DAA CP

1. Data Cleaning
2. Exploratory Data Analysis (EDA)
3. Feature engineering
4. model building
5. model evaluation
6. Improvements depending on the evaluation
7. ensemble learning
8. Converting the project into a website
9. Deploying the website on Heruko

DATA CLEANING

1. dropping last 3 columns in spam.csv
2. renaming the columns
3. checking the missing values
4. checking the duplicate values
5. removing the duplicates

EDA

1. Data is imbalanced
2. ‘apply’ is a function in ‘pandas’ library
3. Fetching the number of words in each SPAM
4. Fetching the number of sentences in each SPAM
5. Analysing ham
6. Analysing SPAM
7. Analysing the ham and SPAM messages with the help of a histogram on the basis of number of characters and words distinctly.
8. To know the relation between number of characters, words, and sentences using correlation map

DATA PREPROCESSING

1. Lower case
2. Tokenization
3. Removing special characters
4. Removing stop words and punctuation
5. Stemming

MODEL BUILDING

1. Vectorising the data using bag of words/ TfidfVectorizer
2. Give this as input in Naïve Bayes
3. Find the accuracy to check the performance
4. Precision Score must be high. It has more priority than Accuracy. So, hence we use “Multinomial Naïve Bayes”
5. Tfidf 🡪 MNB
6. ETC: Extra Tree Classifier has the highest accuracy score. ETC is similar to Random Forest Classifier. Variation of RF. More random than RF.
7. Naïve Bayes has the highest Precision
8. Improving the model
9. Change the max\_featuers parameter of TFIDF, max\_features=3000
10. While vectorizing the text, we can restrict the no. of words
11. Now, Naïve Bayes also has comparable accuracy score.
12. Hence, the most powerful algorithm on this problem is ‘NB’.
13. Scaling the X array for increasing the accuracy of NB
14. But, the precision score decreased.
15. So, scaling won’t help.
16. Appending the number of characters column to X
17. Therefore, Accuracy and Precision decreased again.
18. Using Voting Classifier by combing the best performing algorithms. Voting combinations have equal weightages.
19. Accuracy increased bust precision decreased.
20. Used Stacking, like Voting. Making combinations of algorithms based on your given weightages which are decided using a Final Estimator.
21. This did not give improved results.

CONCLUSION

1. Using Multinomial Naïve Bayes because that is the best performing model.
2. Creating a pipeline which would be later converted into a website.
3. Preprocessing(transforming) an email. [Step 1]
4. Lamitization
5. Removing stop words
6. Converting text into Lower case
7. Vectorizing [Step 2]
8. Applying the algorithm on the text
9. Pickling(pickel) two files.
10. First, pickling the ‘tfidf’ file, naming it as Vectorizer.pkl (wb)
11. Second, pickling the ‘mnb’ bile, naming it as Model.pkl (wb)
12. Open the ipynb jupyter file in chrome (frequent browser, not the default one)
13. Predict [Step 3]
14. Display [Step 4]

DAA\_CP.ipynb

This code is a Python script for performing a variety of tasks related to text classification, including data cleaning, exploratory data analysis, data preprocessing, model building, and model evaluation. It appears to be working with a dataset of text messages (likely SMS messages) and aims to classify them as either "ham" or "spam." Below is a detailed explanation of each section of the code:

1. Importing Libraries:

- The script starts by importing the necessary libraries, including NumPy, Pandas, Matplotlib, Seaborn, and scikit-learn's various modules.

2. Loading the Dataset:

- It reads a CSV file named 'spam.csv' using Pandas with a specific encoding.

- It displays a random sample of 5 rows from the DataFrame.

- It prints the shape of the dataset.

3. Data Cleaning:

- Calls `df.info()` to get information about the dataset, including data types and missing values.

- Drops the columns 'Unnamed: 2', 'Unnamed: 3', and 'Unnamed: 4' since they appear to be irrelevant.

- Renames columns 'v1' to 'target' and 'v2' to 'text'.

- Encodes the 'target' column (ham or spam) as numerical values using LabelEncoder.

- Checks for missing values using `df.isnull().sum()`.

- Checks for and removes duplicate rows.

- Displays the shape of the cleaned dataset.

4. Exploratory Data Analysis:

- Displays the first 5 rows of the dataset.

- Counts and plots the distribution of 'ham' and 'spam' messages using a pie chart.

- Installs and downloads NLTK, then calculates and adds new columns to the DataFrame: 'num\_characters' (character count), 'num\_words' (word count), and 'num\_sentences' (sentence count).

- Provides descriptive statistics for these new columns for both 'ham' and 'spam' messages.

- Creates histograms and a pair plot to visualize the distributions and relationships.

5. Data Preprocessing:

- Downloads NLTK stopwords.

- Defines a function `transform\_text` for text preprocessing. This function converts text to lowercase, tokenizes it, removes punctuation and stopwords, and applies stemming.

- Applies the `transform\_text` function to the 'text' column of the DataFrame, creating a new 'transformed\_text' column.

- Generates WordClouds for 'ham' and 'spam' messages to visualize frequently used words.

6. Text Analysis:

- Extracts the transformed words and performs word frequency analysis on spam messages.

- Plots the top 30 most common words in spam messages.

7. Text Vectorization:

- Uses TF-IDF vectorization to convert text data into numerical features.

8. Model Building:

- Splits the dataset into training and testing sets.

- Initializes several classifiers, including Gaussian Naive Bayes, Multinomial Naive Bayes, and Bernoulli Naive Bayes.

- Fits these classifiers on the training data and evaluates their performance using accuracy, confusion matrix, and precision.

- Initializes additional classifiers, such as Support Vector Machine, Decision Tree, Random Forest, etc.

- Defines a function to train and evaluate various classifiers and prints their accuracy and precision.

- Creates a DataFrame summarizing the performance of all classifiers.

- Visualizes the performance using a bar plot.

9. Model Improvement:

- The code explores different techniques for improving the model's performance, including changing the max\_features parameter for TF-IDF and feature scaling.

- It then evaluates and compares the performance of these modified models.

10. Ensemble Models:

- The script constructs ensemble models, including a Voting Classifier and a Stacking Classifier, combining multiple base classifiers.

11. Model Serialization:

- The final section saves the TF-IDF vectorizer and the Multinomial Naive Bayes model to pickle files for future use.

Overall, this code is a comprehensive example of text classification, covering data preprocessing, feature engineering, model selection, and evaluation. It explores multiple classifiers and ensemble techniques to improve performance.

App.py  
  
This code is a Python script that uses the Streamlit framework to create a simple web application for classifying email messages as either spam or not spam. The code loads a pre-trained text classification model and a TF-IDF vectorizer, allowing users to input a message and get a spam classification prediction. Let's break down the code step by step:

1. Importing Libraries:

- The code begins by importing the necessary libraries, including Streamlit for creating the web application, pickle for loading pre-trained models, and NLTK for natural language processing tasks.

2. Text Preprocessing Function:

- The code defines a function `transform\_text(text)` for text preprocessing. This function takes a text input and performs the following steps:

- Converts the text to lowercase.

- Tokenizes the text using NLTK's word\_tokenize.

- Removes non-alphanumeric characters.

- Removes stopwords and punctuation.

- Applies stemming using the Porter Stemmer.

- Joins the processed words back into a string.

3. Loading Pre-trained Model and Vectorizer:

- It uses `pickle` to load a pre-trained TF-IDF vectorizer (`vectorizer.pkl`) and a text classification model (`model.pkl`) into memory. These were likely trained in a separate script, as indicated in the previous code you shared.

4. Streamlit Application:

- It starts the Streamlit application with `st.title` to set the title of the web application.

5. User Input:

- It creates a text input area using `st.text\_area` where the user can enter the email message they want to classify.

6. Prediction Button:

- It creates a "Predict" button using `st.button`. When the button is clicked, the application will perform the prediction.

7. Prediction Logic:

- When the "Predict" button is clicked, the following steps are executed:

- It preprocesses the user's input message using the `transform\_text` function.

- It vectorizes the preprocessed message using the TF-IDF vectorizer loaded from the pickled model.

- It uses the pre-trained model to make a prediction on the vectorized input.

- If the prediction result is 1, it displays "SPAM!" using `st.header`. If the result is 0, it displays "NOT Spam!".

In summary, this code creates a user-friendly web application using Streamlit, which takes an email message as input, processes it, and predicts whether it's spam or not. The prediction is made using a pre-trained model and vectorizer that have been pickled and loaded into the application. This type of application can be useful for end-users who want to quickly classify their email messages.