**Machine Learning Analysis of COVID-19 Research Papers: Insights and Performance Comparison**

**Kevin Zhou Mastascusa – INFO371 – 6/11/2023**

# Abstract

In this project, I analyzed COVID-19 research papers using machine learning models. The Random Forest model showed the best performance, accurately classifying the papers. The Decision Tree model with sentiment scores also performed well. However, KNN models struggled to predict accurately. These findings demonstrate the effectiveness of machine learning in organizing and analyzing research literature for COVID-19.

# CORD-19: COVID-19 Open Research Dataset

CORD-19 is a free resource of tens of thousands of scholarly articles about COVID-19, SARS-CoV-2, and related coronaviruses for use by the global research community.

*“In response to the COVID-19 pandemic, the White House and a coalition of leading research groups have prepared the COVID-19 Open Research Dataset (CORD-19). CORD-19 is a resource of over 1,000,000 scholarly articles, including over 400,000 with full text, about COVID-19, SARS-CoV-2, and related coronaviruses. This freely available dataset is provided to the global research community to apply recent advances in natural language processing and other AI techniques to generate new insights in support of the ongoing fight against this infectious disease. There is a growing urgency for these approaches because of the rapid acceleration in new coronavirus literature, making it difficult for the medical research community to keep up.”* [https://www.kaggle.com/datasets/allen-institute-for-ai/CORD-19-research-challenge]

# **Data with a set of feature attributes and class attributes.**

A screenshot of a computer program

Description automatically generated with medium confidencemetadata.csv(1.65 GB)

About this file

Metadata for 59k articles

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| cord\_uid | sha | source\_x | title | doi | pmcid | pubmed\_id | license | abstract | publish\_time | authors |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| journal | mag\_id | who\_covidence\_id | arxiv\_id | pdf\_json\_files | pmc\_json\_files | url | s2\_id |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

|  |
| --- |
| cord\_uid: A unique identifier for each research article. |
| sha: The unique identifier of the specific version of the article (usually associated with a specific revision or format). |
| source\_x: The source of the article (e.g., PMC - PubMed Central). |
| title: The title of the research article. |
| doi: The Digital Object Identifier, a unique alphanumeric string assigned to the article to provide a persistent link to its location on the internet. |
| pmcid: The PubMed Central identifier, a unique identifier assigned to articles in the PubMed Central database. |
| pubmed\_id: The PubMed identifier, a unique identifier assigned to articles in the PubMed database. |
| license: The license under which the article is distributed (e.g., no-cc - no Creative Commons license specified). |
| abstract: A summary or brief description of the research article. |
| publish\_time: The date of publication of the research article. |
| authors: The names of the authors who contributed to the research article. |
| journal: The name of the journal in which the research article was published. |
| mag\_id: The identifier of the Microsoft Academic Graph (MAG) associated with the article. |
| who\_covidence\_id: The identifier of the research article in the WHO COVID-19 database (if applicable). |
| arxiv\_id: The identifier of the article in the arXiv preprint repository (if applicable). |
| pdf\_json\_files: The file path or location of the PDF version of the research article. |
| pmc\_json\_files: The file path or location of the PMC XML version of the research article. |
| url: The URL or web link to access the full text of the research article. |
| s2\_id: The identifier of the article in the Semantic Scholar database (if applicable). |

# **Data pre-processing methods**

I decided to remove the unnecessary columns from the dataset to focus only on the relevant features for my analysis. I used the drop() function to remove columns such as 'cord\_uid', 'sha', 'source\_x', 'doi', and more. By doing this, I reduced the dimensionality of the dataset and kept only the columns that are important for my analysis.

Additionally, I wanted to ensure that my data is clean and reliable, so I removed any rows that had missing values. I used the dropna() function to drop these rows and ensure that I am working with complete data.

First, I checked the dataset for missing values by using the df.isnull().sum() function, which provided the number of missing values in each column. To ensure data completeness, I removed the rows with missing values using the df.dropna(inplace=True) function. This step ensures that the remaining data is reliable and suitable for analysis.

Next, I dealt with duplicate rows in the dataset. By utilizing the df.drop\_duplicates(inplace=True) function, I removed any duplicate records. This process eliminates redundant data, ensuring that each observation is unique and preventing biases in subsequent analysis.

In order to facilitate classification tasks, I created a target column based on the 'title' column. By applying a case-insensitive condition to check if the word 'covid-19' exists in each title, I assigned a value of 1 to the target column for the corresponding records that met the condition. For the remaining records, I assigned a value of 0 to the target column. This target column will serve as the basis for classifying the data into COVID-19 related and non-related categories.

# **Models**

1. Decision Tree Classifier,
2. Random Forest Classifier
3. Multinomial Naive Bayes
4. and K-Nearest Neighbors (with k as 5, 10, 15) models.

# Experiment Design

The goal of the project is to systematically evaluate different combinations of data pre-processing methods and classification models for sentiment analysis on self-selected data. I will start with a design of the experiments.

* Data Preprocessing:
  + I will start by loading the dataset containing research paper information (metadata).
  + Then, I'll perform data cleaning and preprocessing steps. This involves removing unnecessary columns, handling missing values, and removing duplicates.
  + Next, I'll create the target column based on the presence of the term "covid-19" in the paper titles.
  + To prepare the title attribute for classification, I'll convert it to lowercase, remove punctuation, numbers, and stopwords, and perform lemmatization.
* Feature Engineering:
  + For my classification task, I'll select the feature attributes that include the transformed title (using TF-IDF or CountVectorizer features) and the sentiment score.
  + To ensure the model's performance, I'll split the dataset into training and testing sets.
* Model Selection and Training:
  + I will choose the machine learning models to be evaluated, such as K-Nearest Neighbors (KNN) and Naive Bayes (MultinomialNB).
  + Using the training set and the selected feature attributes, I'll train each model.
  + If needed, I'll tune the model parameters, such as the value of K for KNN, to find the optimal configuration.
* Model Evaluation:
  + Once the models are trained, I will evaluate their performance using the testing set.
  + I'll calculate performance metrics like accuracy, precision, recall, F1-score, and the confusion matrix for each model.
  + By comparing the performance of the different models, I can determine the best-performing one.
* Further Analysis and Optimization:
  + To gain more insights into the classification performance and the impact of different feature attributes, I'll analyze the results obtained.
  + If necessary, I'll experiment with different preprocessing techniques, feature selections, or model configurations to optimize the classification performance.
  + Additional cross-validation or experiments might be conducted to validate the model's performance and generalizability.
* Reporting and Conclusion:
  + Finally, I will summarize my findings and draw conclusions from the experiment.
  + I'll present the classification results, including the performance metrics and any insights gained from the analysis.
  + I will also discuss the strengths and limitations of my chosen approach and suggest potential areas for future improvements.
  + Following this experiment design, I will be able to systematically evaluate the performance of different machine learning models using the selected feature attributes and preprocessing methods. The results obtained will provide valuable insights into the effectiveness of my chosen approach for classifying research papers related to COVID-19.

# **CHOOSE THE CLASS ATTRIBUTE.**

I created the target column for classification by checking if the term "covid-19" was present in the "title" column of the metadata. If it was, I assigned a value of 1 to indicate the positive class. Otherwise, I assigned a value of 0 to indicate the negative class. This allowed me to label instances in the dataset and use them for training and evaluating machine learning models.

The class attribute (or target variable) is target.

# **CHOOSE SUBSETS OF FEATURE ATTRIBUTES.**

Currently, I have selected the title attribute as one of the feature attributes for my analysis. To represent the textual content of the research paper titles, I have transformed the title using either TF-IDF or CountVectorizer. This transformation allows me to convert the titles into numerical features that can be utilized by machine learning algorithms.

Additionally, I have incorporated the sentiment score as another feature attribute. By calculating the sentiment polarity of each title, I can capture the sentiment associated with the research papers. This sentiment score provides valuable information that can potentially enhance the classification task.

In summary, my chosen feature attribute subsets consist of the transformed title (TF-IDF or CountVectorizer features) and the sentiment score. By leveraging both the textual content and sentiment information, I aim to create a comprehensive set of features that can contribute to accurate classification results.

# **CHOOSE PRE-PROCESSING METHODS.**

For text cleaning, I applied several preprocessing methods to the 'title' column:

1. Convert to lower case: I converted all the text to lowercase using the lambda function **lambda x: x.lower()**. This helps in standardizing the text and avoiding issues with case sensitivity.
2. Remove punctuation: I used the **translate** method with **str.maketrans('', '', string.punctuation)** to remove all punctuation marks from the text. This step helps in removing special characters that may not contribute to the overall meaning of the text.
3. Remove numbers: I used the **translate** method with **str.maketrans('', '', string.digits)** to remove any numerical digits from the text. This step can be useful if the numbers are not relevant for the analysis.
4. Remove whitespaces: I removed any leading or trailing whitespaces from the text using the **strip()** method. This ensures that there are no unnecessary spaces before or after the text.
5. Remove stopwords: I removed stopwords from the text using the NLTK library. I first loaded the set of stopwords for the English language using **stopwords.words('english')**. Then, I used a list comprehension to remove stopwords from each word in the text.
6. Remove short words: I filtered out short words with a length less than 4 characters using a list comprehension. This step helps in removing very short words that may not provide much information.
7. Lemmatization: I performed lemmatization on the text using the WordNetLemmatizer class from the NLTK library. I initialized an instance of the lemmatizer using **lemmatizer = WordNetLemmatizer()**. Then, I applied lemmatization to each word in the text using a list comprehension.

These preprocessing steps aim to clean and normalize the text data, making it more suitable for further analysis and machine learning tasks.

When it comes to preprocessing the text data, I opted for a simple approach without explicit tokenization. Instead, I relied on the TF-IDF vectorizer and CountVectorizer provided by scikit-learn. These vectorizers automatically perform tokenization as part of their process.

Using the TF-IDF vectorizer or CountVectorizer, I transformed the text data into numerical representations by considering each word as a token. This means that the text was split into individual words or tokens, with whitespace and punctuation marks used as delimiters.

By leveraging these vectorizers, I was able to extract features from the text data without explicitly performing tokenization myself. This allowed me to proceed with further analysis and modeling using the numerical representations of the text.

|  |
| --- |
| * Data Pre-processing Methods: |
| * + Binary: I used the CountVectorizer with binary=True to create a binary representation of the text data, where each word is represented by 0 or 1 based on its presence in the title. |
| * + TF: I used the CountVectorizer without specifying the binary parameter, which creates a matrix where each entry represents the count of a word in the title. |
| * + TF-IDF: I used the TfidfVectorizer to create a TF-IDF representation of the text data, where each word is weighted based on its frequency in the title and rarity in the entire corpus. |
| * + WordsToKeep: I applied various text cleaning techniques like converting to lowercase, removing punctuation, numbers, whitespaces, stopwords, and short words. I also performed lemmatization to reduce words to their base form. |
| * Models: |
| * + KNN: I used the KNeighborsClassifier and experimented with different values of n\_neighbors (k) to determine the optimal number of neighbors for the KNN algorithm. |
| * + Naïve Bayes: I used the MultinomialNB classifier, which is a variant of Naïve Bayes specifically designed for discrete features. |
| * + Decision Tree: I used the DecisionTreeClassifier to build a decision tree model and explore different parameters like min\_samples\_leaf, subtree\_raising, and J48 options. |
| * Experimental Design: |
| * + I applied different pre-processing methods to the data, including binary, TF, TF-IDF, and WordsToKeep, to observe their impact on model performance. |
| * + For KNN, I experimented with different values of n\_neighbors (k) to find the optimal number of neighbors. |
| * + For the decision tree model, I explored different parameters like min\_samples\_leaf and subtree\_raising to evaluate their effect on the model's performance. |
| * Model Evaluation and Comparison: |
| * + After training the models, I evaluated their performance using various metrics such as accuracy, precision, recall, F1-score, mean squared error, mean absolute error, R2 score, and explained variance score. |
| * + I also saved the classification reports and confusion matrices to CSV files for further analysis and comparison. |
| * Choosing the Best Model: |
| * + Based on the performance metrics and analysis of different models and configurations, I can compare their results and consider factors like accuracy, precision, recall, and other evaluation metrics to determine the best model for the given problem. |

# TF

A screen shot of a computer

Description automatically generated with low confidence

# TF-IDF

A screenshot of a computer screen

Description automatically generated with low confidence

# Sentiment

A screenshot of a computer

Description automatically generated

# **CHOOSE DIFFERENT K FOR KNN.**

The code currently experiments with different values of K for the K-Nearest Neighbors (KNN) classifier, specifically using K values of 5, 10, and 15.

# **CHOOSE NAÏVEBAYES OR NAIVEBAYESMULTINOMIALTEXT.**

In my current code, I’m using a Multinomial Naive Bayes classifier. I were to use a simple Naive Bayes classifier instead, it would not be able to handle the text data directly, and it might not perform as well given that the Multinomial Naive Bayes is particularly designed for text data.

# **Choose different parameters, such as minNumObj, subtree raising, for the decision tree model, J48.**

J48 is an implementation of the C4.5 algorithm in Weka, but my current code uses a Decision Tree classifier from scikit-learn in Python (which is similar). I could experiment with different parameters, such as max\_depth (maximum depth of the tree), min\_samples\_split (minimum number of samples required to split an internal node), min\_samples\_leaf (minimum number of samples required to be at a leaf node), etc. For the subtree raising (a parameter in J48), the equivalent in scikit-learn's decision trees would be presort=False to disable presorting of data to speed up the finding of best splits in fitting.

# Data

**CORD19\_abstract.csv**

|  |  |
| --- | --- |
| title | target |
| Clinical features of culture-proven Mycoplasma pneumoniae infections at King Abdulaziz University Hospital, Jeddah, Saudi Arabia | 0 |
| Nitric oxide: a pro-inflammatory mediator in lung disease? | 0 |
| Surfactant protein-D and pulmonary host defense | 0 |
| … |  |
| Machine Learning-Based Decision Model to Distinguish Between COVID-19 and Influenza: A Retrospective, Two-Centered, Diagnostic Study | 1 |
| Linked HIV/SARS-CoV-2 testing could reduce incidence of HIV and costs | 0 |
| Can we imagine the meal-sharing economy without service providers? The impact of COVID-19 | 1 |
| Efficacy and Safety of Heterologous Booster Vaccination after Ad5-nCoV (CanSino Biologics) Vaccine: A Preliminary Descriptive Study | 0 |
| Negative Impacts of the Current COVID-19 Crisis on Science Education in Kenya: How Certain Can We Be About the Efficacy of the Science Learning Framework Online? | 1 |
| The SARS‚ÄêCoV‚Äê2 pandemic is associated with increased severity of presentation of childhood onset type 1 diabetes mellitus: A multi‚Äêcentre study of the first COVID‚Äê19 wave | 0 |
| The mediator role of stigma in the association of mindfulness and social engagement among breast cancer survivors in China | 0 |
| Microbes and Parkinson's disease: from associations to mechanisms. | 0 |
| Plant-Derived Lactobacillus paracasei IJH-SONE68 Improves Chronic Allergy Status: A Randomized, Double-Blind, Placebo-Controlled Clinical Trial | 0 |
| Exploring the role of R&D collaborations and non-patent IP policies in government technology transfer performance: Evidence from U.S. federal agencies (1999‚Äì2016) | 0 |
| The economic burden of COVID-19 in the United States: Estimates and projections under an infection-based herd immunity approach | 1 |
| Reassortant Influenza A(H1N1)pdm09 Virus in Elderly Woman, Denmark, January 2021 | 0 |
| Myocarditis and Pericarditis in Adolescents after First and Second doses of mRNA COVID-19 Vaccines | 1 |
| Epilepsy by the numbers ‚Äì From the US Centers for Disease Control and Prevention: Six in 10 adults with active epilepsy saw a neurologist or epilepsy specialist in the past year, United States, 2017 | 0 |
| Presumed COVID-19 Index Case on Diamond Princess cruise ship and Evacuees to Hong Kong | 1 |
| Analyzing Iranian Opinions toward COVID-19 Vaccination | 1 |
| Axillary Lymphadenopathy After mRNA COVID-19 Vaccination | 1 |
| Trends and projections of caesarean section rates: global and regional estimates | 0 |
| The COVID-19 pandemics and the relevance of biosafety facilities for metagenomics surveillance, structured disease prevention and control | 1 |
| COVID-19 vaccine and anaphylaxis\* | 1 |

# Results

KNN N-5 Classification Report & Confusion Matrix

TF-IDF

|  |  |  |  |
| --- | --- | --- | --- |
| precision | recall | f1-score | support |
| 0.55778894 | 1 | 0.71612903 | 111 |
| 1 | 0.01123596 | 0.02222222 | 89 |
| 0.56 | 0.56 | 0.56 | 0.56 |
| 0.77889447 | 0.50561798 | 0.36917563 | 200 |
| 0.75457286 | 0.56 | 0.4073405 | 200 |

|  |  |
| --- | --- |
| 0 | 1 |
| 111 | 0 |
| 88 | 1 |

Naïve Bayes Classification Report & Confusion Matrix

TF-IDF

|  |  |  |  |
| --- | --- | --- | --- |
| precision | recall | f1-score | support |
| 0.73109244 | 0.78378378 | 0.75652174 | 111 |
| 0.7037037 | 0.64044944 | 0.67058824 | 89 |
| 0.72 | 0.72 | 0.72 | 0.72 |
| 0.71739807 | 0.71211661 | 0.71355499 | 200 |
| 0.71890445 | 0.72 | 0.71828133 | 200 |

|  |  |
| --- | --- |
| 0 | 1 |
| 87 | 24 |
| 32 | 57 |

Decision Tree Classification Report & Confusion Matrix

TF-IDF

|  |  |  |  |
| --- | --- | --- | --- |
| precision | recall | f1-score | support |
| 0.97142857 | 0.91891892 | 0.94444444 | 111 |
| 0.90526316 | 0.96629213 | 0.93478261 | 89 |
| 0.94 | 0.94 | 0.94 | 0.94 |
| 0.93834586 | 0.94260553 | 0.93961353 | 200 |
| 0.94198496 | 0.94 | 0.94014493 | 200 |

|  |  |
| --- | --- |
| 0 | 1 |
| 102 | 9 |
| 3 | 86 |

# Discussion

|  |
| --- |
| Classification Report: |
| * Precision: Of all the examples the classifier labeled as positive, what fraction were actually positive. |
| * Recall: Of all the actual positives, what fraction did the classifier label as positive. |
| * F1-Score: The harmonic mean of precision and recall. It provides a single score that balances both the concerns of precision and recall in one number. |
| * Support: The number of samples of the true response that lie in that class. |

|  |
| --- |
| Confusion Matrix: A table layout that allows visualization of the performance of an algorithm. In binary classification: |
| * True negatives is C(0,0) |
| * False negatives is C(1,0) |
| * True positives is C(1,1) |
| * False positives is C(0,1) |

It seems like the Decision Tree model performed the best among the three in terms of precision, recall, and F1-score. The F1-score of the Decision Tree model is 0.94, which is considerably higher than that of the K-Nearest Neighbors model (0.56) and Naive Bayes model (0.72).

Similarly, the confusion matrix for the Decision Tree model shows that it has correctly classified the majority of the instances, with just 12 instances incorrectly classified out of 200, as opposed to the K-Nearest Neighbors model which has incorrectly classified 88 instances, and the Naive Bayes model which incorrectly classified 56 instances.

These metrics are widely used to evaluate the performance of classification models and provide a good summary of how well the model can predict the different classes.

In my project, I used a train-test split with a test size of 0.2, which means that I allocated 20% of the data for testing the performance of my classification models, while I used the remaining 80% for training. This approach allowed me to assess the generalization ability of the models and evaluate their performance on unseen data.

# Sentiment Analysis

* Import the SentimentIntensityAnalyzer from nltk.sentiment. This is a class for performing sentiment analysis. It gives a sentiment score ranging from -1 (most negative) to +1 (most positive).
* Define a function get\_sentiment that takes a text string as input and returns a sentiment score for that text. The score is computed using the SentimentIntensityAnalyzer's polarity\_scores method, which returns a dictionary containing positive, negative, neutral, and compound scores. The function returns only the compound score, which is a single metric that calculates the sum of all the lexicon ratings and normalizes it between -1(most extreme negative) and +1 (most extreme positive).
* Load a cleaned dataset from a csv file named 'CORD19\_abstract\_cleaned.csv' into a pandas dataframe df.
* Print the first 5 rows of the dataframe for inspection.
* Add a new column 'sentiment' to the dataframe. This column is calculated by applying the get\_sentiment function to the 'title' column of the dataframe. If there are any NaN values in the 'title' column, they are first replaced with an empty string before applying the function.
* Print the first 5 rows of the dataframe again to check that the new 'sentiment' column has been added correctly.
* Save the modified dataframe to a new csv file named 'sentiments.csv'. The index=False parameter ensures that the dataframe index is not saved into the file.

**from** nltk.sentiment **import** SentimentIntensityAnalyzer  
  
sia **=** SentimentIntensityAnalyzer**()  
  
  
def get\_sentiment(**text**):** sentiment **=** sia.polarity\_scores**(**text**)  
 return** sentiment**['compound']** # return the compound score  
  
  
print**(' ------------------------------------ ')**print**(' ------------------------------------ ')**# Load cleaned dataset  
print**('Load cleaned dataset:')**df **=** pd.read\_csv**('CORD19\_abstract\_cleaned.csv')**# print the first 5 rows of the dataframe  
print**('Print the first 5 rows of the dataframe:')**print**(**df.head**())**# add a new column to the dataframe  
print**('Add a new column to the dataframe:')**df**['title'] =** df**['title']**.fillna**('')**df**['sentiment'] =** df**['title']**.apply**(**get\_sentiment**)**# print the first 5 rows of the dataframe  
print**('Print the first 5 rows of the dataframe:')**print**(**df.head**())**# save the dataframe to csv file  
print**('Save the dataframe to csv file:')**df.to\_csv**('sentiments.csv'**, index**=False)**

sentiments.csv

|  |  |  |
| --- | --- | --- |
| title | target | sentiment |
| clinical feature cultureproven mycoplasma pneumoniae infection king abdulaziz university hospital jeddah saudi arabia | 0 | 0.0 |
| nitric oxide proinflammatory mediator lung disease | 0 | 0.0 |
| surfactant proteind pulmonary host defense | 0 | 0.128 |
| role endothelin lung disease | 0 | 0.0 |
| gene expression epithelial cell response pneumovirus infection | 0 | 0.0 |
| sequence requirement strand transfer nidovirus discontinuous subgenomic synthesis | 0 | 0.0 |
| debate transfusing normal haemoglobin level improve outcome | 0 | 0.4404 |
| international symposium intensive care emergency medicine brussels belgium march | 0 | 0.1531 |
| heme oxygenase carbon monoxide pulmonary medicine | 0 | 0.0 |
| technical description rod realtime public health surveillance system | 0 | 0.0 |
| conservation polyamine regulation translational frameshifting yeast mammal | 0 | 0.0 |
| heterogeneous nuclear ribonucleoprotein regulates synthesis cytoplasmic virus | 0 | 0.0 |
| method identify domain interacting protein | 0 | 0.0 |
| vaccinia virus infection disrupts microtubule organization centrosome function | 0 | 0.0 |
| site origin influenza pandemic public health implication | 0 | 0.0 |
| multifaceted multiversatile microarray simultaneous detection many virus expression profile | 0 | 0.0 |
| herpes simplex virus type normal protein permeability lung critically patient case pathogenicity | 0 | 0.0 |
| logistics community smallpox control contact tracing ring vaccination stochastic network model | 0 | 0.0 |
| protection pulmonary epithelial cell oxidative stress hmyh adenine glycosylase | 0 | -0.4215 |
| bioinformatic mapping alkb homology domain virus | 0 | 0.0 |
| managing emerging infectious disease federal system impediment effective law | 0 | 0.4767 |
| protein secretion lactococcus lactis efficient increase overall heterologous protein production | 0 | 0.6249 |
| detection characterization horizontal transfer prokaryote using genomic signature | 0 | 0.0 |
| comparison substitution insertion deletion probe resequencing mutational analysis using oligonucleotide microarrays | 0 | 0.0 |

**import** numpy **as** np  
  
# Load sentiments dataset  
print**('Load sentiments dataset:')**df **=** pd.read\_csv**('sentiments.csv')**# Sentiment Analysis  
print**('Sentiment Analysis:')**print**('=' \*** 50**)**# Take a small sample of the data  
print**('Take a small sample of the data:')**df\_sample **=** df.sample**(**n**=**1000, random\_state**=**42**)**# TF  
print**('TF Text Vectorizer:')**tf\_vectorizer **=** CountVectorizer**(**max\_features**=**1000**)**tf\_vectorizer.fit**(**df\_sample**['title'])**X\_text **=** tf\_vectorizer.transform**(**df\_sample**['title'])**# Include the sentiment scores  
print**('Including the sentiment scores:')**X\_sentiment **=** df\_sample**['sentiment']**.values.reshape**(-**1, 1**)**X **=** np.concatenate**((**X\_text.toarray**()**, X\_sentiment**)**, axis**=**1**)**# Prepare data for sentiment analysis  
print**('Prepare data for sentiment analysis:')**X\_train, X\_test, y\_train, y\_test **=** train\_test\_split**(**X, df\_sample**['target']**, test\_size**=**0.2, random\_state**=**42**)**# Train the model  
print**('Train the model:')**model **=** RandomForestClassifier**()**model.fit**(**X\_train, y\_train**)**# Predict the labels  
print**('Predict the labels:')**y\_pred\_binary **=** model.predict**(**X\_test**)**print**('Print the predicted labels:')**print**(**y\_pred\_binary**)**# Evaluate the model  
print**('Evaluate the model:')**print**('Accuracy:')**print**(**accuracy\_score**(**y\_test, y\_pred\_binary**))**print**('Precision, Recall, F1-Score:')**print**(**precision\_recall\_fscore\_support**(**y\_test, y\_pred\_binary, average**='macro'))**print**('Confusion Matrix:')**print**(**confusion\_matrix**(**y\_test, y\_pred\_binary**))**print**('Classification Report:')**print**(**classification\_report**(**y\_test, y\_pred\_binary**))**print**('Mean Squared Error:')**print**(**mean\_squared\_error**(**y\_test, y\_pred\_binary**))**print**('Mean Absolute Error:')**print**(**mean\_absolute\_error**(**y\_test, y\_pred\_binary**))**print**('R2 Score:')**print**(**r2\_score**(**y\_test, y\_pred\_binary**))**print**('Explained Variance Score:')**print**(**explained\_variance\_score**(**y\_test, y\_pred\_binary**))**# Save classification report to csv file  
print**('Save classification report to csv file:')**report **=** classification\_report**(**y\_test, y\_pred\_binary, output\_dict**=True)**df\_report **=** pd.DataFrame**(**report**)**.transpose**()**df\_report.to\_csv**('classification\_reportSent.csv'**, index**=False)**# Save confusion matrix to csv file  
print**('Save confusion matrix to csv file:')**df\_confusion\_matrix **=** pd.DataFrame**(**confusion\_matrix**(**y\_test, y\_pred\_binary**))**df\_confusion\_matrix.to\_csv**('confusion\_matrixSent.csv'**, index**=False)**print**(' ------------------------------------ ')**

This script is modified to include the sentiment score as an additional feature in my classification model. It now follows these steps:

1. Load the sentiments dataset which has the calculated sentiment scores.
2. Choose a random sample of the data.
3. Use the TF Text Vectorizer to convert the text into features.
4. The sentiment scores are then included as an additional feature by reshaping them into the correct shape and concatenating them with the text features.
5. Split the data into training and testing sets.
6. Train a RandomForestClassifier model on the training data.
7. Predict the labels on the testing data.
8. Evaluate the model by calculating several metrics like Accuracy, Precision, Recall, F1-Score, and also generating a confusion matrix and classification report.
9. The classification report and confusion matrix are then saved to csv files.

The primary modification here is the inclusion of sentiment scores as an additional feature. By including this additional feature, the model now has more information to use for classification, which may help improve its performance.

Decision tree classification report & confusion matrix

TF-IDF

|  |  |  |  |
| --- | --- | --- | --- |
| precision | recall | f1-score | support |
| 0.97142857 | 0.91891892 | 0.94444444 | 111 |
| 0.90526316 | 0.96629213 | 0.93478261 | 89 |
| 0.94 | 0.94 | 0.94 | 0.94 |
| 0.93834586 | 0.94260553 | 0.93961353 | 200 |
| 0.94198496 | 0.94 | 0.94014493 | 200 |

|  |  |
| --- | --- |
| 0 | 1 |
| 102 | 9 |
| 3 | 86 |

Decision tree classification report & confusion matrix (sentiment scores as an additional feature)

TF-IDF

|  |  |  |  |
| --- | --- | --- | --- |
| precision | recall | f1-score | support |
| 0.96296296 | 0.93693694 | 0.94977169 | 111 |
| 0.92391304 | 0.95505618 | 0.93922652 | 89 |
| 0.945 | 0.945 | 0.945 | 0.945 |
| 0.943438 | 0.94599656 | 0.9444991 | 200 |
| 0.94558575 | 0.945 | 0.94507909 | 200 |

|  |  |
| --- | --- |
| 0 | 1 |
| 104 | 7 |
| 4 | 85 |

Random Forest classification report & confusion matrix

TF-IDF

|  |  |  |  |
| --- | --- | --- | --- |
| **precision** | **recall** | **f1-score** | **support** |
| 0.98130841 | 0.94594595 | 0.96330275 | 111 |
| 0.93548387 | 0.97752809 | 0.95604396 | 89 |
| 0.96 | 0.96 | 0.96 | 0.96 |
| 0.95839614 | 0.96173702 | 0.95967335 | 200 |
| 0.96091649 | 0.96 | 0.96007259 | 200 |

|  |  |
| --- | --- |
| 0 | 1 |
| 105 | 6 |
| 2 | 87 |

Random Forest classification report & confusion matrix (sentiment scores as an additional feature)

TF

|  |  |
| --- | --- |
| 0 | 1 |
| 105 | 6 |
| 2 | 87 |

|  |  |  |  |
| --- | --- | --- | --- |
| **precision** | **recall** | **f1-score** | **support** |
| 0.98130841 | 0.94594595 | 0.96330275 | 111 |
| 0.93548387 | 0.97752809 | 0.95604396 | 89 |
| 0.96 | 0.96 | 0.96 | 0.96 |
| 0.95839614 | 0.96173702 | 0.95967335 | 200 |
| 0.96091649 | 0.96 | 0.96007259 | 200 |

Decision tree classification report & confusion matrix (sentiment scores as an additional feature using model = DecisionTreeClassifier(max\_depth=10, min\_samples\_split=2, min\_samples\_leaf=1))

TF-IDF

|  |  |  |  |
| --- | --- | --- | --- |
| precision | recall | f1-score | support |
| 0.98095238 | 0.92792793 | 0.9537037 | 111 |
| 0.91578947 | 0.97752809 | 0.94565217 | 89 |
| 0.95 | 0.95 | 0.95 | 0.95 |
| 0.94837093 | 0.95272801 | 0.94967794 | 200 |
| 0.95195489 | 0.95 | 0.95012077 | 200 |

|  |  |
| --- | --- |
| 0 | 1 |
| 103 | 8 |
| 2 | 87 |

# Discussion

It seems that adding sentiment scores as an additional feature improved the performance of the decision tree model.

In the first model (without sentiment scores), we had a weighted average precision of 0.9383, a recall of 0.9426, and an F1-score of 0.9396. Looking at the confusion matrix, there were 102 true positives and 9 false negatives for class 0, and 3 false positives and 86 true negatives for class 1.

In the second model (with sentiment scores), the weighted average precision improved to 0.9434, recall improved to 0.9460, and the F1-score improved to 0.9445. The confusion matrix also shows improvement with 104 true positives and 7 false negatives for class 0, and 4 false positives and 85 true negatives for class 1.

These improvements are marginal, but they indicate that adding sentiment scores as a feature may enhance the decision tree's predictive capability.

The classification report and confusion matrix show that using a DecisionTreeClassifier with the specified parameters (max\_depth=10, min\_samples\_split=2, min\_samples\_leaf=1) and using sentiment scores as an additional feature, along with TF-IDF, has resulted in a reasonably good performance.

The precision, recall, and F1-score values are quite high overall.

The model has a weighted average precision of 0.9484, a recall of 0.9527, and an F1-score of 0.9497.

The confusion matrix also shows a strong performance: there are 103 true positives and 8 false negatives for class 0, and 2 false positives and 87 true negatives for class 1.

These metrics suggest that the model is good at correctly predicting the classes, with a slight tendency to falsely classify instances of class 0 as class 1 (as evidenced by the 8 false negatives).

If we consider all metrics (precision, recall, F1-score, and confusion matrix) across all models, the ranking would be as follows:

1. **Random Forest Classifier (without and with sentiment scores as an additional feature)**: The Random Forest models both have the highest precision, recall, and F1-score, indicating strong overall performance. Furthermore, their confusion matrices show fewer misclassifications compared to the Decision Tree models.
2. **Decision Tree Classifier with sentiment scores as an additional feature and specific parameters (max\_depth=10, min\_samples\_split=2, min\_samples\_leaf=1)**: Although this model's metrics are slightly lower than the Random Forest models, it still shows better performance than the other two Decision Tree models. Its confusion matrix also shows fewer misclassifications than the simpler Decision Tree models.
3. **Decision Tree Classifier with sentiment scores as an additional feature**: This model shows a modest decrease in performance metrics compared to the other two models. The confusion matrix indicates a higher number of misclassifications compared to the Random Forest models and the tuned Decision Tree model.
4. **Decision Tree Classifier without sentiment scores as an additional feature**: This model has the lowest precision, recall, and F1-score of all models, indicating the weakest overall performance. Its confusion matrix shows the highest number of misclassifications.

When I trained the Random Forest model, I didn't observe any improvement when adding sentiment scores as an additional feature. Two key factors might explain why this happened:

Firstly, the complexity of my model plays a role. Random Forest is an ensemble model and inherently more complex than a simple Decision Tree. It aggregates the results of multiple Decision Trees, each trained on a random subset of the data. This allows it to capture complex patterns and relationships between features, which a single Decision Tree might miss. As a result, the addition of sentiment scores, which might have a more noticeable effect in a simpler model, didn't significantly change the predictive power of my already complex Random Forest model.

Secondly, the importance of the added feature is crucial. In Random Forest, every feature gets an importance score that reflects how useful it is in making accurate predictions. This score is computed based on how much the tree nodes, which use that feature, reduce impurity across all trees in the forest. So, if the sentiment scores had a low feature importance score, it's likely that they didn't contribute much to the final prediction, and thus, didn't improve the performance of my model.

In my scenario, when training the Random Forest model, both TF and TF-IDF resulted in similar model performance. This may suggest that in this specific context, less frequent words (which would be given more weight in TF-IDF) did not provide additional predictive value over simply using term frequencies (TF).

It could also be due to the inherent characteristics of the Random Forest model. Random Forests are good at handling a large variety of features and can automatically select the most important features for making predictions. Therefore, the model might have been able to pick out the most informative features regardless of whether TF or TF-IDF was used.

Indeed, my experience with the Random Forest model confirms its robustness. Here's why:

Firstly, thanks to bagging, where I create multiple subsets of the original data and train a Decision Tree on each, my model becomes more stable and less likely to overfit.

Secondly, the model's use of feature randomness - selecting a random subset of features at each candidate split - adds an extra layer of diversity to my model, making it more resilient to noise and outliers in the data.

Thirdly, the model's ability to handle unbalanced data is quite helpful in many scenarios. I can deal with unbalanced datasets by balancing the error function or by class weighting.

Given these characteristics, I find Random Forest to be a powerful tool for my machine learning tasks.

Looking at all these models and their metrics, the one that stands out as the best performer is the Random Forest model (with or without sentiment scores as an additional feature). It consistently delivers high precision, recall, and F1 scores across both classes.

Just to elaborate a bit:

1. **Random Forest (with or without sentiment scores)**: Both precision and recall are well balanced and high (around 0.96), meaning the model is correctly identifying positive and negative cases while minimizing both false positives and false negatives.
2. **Decision Tree (and with sentiment scores as additional feature)**: This model performs almost as well as the Random Forest model with precision, recall, and F1-scores all being around 0.94.
3. **Naïve Bayes**: This model's performance is decent with precision, recall, and F1-scores all around 0.72, but not as good as Random Forest or Decision Tree.
4. **KNN (N=5, N=10, N=15)**: These models performed the worst among all the models. The KNN with N=5 has an unbalanced precision and recall, heavily favoring one class. For N=10 and N=15, the performance dropped further as it couldn't predict the negative class at all.

So, based on these metrics, I would choose the Random Forest model for this task as it gives the best overall performance.

# KNN

KNN N-10 Classification Report & Confusion Matrix

TF-IDF

|  |  |  |  |
| --- | --- | --- | --- |
| precision | recall | f1-score | support |
| 0.555 | 1 | 0.71382637 | 111 |
| 0 | 0 | 0 | 89 |
| 0.555 | 0.555 | 0.555 | 0.555 |
| 0.2775 | 0.5 | 0.35691318 | 200 |
| 0.308025 | 0.555 | 0.39617363 | 200 |

|  |  |
| --- | --- |
| 0 | 1 |
| 111 | 0 |
| 89 | 0 |

KNN N-15 Classification Report & Confusion Matrix

|  |  |
| --- | --- |
| 0 | 1 |
| 111 | 0 |
| 89 | 0 |

TF-IDF

|  |  |  |  |
| --- | --- | --- | --- |
| precision | recall | f1-score | support |
| 0.555 | 1 | 0.71382637 | 111 |
| 0 | 0 | 0 | 89 |
| 0.555 | 0.555 | 0.555 | 0.555 |
| 0.2775 | 0.5 | 0.35691318 | 200 |
| 0.308025 | 0.555 | 0.39617363 | 200 |

Based on the provided classification reports and confusion matrices, it appears that the KNN model with N = 5 performed the best among the three models. Here are the evaluation metrics for KNN N=5:

While KNN N=5 achieved the highest precision, recall, and F1-score for the positive class (support: 111), it's important to note that the performance for the negative class is relatively low with a precision, recall, and F1-score of 1.0, 0.01123596, and 0.02222222, respectively.

# Conclusion

Based on my extensive exploration of different machine learning models and evaluation metrics, I have gained valuable insights into the classification of research papers related to COVID-19. The goal was to identify the best model for accurately categorizing these papers.

I started by preprocessing the data, which involved cleaning the text and selecting relevant features such as TF, TF-IDF, and sentiment scores. I then trained and evaluated various models, including KNN, Naïve Bayes, Decision Tree, and Random Forest, while experimenting with different parameters and feature combinations.

After a thorough analysis, it is clear that the Random Forest model outperformed the others in terms of precision, recall, and F1-scores. It consistently demonstrated a balanced performance, correctly identifying both positive and negative cases while minimizing false positives and false negatives. This makes it the most reliable and accurate model for the task at hand.

The Decision Tree model, particularly when including sentiment scores as an additional feature, showed promising results as well. It achieved similar performance metrics to the Random Forest model, indicating its potential as an alternative choice.

The Naïve Bayes model, while decent, did not match the performance of the Random Forest or Decision Tree models. It achieved lower precision, recall, and F1-scores, suggesting limited accuracy in classifying the research papers.

On the other hand, the KNN models with different values of N (5, 10, 15) performed poorly, with unbalanced precision and recall and a significant inability to predict the negative class accurately.

In conclusion, the Random Forest model is the optimal choice for accurately classifying research papers related to COVID-19. Its robust performance, balanced precision and recall, and ability to handle complex relationships among features make it the most reliable model for this specific task. This study highlights the importance of comprehensive model evaluation and the impact of different features and algorithms in achieving accurate classifications.

Moving forward, the findings from this study can be applied to automate the categorization of research papers, enabling researchers and healthcare professionals to efficiently analyze and extract relevant information from vast amounts of scientific literature in the context of COVID-19.

# GitHub

The GitHub link is included in the report to allow others to replicate and build upon my work independently. By sharing the code and resources, it enables researchers and developers to reproduce the results and further advance the research in the field of CORD-19 classification. This fosters collaboration and contributes to the overall progress of scientific knowledge.

[https://github.com/kevinmastascusa/CORD\_19\_Research]

The GitHub link provided in the report focuses on sharing the code and resources related to the CORD-19 research paper classification project. Due to restrictions on file size imposed by GitHub, the actual data used in the project cannot be uploaded directly to the repository. However, the code and documentation provided in the repository offer a clear overview of the methodology and steps involved, allowing others to understand and replicate the project using their own data.

# Python

I am including this brief overview of the main steps in my report because the Python script (main.py) is too long to include in its entirety. However, these steps provide a summary of the key processes involved in the script.

1. Import the necessary libraries and modules for data processing and machine learning.
2. Define a preprocessing function to handle tasks such as lowercase conversion, punctuation removal, and tokenization.
3. Load the dataset from a CSV file into a pandas DataFrame.
4. Preprocess the text data in the DataFrame using the defined preprocessing function.
5. Split the dataset into training and testing sets.
6. Apply a text vectorization technique (such as TF-IDF or CountVectorizer) to convert the text data into numerical features.
7. Initialize and train a machine learning model (such as Random Forest or Naive Bayes) using the training data.
8. Evaluate the performance of the trained model using various metrics, including accuracy, precision, recall, and F1-score, on the testing data.
9. Save the trained model to a file for future use.
10. Use the trained model to make predictions on new, unseen data.
11. Display the results of the predictions.

# Endnote

In my project, I used a fixed train-test split with a test size of 0.2 to evaluate the performance of my classification models. However, if I were to repeat this project in the future, I would consider testing the effects of different training and testing splits. By varying the split ratios, such as 70% training and 30% testing or 90% training and 10% testing, I could explore how the distribution of data between the training and testing sets affects the model's performance. This analysis would provide insights into the optimal split ratio that yields the best results for my specific classification task.

In retrospect, if I were to redo this project, I would have experimented with different methods for selecting the optimal K value in the K-nearest neighbors (KNN) algorithm. While I used a fixed set of K values in my initial experiment, such as K=5, K=10, and K=15, I now realize that there are more sophisticated approaches available for determining the best K value. Techniques like cross-validation, grid search, or even automated algorithms like the elbow method or silhouette analysis could have been utilized to find the most suitable K value for my specific dataset and classification problem. By exploring these alternative methods, I could have potentially identified an even better K value that improves the performance of the KNN model.

While macro-average precision, recall, and F1-score provide an overall performance measure, they may not capture the nuances and imbalances within each class. Therefore, I would consider using other evaluation techniques such as micro-average, weighted-average, or stratified sampling to ensure a more comprehensive assessment of the models' effectiveness.

In my project, I did not specifically examine the feature importance scores, but it is something that I would consider as an important aspect for future analysis. By assessing the feature importance scores, I would be able to gain insights into the variables that have the most significant impact on the classification outcomes. This analysis would provide a better understanding of the key factors driving the predictions and allow for more informed decision-making. Examining feature importance would be a valuable addition to my project, providing further insights into the relevance and influence of different variables in the classification models.

In retrospect, I would have further explored the interpretation of the models, delving into specific decision rules and feature interactions to gain more granular insights. Additionally, I would have conducted a more detailed analysis of the sentiment scores, examining their relationship with the classification outcomes and exploring potential correlations or patterns.

In my project, I would have also focused on hyperparameter tuning for the Random Forest classifier to optimize its performance. By experimenting with different hyperparameters such as the number of trees in the forest, maximum depth of the trees, and minimum number of samples required to split a node, I could have fine-tuned the model to achieve even better classification results. Hyperparameter tuning plays a crucial role in optimizing the model's ability to capture complex patterns and generalize well to unseen data. Therefore, incorporating this step would have allowed me to unlock the full potential of the Random Forest classifier for classifying CORD-19 research papers.

In my project, I encountered challenges when attempting to load the dataset into WEKA for analysis. As a result, I had to switch to Python as my primary programming language for data processing and modeling. While WEKA is a powerful tool for data analysis, its limitations in handling large or complex datasets can arise. Despite this obstacle, Python provided a versatile and flexible environment for data manipulation, preprocessing, and model development. The extensive libraries and frameworks available in Python allowed me to overcome the dataset loading issues and continue with my analysis effectively. This adaptation demonstrates the importance of being adaptable and resourceful in addressing technical challenges during the research process.

Working with large and complex datasets can often pose challenges, even for commonly used tools like Excel. In my project, I encountered difficulties when attempting to load the dataset into Excel due to its size and complexity. The loading process was time-consuming and cumbersome, causing delays in data exploration and analysis. As a result, I had to seek alternative solutions and turn to more specialized tools and programming languages like Python to handle the data effectively.

In my project, I found the Natural Language Toolkit (NLTK) to be an invaluable resource for my text analysis tasks. NLTK provided a wide range of tools and functionalities that greatly facilitated my text processing and sentiment analysis workflows. I was able to leverage NLTK's pre-trained models, such as the SentimentIntensityAnalyzer, to easily compute sentiment scores for the research paper titles. The library also offered various text preprocessing capabilities, such as tokenization, stemming, and stop word removal, which allowed me to clean and prepare the text data efficiently. Overall, NLTK proved to be a powerful and user-friendly tool that enhanced the accuracy and efficiency of my text analysis tasks.

To improve the structure of my main script, I would break it down into different files, each dedicated to a specific stage or task. This modular approach would enhance code organization, readability, and reusability. I would have separate files for data preprocessing, feature extraction, model training, evaluation, and any other relevant steps. This way, I could easily manage and update individual components without affecting the entire script, leading to better code maintenance and scalability.