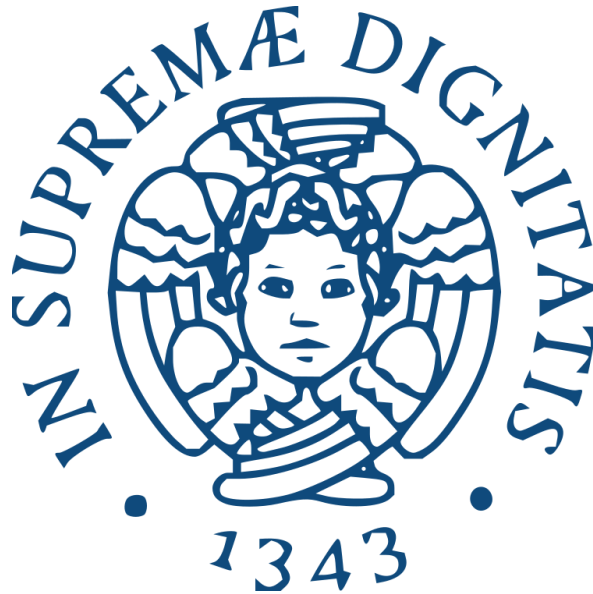


Mirko Di Lucia

Francesco Marcelloni

AIDE

26/05/2024



University of Pisa

Laurea Magistrale (MSc) in Artificial Intelligence and Data Engineering

Project

Data Mining and Machine Learning

Articles Categorizer

A python application to categorize articles from open news site

Introduction

The scope of this project is to develop an application with the following capabilities:

- Collect news articles from the renowned Italian online newspaper, <https://open.online>.
- Build a comprehensive dataset from the collected articles.
- Train various classifier algorithms using this dataset.
- Evaluate each model to identify the most accurate machine learning algorithm for the application's requirements.
- Use cross-validation to discover overfitted models

Dataset

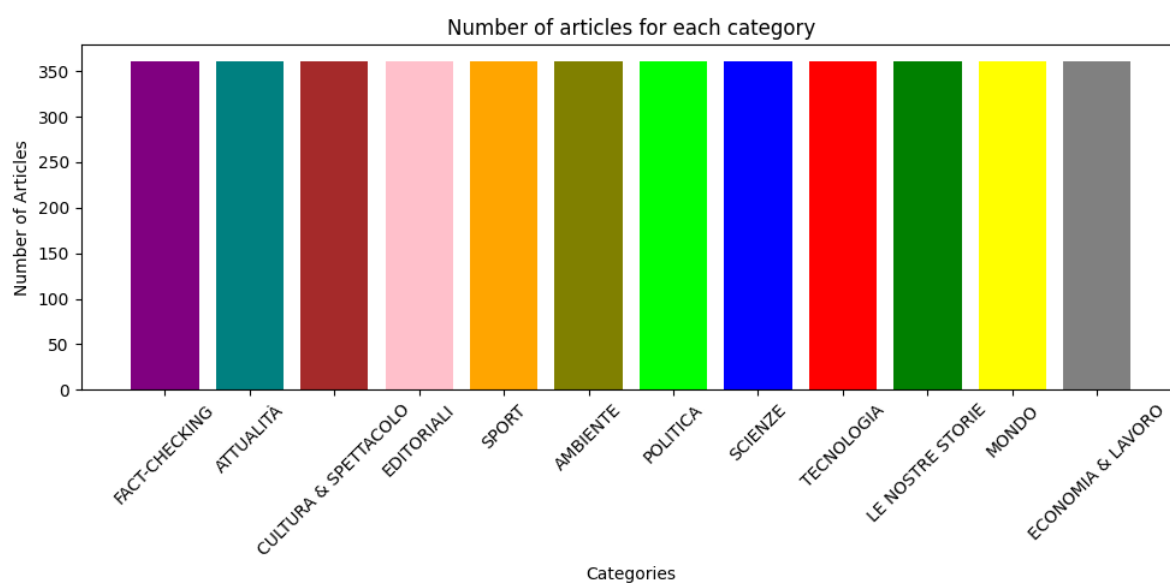
The dataset consists of a collection of folders, each labeled with a category name. Each folder contains text files with news articles collected from the news website.

From the news website, we can collect articles from various categories:

- Ambiente
- Attualità
- Cultura e SpettacoloEconomia e Lavoro
- Editoriali
- Fact-Checking
- Le nostre storie
- Mondo
- Politica
- Scienze
- Sport
- Tecnologia

We have a total of 4332 articles divided in each category, in details we have

- ATTUALITÀ: 361
- CULTURA & SPETTACOLO: 361
- SPORT: 361
- FACT-CHECKING: 361
- EDITORIAL: 361
- LE NOSTRE STORIE: 361
- AMBIENTE': 361
- POLITICA: 361
- SCIENZE: 361
- TECNOLOGIA': 361
- MONDO': 361
- ECONOMIA & LAVORO: 361



After the collections process using a custom crawler, we identify the most commons words for each news category. Using the preprocess script, we can obtain the data that we give to every model for training.

These are the results obtained from the process

- FACT-CHECKING: (stat, 1408), (articol, 919), (facebook, 907), (vide, 879), (qui, 848), (legg, 823), (post, 768), (pubblic, 712), (vaccin, 672), (fals, 618),
- ATTUALITÀ: (stat, 950), (cas, 396), (prim, 386), (poi, 355), (pi, 320), (fatt, 286), (trov, 284), (anni, 266), (person, 265), (due, 259),
- CULTURA & SPETTACOLO: (stat, 592), (pi, 527), (cos, 447), (prim, 375), (anni, 338), (rai, 323), (due, 321), (pubblic, 320), (poi, 319), (perc, 317)
- EDITORIALI: (pi, 518), (prim, 508), (salvin, 495), (part, 489), (cos, 482), (fatt, 460), (stat, 451), (polit, 433), (govern, 421), (ital, 337)
- SPORT: (stat, 603), (atlet, 490), (parig, 414), (final, 394), (dop, 384), (part, 365), (prim, 349), (olimpiad, 332), (pi, 331), (gar, 297)
- AMBIENTE : (pi, 1181), (stat, 805), (climat, 533), (paes, 446), (camb, 431), (ital, 409), (nuov, 407), (esser, 376), (europe, 374), (second, 368)
- POLITICA: (stat, 770), (part, 696), (ital, 599), (pi, 591), (polit, 463), (govern, 422), (prim, 391), (president, 380), (melon, 361), (sol, 325)
- SCIENZE: (pi, 1005), (stat, 940), (ricerc, 615), (stud, 593), (vaccin, 588), (prim, 581), (esser, 504), (cas, 489), (nuov, 464), (sol, 448)
- TECNOLOGIA: (pi, 628), (stat, 598), (utent, 371), (esser, 342), (dat, 341), (nuov, 340), (piattaform, 327), (prim, 320), (propr, 314), (artificial, 304)
- LE NOSTRE STORIE: (pi, 1766), (stat, 1365), (lavor, 1181), (cos, 1043), (prim, 967), (anni, 860), (cas, 813), (sol, 773), (perc, 772), (far, 749)
- MONDO: (stat, 1124), (pi, 461), (israel, 448), (prim, 439), (russ, 403), (part, 393), (second, 381), (president, 341), (harris, 337), (dop, 331)
- ECONOMIA & LAVORO: (pi, 960), (lavor, 754), (stat, 639), (eur, 618), (ital, 583), (anni, 462), (europe, 454), (part, 445), (prim, 432), (nuov, 418)

Application Structure

The application is structured as a Python monorepo and includes two distinct applications along with a shared folder containing common utility functions. This structure allows for easy expansion, such as adding a RESTful API or a CLI.

1. **Classifier:** This application comprises the core model used for training and evaluating machine learning models.
2. **Scraper:** This application consists of a web scraper capable of collecting news from the online newspaper.

Crawler

The scraper includes scripted commands designed to perform a thorough search of the newspaper's website using two primary libraries:

- **Selenium:** This library is utilized for navigating and searching within HTML pages. Selenium provides a robust framework for automating web browser interactions, allowing the scraper to dynamically access and extract content from web pages.
- **Chrome WebDriver:** This library enables the control of the Google Chrome browser, allowing the scraper to interact with web pages in a human-like manner. It can perform various actions such as clicking buttons, accepting cookies, and executing scripts. This includes handling AJAX calls, ensuring that the scraper can interact with and retrieve content from dynamically loaded web pages.

By leveraging these libraries, the scraper can efficiently collect comprehensive and up-to-date news articles from the newspaper's website, simulating a user's interaction with the site to ensure all relevant content is captured.

The crawler can accept cookie and open the categories menu to retrieve all categories from the html tags

Every category is stored in a json file format with its link.

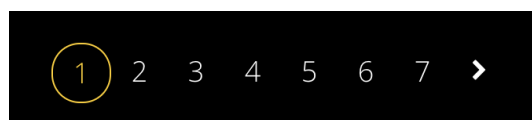


The menu of the news website that contains all the categories

Starting from the list saved in before step the crawler opens every category and start to scrape all articles title, the crawler navigates between pages. Same as before step the crawler saves each article title, category and link in another json file, to use it in the next step





The category page that contains all the news related to the category



Pagination menu present in every category page

For each article saved in the previous step the scraper opens every single article page and collects title, and the article contents. Every article is stored in a text file in the


correct folder.




AMBIENTE ECONOMIA CIRCOLARE • EUTOPIA • MODA • PARLAMENTO EUROPEO • POLITICHE AMBIENTALI • UNIONE EUROPEA

Ecodesign e «diritto alla riparazione», via libera del Parlamento Ue alla stretta su fast fashion e consumo di massa

23 APRILE 2024 - 13:35 di Gianluca Brambilla



Luce verde dell'Eurocamera a due provvedimenti che puntano a garantire la sostenibilità ambientale dei prodotti. Ecco cosa prevedono



Da Strasburgo – Da un lato, nuove regole per rendere i prodotti più sostenibili fin dalla fase di progettazione. Dall'altro, garantire ai cittadini un vero e proprio «diritto alla riparazione». Oggi il

Page of an article

Classifier

Here's an expanded version of the text about the classifier:

The classifier is a Python application capable of performing the following tasks:

1. **Loading Articles:** It loads all the news articles from the specified source.
2. **Data Cleaning:** It cleans the data using a stemmer to preprocess the text, reducing words to their base or root form.
3. **Dataset Construction:** It builds the dataset by organizing the articles into different folders, with each folder named after the category of the articles it contains.
4. **Data Splitting:** It splits the dataset into training and testing sets to ensure the models can be properly trained and evaluated.
5. **Model Training:** It trains various classifier algorithms using the same dataset to ensure consistency in the training process.
6. **Model Evaluation:** It evaluates the models to determine their performance and accuracy.
7. **Cross Validation:** To find if the models are overtrained against the dataset.

By performing these tasks, the classifier ensures a systematic and efficient approach to training and evaluating different machine learning models for categorizing news articles.

Preprocessing

Text Cleaning:

- Convert all text to lowercase to maintain consistency.
- Remove any non-alphanumeric characters (such as punctuation) to reduce noise in the data.
- Remove stopwords (common words like "e", "o", "il") that do not carry significant meaning for classification tasks.

Stemming:

- Stemming is the process of reducing words to their root form. For example, "running" and "runner" are both reduced to "run".
- Use the nltk library's PorterStemmer for stemming.

Build the Feature matrix:

This process consist on building a feature matrix using both CountVectorizer and TfidfTransformer involves transforming text data into a numerical format that machine learning algorithms can understand. Here's an explanation of the code and the role of both CountVectorizer and TfidfTransformer:

Count Vectorization (Bag of Words Representation)

The first step is using CountVectorizer, which is a method of converting a collection of text documents into a matrix of token counts. This is known as the Bag of Words model, where each document is represented by a vector of word counts, ignoring grammar and word order.

`CountVectorizer(analyzer=preprocess, min_df=2, max_df=100000000):`

The CountVectorizer tokenizes the text, builds a vocabulary of known words, and encodes new documents using this vocabulary.

- `analyzer=preprocess` means that a custom preprocessing function is being used to analyze the text before counting the tokens.
- `min_df=2` ensures that words that appear in fewer than 2 documents are ignored, as they may be too rare to be useful.
- `max_df=100000000` sets a very high upper limit, meaning no words are excluded based on appearing in too many documents (often used to exclude common stopwords).
- `train_counts = count_vect.fit_transform(train_set.data):`

This fits the CountVectorizer to the training dataset (`train_set.data`), which learns the vocabulary from the training data.

It then transforms the text data into a sparse matrix where each row represents a document, and each column represents a word from the vocabulary. The values in the matrix represent the number of times each word appears in a document.

TF-IDF Transformation (Term Frequency-Inverse Document Frequency)

The next step is to apply a TF-IDF transformation to the document-term matrix. While the Count Vectorizer represents the raw counts of words, TF-IDF adjusts these counts by the importance of words in the document set.

`tfidf_transformer = TfidfTransformer():`

TfidfTransformer takes the output of CountVectorizer (i.e., the raw word counts) and transforms it into TF-IDF values.

TF-IDF measures how important a word is in a document relative to the rest of the documents. It reduces the influence of very common words (like "the" or "is") and increases the importance of rare but meaningful words.

```
train_tfidf = tfidf_transformer.fit_transform(train_counts):
```

This fits the TF-IDF transformer to the document-term matrix (`train_counts`) and transforms the raw counts into TF-IDF values.

The resulting `train_tfidf` is a TF-IDF weighted document-term matrix where each value represents the importance of a word in the document relative to the entire dataset.

Transforming the Test Set

```
test_counts = count_vect.transform(test_set.data):
```

The `CountVectorizer` that was trained on the training data is used to transform the test data into a document-term matrix. This ensures that the test data is represented using the same vocabulary learned from the training data (i.e., no new words are added).

```
test_tfidf = tfidf_transformer.transform(test_counts):
```

Similarly, the `TfidfTransformer` is applied to the test data to convert the word counts into TF-IDF scores, ensuring that the test set uses the same transformation as the training set.

Train the models

The models are built by executing the script `classifier_building.py`, which creates the following classifiers:

- Multinomial Naïve Bayes classifier
- Decision Tree classifier
- Random Forest classifier
- K-Neighbors classifier
- For the K-Neighbors classifier, the optimal value of `k` is determined by testing the classifier on the test set with `k` values ranging from 2 to 100. The `k` that results in the best accuracy is selected (tests have shown that the optimal parameter is `k=3`).

After selecting the best `k`, all classifiers are fitted on the training set and then stored in permanent file.

Evaluation

The script `evaluator.py` evaluates different classifiers using a test dataset and saves the results into files. The function `evaluate` is designed to handle this process given a classifier by:

1. Making predictions on the test data.
2. Calculating accuracy.
3. Generating a detailed classification report and confusion matrix.
4. Printing and saving the evaluation results to a file.

By using this function, you can systematically assess the performance of different classifiers and store the results for further analysis or reporting.

Results

Here we report the results of the four classifiers: Decision Tree, k-Nearest Neighbors (kNN), Multinomial Naive Bayes, and Random Forest, based on their accuracy and performance metrics on the same dataset.

Decision Tree Classifier

Accuracy: 77.07%

Precision, Recall, F1-Score: The classifier showed reasonable performance for most classes, but some categories (like "ECONOMIA & LAVORO" and "MONDO") struggled with recall and F1-score, indicating that the model had difficulty correctly identifying all instances of these classes.

Accuracy on tests set:

0.7707317073170732

Metrics per class on tests set:

	precision	recall	f1-score	support
AMBIENTE	0.74	0.76	0.75	70
ATTUALITÀ	0.96	0.96	0.96	68
CULTURA & SPETTACOLO	0.96	0.97	0.97	72
ECONOMIA & LAVORO	0.64	0.62	0.63	55
MONDO	0.61	0.63	0.62	84
POLITICA	0.83	0.89	0.86	64
SCIENZE	0.72	0.72	0.72	64
SPORT	0.83	0.75	0.79	67
TECNOLOGIA	0.67	0.65	0.66	71
accuracy			0.77	615
macro avg	0.77	0.77	0.77	615
weighted avg	0.77	0.77	0.77	615

Confusion matrix:

	amb	att	c&s	eco	mon	pol	sci	spo	tec
amb	[53	0	0	3	2	1	5	2	4]
att	[1	65	0	0	0	1	0	0	1]
c&s	[0	0	70	0	0	0	0	0	2]
eco	[3	0	1	34	7	1	3	0	6]
mon	[7	1	0	2	53	7	4	5	5]
pol	[0	1	0	2	2	57	1	1	0]
sci	[6	1	0	1	5	0	46	2	3]
spo	[1	0	0	2	8	2	2	50	2]
tec	[1	0	2	9	10	0	3	0	46]

Confusion Matrix: The decision tree struggled particularly with misclassifications between related categories. For example, the "ECONOMIA & LAVORO" class had multiple misclassifications across other categories, suggesting that the tree might have overfitted on certain splits of the data.

Random Forest Classifier

Accuracy: 90.57%

Precision, Recall, F1-Score: Random Forest performed the best overall, with nearly all categories showing high precision, recall, and F1-scores. The improvement in performance across classes can be attributed to the robustness of random forests, which average out the predictions from multiple decision trees and mitigate overfitting.

```

Accuracy on tests set:
0.9056910569105691
Metrics per class on tests set:

```

	precision	recall	f1-score	support
AMBIENTE	0.86	0.90	0.88	70
ATTUALITÀ	0.92	0.96	0.94	68
CULTURA & SPETTACOLO	0.99	1.00	0.99	72
ECONOMIA & LAVORO	0.79	0.91	0.85	55
MONDO	0.94	0.79	0.86	84
POLITICA	0.91	0.92	0.91	64
SCIENZE	0.88	0.88	0.88	64
SPORT	0.94	0.97	0.96	67
TECNOLOGIA	0.91	0.86	0.88	71
accuracy			0.91	615
macro avg	0.90	0.91	0.90	615
weighted avg	0.91	0.91	0.91	615

```

Confusion matrix:

```

	amb	att	c&s	eco	mon	pol	sci	spo	tec
amb	[63	0	0	2	1	0	3	0	1]
att	[0	65	0	0	1	1	1	0	0]
c&s	[0	0	72	0	0	0	0	0	0]
eco	[2	2	0	50	0	1	0	0	0]
mon	[2	4	0	1	66	4	1	3	3]
pol	[0	0	0	5	0	59	0	0	0]
sci	[6	0	0	0	0	0	56	1	1]
spo	[0	0	1	0	0	0	0	65	1]
tec	[0	0	0	5	2	0	3	0	61]

Confusion Matrix: Random forest had fewer misclassifications than any other model, with most errors being minor and occurring in classes with overlapping content. For example, "ECONOMIA & LAVORO" had some confusion with other categories, but this was less significant than in the other models.

k-Nearest Neighbors (kNN) Classifier:

Accuracy: 86.67%

Precision, Recall, F1-Score: kNN showed a significant improvement over the decision tree, particularly in terms of overall accuracy and recall for most classes. For example, "TECNOLOGIA" and "SPORT" showed high recall, meaning that most instances of these classes were correctly classified.

Accuracy on tests set:					
0.8666666666666667					
Metrics per class on tests set:					
	precision	recall	f1-score	support	
AMBIENTE	0.79	0.87	0.83	70	
ATTUALITÀ	0.86	0.84	0.85	68	
CULTURA & SPETTACOLO	0.94	1.00	0.97	72	
ECONOMIA & LAVORO	0.76	0.82	0.79	55	
MONDO	0.87	0.79	0.82	84	
POLITICA	0.84	0.91	0.87	64	
SCIENZE	0.86	0.84	0.85	64	
SPORT	0.93	0.93	0.93	67	
TECNOLOGIA	0.95	0.82	0.88	71	
accuracy			0.87	615	
macro avg	0.87	0.87	0.87	615	
weighted avg	0.87	0.87	0.87	615	

Confusion matrix:										
	amb	att	c&s	eco	mon	pol	sci	spo	tec	
amb	[61	1	0	3	0	1	4	0	0]	
att	[2	57	4	1	2	1	1	0	0]	
c&s	[0	0	72	0	0	0	0	0	0]	
eco	[4	0	0	45	0	4	1	0	1]	
mon	[3	5	1	1	66	3	2	3	0]	
pol	[1	2	0	1	2	58	0	0	0]	
sci	[4	1	0	2	1	0	54	1	1]	
spo	[2	0	0	1	1	0	0	62	1]	
tec	[0	0	0	5	4	2	1	1	58]	

Confusion Matrix: There were still some misclassifications, particularly in overlapping or similar categories, such as between "MONDO" and "POLITICA." However, kNN performed better across most categories compared to the decision tree

Multinomial Naive Bayes Classifier:

Accuracy: 89.92%

Precision, Recall, F1-Score: Naive Bayes performed consistently well, with most categories achieving high precision and recall scores. Notably, categories like "CULTURA & SPETTACOLO" and "TECNOLOGIA" achieved near-perfect results. This performance is indicative of Naive Bayes' ability to handle high-dimensional data well (such as text data after TF-IDF transformation).

Accuracy on tests set:

0.8991869918699187

Metrics per class on tests set:

	precision	recall	f1-score	support
AMBIENTE	0.86	0.91	0.89	70
ATTUALITÀ	0.90	0.91	0.91	68
CULTURA & SPETTACOLO	0.95	1.00	0.97	72
ECONOMIA & LAVORO	0.77	0.91	0.83	55
MONDO	0.97	0.76	0.85	84
POLITICA	0.83	0.89	0.86	64
SCIENZE	0.91	0.91	0.91	64
SPORT	0.94	0.96	0.95	67
TECNOLOGIA	0.97	0.87	0.92	71
accuracy			0.90	615
macro avg	0.90	0.90	0.90	615
weighted avg	0.90	0.90	0.90	615

Confusion matrix:

	amb	att	c&s	eco	mon	pol	sci	spo	tec
amb	[64	0	1	1	0	0	3	0	1]
att	[0	62	1	2	1	1	1	0	0]
c&s	[0	0	72	0	0	0	0	0	0]
eco	[2	0	0	50	0	3	0	0	0]
mon	[2	7	0	2	64	6	0	3	0]
pol	[1	0	0	6	0	57	0	0	0]
sci	[5	0	0	0	0	0	58	0	1]
spo	[0	0	2	0	1	0	0	64	0]
tec	[0	0	0	4	0	2	2	1	62]

Confusion Matrix: Misclassifications were few and tended to occur in cases with overlapping topics (e.g., "MONDO" being confused with "POLITICA"). However, these errors were less frequent than in kNN and decision tree results

Cross Validation

By utilizing k-fold cross-validation, we mitigate the risk of the model being overly sensitive to a particular train-test split, leading to a more accurate and generalizable evaluation of the model's capabilities. In this analysis, various classifiers, including Decision Tree, Random Forest, k-Nearest Neighbors (kNN), and Naive Bayes, were subjected to cross-validation to assess their performance.

Decision Tree Results:

The decision tree classifier achieved the following cross-validation scores across 10 folds:

- Cross-Validation Scores: [0.79545455, 0.78030303, 0.77272727, 0.83333333, 0.76425856, 0.74144487, 0.75285171, 0.82509506, 0.79087452, 0.78326996]
- The accuracy across folds ranged from 74.1% to 83.3%, indicating moderate variability in performance depending on the specific split of the data. The decision tree classifier exhibited an average accuracy of:
- Mean Accuracy: 78.40%

This suggests that while the decision tree provides a reasonable level of accuracy, it may struggle with generalization compared to more advanced models due to its tendency to overfit, especially on smaller datasets.

Random Forest Results:

The random forest classifier demonstrated consistently strong performance across the 10 cross-validation folds:

- Cross-Validation Scores: [0.90151515, 0.92045455, 0.92045455, 0.92045455, 0.91254753, 0.87452471, 0.88973384, 0.92775665, 0.88212928, 0.90874525]
- The accuracy ranged from 87.4% to 92.7%, showcasing the robustness of random forests, which benefit from reducing overfitting by averaging the results of multiple decision trees. The mean accuracy across all folds was:
- Mean Accuracy: 90.58%

This strong performance indicates that random forests generalize well across different data splits and are a reliable choice for classification tasks.

k-Nearest Neighbors (kNN) Results:

The kNN classifier also performed well, with the following cross-validation scores:

- Cross-Validation Scores: [0.85606061, 0.82575758, 0.86742424, 0.84848485, 0.87452471, 0.79467681, 0.84410646, 0.84790875, 0.79847909, 0.8365019]
- Accuracy for kNN varied between 79.4% and 87.4% across the folds, with the classifier achieving a respectable average accuracy of:
- Mean Accuracy: 83.94%

Although kNN performed well, its accuracy was slightly lower and more variable compared to the random forest, likely due to its sensitivity to the choice of k and its dependence on local relationships between data points.

Naive Bayes Results:

The Naive Bayes classifier performed consistently across the cross-validation folds, with scores as follows:

- Cross-Validation Scores: [0.90151515, 0.89772727, 0.90151515, 0.89015152, 0.91634981, 0.86311787, 0.87072243, 0.92395437, 0.88212928, 0.8973384]
- Naive Bayes maintained high accuracy, ranging from 86.3% to 92.3%, which is impressive given the simplicity of the model. The average accuracy across all folds was:
- Mean Accuracy: 89.45%

This result highlights that Naive Bayes is particularly effective when dealing with text data, as it often assumes feature independence, which can work well for high-dimensional datasets, such as those generated by TF-IDF vectorization.

Decision Tree: The decision tree classifier is relatively simple and can overfit smaller datasets or datasets with more complex relationships. Its average accuracy (75.8%) is lower than the other classifiers.

Random Forest: Random forest significantly outperforms the decision tree with a mean accuracy of 78.40%. Random forests are more robust and reduce the risk of overfitting by averaging multiple decision trees.

kNN: The kNN classifier has a mean accuracy of 83.94%, which is good but not as high as random forest or Naive Bayes. kNN's performance can be sensitive to how data is scaled and the choice of k.

Naive Bayes: Naive Bayes achieved the highest mean accuracy (89.45%), which indicates that it's a good fit for this dataset. Naive Bayes often works well for text classification problems (likely the case with TF-IDF features), especially when the feature independence assumption holds reasonably well.

Final Thoughts:

Naive Bayes and Random Forest perform the best on this dataset. If interpretability is important, a decision tree could still be useful, but its lower accuracy suggests that you might want to prioritize models like Random Forest or Naive Bayes.

Can be useful to fine-tune the hyperparameters of these models or experiment with feature selection to further improve performance.