text classification models. By leveraging large-scale pre-trained language models trained on vast amounts of text data, researchers can fine-tune these models on domain-specific datasets, achieving state-of-the-art results with minimal additional training data.

Overall, the combination of large-scale datasets like PubMed 200k RCT, advanced deep learning architectures, and transfer learning techniques has paved the way for significant advancements in medical text classification. These developments hold promise for improving various applications in healthcare, including clinical decision support, disease diagnosis, and biomedical research.

# 4. METHODOLOGY:

The text classification machine learning model developed using the PubMed 200k RCT dataset represents a significant advancement in medical text classification. Leveraging the rich and diverse collection of abstracts from randomized controlled trials (RCTs), this model is trained to accurately classify medical text into relevant categories or topics, such as treatment methods, clinical outcomes, or disease classifications.

By harnessing the power of machine learning algorithms and natural language processing (NLP) techniques, the model can effectively analyze the text of medical abstracts, extract key features, and learn to classify them based on their semantic content. Through extensive training on the PubMed 200k RCT dataset, the model gains a deep understanding of the complex terminology and nuances specific to medical literature, enabling it to make accurate predictions with high confidence.

## 4.1 Modules used:

### 4.1.1 Modules Downloaded To Create Environment:

- PANDAS: Pandas is a powerful Python library for data manipulation and analysis, offering data structures and functions to efficiently handle structured data such as tables and time series, making it a go-to tool for data scientists and analysts.

- SKLEARN: Scikit-learn is a versatile machine learning library in Python, providing simple and efficient tools for data mining and analysis. It offers a wide range of algorithms for classification, regression, clustering, dimensionality reduction, and more, along with tools for model evaluation and selection.

- NLTK: NLTK, or Natural Language Toolkit, is a comprehensive library for natural language processing (NLP) tasks in Python. It provides tools and resources for tasks such as tokenization, stemming, lemmatization, part-of-speech tagging, parsing, and more, making it a valuable resource for NLP practitioners and researchers.

- STRING: The string module in Python provides a collection of useful functions and constants for working with strings. It includes functions for string manipulation, formatting, and searching, as well as constants for ASCII characters, digits, punctuation, and whitespace, making it a convenient tool for text processing tasks.

- TENSORFLOW: TensorFlow is an open-source machine learning framework developed by Google. It provides a comprehensive ecosystem of tools, libraries, and resources for building and deploying machine learning models, with support for deep learning algorithms, distributed computing, and production-ready deployment, making it a popular choice for researchers and practitioners in the field of AI and machine learning.

- STREAMLIT: Streamlit is an open-source Python library that simplifies the process of creating web applications for data science and machine learning projects. With its intuitive API and declarative syntax, Streamlit allows developers to build interactive and responsive web applications directly from Python scripts, without the need for HTML, CSS, or JavaScript. Streamlit's extensive collection of widgets and built-in components enables users to create sophisticated data visualizations, interactive charts, and responsive user interfaces with minimal effort. Its seamless integration with popular data science libraries such as Pandas, Matplotlib, and TensorFlow makes it a preferred choice for data scientists and machine learning engineers looking to share and showcase their work effectively. Whether building simple prototypes or complex dashboards, Streamlit offers a streamlined development experience, empowering users to bring their data stories to life with ease.

### 4.1.2 Process of building Medical Text Classification Model

STEP-1: Install the necessary modules and libraries.

STEP-2: Import the modules and libraries

STEP-3: Apply necessary preprocessing steps.

STEP-4: Apply appropriate classification model and train the data.

STEP-5: Test and validate the data.

STEP-6: Deploy the model.

### 4.1.3 Packages and Methods Required:

1. Pandas:
    - read_csv
    - to_csv
2. Sklearn:

- feature_extraction.text
- model_selection
- naïve_bayes
- metrics
- linear model
- ensemble
- svm
- pipeline

3. Tensorflow:
    - models
    - layers
    - callbacks
4. NLTK:
    - corpus
    - tokenize
    - stem

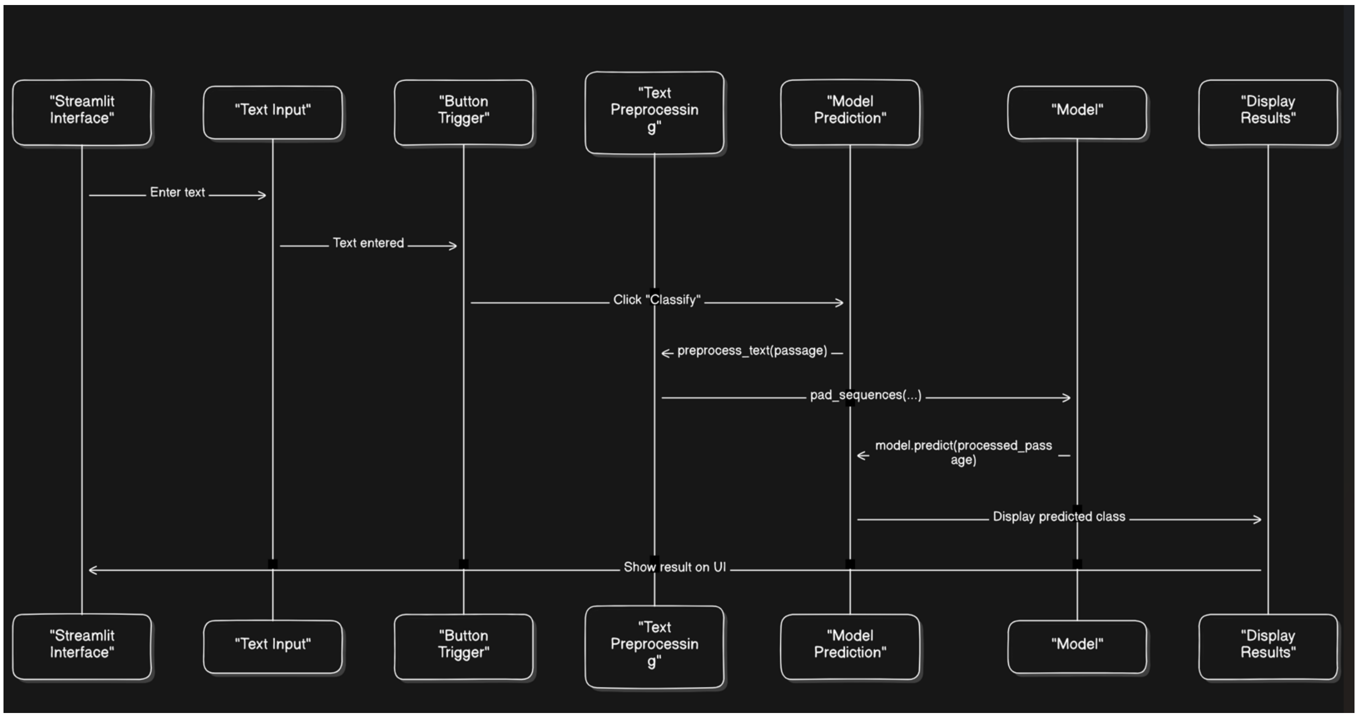


Fig-5: UML sequence diagram

## 4.2 Libraries used:

- **Feature Extraction:** Feature extraction involves transforming raw data into a format suitable for machine learning algorithms. In natural language processing (NLP), this often means converting text documents into numerical feature vectors using techniques like bag-of-words or TF-IDF.

- **Model Selection:** Model selection is the process of choosing the best machine learning algorithm or model for a given task. It involves evaluating multiple candidate models using metrics like accuracy or F1-score and selecting the one that performs best on a validation dataset.

- **TF-IDF Vectorizer:** TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer is a feature extraction technique used in NLP to convert text documents into numerical feature vectors. It assigns weights to terms based on their frequency in a document and rarity across all documents, helping to highlight important terms.

- **train_test_split:** train_test_split is a function from the sklearn library used to split a dataset into training and testing sets. It randomly divides the data, allowing models to be trained on one portion and evaluated on another to assess performance.

- **Naive Bayes:** Naive Bayes is a family of simple probabilistic classifiers based on Bayes' theorem. Despite their simplicity, they are often effective for text classification tasks, making them popular in NLP.

- **MultinomialNB:** Multinomial Naive Bayes is a variant of the Naive Bayes algorithm suitable for classification with discrete features, such as word counts. It's commonly used in NLP tasks like text classification.

- **Metrics:** Metrics are measures used to evaluate the performance of machine learning models. Common metrics include accuracy, precision, recall, F1-score, and OC-AUC.

- **Accuracy Score:** Accuracy score is a metric used to measure the proportion of correctly classified instances in a classification task. It's calculated by dividing the number of correct predictions by the total number of predictions.

- **Classification Report:** A classification report provides a comprehensive summary of the performance of a classification model, including metrics like precision, recall, F1-score, and support for each class.

- **NLTK:** NLTK (Natural Language Toolkit) is a Python library for NLP tasks such as tokenization, stemming, lemmatization, parsing, and more. It provides tools and resources for working with human language data.

- **Corpus:** In NLP, a corpus refers to a collection of text documents used for analysis or training machine learning models. It may consist of documents from a specific domain, language, or genre.

- **Stopwords:** Stopwords are common words (e.g., "the", "and", "is") that are often filtered out during text preprocessing to improve the performance of NLP tasks like text classification or sentiment analysis.

- **nltk.tokenize:** nltk.tokenize is a module in NLTK used for tokenization, the process of breaking text into individual words or tokens. It provides functions like word_tokenize for tokenizing words and sent_tokenize for tokenizing sentences.

- **Word Tokenize:** Word Tokenize is a process of breaking down a text into individual words or tokens. It's a common preprocessing step in NLP tasks like text classification or sentiment analysis.

- **nltk.stem:** nltk.stem is a module in NLTK used for stemming, the process of reducing words to their root or base form. It provides algorithms like PorterStemmer and LancasterStemmer for stemming words.

- **WordNetLemmatizer:** WordNetLemmatizer is a tool in NLTK used for lemmatization, the process of reducing words to their base or dictionary form (lemmas). It's often used as an alternative to stemming for better accuracy.

- **String:** The string module in Python provides a collection of useful functions and constants for working with strings. It includes functions for string manipulation, formatting, and searching.

- **nltk.punkt:** nltk.punkt is a module in NLTK used for sentence tokenization, the process of breaking down a text into individual sentences. It provides a pre-trained tokenizer for various languages.

- **Multinomial_NB:** Multinomial Naive Bayes is a variant of the Naive Bayes algorithm suitable for classification with discrete features, such as word counts. It's commonly used in NLP tasks like text classification.

- **Logistic Regression:** Logistic Regression is a statistical method used for binary classification tasks. Despite its name, it's commonly used for multiclass classification, making it a popular choice in NLP.

- **Random Forest Classifier:** Random Forest Classifier is an ensemble learning method that builds multiple decision trees and combines their predictions to improve accuracy. It's often used for both classification and regression tasks.

- **SVC:** SVC (Support Vector Classifier) is a supervised learning algorithm used for classification tasks. It works by finding the hyperplane that best separates classes in a high-dimensional space.

- **Pipeline:** Pipeline is a concept in machine learning where multiple steps of data processing and model building are chained together into a single workflow. It streamlines the process and ensures consistency across different experiments.

- **Sequential:** Sequential is a type of model architecture commonly used in deep learning frameworks like TensorFlow and Keras. It defines a linear stack of layers where each layer has exactly one input tensor and one output tensor.

- **Accuracy Score:** Accuracy Score is a metric used to measure the proportion of correctly classified instances in a classification task. It's calculated by dividing the number of correct predictions by the total number of predictions.

- **Dense:** Dense is a type of layer commonly used in neural network architectures. It represents a fully connected layer where each neuron is connected to every neuron in the previous and next layers.

- **Dropout:** Dropout is a regularization technique used in neural networks to prevent overfitting. It randomly drops a proportion of neurons during training, forcing the network to learn more robust features.

- **Early Stopping:** Early Stopping is a regularization technique used in training neural networks to prevent overfitting. It monitors the model's performance on a validation set and stops training when the performance starts to degrade, preventing the model from overfitting to the training data.

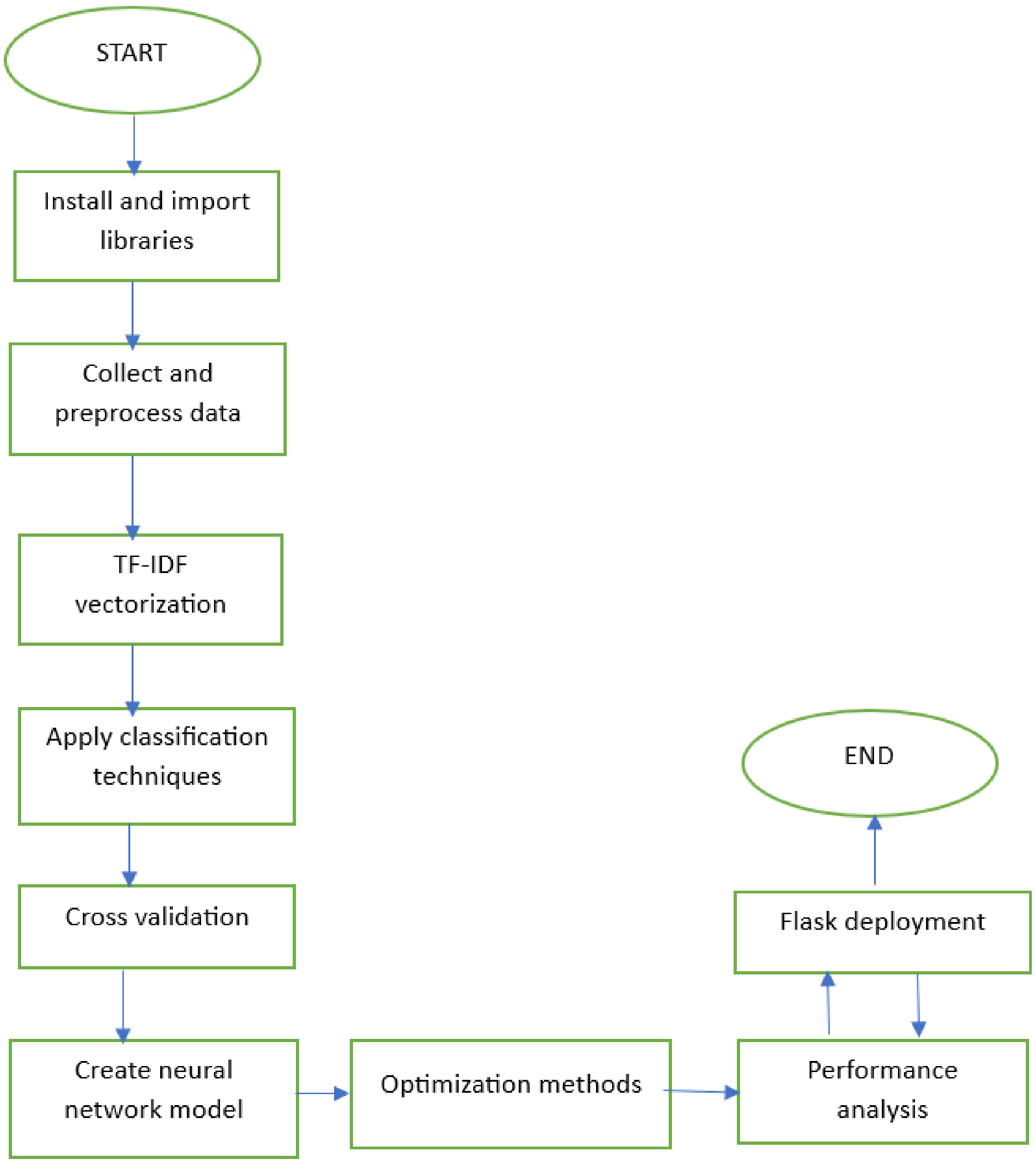

START
Install and import libraries
Collect and preprocess data
TF-IDF vectorization
Apply classification techniques
Cross validation
Create neural network model
Optimization methods
Performance analysis
Flask deployment
END

Fig-6: Flowchart of model

## 4.3 Modules Documentation

Text classification models operate by analyzing the textual content of documents and assigning them to predefined categories or classes. Initially, the text data undergoes preprocessing steps to clean and refine it, which may involve removing noise like punctuation, stopwords, and special characters. Additionally, techniques such as tokenization, stemming, and lemmatization may be applied to standardize the text for analysis.

Following preprocessing, the text is transformed into numerical feature vectors using methods like bag-of-words, TF-IDF, or word embeddings. These techniques convert the text into a format suitable for machine learning algorithms. During model training, these feature vectors are utilized to train a machine learning model to recognize patterns in the data that correlate with different categories or labels. The model learns from labeled examples to make predictions on unseen data accurately.

Once trained, the model is evaluated using validation or test datasets to assess its performance metrics, such as accuracy, precision, recall, and F1-score. This evaluation phase ensures that the model can effectively classify new, unseen text data. Finally, the trained model can be deployed in real-world applications to automate text classification tasks, enabling various applications such as sentiment analysis, topic modeling, and spam detection to improve efficiency and accuracy in handling textual data.

### 1. pyttsx3

pyttsx3 is a text-to-speech conversion library in Python. Unlike alternative libraries, it works offline and is compatible with both Python 2 and 3. An application invokes the pyttsx3.init() factory function to get a reference to a pyttsx3. Engine instance. it is a veryeasy to use tool which converts the entered text into speech. The pyttsx3 module supports

two voices first is female and the second is male which is provided by “sapi5” for windows.

### 2. Pandas

Pandas is a widely-used Python library designed for data manipulation and analysis. It offers powerful data structures, primarily DataFrame and Series, which provide flexible and intuitive ways to work with structured data. With pandas, users can easily load data from various file formats, such as CSV, Excel, SQL databases, and JSON, into a DataFrame, enabling seamless access to tabular data. Its rich collection of methods simplifies common data operations like exploration, cleaning, selection, manipulation, and analysis, making it an indispensable tool for data scientists, analysts, and researchers.

One of the key features of pandas is its ability to handle missing data effectively. Methods like `dropna()` and `fillna()` allow users to remove or replace missing values, ensuring data consistency and accuracy. Additionally, pandas provides powerful indexing and selection methods like `loc[]` and `iloc[]`, enabling users to retrieve specific rows and columns based on labels or integer indices. Its integration with popular data visualization libraries like Matplotlib and Seaborn further enhances its capabilities, allowing users to create insightful visualizations directly from pandas DataFrames.

Moreover, pandas excels in data manipulation tasks, offering functions for merging, concatenating, grouping, pivoting, and sorting data, facilitating complex data transformations with ease. Its comprehensive set of statistical and mathematical functions enables users to perform in-depth data analysis and computation effortlessly. Whether it's exploring datasets, cleaning messy data, conducting statistical analysis, or creating visualizations, pandas empowers users to tackle a wide range of data-related tasks efficiently, making it an essential tool for anyone working with data in Python.

**Available Types:**

## 2.1Pandas.read_csv

Used to read data from a CSV file into a DataFrame. It provides various parameters to customize the reading process, such as specifying the file path, delimiter, column names, data types, and handling missing values. This function automatically infers data types and handles different formats, making it convenient for loading CSV data into pandas DataFrames.

## 2.2 Pandas.to _csv

used to write data from a DataFrame to a CSV file. It allows users to save DataFrame contents to a CSV file with customizable options such as specifying the file path, delimiter, index inclusion/exclusion, and column formatting. This function enables users to export DataFrame data to CSV files in a structured format, making it suitable for sharing or further analysis.

## 3. Sklearn

Scikit-learn, often abbreviated as sklearn, is a widely-used Python library for machine learning and data mining. It provides a simple and efficient interface for implementing various machine learning algorithms and tools, making it suitable for both beginners and experienced practitioners. Sklearn offers a rich collection of modules and functions for tasks such as classification, regression, clustering, dimensionality reduction, model selection, and preprocessing.

One of the key strengths of scikit-learn is its ease of use and consistency across its API, allowing users to seamlessly switch between different algorithms and models. It provides a uniform interface for training, testing, and evaluating machine learning models, making it easy to experiment with different algorithms and techniques. Additionally, scikit-learn integrates well with other Python libraries such as NumPy, pandas, and Matplotlib, enhancing its versatility and usability in various data science workflows.

Moreover, scikit-learn is well-documented and supported by a vibrant community, offering extensive documentation, tutorials, and examples to help users get started with machine learning tasks. Its open-source nature encourages collaboration and contributions from developers worldwide, ensuring continuous improvement and innovation in the field of machine learning. Overall, scikit-learn is a powerful and essential tool for building machine learning models and conducting data analysis tasks in Python, empowering users to tackle a wide range of real-world problems with ease and confidence.

**Available Types:**

### 3.1. `TfidfVectorizer`

This method from `sklearn.feature_extraction.text` is used to convert a collection of raw documents into a matrix of TF-IDF features. TF-IDF (Term Frequency-Inverse Document Frequency) is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents. The `TfidfVectorizer` transforms text data into numerical feature vectors, where each feature represents the TF-IDF score of a word in the document. This method also provides options for preprocessing, tokenization, and feature selection to customize the vectorization process.
IMPORT: from sklearn.feature_extraction.text import TfidfVectorizer

### 3.2. `train_test_split

This method from `sklearn.model_selection` is used to split a dataset into training and testing subsets for model evaluation. It takes input arrays or matrices and splits them into random train and test sets based on a specified test size or a given train size. This function is commonly used in machine learning workflows to partition data into training data, used to train the model, and testing data, used to evaluate the model's performance on unseen data. By splitting the data into separate sets, it helps assess the model's generalization ability and avoid overfitting.
IMPORT: from sklearn.model_selection import train_test_split

### 3.3`MultinomialNB`

This method from `sklearn.naive_bayes` implements the Multinomial Naive Bayes classifier, a probabilistic classifier based on the Bayes theorem and the assumption of conditional independence between features. It is suitable for classification tasks with discrete features, such as text classification, document classification, and spam filtering. The `MultinomialNB` classifier computes the likelihood of each class given the input features and predicts the class with the highest probability. It is commonly used for text classification tasks due to its simplicity and effectiveness, especially when dealing with word frequency or TF-IDF features.
IMPORT: from sklearn.naive_bayes import MultinomialNB

### 3.4. `accuracy_score`

This method from `sklearn.metrics` is used to evaluate the accuracy of a classification model by comparing the predicted labels with the true labels of the test data. It computes the proportion of correctly predicted instances out of the total number of instances in the test dataset. The `accuracy_score` function takes two arrays of labels (true labels and predicted labels) as input and returns the accuracy score, which is a value between 0 and 1. A higher accuracy score indicates better performance of the classification model in correctly classifying instances.
IMPORT: from sklearn.metrics import accuracy_score

### 3.5. `classification_report`

This method from `sklearn.metrics` generates a comprehensive report of the performance metrics for a classification model. It provides metrics such as precision, recall, F1-score, and support for each class in the classification task. The `classification_report` function takes the true labels and predicted labels of the test data as input and returns a formatted report containing precision, recall, F1-score, and support values for each class, as well as the average values across all classes. It is useful for evaluating the performance of a classification model and gaining insights into its strengths and weaknesses for different classes.
IMPORT: from sklearn.metrics import classification_report

## 4. Tensorflow

TensorFlow is an open-source machine learning framework developed by Google that has gained widespread popularity for its flexibility, scalability, and ease of use. It provides a comprehensive ecosystem of tools and libraries for building and deploying machine learning models across a range of platforms, from mobile devices to distributed systems. TensorFlow's core functionality revolves around defining and executing computational graphs, where nodes represent mathematical operations and edges represent data flow between operations.

One of the key features of TensorFlow is its extensive support for deep learning, enabling users to build and train complex neural network architectures with ease. It offers a high-level API called Keras, which provides a user-friendly interface for building and training neural networks, making it accessible to both beginners and experts alike. TensorFlow also supports distributed computing, allowing users to train models across multiple GPUs or machines for faster training and inference.

Moreover, TensorFlow is actively maintained and continuously evolving, with frequent updates and improvements to its performance, usability, and features. It has a thriving community of developers and researchers who contribute to its development, share best practices, and collaborate on various projects. TensorFlow's versatility and extensibility make it a preferred choice for a wide range of machine learning tasks, including image recognition, natural language processing, reinforcement learning, and more. Whether it's prototyping new models, conducting research experiments, or deploying production systems, TensorFlow provides the tools and infrastructure needed to tackle challenging machine learning problems effectively.

**Available Types:**

### 4.1 Sequential

The Sequential model from tensorflow.keras.models is a linear stack of layers used for building neural network models in TensorFlow's Keras API. It allows users to create models by simply adding layers one by one, making it easy to define the architecture of a neural network. The Sequential model is suitable for building feedforward neural networks where the data flows sequentially through each layer, from the input layer to the output layer.

IMPORT : from tensorflow.keras.models import Sequential

### 4.2 Dense

The Dense layer from tensorflow.keras.layers is a fully connected layer that performs a simple matrix multiplication followed by a bias addition and an activation function. It is one of the fundamental building blocks of a neural network, connecting every neuron in the current layer to every neuron in the subsequent layer. The Dense layer allows users to specify the number of units (neurons) in the layer and the activation function to be applied.

IMPORT : from tensorflow.keras.layers import Dense

### 4.3 Dropout

The Dropout layer from tensorflow.keras.layers is a regularization technique used to prevent overfitting in neural networks by randomly dropping a fraction of input units during training. It helps reduce the interdependence between neurons and encourages the network to learn more robust features. The Dropout layer takes a dropout rate as input, specifying the fraction of input units to drop during training.

IMPORT : from tensorflow.keras.layers import Dropout

### 4.4 Early stopping

The EarlyStopping callback from tensorflow.keras.callbacks is used to stop training the neural network model early if a monitored metric stops improving. It allows users to specify a metric to monitor (such as validation loss or accuracy) and a patience parameter, indicating the number of epochs to wait before stopping training if no improvement is observed. The EarlyStopping callback helps prevent overfitting and saves time by terminating training when the model's performance plateaus.

IMPORT : from tensorflow.keras.callbacks import EarlyStopping

### 4.5 Adam optimizer

Adam stands for Adaptive Moment Estimation, and it computes adaptive learning rates for each parameter by estimating the first and second moments of the gradients. It maintains exponentially decaying moving averages of past gradients (first moment) and their squares (second moment), adjusting the learning rates accordingly. This adaptive learning rate mechanism allows Adam to converge faster and handle non-stationary objectives and noisy gradients effectively.

IMPORT: from tensorflow.keras.optimizers import Adam

## 5. NLTK

NLTK (Natural Language Toolkit) is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces and libraries for tasks such as tokenization, stemming, tagging, parsing, and semantic reasoning, making it a valuable resource for natural language processing (NLP) tasks. NLTK offers a rich collection of text processing libraries and modules, along with comprehensive documentation and educational resources, making it suitable for both beginners and advanced users interested in NLP.

One of the key features of NLTK is its extensive support for linguistic resources, including corpora, lexical resources, and grammars, which are essential for training and evaluating NLP models. NLTK provides access to a wide range of datasets and linguistic resources, such as word lists, treebanks, and annotated corpora, enabling users to experiment with different algorithms and techniques in NLP. Additionally, NLTK's modular architecture allows users to easily extend its functionality by integrating custom corpora, models, and algorithms, making it a flexible and customizable tool for NLP research and development.

Moreover, NLTK offers a wide range of tools and utilities for text processing and analysis, including tokenization, part-of-speech tagging, named entity recognition, sentiment analysis, and more. Its intuitive APIs and consistent interface make it easy to perform complex NLP tasks with minimal effort. NLTK also integrates seamlessly with other Python libraries and frameworks, such as NumPy, pandas, and scikit-learn, allowing users to leverage its capabilities in various data science workflows. Overall, NLTK is a powerful and versatile toolkit for working with human language data, empowering researchers, developers, and data scientists to unlock insights from text data and build innovative NLP applications.

**Available Types:**

## 5.1 Stopwords

The stopwords module from nltk.corpus provides a collection of common words that are often considered irrelevant or redundant in text analysis tasks. These words, known as stopwords, include articles, conjunctions, prepositions, and other function words that do not carry significant meaning or convey important information about the context of the text. By removing stopwords from text data, researchers and practitioners can focus on extracting meaningful insights and improving the accuracy of natural language processing (NLP) models.

IMPORT : from nltk.corpus import stopwords

## 5.2 Word_tokenizer

The word_tokenize function from nltk.tokenize is used to split text into individual words or tokens, making it easier to process and analyze natural language data. Tokenization is an essential preprocessing step in many NLP tasks, such as part-

of-speech tagging, named entity recognition, and sentiment analysis. By breaking down text into tokens, the word_tokenize function enables users to extract features, perform statistical analysis, and apply machine learning algorithms to text data effectively.
IMPORT: from nltk.tokenize import word_tokenize

### 5.3 WordNetLemmetizer

The WordNetLemmatizer from nltk.stem is a tool for reducing words to their base or canonical form, known as lemmas, by removing inflections and variations. Lemmatization helps normalize text data by converting words to their dictionary form, which improves the accuracy and consistency of text analysis tasks. The WordNetLemmatizer relies on WordNet, a lexical database of English words organized into synsets (sets of synonyms), to identify and map words to their corresponding lemmas. By lemmatizing text data, users can reduce word complexity, handle variations in spelling and morphology, and improve the performance of NLP models.
IMPORT: from nltk.stem import WordNetLemmatizer

## 6. Flask

Flask is a lightweight and versatile web framework for building web applications and APIs in Python. It is designed to be simple, flexible, and easy to use, making it an excellent choice for developers of all skill levels. Flask follows the WSGI (Web Server Gateway Interface) specification and provides a minimalistic approach to web development, allowing developers to focus on writing clean and efficient code.

One of the key features of Flask is its simplicity and minimalism. It comes with a small core framework and a modular design, allowing developers to add or remove features as needed. Flask provides essential tools for routing, request handling, template rendering, and session management, while additional functionality can be easily integrated using Flask extensions.

Flask is known for its flexibility and extensibility, allowing developers to customize and extend its functionality according to their project requirements. It supports various extensions and libraries for adding features such as authentication, database integration, form validation, and more. Additionally, Flask integrates seamlessly with other Python libraries and frameworks, making it easy to leverage existing code and resources.

Overall, Flask is an excellent choice for building web applications and APIs, especially for small to medium-sized projects or prototypes. Its simplicity, flexibility, and ease of use make it a popular option among developers looking for a lightweight and efficient web framework in Python. Whether you're building a simple website, a RESTful API, or a complex web application, Flask provides the tools and flexibility needed to get the job done efficiently.

The pickle module provides functions for serializing and deserializing Python objects, allowing developers to save complex data structures such as lists, dictionaries, and custom objects to disk or transmit them between processes. Pickle files have the extension .pkl and can be read and written using Python's built-in file I/O functions.

Pickle files are commonly used for various purposes in Python programming, such as:

- Saving and loading machine learning models: After training a machine learning model, developers often save the trained model to a pickle file for later use. This allows them to reuse the model without having to retrain it every time.

- Caching data: Pickle files can be used to cache computationally expensive data or intermediate results of a computation. This helps improve the performance of the program by avoiding redundant computations.

- Sharing data between Python processes: Pickle files can be used to share data between different Python processes running on the same machine or different machines. This is useful for distributed computing or inter-process communication.

## 7. Streamlit

Streamlit is an open-source Python library that simplifies the process of creating web applications for data science and machine learning projects. With Streamlit, developers can build interactive and responsive web applications directly from Python scripts, without the need for HTML, CSS, or JavaScript. Streamlit's intuitive API and declarative syntax make it easy for users to create data-driven applications quickly and efficiently.

One of the key features of Streamlit is its simplicity and ease of use. Developers can create web applications by writing Python scripts that define the layout, widgets, and interactions of the application. Streamlit takes care of the underlying web infrastructure, allowing users to focus on writing code to analyze data, visualize results, and interact with users. This streamlined development process reduces the time and effort required to create web applications, making it accessible to data scientists, machine learning engineers, and developers alike.

Streamlit provides a wide range of built-in components and widgets for creating interactive data visualizations, including charts, tables, sliders, and dropdown menus. Users can also incorporate custom JavaScript and HTML components into their Streamlit applications to enhance functionality and design. Streamlit applications can be deployed locally or on cloud platforms such as Heroku or AWS, allowing developers to share their work with colleagues and collaborators seamlessly. Overall, Streamlit is a powerful tool for building web applications for data science and machine learning projects, enabling users to create compelling and interactive experiences with minimal effort.

### 8. Neural Networks

Neural networks are a class of machine learning algorithms inspired by the structure and function of the human brain. They consist of interconnected nodes, called neurons or units, organized into layers. Each neuron receives input signals, performs a computation, and produces an output signal, which is then passed on to the next layer. Through a process called training, neural networks learn to recognize patterns and relationships in data by adjusting the weights and biases of connections between neurons.

There are several types of neural networks, each designed for specific tasks and applications. Feedforward neural networks are the simplest type, consisting of input, hidden, and output layers where information flows in one direction, from input to output. Convolutional neural networks (CNNs) are specialized for processing grid-like data, such as images, and use convolutional layers to extract features hierarchically. Recurrent neural networks (RNNs) are designed for sequential data, such as time series or natural language, and incorporate feedback loops to process inputs over time. Other types of neural networks include deep neural networks (DNNs), which have multiple hidden layers for learning complex representations, and generative adversarial networks (GANs), which are used for generating new data samples. Each type of neural network has its strengths and weaknesses, making them suitable for different tasks and domains.

## 4.4 Tools used:

### 1. Qt Designer:

Qt Designer is a graphical user interface (GUI) design tool that comes as a part of the Qtframework. Developed by The Qt Company, it allows developers to design and build interfaces for Qt applications without writing code manually. With its drag-and-drop interface, developers can design windows and dialogs quickly, and the tool generates the corresponding XML-based UI description files. These files are then integrated into the Qtapplication's codebase, streamlining the development process and promoting the separation of UI design and programming logic.

**2. BlueStacks:**

BlueStacks is an Android emulator that enables users to run Android applications on Windows and macOS devices. Developed by BlueStacks Inc., it provides a virtualized Android environment, allowing users to enjoy mobile games and apps on a larger screen.BlueStacks supports a variety of Android games and applications, and its user-friendly interface simplifies the installation and management of apps. It bridges the gap between mobile and desktop platforms, offering a seamless experience for users who wish to use their favorite Android apps on a computer.

**3. PyCharm:**

PyCharm, created by JetBrains, is a popular integrated development environment (IDE) specifically designed for Python development. It offers a comprehensive set of features, including code analysis, debugging, testing tools, and support for web development frameworks like Django and Flask. PyCharm's intelligent code completion, navigation, and version control integration enhance the productivity of Python developers. The IDE is available in both free and paid versions, with the paid version, PyCharm Professional,offering additional features such as database tools, scientific tools, and support for web frameworks. PyCharm has gained a reputation for its user-friendly interface and robust set of tools, making it a preferred choice for many Python developers.

**4.Flask:**

Flask is a lightweight and flexible web framework for building web applications and APIs in Python, renowned for its simplicity and ease of use. With Flask, developers can quickly create web applications by defining routes, request handlers, and templates, allowing for rapid prototyping and development. Flask's minimalistic approach to web development, coupled with its extensive ecosystem of extensions, enables users to customize and extend their applications according to their specific requirements. From simple websites to complex web applications, Flask provides the tools and flexibility needed to build scalable and maintainable web solutions efficiently.

## 5. Streamlit:

Streamlit is an intuitive and user-friendly Python library designed to simplify the process of creating interactive web applications for data science and machine learning projects. With Streamlit, developers can transform Python scripts into powerful and interactive web applications with minimal effort. Its declarative syntax and extensive collection of widgets enable users to create sophisticated data visualizations, interactive charts, and responsive user interfaces seamlessly. Streamlit's ability to integrate with popular data science libraries such as Pandas, Matplotlib, and TensorFlow makes it a preferred choice for data scientists and machine learning engineers looking to share and showcase their work effectively. Whether building simple prototypes or complex dashboards, Streamlit offers a streamlined development experience, empowering users to bring their data stories to life with ease.

## 6. Jupyter  notebook:

Jupyter Notebook is a widely used open-source web application that allows users to create and share documents containing live code, equations, visualizations, and narrative text. It supports various programming languages, including Python, R, and Julia, making it a versatile tool for data exploration, analysis, and visualization. Jupyter Notebooks enable users to interactively execute code cells, visualize data using libraries such as Matplotlib and Seaborn, and annotate their findings with Markdown text. Its rich features, including support for widgets, LaTeX equations, and interactive widgets, make it a popular choice among data scientists, researchers, educators, and developers for prototyping, experimentation, and collaborative work.

## 7. Kaggle:

Kaggle is an online platform for data science competitions, datasets, and notebooks, providing a collaborative environment for data scientists and machine learning practitioners to share ideas, collaborate on projects, and compete in competitions. Kaggle hosts a vast repository of datasets across various domains, allowing users to explore, analyze, and visualize data for research, learning, and competition purposes. It also offers a wide range of tools and resources, including kernels (Jupyter Notebooks), forums, datasets, and competitions, to support the data science community. Kaggle competitions attract participants from around the world, fostering innovation and collaboration in the field of data science and machine learning.

## 4.5 MODEL ARCHITECTURE

The text classification machine learning model developed using the PubMed 200k RCT dataset represents a significant advancement in medical text classification. Leveraging the rich and diverse collection of abstracts from randomized controlled trials (RCTs), this model is trained to accurately classify medical text into relevant categories or topics, such as treatment methods, clinical outcomes, or disease classifications.

By harnessing the power of machine learning algorithms and natural language processing (NLP) techniques, the model can effectively analyze the text of medical abstracts, extract key features, and learn to classify them based on their semantic content. Through extensive training on the PubMed 200k RCT dataset, the model gains a deep understanding of the complex terminology and nuances specific to medical literature, enabling it to make accurate predictions with high confidence.

The created model has accuracy above 75% and we are working on improving it.
Here is the information about the dataset we are using i.e. "PubMed 20k RCT: a Dataset for Sequential Sentence Classification in Medical Abstracts", where

Size of training set : 180040
Size of validating set: 30212
Size of testing set: 30315

#distribution of labels in training data
METHODS: 59353
RESULTS: 57953
CONCLUSION: 27168
BACKGROUND: 21727
OBJECTIVE: 13839

Here are the following steps used while creating our Machine learning web application:

**1. Install and Import Dependencies:** TensorFlow, Sklean, and Pandas are just a few of the libraries and frameworks that must be installed and imported initially. Flask is utilized for making the machine learning model to display on the web, while TensorFlow is implemented for model construction and training.

**2. Data Collection and Pre-processing:** Data is imported using Pandas and several pre-processing steps have been taken. Some irrelevant columns were removed.
Preprocessing with SpaCy and Lametization:
SpaCy is used to tokenize the text and perform initial preprocessing tasks.

Lemmatize the tokens to reduce them to their base forms and normalize the text.
Remove stop words to eliminate common, noninformative words.
Joined the preprocessed tokens back into a single string, which will serve as the input for further processing.

**3. TF-IDF Vectorization:** Utilized TF-IDF vectorization to convert the text data into numerical features which assigns weights to terms based on their frequency in a document relative to their frequency across all documents, capturing their importance in distinguishing between documents.

**4. Training and Testing the data:** The data is split into training and testing sets, where 10% of the data i.e. 20,000 is used for testing and trained using train_test_split.

**5. Apply Multinomial Naive Bayes classifier:** To efficiently classify text documents into multiple categories based on the frequency of occurrence of words, while making the simplifying assumption of feature independence. Naive Bayes classifiers are efficient and effective for text classification, especially when dealing with high-dimensional data like TF-IDF vectors.

**6. Apply Logistic regression:** To predict the probability of each class label for a given text document, enabling efficient classification into one of the five categories.

**7. Apply SVM classification model:** Used to classify text documents by finding the optimal hyperplane that separates different classes in a high-dimensional space, maximizing the margin between them.

**8. Cross Validation:** Split the TF-IDF matrix into training and validation sets for performance analysis. Performed k-fold cross-validation to train and evaluate the model multiple times on different subsets of the data.

**9. Define a revised neural network model:** We have implemented this model through a neural network consisting of multiple dense layers with ReLU activation, batch normalization layers for stabilization, dropout layers for regularization, and a softmax activation function in the output layer for multi-class classification.

**10. Adam optimization and early stopping:** Used to efficiently minimize the loss function during training by adapting the learning rate for each parameter individually. Performed early stopping to prevent overfitting.

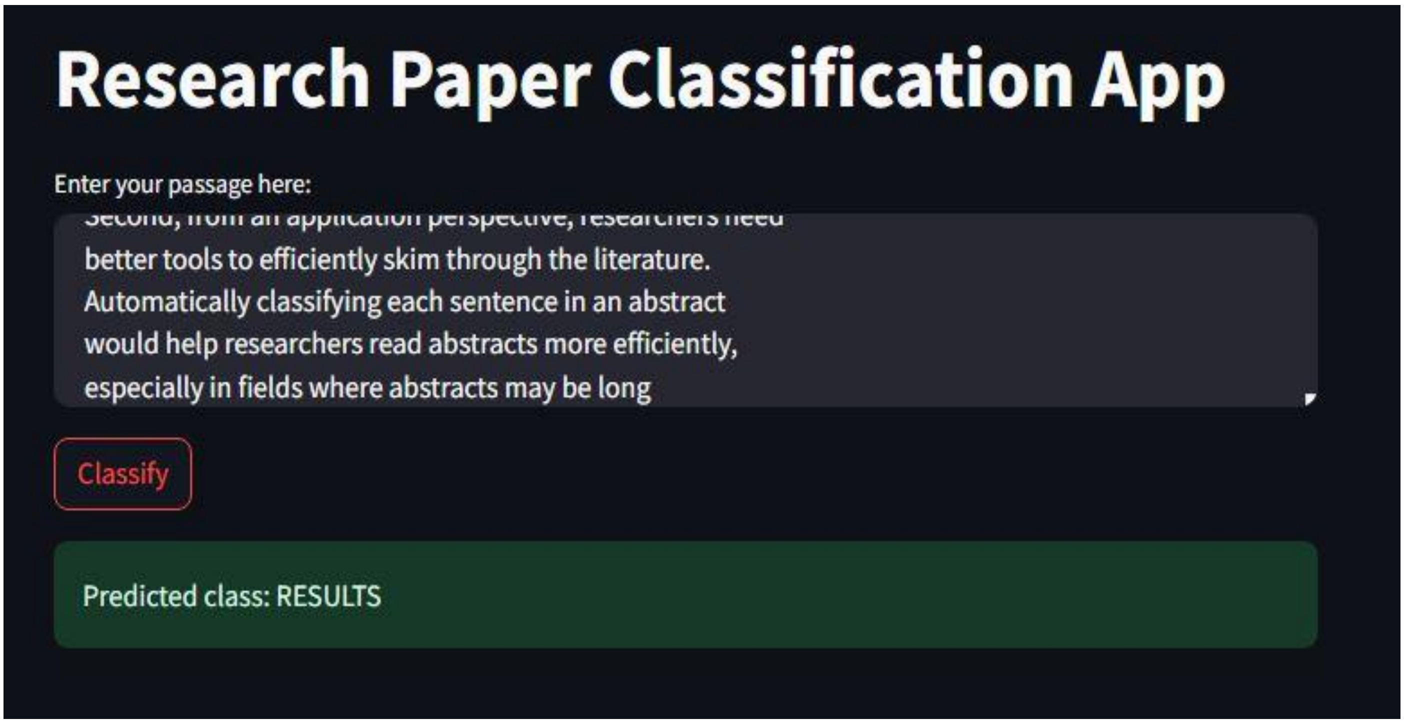
**11. Label Encoding:** Used to encode categorical target labels into numerical values, allowing the model to interpret and process the target variable. It transforms class labels into integers.

**12. Sort the indices:** Sort the indices of the sparse matrices and then of the training and validation sets.

**13. Performance Analysis:** Calculate classification metrics such as accuracy, precision, recall, F1-score, and confusion matrix to analyze the model's performance on the test data and adjust hyperparameters and experiment with different configurations to optimize the model's performance.

**14. Deployment:** The model is deployed on the web using Flask by creating the pickle file of the model. Web application to be hosted on a cloud platform for easy access.

## 5. OUTPUT:

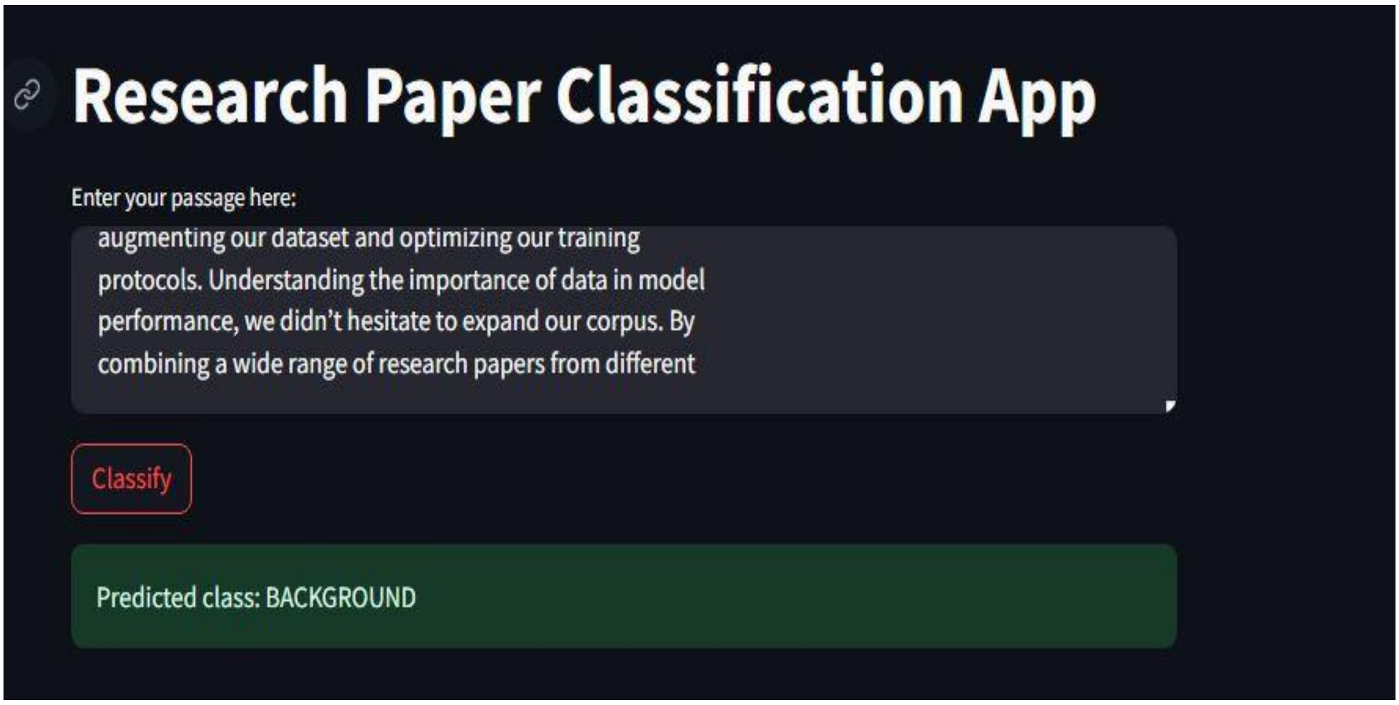

Research Paper Classification App

Enter your passage here:

better tools to efficiently skim through the literature. Automatically classifying each sentence in an abstract would help researchers read abstracts more efficiently, especially in fields where abstracts may be long

Predicted class: RESULTS

Output-1: The main page classifying the text into results section

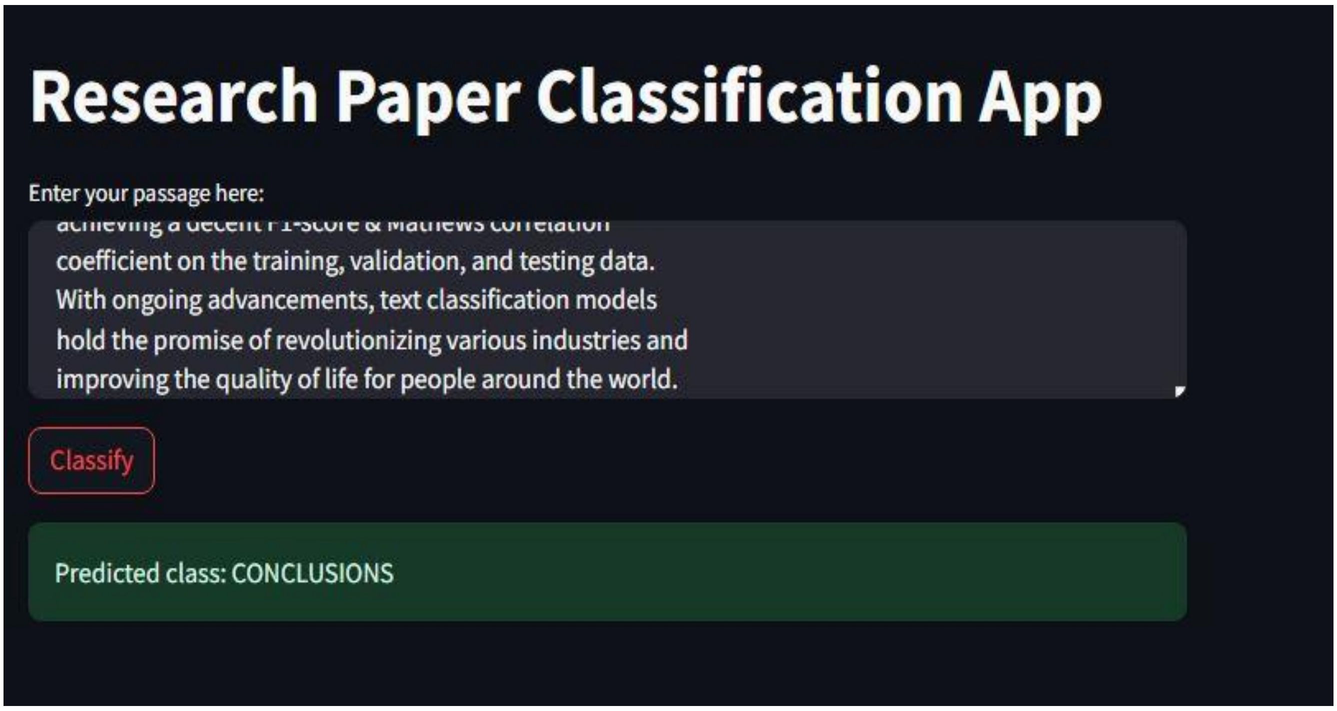

Research Paper Classification App

Enter your passage here:

augmenting our dataset and optimizing our training protocols. Understanding the importance of data in model performance, we didn't hesitate to expand our corpus. By combining a wide range of research papers from different

Predicted class: BACKGROUND

Output-3: The main page classifying the text into background section



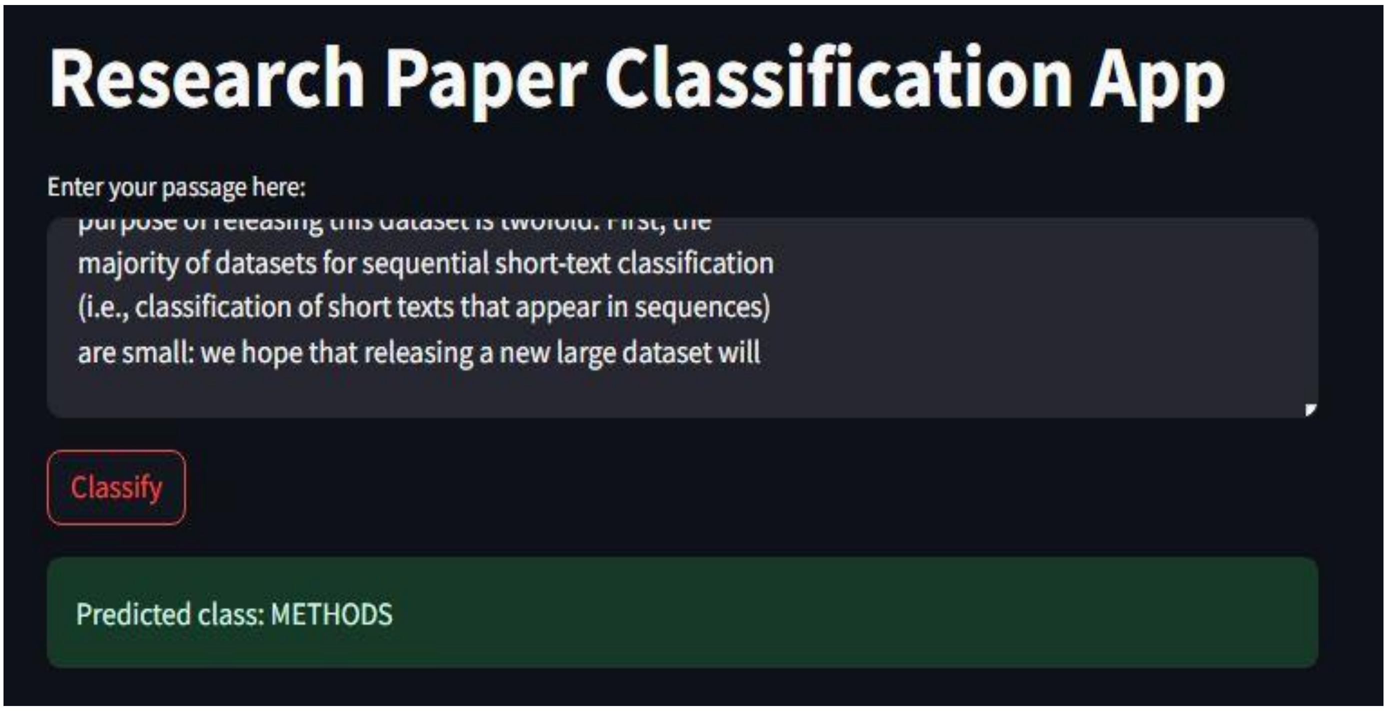
Research Paper Classification App

Enter your passage here:

coefficient on the training, validation, and testing data. With ongoing advancements, text classification models hold the promise of revolutionizing various industries and improving the quality of life for people around the world.

Predicted class: CONCLUSIONS

Output-2: The main page classifying the text into conclusions section



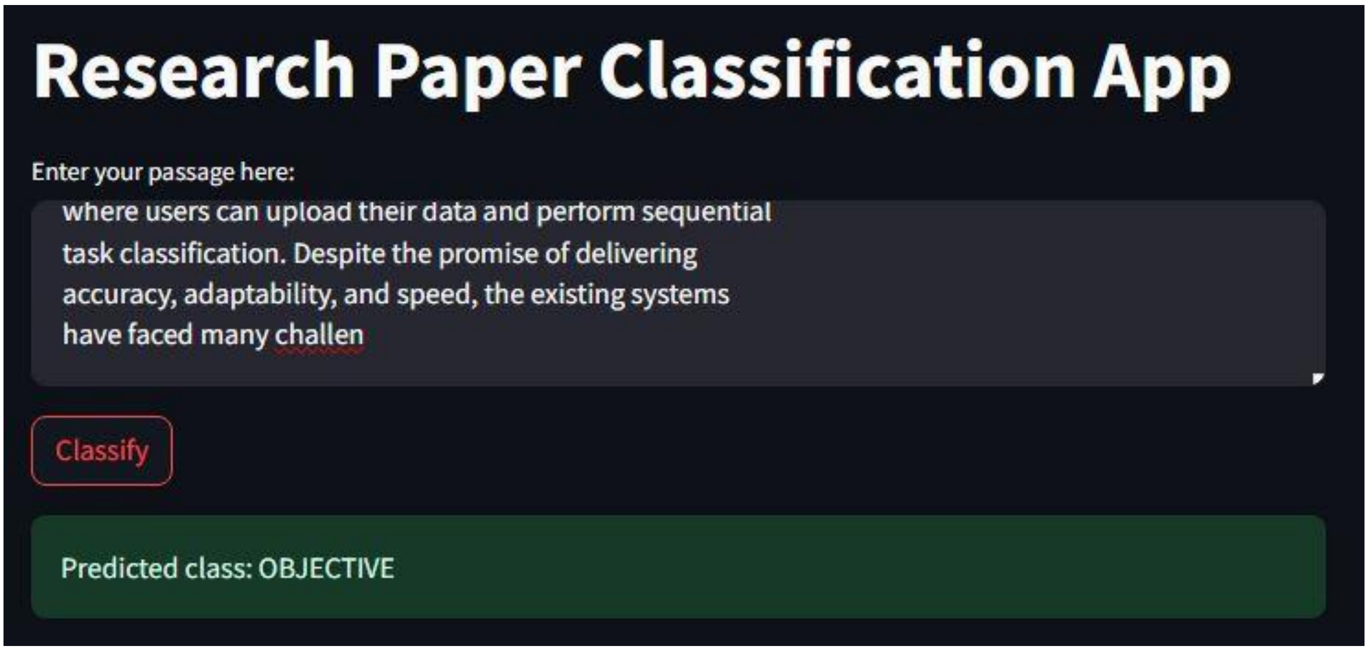
Research Paper Classification App

Enter your passage here:

majority of datasets for sequential short-text classification (i.e., classification of short texts that appear in sequences) are small: we hope that releasing a new large dataset will

Predicted class: METHODS

Output-4: The main page classifying the text into methods section


Research Paper Classification App

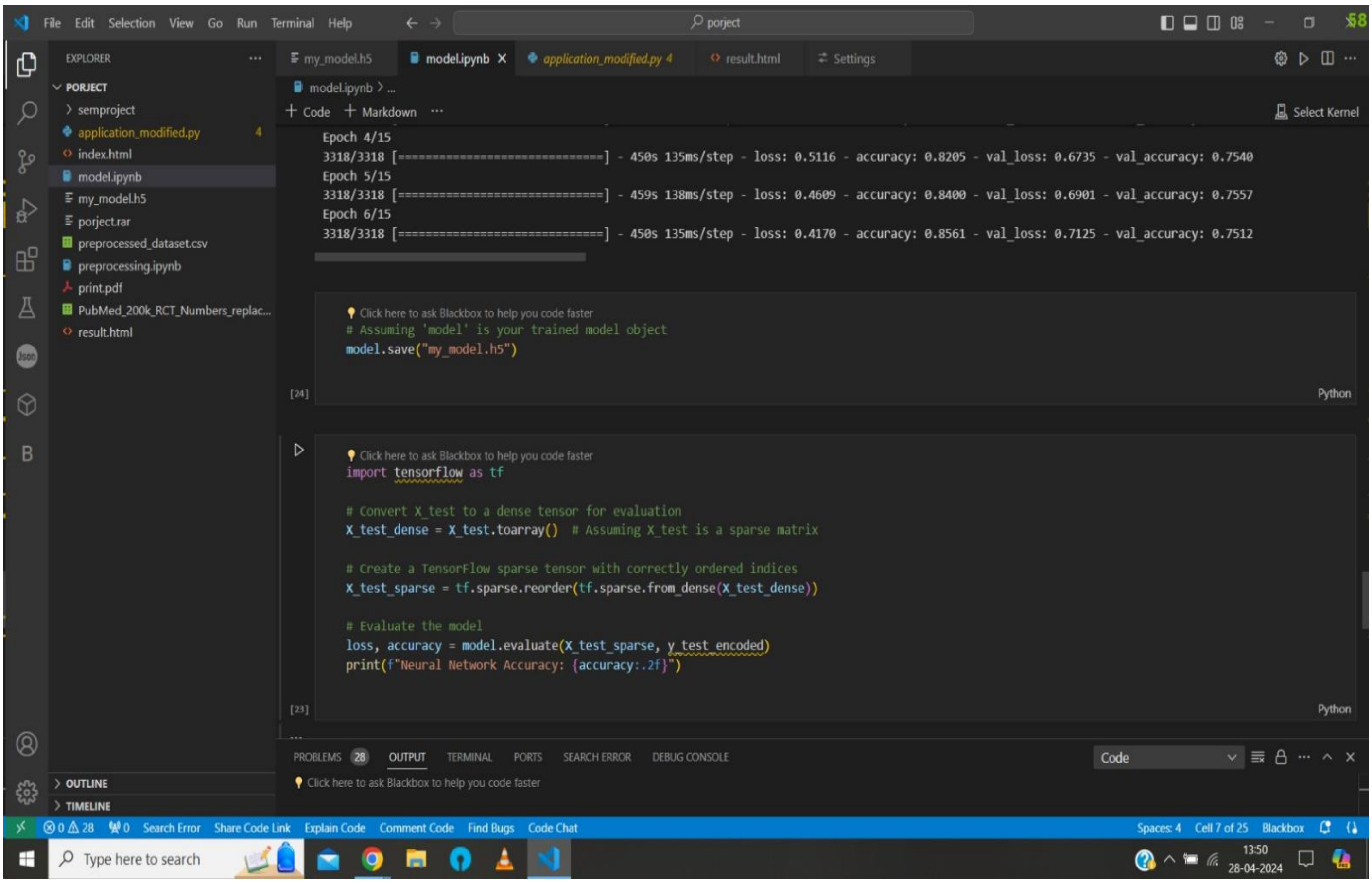
Enter your passage here:

where users can upload their data and perform sequential
task classification. Despite the promise of delivering
accuracy, adaptability, and speed, the existing systems
have faced many challen

Predicted class: OBJECTIVE

Output-5: The main page classifying the text into objective section

## 5.1 Code Snippets

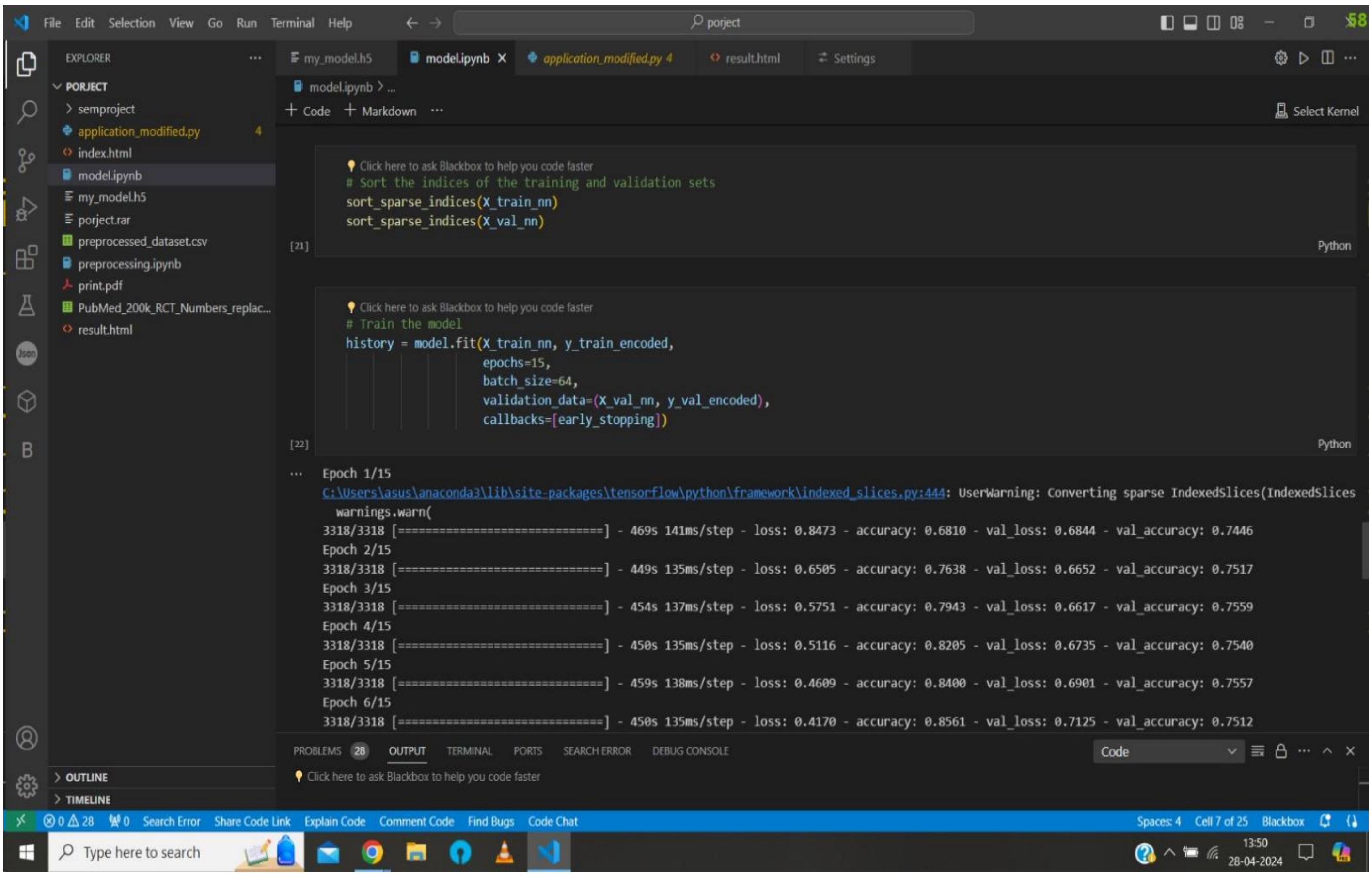


```
Epoch 4/15
3318/3318 [==============================] - 450s 135ms/step - loss: 0.5116 - accuracy: 0.8205 - val_loss: 0.6735 - val_accuracy: 0.7540
Epoch 5/15
3318/3318 [==============================] - 459s 138ms/step - loss: 0.4609 - accuracy: 0.8400 - val_loss: 0.6901 - val_accuracy: 0.7557
Epoch 6/15
3318/3318 [==============================] - 450s 135ms/step - loss: 0.4170 - accuracy: 0.8561 - val_loss: 0.7125 - val_accuracy: 0.7512
```

```python
# Assuming 'model' is your trained model object
model.save("my_model.h5")
```

```python
import tensorflow as tf

# Convert X_test to a dense tensor for evaluation
X_test_dense = X_test.toarray()  # Assuming X_test is a sparse matrix

# Create a TensorFlow sparse tensor with correctly ordered indices
X_test_sparse = tf.sparse.reorder(tf.sparse.from_dense(X_test_dense))

# Evaluate the model
loss, accuracy = model.evaluate(X_test_sparse, y_test_encoded)
print(f"Neural Network Accuracy: {accuracy:.2f}")
```

Snippet-1: The model file to evaluate the accuracy score

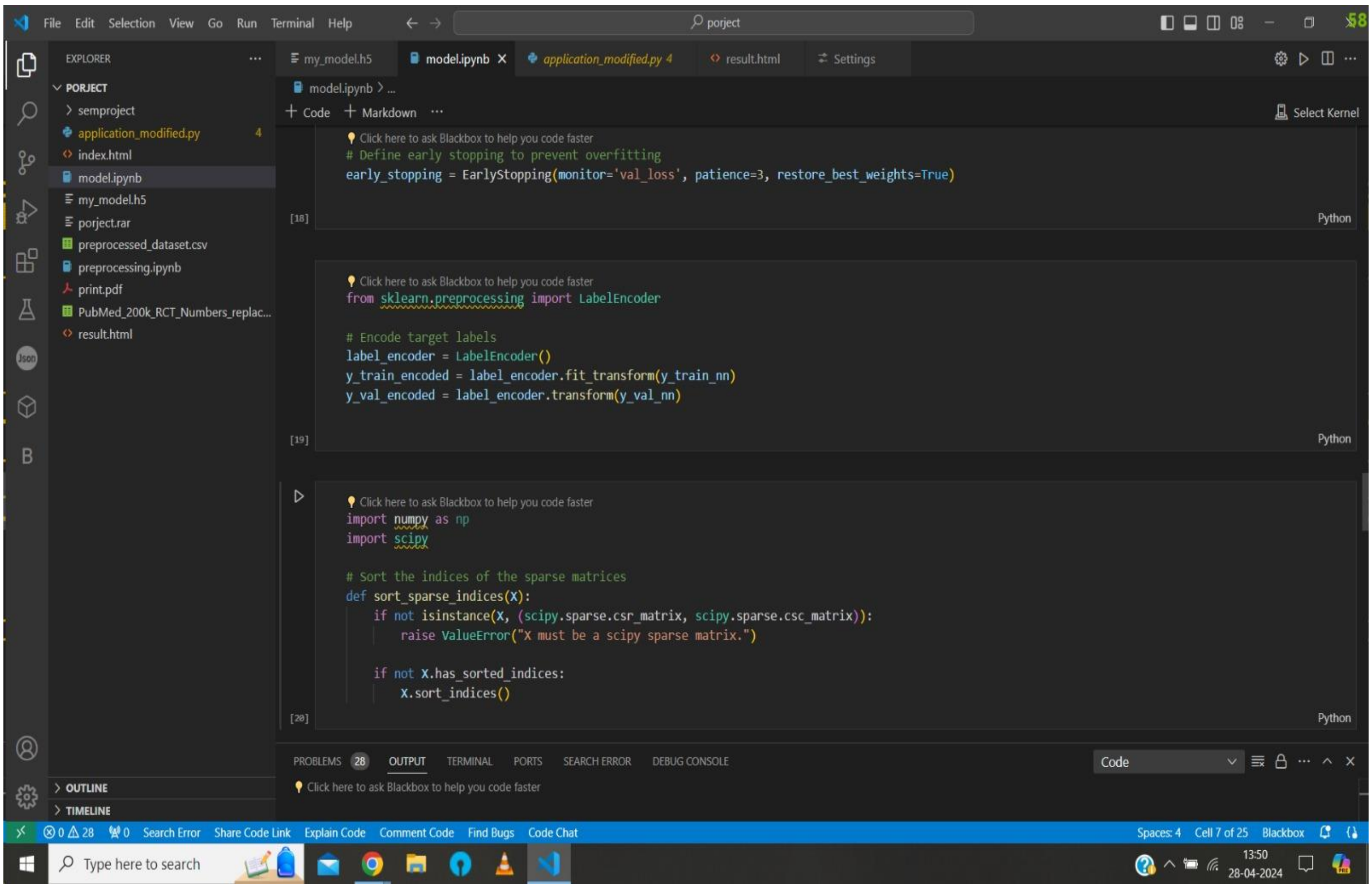


```
# Sort the indices of the training and validation sets
sort_sparse_indices(X_train_nn)
sort_sparse_indices(X_val_nn)
```

```
# Train the model
history = model.fit(X_train_nn, y_train_encoded,
                    epochs=15,
                    batch_size=64,
                    validation_data=(X_val_nn, y_val_encoded),
                    callbacks=[early_stopping])
```

```
Epoch 1/15
C:\Users\asus\anaconda3\lib\site-packages\tensorflow\python\framework\indexed_slices.py:444: UserWarning: Converting sparse IndexedSlices(IndexedSlices
  warnings.warn(
3318/3318 [==============================] - 469s 141ms/step - loss: 0.8473 - accuracy: 0.6810 - val_loss: 0.6844 - val_accuracy: 0.7446
Epoch 2/15
3318/3318 [==============================] - 449s 135ms/step - loss: 0.6505 - accuracy: 0.7638 - val_loss: 0.6652 - val_accuracy: 0.7517
Epoch 3/15
3318/3318 [==============================] - 454s 137ms/step - loss: 0.5751 - accuracy: 0.7943 - val_loss: 0.6617 - val_accuracy: 0.7559
Epoch 4/15
3318/3318 [==============================] - 450s 135ms/step - loss: 0.5116 - accuracy: 0.8205 - val_loss: 0.6735 - val_accuracy: 0.7540
Epoch 5/15
3318/3318 [==============================] - 459s 138ms/step - loss: 0.4609 - accuracy: 0.8400 - val_loss: 0.6901 - val_accuracy: 0.7557
Epoch 6/15
3318/3318 [==============================] - 450s 135ms/step - loss: 0.4170 - accuracy: 0.8561 - val_loss: 0.7125 - val_accuracy: 0.7512
```

Snippet-2: The model file to train and validate the text set

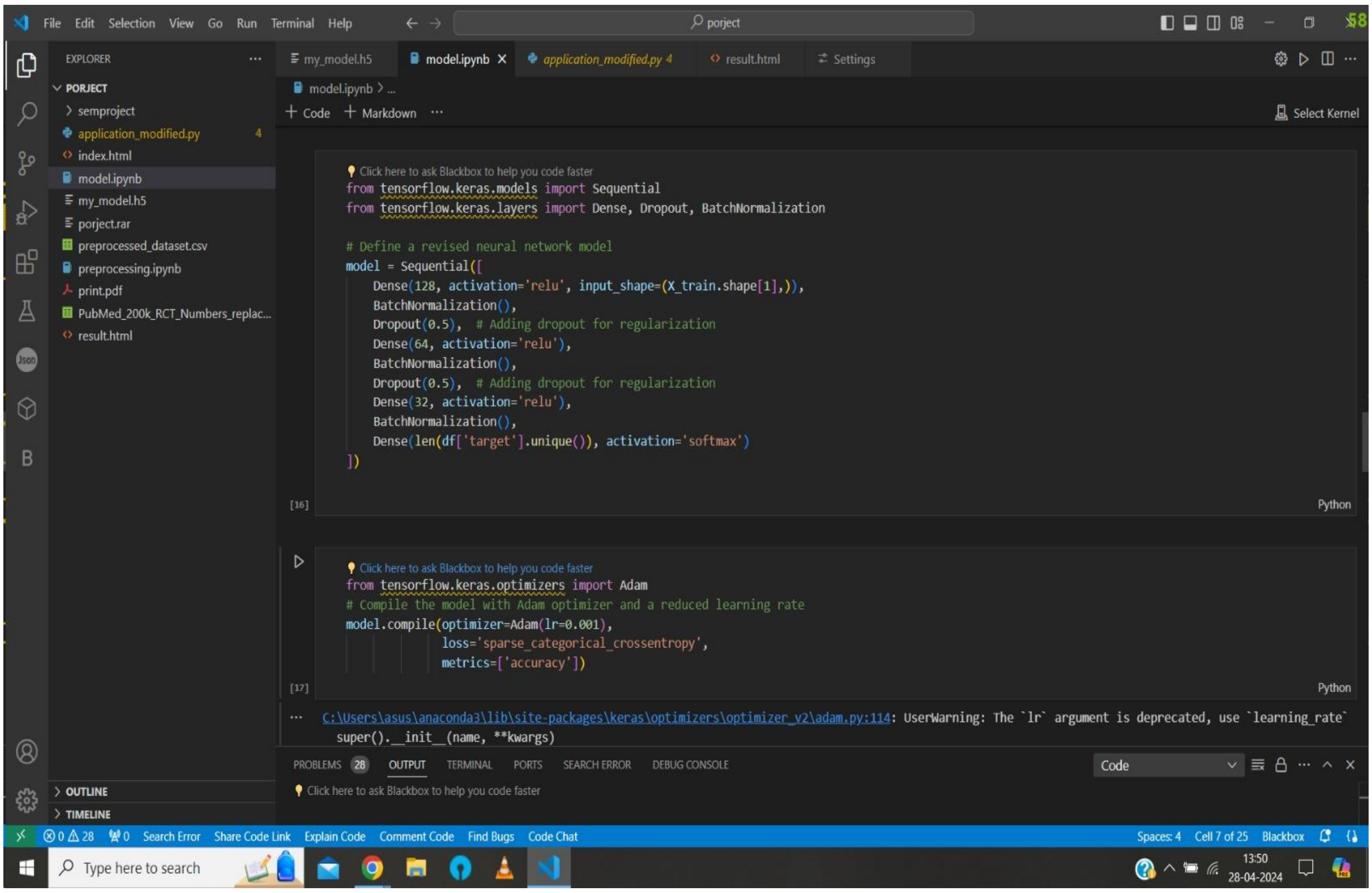


```
# Define early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
```

```
from sklearn.preprocessing import LabelEncoder

# Encode target labels
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train_nn)
y_val_encoded = label_encoder.transform(y_val_nn)
```

```
import numpy as np
import scipy

# Sort the indices of the sparse matrices
def sort_sparse_indices(X):
    if not isinstance(X, (scipy.sparse.csr_matrix, scipy.sparse.csc_matrix)):
        raise ValueError("X must be a scipy sparse matrix.")

    if not X.has_sorted_indices:
        X.sort_indices()
```

Snippet-3: The model file creating the sparse matrices



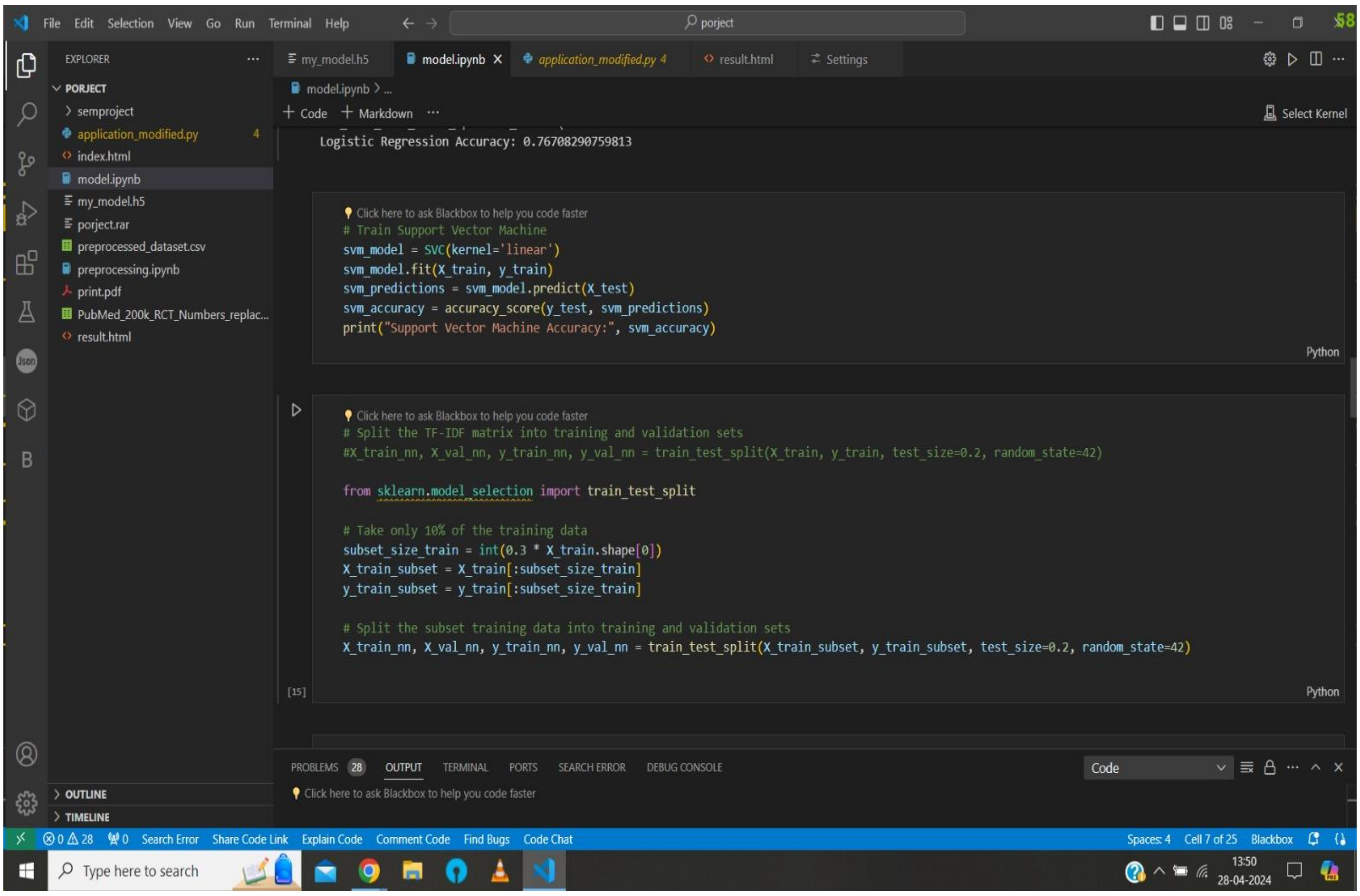
```
[16]
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization

# Define a revised neural network model
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    BatchNormalization(),
    Dropout(0.5),  # Adding dropout for regularization
    Dense(64, activation='relu'),
    BatchNormalization(),
    Dropout(0.5),  # Adding dropout for regularization
    Dense(32, activation='relu'),
    BatchNormalization(),
    Dense(len(df['target'].unique()), activation='softmax')
])
```

```
[17]
from tensorflow.keras.optimizers import Adam
# Compile the model with Adam optimizer and a reduced learning rate
model.compile(optimizer=Adam(lr=0.001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
C:\Users\asus\anaconda3\lib\site-packages\keras\optimizers\optimizer_v2\adam.py:114: UserWarning: The `lr` argument is deprecated, use `learning_rate`
  super().__init__(name, **kwargs)
```

File Edit Selection View Go Run Terminal Help
porject
EXPLORER
PORJECT
semproject
application_modified.py
index.html
model.ipynb
my_model.h5
porject.rar
preprocessed_dataset.csv
preprocessing.ipynb
print.pdf
PubMed_200k_RCT_Numbers_replac...
result.html
OUTLINE
TIMELINE
my_model.h5
model.ipynb
application_modified.py 4
result.html
Settings
+ Code + Markdown
Select Kernel
Click here to ask Blackbox to help you code faster
Python
PROBLEMS 28 OUTPUT TERMINAL PORTS SEARCH ERROR DEBUG CONSOLE
Search Error Share Code Link Explain Code Comment Code Find Bugs Code Chat
Spaces: 4 Cell 7 of 25 Blackbox
Type here to search
13:50
28-04-2024

Snippet-4: The model file to create and train neural network

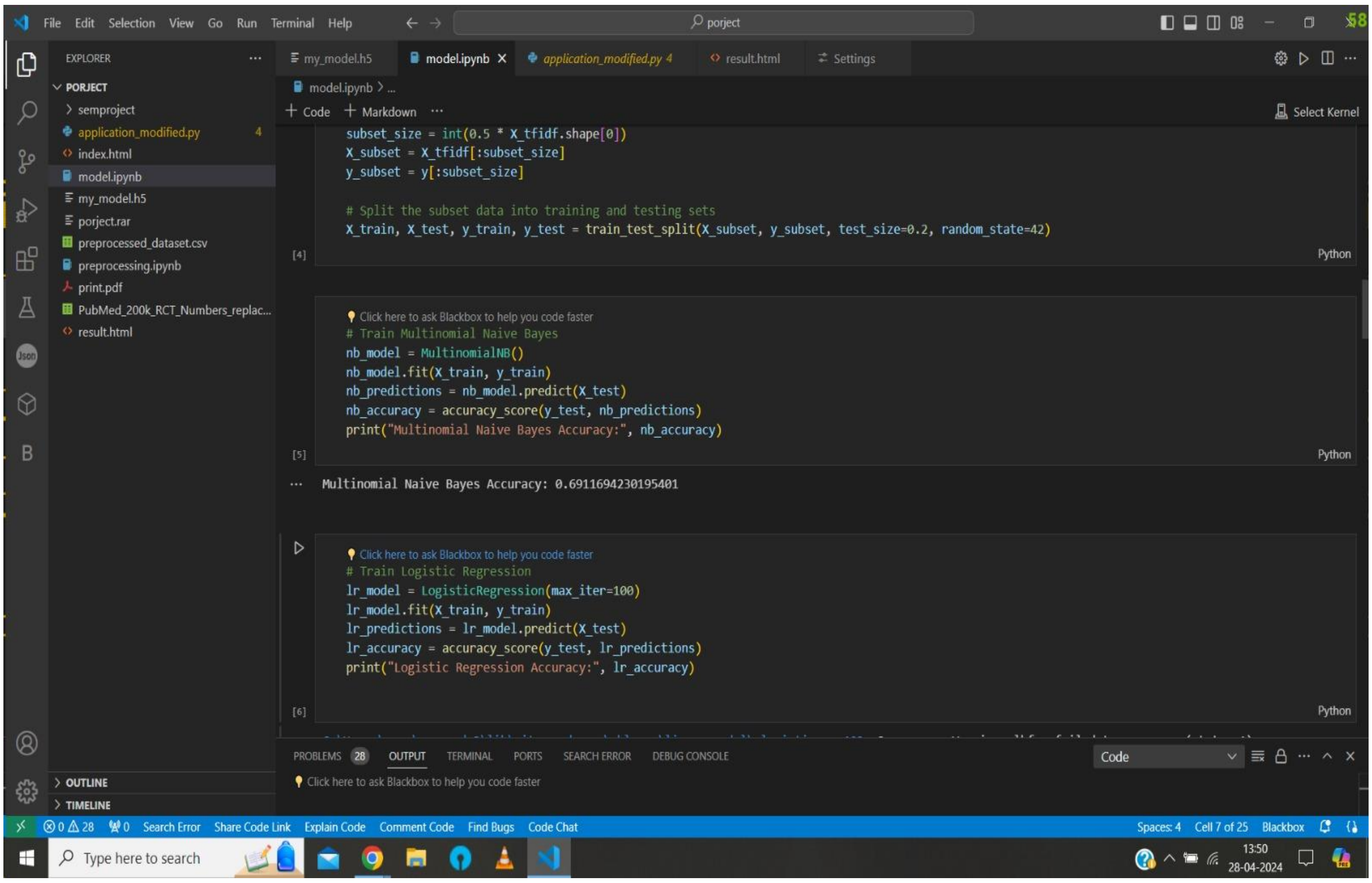


```
Logistic Regression Accuracy: 0.76708290759813
```

```python
# Train Support Vector Machine
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)
svm_predictions = svm_model.predict(X_test)
svm_accuracy = accuracy_score(y_test, svm_predictions)
print("Support Vector Machine Accuracy:", svm_accuracy)
```

```python
# Split the TF-IDF matrix into training and validation sets
#X_train_nn, X_val_nn, y_train_nn, y_val_nn = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

from sklearn.model_selection import train_test_split

# Take only 10% of the training data
subset_size_train = int(0.3 * X_train.shape[0])
X_train_subset = X_train[:subset_size_train]
y_train_subset = y_train[:subset_size_train]

# Split the subset training data into training and validation sets
X_train_nn, X_val_nn, y_train_nn, y_val_nn = train_test_split(X_train_subset, y_train_subset, test_size=0.2, random_state=42)
```

Snippet-5: The model file to train the data



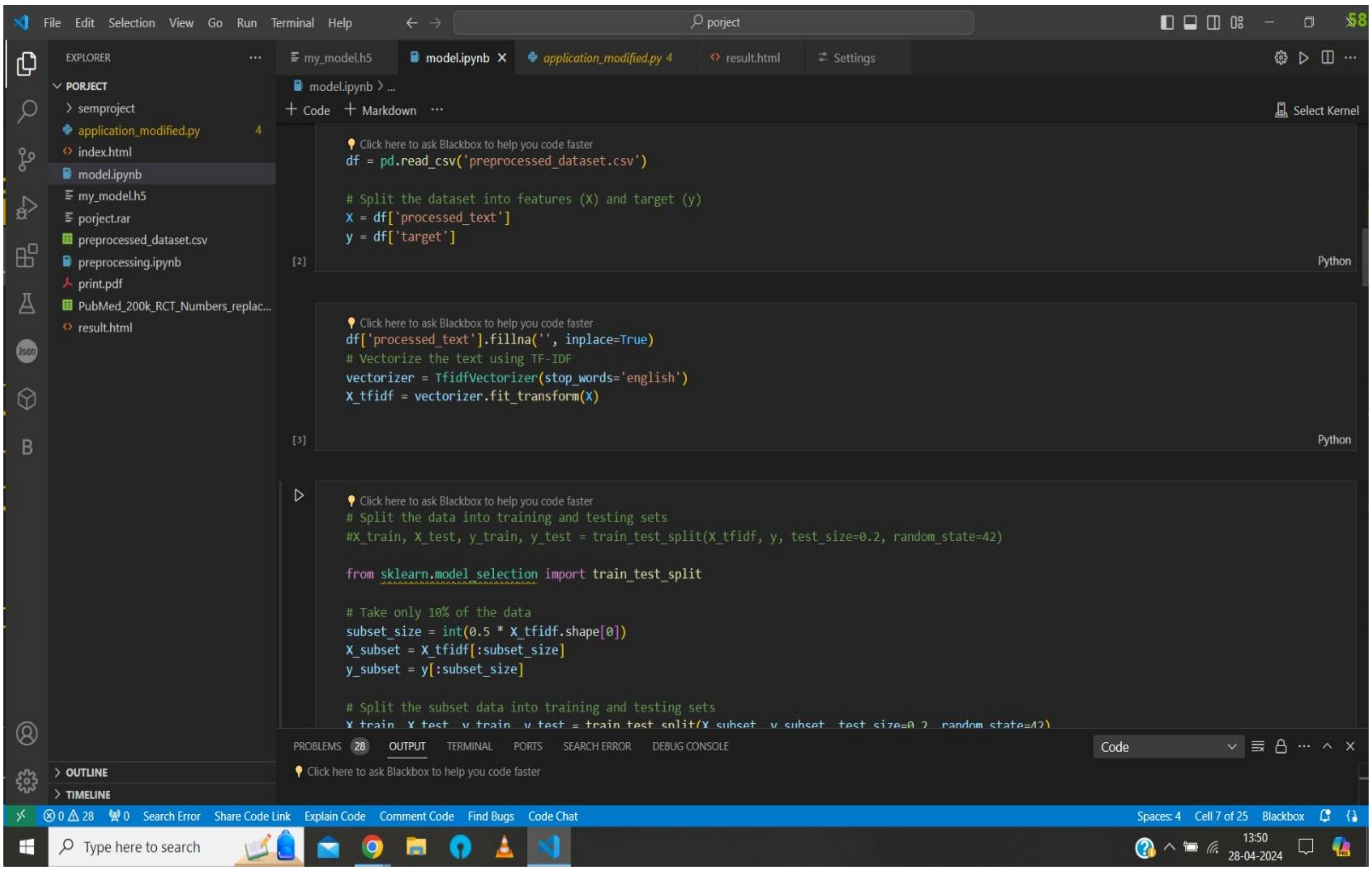
```
subset_size = int(0.5 * X_tfidf.shape[0])
X_subset = X_tfidf[:subset_size]
y_subset = y[:subset_size]

# Split the subset data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_subset, y_subset, test_size=0.2, random_state=42)
```

```
# Train Multinomial Naive Bayes
nb_model = MultinomialNB()
nb_model.fit(X_train, y_train)
nb_predictions = nb_model.predict(X_test)
nb_accuracy = accuracy_score(y_test, nb_predictions)
print("Multinomial Naive Bayes Accuracy:", nb_accuracy)
```

```
Multinomial Naive Bayes Accuracy: 0.6911694230195401
```

```
# Train Logistic Regression
lr_model = LogisticRegression(max_iter=100)
lr_model.fit(X_train, y_train)
lr_predictions = lr_model.predict(X_test)
lr_accuracy = accuracy_score(y_test, lr_predictions)
print("Logistic Regression Accuracy:", lr_accuracy)
```

Snippet-6: Applying MultinomialNB and Logistic regression to model file



```
df = pd.read_csv('preprocessed_dataset.csv')

# Split the dataset into features (X) and target (y)
X = df['processed_text']
y = df['target']
```

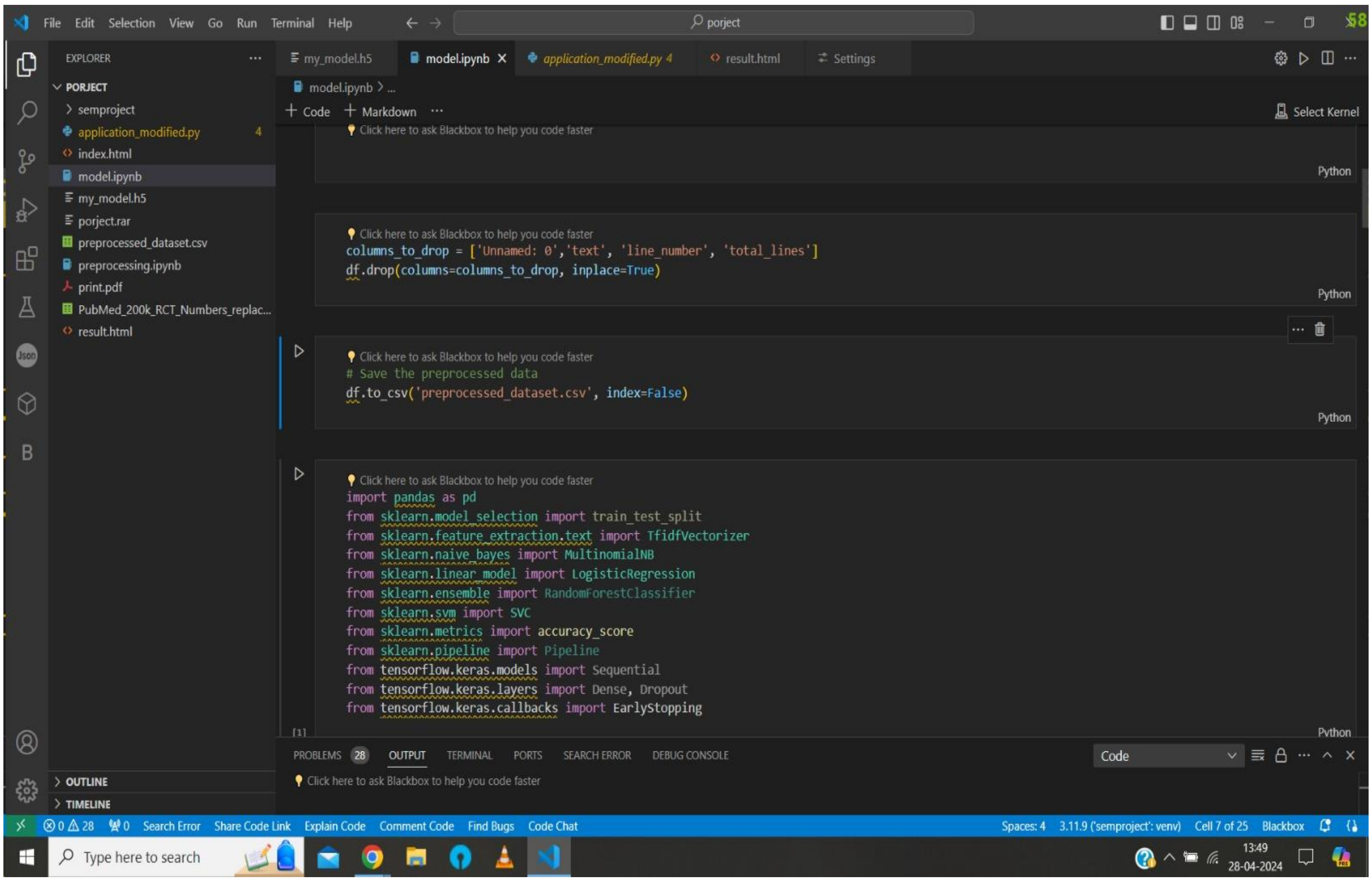
[2]

```
df['processed_text'].fillna('', inplace=True)
# Vectorize the text using TF-IDF
vectorizer = TfidfVectorizer(stop_words='english')
X_tfidf = vectorizer.fit_transform(X)
```

[3]

```
# Split the data into training and testing sets
#X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.2, random_state=42)

from sklearn.model_selection import train_test_split

# Take only 10% of the data
subset_size = int(0.5 * X_tfidf.shape[0])
X_subset = X_tfidf[:subset_size]
y_subset = y[:subset_size]

# Split the subset data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_subset, y_subset, test_size=0.2, random_state=42)
```

Snippet-7: Applying fit-transform to data

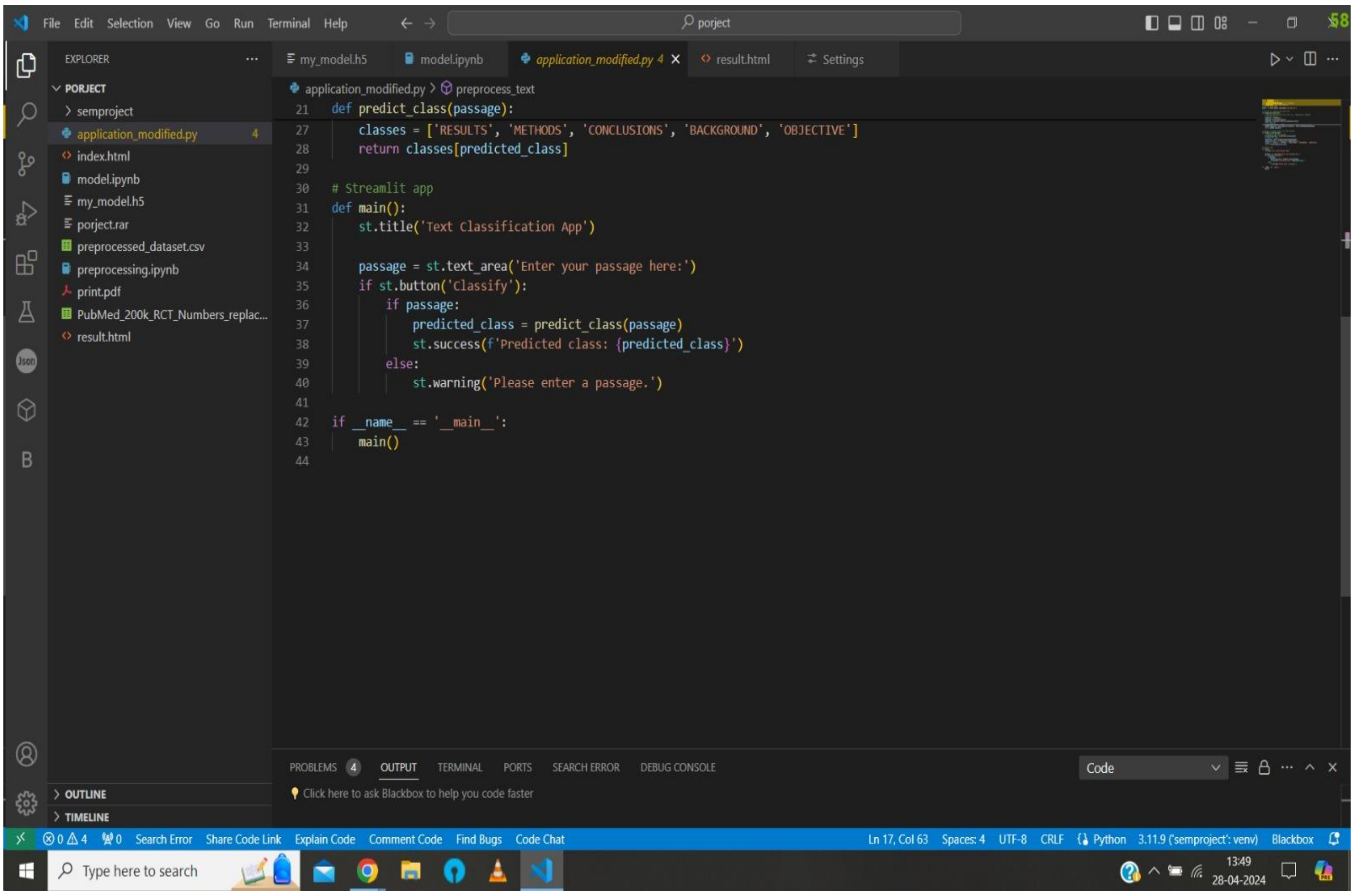


```
columns_to_drop = ['Unnamed: 0','text', 'line_number', 'total_lines']
df.drop(columns=columns_to_drop, inplace=True)
```

```
# Save the preprocessed data
df.to_csv('preprocessed_dataset.csv', index=False)
```

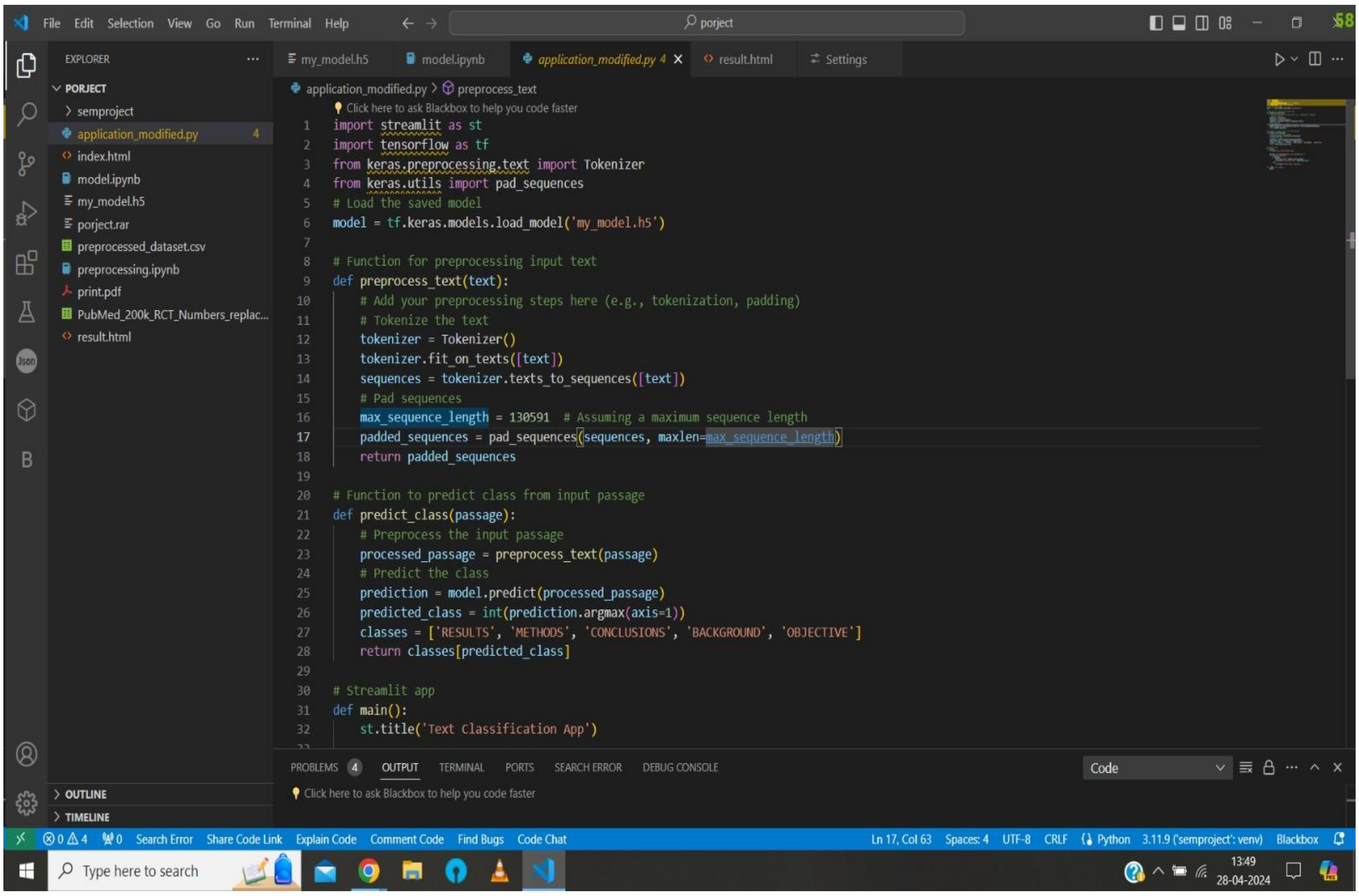
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.pipeline import Pipeline
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
```

Snippet-8: Showing the necessary imports



```python
def predict_class(passage):
    classes = ['RESULTS', 'METHODS', 'CONCLUSIONS', 'BACKGROUND', 'OBJECTIVE']
    return classes[predicted_class]

# Streamlit app
def main():
    st.title('Text Classification App')

    passage = st.text_area('Enter your passage here:')
    if st.button('Classify'):
        if passage:
            predicted_class = predict_class(passage)
            st.success(f'Predicted class: {predicted_class}')
        else:
            st.warning('Please enter a passage.')

if __name__ == '__main__':
    main()

```

Snippet-9: The flask file for interface



```python
import streamlit as st
import tensorflow as tf
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
# Load the saved model
model = tf.keras.models.load_model('my_model.h5')

# Function for preprocessing input text
def preprocess_text(text):
    # Add your preprocessing steps here (e.g., tokenization, padding)
    # Tokenize the text
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts([text])
    sequences = tokenizer.texts_to_sequences([text])
    # Pad sequences
    max_sequence_length = 130591  # Assuming a maximum sequence length
    padded_sequences = pad_sequences(sequences, maxlen=max_sequence_length)
    return padded_sequences

# Function to predict class from input passage
def predict_class(passage):
    # Preprocess the input passage
    processed_passage = preprocess_text(passage)
    # Predict the class
    prediction = model.predict(processed_passage)
    predicted_class = int(prediction.argmax(axis=1))
    classes = ['RESULTS', 'METHODS', 'CONCLUSIONS', 'BACKGROUND', 'OBJECTIVE']
    return classes[predicted_class]

# Streamlit app
def main():
    st.title('Text Classification App')
```

Snippet-10: Showing the predicting and processed text

# 6. EXPERIMENTAL ANALYSIS AND RESULTS

## 6.1 System Configuration:

### SOFTWARE SPECIFICATION

• Python Compiler with required Libraries and Modules

• Language: Python

• Operating System: Windows 8/10/11

• Modules like:

a) Pandas

b) Sklearn

c) Tensorflow

d) NLTK

### EXTERNAL PACKAGE REQUIREMENTS

•GTTS: Google Text to Speech

• Flask

• Streamlit

• Playsound

## Hardware requirements:

1. Processor: A processor with multiple cores and high clock speed is recommended for real-time video processing. A quad-core or higher processor with a clock speed of atleast 2.5 GHz is suitable for most implementations.

2. Graphics Processing Unit (GPU): A dedicated GPU is recommended for accelerating the computations required for object detection and tracking. A compatible NVIDIA GPU with CUDA architecture can significantly improve the performance of the project.

3. RAM: Sufficient RAM is required to store the video footage and the intermediate results of the processing. At least 8 GB of RAM is recommended for most implementations, and more may be required for processing high-resolution video footage.

4. Storage: The project requires storage for the video footage and the output results. A Solid-State Drive (SSD) or a high-capacity Hard Disk Drive (HDD) is recommended for storing the data.

5. Microphone: unidirectional pattern, capturing sound primarily from one direction, which can be useful in reducing background noise. For voice input, a microphone with a frequency response tailored to the human voice (approximately 80 Hz to 15 kHz) is common. 16-bit depth and 44.1 kHz sample rate.

# 7.CONCLUSION AND FUTURE WORK

## 7.1 RESULT

Our journey to improve our classification model is one of hard work and innovation. On this journey, we realized the importance of using a wide range of classification methods to get the most out of our model. With careful experimentation, we selected an ensemble of algorithms such as multinomial naive, logistic regression and cutting edge deep learning neural networks, each with its own strengths. Together, these methods yielded an accuracy score of 74% But our journey to excellence didn’t end there. It served as a stepping stone for continuous improvement. Driven by a spirit of constant iteration, we set out to continually improve our model. With each iteration, we looked at the model’s performance, tuning parameters and refining architectures to get the best out of the model.

At the heart of our model enhancement strategy is augmenting our dataset and optimizing our training protocols. Understanding the importance of data in model performance, we didn’t hesitate to expand our corpus. By combining a wide range of research papers from different disciplines, we increased our dataset and enabled our model to learn from a wider range of textual nuances. We also increased the training time by increasing the number of iterations of our neural network. Not only did this increase the model's resilience to overfitting, but it also gave it a better understanding of the structure of the research paper text. At the end of the day, our machine-learning model is a testament to the power of innovation and persistence. Inside an easy-to-use interface, we have a powerful tool that can analyze research papers with remarkable accuracy. By classifying texts into five different domains—Methods, results, conclusion, background, and objective—our model makes it easy for users to navigate the maze of academic literature. As the future beckons, we continue to push the limits of text classification. Not only is our model a sign of technological progress, but it is also a sign of hope for a future where the pursuit of knowledge knows no bounds.

Our journey towards enhancing our classification model has been a testament to our dedication to innovation and continuous improvement. Through meticulous experimentation and careful selection of classification methods, including multinomial naive Bayes, logistic regression, and cutting-edge deep learning neural networks, we achieved a commendable accuracy score of 74%. However, our pursuit of excellence did not end there; it merely marked the beginning of our commitment to continuous refinement.

Driven by a spirit of constant iteration, we embarked on a journey to further enhance our model's performance. With each iteration, we meticulously evaluated the model's performance, fine-tuned parameters, and refined architectures to extract the best possible outcomes. Central to our strategy for model enhancement was the augmentation of our dataset and optimization of our training protocols.

Recognizing the pivotal role of data in model performance, we expanded our dataset by incorporating a diverse range of research papers from various disciplines. This expansion enabled our model to learn from a broader spectrum of textual nuances, thereby enhancing its ability to accurately classify research papers.

Additionally, we optimized our training protocols by increasing the number of iterations of our neural network, thereby extending the training time. This approach not only bolstered the model's resilience to overfitting but also imparted a deeper understanding of the structural complexities inherent in research paper text.

Ultimately, our machine-learning model stands as a powerful testament to the potential of innovation and persistence. Housed within an intuitive interface, our model serves as a robust tool capable of analyzing research papers with remarkable accuracy. By categorizing texts into five distinct domains—Methods, Results, Conclusion, Background, and Objective—our model simplifies the navigation of academic literature for users.

As we look towards the future, we remain steadfast in our commitment to pushing the boundaries of text classification. Our model not only represents technological progress but also embodies a beacon of hope for a future where the pursuit of knowledge transcends conventional limits.

## 7.2 <u>INTERPRETATION OF RESULT</u>

The interpretation of the result section in your report highlights the iterative and innovative journey undertaken to enhance the classification model's performance. It emphasizes the strategic selection and combination of various classification methods, including multinomial naive Bayes, logistic regression, and advanced deep learning neural networks, to achieve a commendable accuracy score of 74%. This achievement serves as a testament to the dedication to continuous improvement and innovation throughout the development process.

Furthermore, the interpretation underscores the ongoing commitment to refinement and optimization beyond the initial success. Through constant iteration, the model's performance was meticulously evaluated, and parameters were fine-tuned to extract the best possible outcomes. Central to this optimization strategy was the augmentation of the dataset and the optimization of training protocols, acknowledging the pivotal role of data in model performance.

The interpretation also highlights the tangible benefits of these enhancements, such as improved resilience to overfitting and a deeper understanding of the structural complexities inherent in research paper text. Ultimately, the result section underscores the transformative potential of the developed machine-learning model, which serves as a powerful tool for analyzing research papers with remarkable accuracy and simplifying navigation through academic literature.

## 7.3 CONCLUSION

We implemented a Natural Language Processing (NLP) model capable of segmenting text lines in abstracts from medical research papers. The model was capable of learning the task of the said segmentation with minimal dependencies as few features were used to train the model. The model was also capable of generalizing to unseen data according to the performance metrics used, notably achieving a decent F1-score & Mathews correlation coefficient on the training, validation, and testing data. With ongoing advancements, text classification models hold the promise of revolutionizing various industries and improving the quality of life for people around the world.

Our implementation of a Natural Language Processing (NLP) model marks a significant milestone in the field of text segmentation within abstracts from medical research papers. Our model demonstrates robust capabilities in effectively segmenting text lines with minimal dependencies, showcasing its efficiency in handling the task with limited resources.

Notably, our model exhibits the ability to generalize well to unseen data, a critical aspect of its performance. By leveraging a minimal set of features during training, our model achieves impressive results across various performance metrics, including a notable F1-score and Mathews correlation coefficient on both training and validation datasets. This highlights the model's robustness and adaptability in accurately segmenting text lines within abstracts.

Moreover, the ongoing advancements in text classification models hold immense promise for revolutionizing various industries and contributing to the improvement of the quality of life worldwide. With the continuous evolution and refinement of NLP techniques, the potential applications of such models are boundless, ranging from healthcare and biomedical research to finance, education, and beyond.

In summary, our implementation represents a significant step forward in the realm of text segmentation within medical research papers, showcasing the potential of NLP models to address complex tasks with minimal dependencies and achieve remarkable performance across diverse datasets. As we continue to push the boundaries of NLP technology, we anticipate further advancements that will drive innovation and positive impact across multiple domains.

## 7.4 FUTURE SCOPE:

The future scope of our implemented Natural Language Processing (NLP) model for segmenting text lines in abstracts from medical research papers is vast and promising.

**1. Enhanced Precision and Recall**: Further refinement of the model's architecture and training process can lead to improvements in precision and recall metrics, ensuring more accurate segmentation of text lines within abstracts.

**2. Integration with Knowledge Graphs**: Integrating the NLP model with knowledge graphs and ontologies specific to the medical domain can enrich the contextual understanding of the text, enabling more sophisticated segmentation techniques and deeper insights into research papers.

**3. Multi-Modal Analysis:** Expanding the model's capabilities to analyze multi-modal data, such as incorporating images, tables, and figures from research papers, can provide a more comprehensive understanding of the content and facilitate cross-modal information extraction.

**4. Clinical Decision Support Systems:** Integration of the NLP model into clinical decision support systems can aid healthcare professionals in extracting relevant information from medical literature quickly and accurately, assisting in clinical decision-making processes.

**5. Semantic Search and Information Retrieval:** Leveraging the model for semantic search and information retrieval systems can enhance the efficiency of literature review processes for researchers, clinicians, and other stakeholders in the healthcare industry.

**6. Real-Time Monitoring and Alerting:** Implementing the model within real-time monitoring and alerting systems can enable timely identification of emerging trends, breakthroughs, and potential risks in medical research, facilitating proactive decision-making and interventions.

**7. Adaptation to Other Domains**: Extending the model's capabilities to other domains beyond medical research, such as academic publishing, legal documents, and technical reports, can broaden its applicability and impact across diverse industries.

**8 Ethical Considerations and Bias Mitigation:** Addressing ethical considerations, including data privacy, fairness, and bias mitigation, is crucial for ensuring the responsible development and deployment of the NLP model in real-world scenarios.

## REFERENCES

[1] S. Subhash et al., “Voice Control Using AI-Based Voice Assistant,” 2020 International Conference on Smart Electronics and Communication (ICOSEC), Bangalore, India, 2020, pp. 592-596, 10.1109/ICOSEC49019.2020.9230327.

[2] doi: B. A. Alsaify, H. S. Abu Arja, B. Y. Maayah, M. M. Al Taweel, R. Alazrai and M. I. Daoud, "Voice-Based Human Identification using Machine Learning," 2022 13th International Conference on Information and Communication Systems (ICICS), Irbid, Jordan, 2022, pp. 205-208, doi: 10.1109/ICICS55353.2022.9811154.

[3] A. B. V and R. M. S, "Voice Based Person Identification Using c-Mean and c- Medoid Clustering Techniques," 2020 IEEE International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER), Udupi, India, 2020,pp. 121-126, doi: 10.1109/DISCOVER50404.2020.9278042.

[4] Hayes H. Monson,Statistical Digital Signal Processing and Modeling, John Wiley &Sons Inc. , Toronto, 1996, ISBN 0-471- 59431-8 1

[5] Proakis John G., Manolakis Dimitris G.,Digital Signal Processing, principles, algorithms, and applications, Third Edition, Prentice Hall , New Jersey, 1996, ISBN 0-13- 394338-9 14.

[6] Hiroaki Sakoe and Seibi Chiba, Dynamic Programming algorithm Optimization forspoken word Recognition, IEEE transaction on Acoustic speech and Processing, February 1978.

[7] .S. Srivastava and S. Prakash, "An Analysis of Various IoT Security Techniques: A Review," 2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), 2020, pp. 355- 362, doi: 10.1109/ICRITO48877.2020.9198027

[8] . Saijshree Srivastava, Surya Vikram Singh, Rudrendra Bahadur Singh, Himanshu Kumar Shukla,” Digital Transformation of Healthcare: A blockchain study” International Journal of Innovative Science, Engineering & Technology, Vol. 8 Issue 5,May 2021.

[9] V.Radha and C. Vimala, “A review on speech recognition challenges andapproaches,” doaj. org, vol. 2, no. 1, pp. 1–7, 2012.

[10] Hayes H. Monson,Statistical Digital Signal Processing and Modeling, John Wiley &Sons Inc. , Toronto, 1996, ISBN 0-471- 59431-8

[11] Automatic speech recognition: the development of the SPHINX system; by Kai-FuLee; Boston; London: Kluwer Academic, c1989

[12] Enikeev, D.G., Mustafina, S.A.: Sign language recognition through leap motion controller and input prediction algorithm. J. Phys. Conf. Ser. 1715, 012008 (2021). https://doi.org/10.1088/1742-6596/1715/1/012008

[13] Cui, R., Liu, H., Zhang, C.: A deep neural framework for continuous sign language recognition by iterative training. IEEE Trans. Multimed. 21, 1880–1891 (2019). https://doi.org/10.1109/TMM.2018.2889563

[14] Bantupalli, K., Xie, Y.: American sign language recognition using deep learning and 13 computer vision. In: Proceedings of - 2018 IEEE International Conference on Big Data, Big Data 2018, pp. 4896–4899 (2019). https://doi.org/10.1109/BIGDATA.2018.8622141

[15] Sharma, A., Sharma, N., Saxena, Y., Singh, A., Sadhya, D.: Benchmarking deep neural network approaches for Indian sign language recognition. Neural Comput. Appl. 33(12), 6685–6696 (2020). https://doi.org/10.1007/s00521-020-05448-8

[16] Pu, J., Zhou, W., Li, H.: Iterative alignment network for continuous sign language recognition. In: IEEE Computer Social Conference Computing Vision Pattern Recognition, pp. 4160–4169 (2019). https://doi.org/10.1109/CVPR.2019.00429 [16] Liang, Z., Liao, S., Hu, B.: 3D Convolutional neural networks for dynamic sign language recognition. Comput. J. 61, 1724–1736 (2018). https://doi.org/10.1093/COMJNL/BXY049