iNeuron

LOW-LEVEL DOCUMENT

News Article Sorting

Revision number – 1.2

Last date of revision - 11/9/2023

Authored by: Krishna Chandra Yadav

1. Document Version Control

Date	Version	Description	Author
02/9/2023	1.0	Abstract	Krishna Chandra Yadav
08/9/2023	1.1	Design Flow	Krishna Chandra Yadav
11/9/2023	1.2	Performance Evaluation Conclusion	Krishna Chandra Yadav

CONTENTS

ABSTRACT

This project focuses on improving the classification of news articles by leveraging the power of TF-IDF (Term Frequency-Inverse Document Frequency) for text vectorization and employing a range of machine learning algorithms for performance evaluation. In an era of information overload, efficient news article classification plays a pivotal role in helping user’s access relevant content quickly and aiding advertisers in targeting their audience effectively.

The project employs TF-IDF, a widely used technique in natural language processing, to convert the textual content of news articles into numerical vectors. These vectors capture the importance of terms within articles, enabling the algorithms to understand the content better.

To assess the performance of the classification task, four powerful machine learning algorithms—Support Vector Machine (SVM), Random Forest (RF), Gradient Boosting Classifier, and AdaBoost are employed. Each algorithm brings its unique strengths, from SVM's ability to handle high-dimensional data to Random Forest's ensemble-based robustness.

The project evaluates these algorithms based on accuracy to determine their effectiveness in classifying news articles accurately. Through extensive experimentation and analysis, we aim to identify the algorithm that offers the best trade-off between precision and computational efficiency for news article classification.

Keywords: Natural Language Processing, machine learning, TF-IDF, content recommendation, news article classification, Support Vector Machine, Random Forest, Gradient Boosting Classifier, Ada-Boost, classification system.

2. Introduction

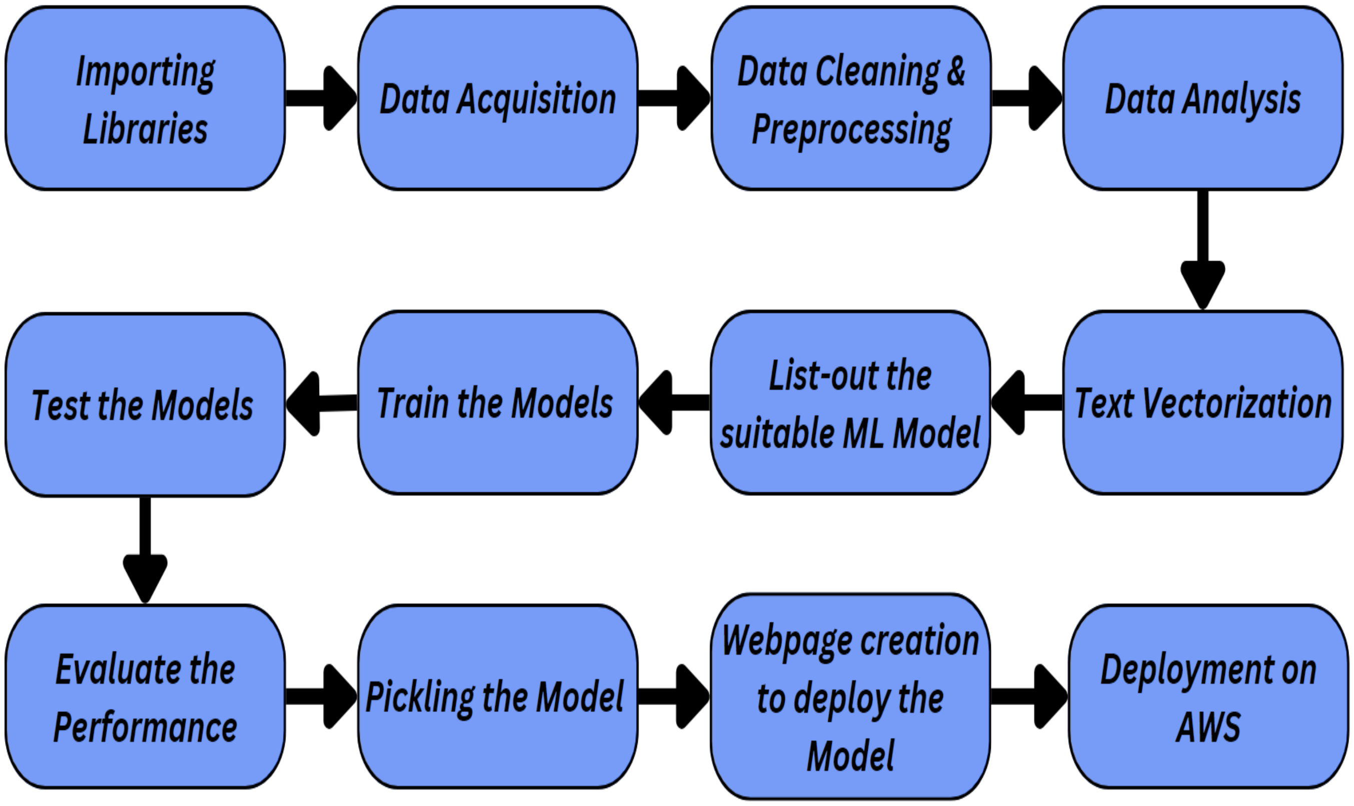
2.1 What is LLD Document?

The main goal of the LLD document is to give the internal logic design of actual code implementation and supply the outline of the machine learning model and its implementation. Additionally, it provides a description of how our project will design end-to-end.

2.2 Scope

Low-level design (LLD) is a component-level design process that follows a step-by-step refinement process. This process can be used for designing data structures, required software architecture, source code, and ultimately, performance algorithms. Overall, the data organization may be defined during requirement analysis and then refined during data design work.

3. Architecture



4. Architecture Design

This project is designed to make an interface for the User to predict customer behavior.

4.1 Data Collection

The data for this project is collected from the Kaggle Dataset, the URL for the dataset-: https://www.kaggle.com/c/learn-ai-bbc/data

4.2 Data Description

4.3 File descriptions

- **BBC News Train.csv** - the training set of 1490 records
- **BBC News Test.csv** - the test set of 736 records
- **BBC News Sample Solution.csv** - a sample submission file in the correct format

4.4 Data fields

- **ArticleId** - Article id unique no. given to the record.
- **Article** - text of the header and article.
- **Category** - cateogry of the article (tech, business, sport, entertainment, politics.

4.5 Database

We are using the CSV file format for storing the data.

4.6 Data Cleaning and Preprocessing

1. **Lowercasing:** This operation involves converting all the text in the dataset to lowercase letters. It's a common step because it ensures that words are treated the same way regardless of their capitalization. For example, "Hello" and "hello" will both be converted to "hello."

2. **Punctuation Removal:** Punctuation marks (such as periods, commas, exclamation marks, etc.) are removed from the text. This helps in simplifying the text and ensures that words like "apple" and "apple!" are treated as the same word.

3. **Stop Word Removal:** Stop words are common words like "the," "and," "in," "of," etc., that don't typically provide meaningful information for text analysis tasks like text classification. Removing these words can reduce noise in the data and improve the efficiency of text processing algorithms.

4. **Tokenization:** Tokenization is the process of breaking down a text into individual words or tokens. It splits a sentence or a paragraph into smaller units (words or phrases), making it easier to analyze and process text data. For example, the sentence "The quick brown fox" would be tokenized into individual tokens: ["The", "quick", "brown", "fox"].

5. **TF-IDF (Term Frequency-Inverse Document Frequency):** TF-IDF is a technique used for text vectorization and feature engineering. It calculates a numerical value for each word in a document, representing its importance in relation to the entire dataset. This is done by considering

both how often the word appears in the document (term frequency) and how unique it is across the entire dataset (inverse document frequency). TF-IDF values are assigned to each token, and these values are often used as features for machine learning models in text classification tasks. TF-IDF helps in capturing the significance of words within the context of the dataset.

4.7 Model Building

3.5.1 **Data Splitting:** Start by dividing your dataset into three subsets: a training set, a validation set, and a test set. The training set is used to train your models and make early assessments of model performance, and the test set is reserved to evaluate the final model's performance.

3.5.2 **Selecting Machine Learning Algorithms:** You've mentioned that you intend to use machine learning algorithms like SVM, Random Forest, Gradient Boosting, and AdaBoost. Each of these algorithms has unique strengths:

SVM: Effective in high-dimensional spaces, it can handle complex decision boundaries.
Random Forest: Robust and able to capture complex

relationships in data.

Gradient Boosting: Builds models sequentially, focusing on correcting errors from the previous models.

AdaBoost: Emphasizes the weaknesses of the model by giving more weight to misclassified samples.

3.5.3 **Model Training:** Train each selected algorithm on the training data using the TF-IDF vectors as input features. Tune hyperparameters using the validation set to optimize each model's performance. You may need to adjust parameters like regularization strength, tree depth, or learning rate.

3.5.4 **Model Evaluation:** Assess the performance of each trained model using accuracy. Evaluate models on the validation set to choose the best-performing one.

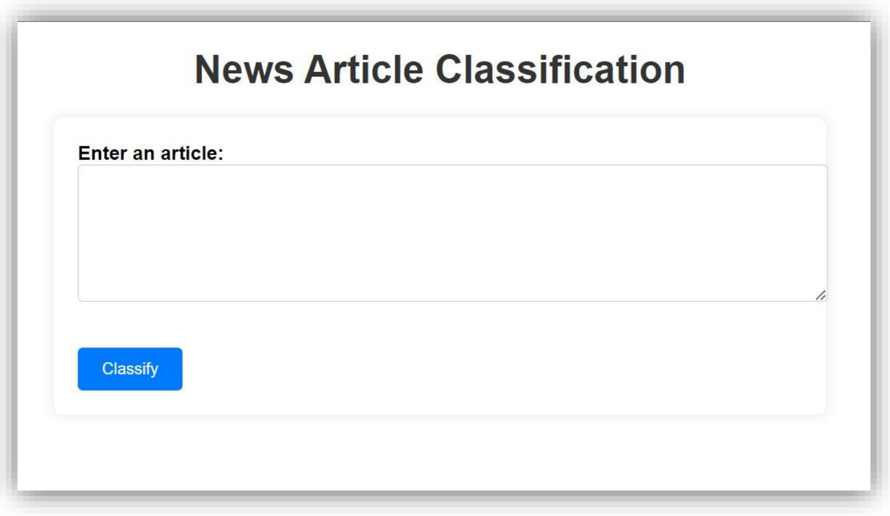
3.5.5 **Final Model Selection:** After evaluating all models on the validation set, choose the best-performing model based on the selected metrics.

3.5.6 **Model Testing:** Evaluate the selected model on the previously held-out test set to get an unbiased estimate of its real-world performance. Ensure that the model generalizes well to new, unseen data.

4.8 Data from the User

The required data from the customer is retrieved by using web pages.

We have used Flask for the creation of web pages for the user interface.



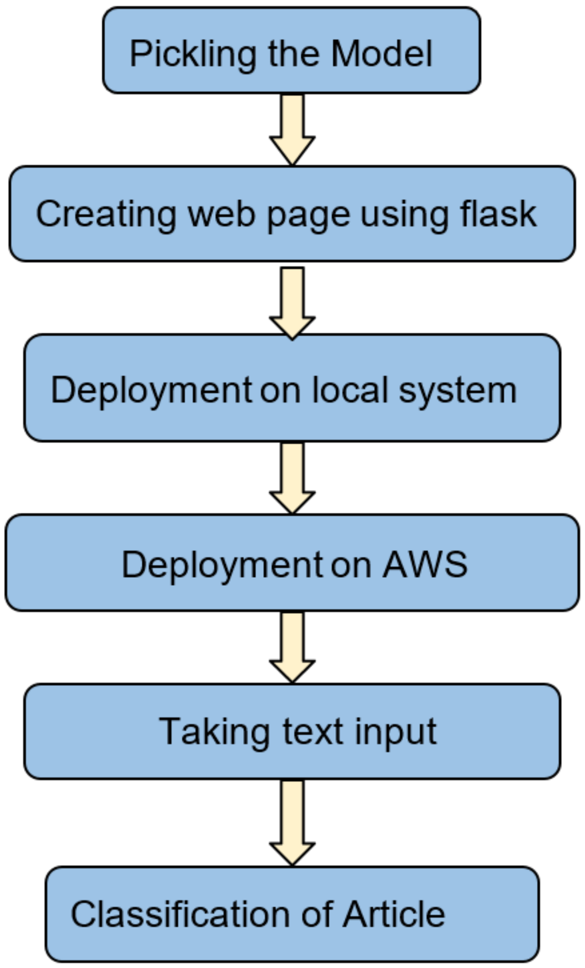
4.9 Data Validation

The article will be passed to the model through web pages.

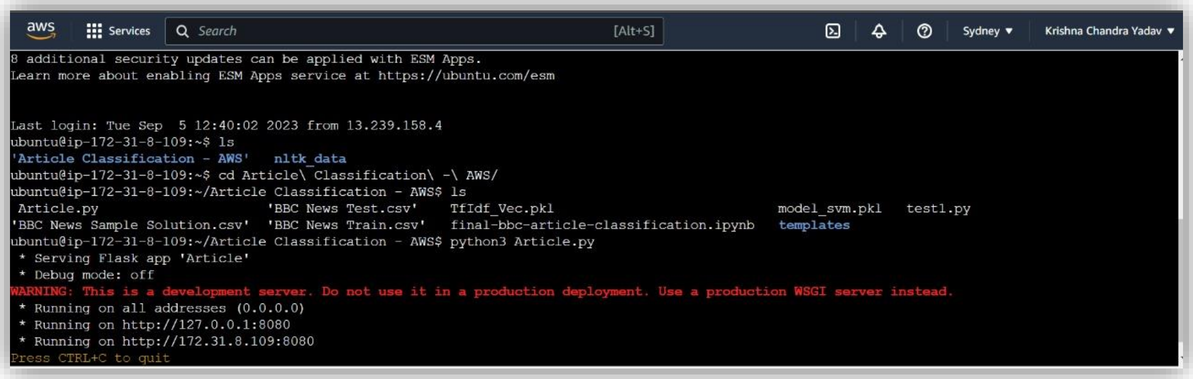
4.10 Results

A trained vector pickle file converts the user input text data into a number format. Then the data is given to a machine learning model (SVM) to classify so we will get the appropriate class.

5. DEPLOYMENT


Pickling the Model
Creating web page using flask
Deployment on local system
Deployment on AWS
Taking text input
Classification of Article

AWS cloud services are used to deploy model



```
8 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm


Last login: Tue Sep  5 12:40:02 2023 from 13.239.158.4
ubuntu@ip-172-31-8-109:~$ ls
'Article Classification - AWS'   nltk_data
ubuntu@ip-172-31-8-109:~$ cd Article\ Classification\ -\ AWS/
ubuntu@ip-172-31-8-109:~/Article Classification - AWS$ ls
 Article.py                       'BBC News Test.csv'    TfIdf_Vec.pkl                            model_svm.pkl   test1.py
'BBC News Sample Solution.csv'   'BBC News Train.csv'    final-bbc-article-classification.ipynb   templates
ubuntu@ip-172-31-8-109:~/Article Classification - AWS$ python3 Article.py
 * Serving Flask app 'Article'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:8080
 * Running on http://172.31.8.109:8080
Press CTRL+C to quit
```

TESTING

Unit Testing

4.1 Unit test case

Test-Case Description	Pre-Requisites	Expected Results
Verify whether the Webpage is accessible to the User or not.	Webpage URL should be defined.	Webpage should be accessible to the User.
Verify whether the Webpage is completely loaded for the User or not	1. Webpage URL is accessible. 2. Webpage is deployed.	The Webpage should be completely loaded for the User when it is accessed.
Verify whether the User is able to enter data in input fields or not.	1. Webpage URL is accessible. 2. Webpage is deployed. 3. WebPage input fields are editable.	The User is able to enter data in input fields.

Verify whether the User is able to submit details or not.	1. Webpage URL is accessible. 2. Webpage is deployed. 3. Webpage input fields are editable.	The User is able to submit details to process.