

kelly casey assignment

by Kelly Casey

Submission date: 24-Apr-2026 11:57AM (UTC+0100)

Submission ID: 280855689

File name: 31343_Kelly_Casey_kelly_casey_assignment_726057_204012777.docx (994.35K)

Word count: 4323

Character count: 27120



TEXT CLASSIFICATION USING ML ALGORITHMS. APPLICATION OF MACHINE LEARNING - ACCA7025

Kelly Casey - 2106451



24 APRIL 2026

Contents

1. Abstract	2
2. Introduction.....	3
3. Literature Review	4
3.1 Sentiment Analysis and Text Classification	4
3.2 Traditional Machine Learning Algorithms.....	5
3.3 Comparison of Algorithms	6
4. Methodology	7
4.1 Dataset Selection.....	7
4.2 Data Pre-processing.....	7
4.3 Feature Extraction	8
4.4 Model Selection and Training	8
4.5 Model Evaluation	9
5. Discussion & Recommendation	10
5.1 Model Performance Evaluation	10
5.2 Comparison of Machine Learning Models	11
5.3 Limitations of the Approach	13
5.4 Recommendation	14
6. Conclusion	15
Bibliography	16
Appendix.....	17
Screenshots.....	17
YouTube link.....	22
Code	23

1. Abstract

This report explores the use of traditional machine learning algorithms for text classification in the context of sentiment analysis. A structured workflow was followed, including data preprocessing, feature extraction using Term Frequency-Inverse Document Frequency (TF-IDF), and model training and evaluation. Three commonly used algorithms:

- Naive Bayes,
- Support Vector Machine (SVM),
- and Decision Tree

were applied and compared to identify the most suitable option for a resource-constrained environment.

The models were evaluated using accuracy, precision, recall, F1-score, confusion matrices, and five-fold cross-validation. The results showed that Naive Bayes achieved a test accuracy of around 65% and performed consistently, with a cross-validation mean of approximately 69%. SVM produced similar results but did not clearly outperform Naive Bayes, while the Decision Tree model performed significantly worse, highlighting its limitations with high-dimensional text data.

Although the dataset used was designed for topic classification rather than sentiment analysis, the study still shows that traditional machine learning methods can be effective for text classification tasks. Based on the results, Naive Bayes is recommended as the most suitable algorithm due to its balance of performance, simplicity, and efficiency. Future work could involve applying the same workflow to a dataset specifically labelled for sentiment to further assess its effectiveness in real-world scenarios.

2.Introduction

Machine learning (ML) is a subfield of artificial intelligence that enables systems to learn patterns from data and make predictions without being explicitly programmed. Unlike traditional rule-based systems, machine learning models improve over time by learning from examples, which makes them useful for complex, data-driven tasks such as text analysis.

One important application of machine learning is sentiment analysis, which involves analysing text to determine the underlying opinion or emotion being expressed. This is widely used in areas such as customer reviews, social media, and user feedback, allowing organisations to better understand customer behaviour and preferences [2]. In most cases, sentiment analysis is treated as a classification problem, where text is categorised into groups such as positive, negative, or neutral sentiment [3].

Traditional machine learning approaches are still commonly used for sentiment analysis due to their efficiency and relatively low computational cost. Algorithms such as Naive Bayes, Support Vector Machines (SVM), K-Nearest Neighbours (KNN), and Decision Trees have been widely applied in text classification tasks. These methods can achieve strong performance while remaining lightweight and easier to implement compared to more complex models [4]. In particular, Naive Bayes is often valued for its simplicity and ability to scale effectively when working with large volumes of text data [5].

That said, different algorithms offer different strengths depending on the dataset and how the text is represented. For example, Support Vector Machines are often associated with high classification accuracy due to their ability to find optimal decision boundaries in high-dimensional spaces. In contrast, Naive Bayes tends to provide faster training and prediction times, making it more suitable for environments with limited computational resources [6].

For a marketing company with constrained resources, selecting the most appropriate algorithm requires balancing performance with efficiency. This report therefore focuses on evaluating traditional machine learning algorithms for sentiment analysis in this context. It will review relevant literature, outline a complete machine learning workflow, and provide a justified recommendation for the most suitable approach based on both theoretical understanding and practical considerations.

3.Literature Review

This section reviews existing research on sentiment analysis and the use of traditional machine learning algorithms for text classification. It explores key approaches discussed in the literature, including probabilistic, distance-based, and tree-based models, while examining their strengths and limitations. The aim is to provide a clear comparison of these methods in order to identify the most suitable approach for sentiment analysis, particularly within a resource-constrained environment.

3.1 Sentiment Analysis and Text Classification

Sentiment analysis is a widely studied area within natural language processing that focuses on identifying the underlying opinion or emotion expressed in textual data. It is commonly applied to customer reviews, social media posts, and online feedback to support decision-making in business environments. Early research by Pang et al. showed that machine learning approaches can significantly outperform traditional rule-based methods, particularly when working with large datasets [3].

As a classification task, sentiment analysis involves assigning text to predefined categories such as positive, negative, or neutral sentiment. Liu highlights that the growing volume of user-generated content has made sentiment analysis increasingly important, as manual analysis is no longer practical at scale [2]. This is where machine learning becomes particularly useful, as it can learn patterns from labelled data and apply this knowledge to new, unseen text.

Traditional machine learning algorithms are widely used in sentiment analysis because they can handle high-dimensional text data effectively. Before classification can happen, the text needs to be turned into a numerical format. This is usually done using methods like bag-of-words or term frequency, which convert words into structured data. Once the text is in this form, machine learning algorithms can process it and start identifying patterns that are useful for classification [4].

3.2 Traditional Machine Learning Algorithms

A range of traditional machine learning algorithms have been applied to sentiment analysis, with each offering its own strengths and limitations.

Naive Bayes is one of the most commonly used algorithms for text classification because it is simple and efficient. It is a probabilistic model based on Bayes' theorem and assumes that features are independent of each other. McCallum and Nigam showed that Naive Bayes performs well in document classification tasks, especially when used with feature extraction techniques like bag-of-words [5]. Even though the independence assumption is quite simplistic, the algorithm still works well in practice because text data is usually high-dimensional and sparse [4].

Support Vector Machines (SVM) are another widely used method for text classification. SVM works by finding an optimal boundary that separates data into different classes. Joachims found that SVM performs particularly well in high-dimensional spaces, which makes it well suited to text-based data [6]. Further studies have also shown that SVM can achieve high accuracy, especially when combined with techniques like TF-IDF to improve how features are represented [7].

The K-Nearest Neighbours (KNN) algorithm takes a different approach by classifying data based on similarity. It assigns a label to a data point based on the labels of its closest neighbours. Cover and Hart introduced this method and showed that it can be effective for classification tasks [8]. However, KNN does not build a model and instead stores all the training data, which can make it slow and computationally expensive when making predictions, especially with large datasets [9].

Decision Trees are often used because they are easy to understand and interpret. They work by splitting data into smaller groups based on feature values, creating a series of decision rules. While this makes them simple to use, they can sometimes overfit the data, particularly when the dataset is complex [4]. This means they may not perform as well on new, unseen data.

To address this, ensemble methods such as Random Forest have been developed. Random Forest, introduced by Breiman, combines multiple decision trees using a technique called bagging to improve performance [10]. By averaging the results from several trees, it reduces the risk of overfitting and generally improves accuracy. However, this also makes the model more complex and less easy to interpret compared to a single decision tree.

3.3 Comparison of Algorithms

The literature shows there is a clear trade-off between accuracy, computational efficiency, and interpretability when choosing a machine learning algorithm for sentiment analysis.

Algorithms such as SVM and Random Forest are often linked to higher accuracy because they can handle more complex patterns and reduce overfitting [6], [7], [10]. However, these methods usually require more computational power, especially during training, which may not be ideal for smaller organisations with limited resources.

On the other hand, simpler algorithms like Naive Bayes have a much lower computational cost while still performing well in text classification tasks [5]. Its fast training and prediction times make it a strong option where efficiency matters, such as real-time analysis or when working with large amounts of data.

KNN is simple to understand, but it is less suitable for larger datasets because it relies on distance calculations during prediction, which can slow things down [9]. Decision Trees are easy to interpret, but they can sometimes overfit the data, which means they may not perform as well on new, unseen data [4].

Overall, the choice of algorithm depends on what is most important for the task. For situations where efficiency and scalability are key, such as sentiment analysis in a resource-constrained marketing environment, simpler models like Naive Bayes offer a good balance between performance and efficiency. More complex models may improve accuracy slightly, but this often comes at the cost of increased computational requirements.

4. Methodology

This section outlines the machine learning workflow used to develop a sentiment analysis model. It follows a structured supervised learning process, starting with dataset selection and moving through data preprocessing, feature extraction, model training, and evaluation. Following this kind of step-by-step approach helps ensure the model is both effective and efficient, making it suitable for use in a resource-constrained environment.

4.1 Dataset Selection

For sentiment analysis tasks, it is important to use a dataset that contains labelled text data. In supervised learning, machine learning models rely on labelled examples to learn the relationship between input features and the expected output [1].

A suitable dataset for this task could include customer reviews or a dataset like the 20 Newsgroups dataset, which contains labelled text across different categories. This type of dataset is commonly used in text classification and provides a clear structure for training and testing machine learning models.

The dataset is usually split into training and testing sets, often using an 80:20 ratio. The training data is used to build the model, while the test data is used to check how well it performs on unseen data. This helps make sure the model can generalise properly rather than just memorising the training data [11].

4.2 Data Pre-processing

Data preprocessing is a critical step in the machine learning workflow, particularly for text data, which often contains noise and irrelevant information [4]. Raw text must be cleaned and transformed into a suitable format before it can be used by machine learning algorithms.

The preprocessing steps typically include:

- converting text to lowercase
- removing punctuation and special characters
- removing stop words
- tokenisation

These steps help to reduce noise and improve the quality of the data, allowing the model to focus on the most relevant information.

4.3 Feature Extraction

Machine learning algorithms work with numerical data, so text needs to be converted into a numerical format before it can be used. This process is known as feature extraction.

One of the most commonly used methods is Term Frequency-Inverse Document Frequency (TF-IDF), which turns text into a matrix of numerical values based on how important each word is within a document compared to the whole dataset [4].

TF-IDF builds on simple bag-of-words models by reducing the impact of very common words and giving more importance to words that carry more meaning. This helps improve the overall performance of the model when classifying text.

4.4 Model Selection and Training

A range of traditional machine learning algorithms can be used for sentiment analysis. Based on the literature review, the following models are considered:

- Naive Bayes [5]
- Support Vector Machine (SVM) [6]
- K-Nearest Neighbours (KNN) [8]
- Decision Tree [4]

Each model is trained using the training dataset, allowing it to learn the relationship between the input features and the sentiment labels.

Out of these, Naive Bayes stands out as a particularly suitable option due to its simplicity, efficiency, and strong performance in text classification tasks [5]. It is computationally lightweight and can handle large amounts of data well, which makes it a good choice for environments with limited resources.

To make sure the model performs reliably, techniques such as k-fold cross-validation can be used during training. This helps reduce overfitting and gives a more accurate estimate of how the model will perform on new, unseen data [11].

4.5 Model Evaluation

Once the models have been trained, their performance needs to be evaluated using the test dataset. This helps check how well the model can generalise to new, unseen data rather than just the data it was trained on.

Several metrics are used to assess performance, including accuracy, precision, recall, and F1-score [11]. Each of these gives a slightly different view of how well the model is performing.

A confusion matrix is also used to give a more detailed breakdown of the predictions. It shows values such as true positives, false positives, true negatives, and false negatives, helping to understand where the model is making mistakes [11].

Together, these metrics provide a clear overall evaluation of each model, making it easier to compare their performance and choose the most suitable algorithm for sentiment analysis.

Using multiple evaluation metrics provides a more comprehensive assessment of model performance, as accuracy alone may not fully capture classification effectiveness, particularly in multi-class problems.

In addition to the workflow used, a machine learning pipeline could be introduced to make the model more robust. A pipeline brings together preprocessing steps, such as TF-IDF feature extraction, with model training into a single process. This ensures that the same transformations are applied consistently to both the training and test data.

Using a pipeline also helps prevent data leakage and reflects best practice in real-world machine learning systems.

5. Discussion & Recommendation

This section evaluates how well the machine learning models perform in text classification and considers how suitable they are for sentiment analysis, particularly in a resource-constrained environment.

The section presents the results of the implemented machine learning workflow and evaluates the performance of the models.

5.1 Model Performance Evaluation

The Naive Bayes model achieved a test accuracy of around 65%, with a cross-validation mean of approximately 69%. This suggests that the model performs consistently across different subsets of the data, indicating good generalisation.

The classification report (Figure 7) shows that precision, recall, and F1-scores are relatively balanced across several categories, although performance does vary depending on the class. Some categories are predicted more accurately than others, which is expected in text classification tasks.

The confusion matrix (Figure 8) gives a clearer picture of how the model is performing. A strong diagonal pattern shows that many predictions are correct, while the lighter off-diagonal values highlight where misclassifications occur. These errors are more noticeable between categories with similar content, which is common when working with text data.

	precision	recall	f1-score	support
alt.atheism	0.55	0.31	0.40	319
comp.graphics	0.57	0.65	0.61	389
comp.os.ms-windows.misc	0.61	0.62	0.61	394
comp.sys.ibm.pc.hardware	0.60	0.66	0.63	392
comp.sys.mac.hardware	0.71	0.62	0.66	385
comp.windows.x	0.76	0.73	0.75	395
misc.forsale	0.74	0.78	0.76	390
rec.autos	0.70	0.68	0.69	396
rec.motorcycles	0.74	0.71	0.72	398
rec.sport.baseball	0.80	0.77	0.78	397
rec.sport.hockey	0.55	0.89	0.68	399
sci.crypt	0.76	0.70	0.73	396
sci.electronics	0.62	0.48	0.54	393
sci.med	0.75	0.69	0.72	396
sci.space	0.74	0.69	0.72	394
soc.religion.christian	0.45	0.86	0.59	398
talk.politics.guns	0.53	0.68	0.60	364
talk.politics.mideast	0.81	0.75	0.78	376
talk.politics.misc	0.71	0.34	0.46	310
talk.religion.misc	0.65	0.04	0.08	251
accuracy			0.65	7532
macro avg	0.67	0.63	0.63	7532
weighted avg	0.67	0.65	0.64	7532

Figure 7. Classification report output from Naive Bayes model showing precision, recall, and F1-score across categories

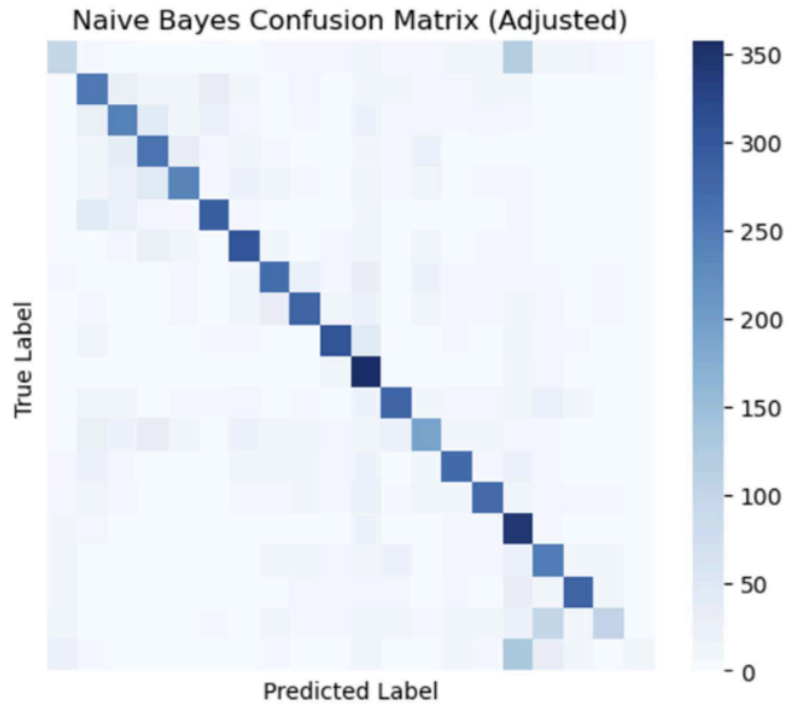


Figure 8. Confusion matrix output from Naive Bayes model

The variation in performance across different categories suggests that the model is affected by how the data is distributed and how distinct the features are. Classes with more unique vocabulary tend to be easier to classify, while those with similar or overlapping terms are harder to distinguish. This highlights a limitation of models like Naive Bayes when dealing with text that is very similar in meaning.

5.2 Comparison of Machine Learning Models

A comparison of model performance is shown in Figure 14. The results indicate that Naive Bayes achieved a test accuracy of around 65%, while the Support Vector Machine (SVM) reached approximately 62.7%. The Decision Tree performed much worse, with an accuracy of around 43.7%.

Although SVM achieved a similar cross-validation mean to Naive Bayes, it did not lead to any clear improvement in test accuracy. It is also more computationally complex, meaning it requires more processing time and resources. In comparison, the Decision Tree model performed poorly, which is likely due to overfitting and its difficulty in handling high-dimensional, sparse text data.

Overall, these results match what is commonly found in the literature. Simpler models like Naive Bayes tend to perform well for text classification, while tree-based models often find this type of data more difficult to handle.

```
[13]: results = pd.DataFrame({
    'Model': ['Naive Bayes', 'SVM', 'Decision Tree'],
    'Test Accuracy': [
        accuracy_score(y_test, y_pred_nb),
        accuracy_score(y_test, y_pred_svm),
        accuracy_score(y_test, y_pred_dt)
    ],
    'CV Mean Accuracy': [
        cv_scores_nb.mean(),
        cv_scores_svm.mean(),
        cv_scores_dt.mean()
    ]
})

print(results)
```

	Model	Test Accuracy	CV Mean Accuracy
0	Naive Bayes	0.650558	0.693301
1	SVM	0.627058	0.694008
2	Decision Tree	0.437201	0.473485

Figure 14. Comparison of Naive Bayes, Support Vector Machine, and Decision Tree model performance

While SVM shows similar performance, it is more complex and takes longer to train, which makes it less suitable for real-time or resource-limited environments. In comparison, Naive Bayes is faster to train and make predictions, making it a more practical choice for smaller systems.

These results are in line with the literature, which shows that Naive Bayes performs well in high-dimensional text classification tasks because of its probabilistic approach [5].

Further improvements in model performance could be achieved through hyperparameter tuning. For example, adjusting the smoothing parameter in Naive Bayes or the regularisation parameter in SVM may improve classification accuracy. However, this kind of optimisation also increases computational cost and complexity. In resource-constrained environments, where efficiency is a priority, this may not always be practical.

5.3 Limitations of the Approach

A key limitation of this study is the use of the 20 Newsgroups dataset, which is designed for topic classification rather than sentiment analysis. This can be seen in the custom text predictions (Figure 13), where the model predicts topic-based categories instead of identifying positive or negative sentiment.

This highlights the importance of dataset selection in machine learning. While the workflow and models used are suitable for sentiment analysis, the results are influenced by the type of data being used. A dataset specifically labelled for sentiment would likely lead to more accurate and meaningful predictions.

While a sentiment-labelled dataset, such as IMDB or Amazon reviews, may have been more suitable for this task, the chosen dataset was still useful for demonstrating the machine learning workflow and allowing a comparison of model performance.

Overall, this reinforces that model performance depends heavily on the quality and relevance of the training data, and even well-performing models can produce misleading results if they are trained on the wrong type of dataset.

```
[12]: sample_texts = [
        "This product is amazing and I love it",
        "Terrible experience, very disappointed",
        "It was okay, nothing special"
    ]

    sample_features = vectorizer.transform(sample_texts)
    predictions = nb_model.predict(sample_features)

    for text, pred in zip(sample_texts, predictions):
        print("Text:", text)
        print("Predicted category:", train_data.target_names[pred])
        print()

    Text: This product is amazing and I love it
    Predicted category: soc.religion.christian

    Text: Terrible experience, very disappointed
    Predicted category: soc.religion.christian

    Text: It was okay, nothing special
    Predicted category: rec.motorcycles
```

Figure 13. Example predictions generated by Naive Bayes model using custom input text

In addition, real-world sentiment analysis comes with challenges that are not reflected in this dataset. Text can include sarcasm, ambiguity, informal language, and meanings that depend on context, which are difficult for traditional models like Naive Bayes to handle.

Since these models mainly rely on word frequency rather than understanding context, they can struggle to correctly interpret more subtle or complex sentiment.

5.4 Recommendation

Based on the results, Naive Bayes is the most suitable model for sentiment analysis in a resource-constrained environment. It achieves good performance while remaining efficient and easy to implement.

Although SVM produces similar results, it is more complex and requires more resources, which makes it less suitable for smaller organisations. The Decision Tree model is not recommended, as it showed much lower accuracy and did not generalise well.

Overall, Naive Bayes offers the best balance between accuracy, efficiency, and simplicity, making it a practical choice for real-world sentiment analysis in a marketing context.

In a real-world setting, the recommended model could be integrated into a system that automatically analyses customer reviews and feedback in real time. This would allow the company to quickly identify trends in customer sentiment and respond to issues more efficiently.

Because of its low computational requirements, the Naïve Bayes model can run on standard hardware without needing specialised infrastructure, making it a cost-effective option for smaller organisations.

6. Conclusion

This report has explored the use of traditional machine learning algorithms for text classification in the context of sentiment analysis. A structured workflow was followed, including data preprocessing, feature extraction using TF-IDF, model training, and evaluation using different performance metrics.

The results showed that while all models were able to perform text classification to some extent, Naive Bayes offered the best balance between accuracy, efficiency, and simplicity. Although Support Vector Machine achieved similar results, it did not provide enough improvement to justify its higher computational cost. In comparison, the Decision Tree model performed significantly worse, showing its limitations when working with high-dimensional text data.

One key limitation of this study was the use of a topic-based dataset instead of a true sentiment dataset. This affected the model's ability to accurately predict sentiment and highlights how important dataset selection is in machine learning tasks.

Overall, the findings support the use of Naive Bayes as a suitable model for sentiment analysis in resource-constrained environments. Future work could involve applying the same workflow to a dataset specifically labelled for sentiment to see how well the model performs in a more realistic setting.

This demonstrates that selecting an appropriate model is not only about maximising accuracy, but also about considering efficiency, scalability, and the specific requirements of the application.

This makes Naïve Bayes particularly suitable for applications such as real-time customer feedback analysis, where speed, scalability, and efficiency are more important than marginal improvements in classification accuracy

Bibliography

- [1] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, 1997. [Online]. Available: <https://www.cs.cmu.edu/~tom/mlbook.html>. [Accessed: Apr. 1, 2026].
- [2] B. Liu, *Sentiment Analysis and Opinion Mining*. San Rafael, CA, USA: Morgan & Claypool Publishers, 2012. [Online]. Available: <https://www.cs.uic.edu/~liub/FBS/SentimentAnalysis-and-OpinionMining.pdf>. [Accessed: Apr. 1, 2026].
- [3] B. Pang, L. Lee, and S. Vaithyanathan, “Thumbs up? Sentiment classification using machine learning techniques,” in *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2002, pp. 79–86. [Online]. Available: <https://aclanthology.org/W02-1011.pdf>. [Accessed: Apr. 1, 2026].
- [4] F. Sebastiani, “Machine learning in automated text categorization,” *ACM Computing Surveys*, vol. 34, no. 1, pp. 1–47, 2002. [Online]. Available: <https://dl.acm.org/doi/10.1145/505282.505283>. [Accessed: Apr. 12, 2026].
- [5] A. McCallum and K. Nigam, “A comparison of event models for naive Bayes text classification,” in *AAAI-98 Workshop on Learning for Text Categorization*, Madison, WI, USA, 1998. [Online]. Available: <https://cdn.aaai.org/Workshops/1998/WS-98-05/WS98-05-007.pdf>. [Accessed: Apr. 1, 2026]
- [6] T. Joachims, “Text categorization with support vector machines: Learning with many relevant features,” in *Machine Learning: ECML-98*, C. Nédellec and C. Rouveirol, Eds., Lecture Notes in Computer Science, vol. 1398, Berlin, Heidelberg: Springer, 1998, pp. 137–142. [Online]. Available: <https://link.springer.com/chapter/10.1007/BFb0026683>. [Accessed: Apr. 1, 2026].
- [7] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995. [Online]. Available: <https://link.springer.com/article/10.1007/BF00994018>. [Accessed: Apr. 5, 2026].
- [8] T. M. Cover and P. E. Hart, “Nearest neighbor pattern classification,” *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967. [Online]. Available: <https://ieeexplore.ieee.org/document/1053964>. [Accessed: Apr. 5, 2026].
- [9] D. W. Aha, D. Kibler, and M. K. Albert, “Instance-based learning algorithms,” *Machine Learning*, vol. 6, no. 1, pp. 37–66, 1991. [Online]. Available: <https://link.springer.com/article/10.1007/BF00153759>. [Accessed: Apr. 5, 2026].

[10] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. [Online]. Available:

<https://link.springer.com/article/10.1023/A:1010933404324>. [Accessed: Apr. 5, 2026].

[11] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed. Waltham, MA, USA: Morgan Kaufmann, 2011. [Online]. Available:

<https://www.sciencedirect.com/book/monograph/9780123814791/data-mining-concepts-and-techniques>. [Accessed: Apr. 12, 2026].

Appendix

Screenshots

```
[1]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

[1]:

Figure 1. imports

```
[2]: train_data = fetch_20newsgroups(subset='train', remove=('headers', 'footers', 'quotes'))
test_data = fetch_20newsgroups(subset='test', remove=('headers', 'footers', 'quotes'))

print("Number of training documents:", len(train_data.data))
print("Number of test documents:", len(test_data.data))
print("Number of categories:", len(train_data.target_names))

Number of training documents: 11314
Number of test documents: 7532
Number of categories: 20
```

[1]:

Figure 2. Dataset loading and overview of training and test data

```
[3]: print("First category names:", train_data.target_names[:5])
      print("\nExample document:\n")
      print(train_data.data[0][:500])
```

First category names: ['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware']

Example document:

I was wondering if anyone out there could enlighten me on this car I saw the other day. It was a 2-door sports car, looked to be from the late 60s/early 70s. It was called a Bricklin. The doors were really small. In addition, the front bumper was separate from the rest of the body. This is all I know. If anyone can tell me a model name, engine specs, years of production, where this car is made, history, or whatever info you have on this funky looking car, please e-mail.

[]:

Figure 3. Example categories and sample document from the dataset

```
[4]: vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)

      X_train = vectorizer.fit_transform(train_data.data)
      X_test = vectorizer.transform(test_data.data)

      y_train = train_data.target
      y_test = test_data.target

      print("Training feature matrix shape:", X_train.shape)
      print("Test feature matrix shape:", X_test.shape)
```

Training feature matrix shape: (11314, 5000)
Test feature matrix shape: (7532, 5000)

[]:

Figure 4. TF-IDF feature extraction and resulting feature matrix dimensions

```
test feature matrix shape: (7532, 5000)
```

```
[5]: nb_model = MultinomialNB()

      nb_model.fit(X_train, y_train)

      y_pred_nb = nb_model.predict(X_test)

      print("Naive Bayes Accuracy:", accuracy_score(y_test, y_pred_nb))
```

Naive Bayes Accuracy: 0.6505576208178439

[]:

Figure 5. Naive Bayes model training and accuracy on test data

```

naive_bayes_classifier = NaiveBayesClassifier()

[6]: print(classification_report(y_test, y_pred_nb, target_names=test_data.target_names))

cm_nb = confusion_matrix(y_test, y_pred_nb)

plt.figure(figsize=(10, 8))
sns.heatmap(cm_nb, cmap="Blues")
plt.title("Naive Bayes Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

```

Figure 6. Implementation of classification report and confusion matrix for Naive Bayes model

	precision	recall	f1-score	support
alt.atheism	0.55	0.31	0.40	319
comp.graphics	0.57	0.65	0.61	389
comp.os.ms-windows.misc	0.61	0.62	0.61	394
comp.sys.ibm.pc.hardware	0.60	0.66	0.63	392
comp.sys.mac.hardware	0.71	0.62	0.66	385
comp.windows.x	0.76	0.73	0.75	395
misc.forsale	0.74	0.78	0.76	390
rec.autos	0.70	0.68	0.69	396
rec.motorcycles	0.74	0.71	0.72	398
rec.sport.baseball	0.80	0.77	0.78	397
rec.sport.hockey	0.55	0.89	0.68	399
sci.crypt	0.76	0.70	0.73	396
sci.electronics	0.62	0.48	0.54	393
sci.med	0.75	0.69	0.72	396
sci.space	0.74	0.69	0.72	394
soc.religion.christian	0.45	0.86	0.59	398
talk.politics.guns	0.53	0.68	0.60	364
talk.politics.mideast	0.81	0.75	0.78	376
talk.politics.misc	0.71	0.34	0.46	310
talk.religion.misc	0.65	0.04	0.08	251
accuracy			0.65	7532
macro avg	0.67	0.63	0.63	7532
weighted avg	0.67	0.65	0.64	7532

Figure 7. Classification report output from Naive Bayes model showing precision, recall, and F1-score across categories

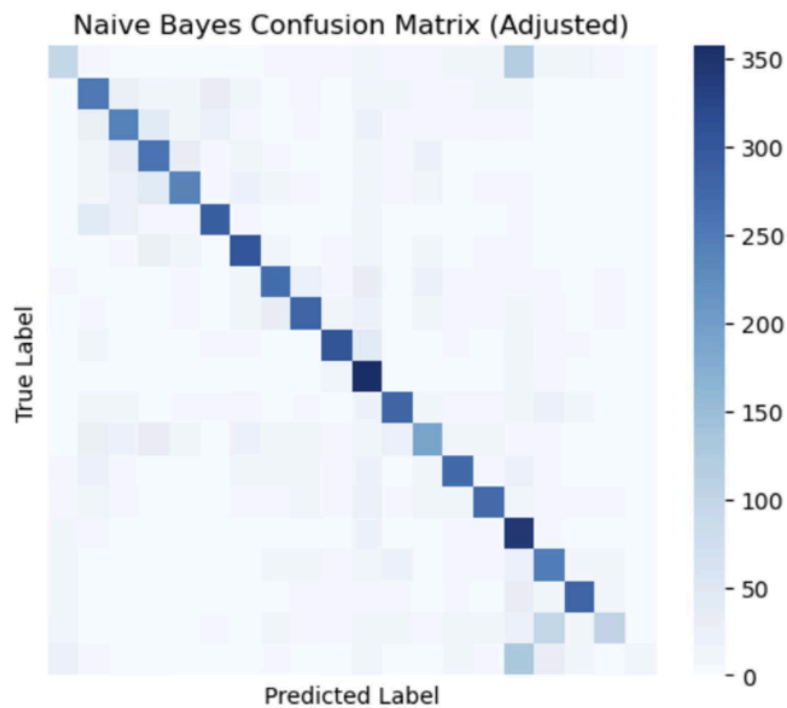


Figure 8. Confusion matrix output from Naive Bayes model

```
[8]: cv_scores_nb = cross_val_score(nb_model, X_train, y_train, cv=5, scoring='accuracy')
print("Naive Bayes CV scores:", cv_scores_nb)
print("Naive Bayes CV mean:", cv_scores_nb.mean())
print("Naive Bayes CV standard deviation:", cv_scores_nb.std())
```

Naive Bayes CV scores: [0.6955369 0.67918692 0.69509501 0.7003977 0.69628647]
 Naive Bayes CV mean: 0.6933005997765929
 Naive Bayes CV standard deviation: 0.007303439981597496

Figure 9. Five-fold cross-validation results for Naive Bayes model

```

9]: svm_model = LinearSVC()

svm_model.fit(X_train, y_train)

y_pred_svm = svm_model.predict(X_test)

print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))

cv_scores_svm = cross_val_score(svm_model, X_train, y_train, cv=5, scoring='accuracy')
print("SVM CV mean:", cv_scores_svm.mean())

SVM Accuracy: 0.6270578863515667
SVM CV mean: 0.694007703989876

]:

```

Figure 10. Support Vector Machine model performance and cross-validation results

```

[10]: results = pd.DataFrame({
    'Model': ['Naive Bayes', 'SVM'],
    'Test Accuracy': [
        accuracy_score(y_test, y_pred_nb),
        accuracy_score(y_test, y_pred_svm)
    ],
    'CV Mean Accuracy': [
        cv_scores_nb.mean(),
        cv_scores_svm.mean()
    ]
})

print(results)

```

	Model	Test Accuracy	CV Mean Accuracy
0	Naive Bayes	0.650558	0.693301
1	SVM	0.627058	0.694008

Figure 11. Comparison of Naive Bayes and Support Vector Machine model performance

```

[11]: dt_model = DecisionTreeClassifier(random_state=42)

dt_model.fit(X_train, y_train)

y_pred_dt = dt_model.predict(X_test)

print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_dt))

cv_scores_dt = cross_val_score(dt_model, X_train, y_train, cv=5, scoring='accuracy')
print("Decision Tree CV mean:", cv_scores_dt.mean())

Decision Tree Accuracy: 0.43720127456186936
Decision Tree CV mean: 0.4734846469147899

```

Figure 12. Decision Tree model performance and cross-validation results

```
[12]: sample_texts = [
        "This product is amazing and I love it",
        "Terrible experience, very disappointed",
        "It was okay, nothing special"
    ]

    sample_features = vectorizer.transform(sample_texts)
    predictions = nb_model.predict(sample_features)

    for text, pred in zip(sample_texts, predictions):
        print("Text:", text)
        print("Predicted category:", train_data.target_names[pred])
        print()

    Text: This product is amazing and I love it
    Predicted category: soc.religion.christian

    Text: Terrible experience, very disappointed
    Predicted category: soc.religion.christian

    Text: It was okay, nothing special
    Predicted category: rec.motorcycles
```

Figure 13. Example predictions generated by Naive Bayes model using custom input text

```
[13]: results = pd.DataFrame({
        'Model': ['Naive Bayes', 'SVM', 'Decision Tree'],
        'Test Accuracy': [
            accuracy_score(y_test, y_pred_nb),
            accuracy_score(y_test, y_pred_svm),
            accuracy_score(y_test, y_pred_dt)
        ],
        'CV Mean Accuracy': [
            cv_scores_nb.mean(),
            cv_scores_svm.mean(),
            cv_scores_dt.mean()
        ]
    })

    print(results)
```

	Model	Test Accuracy	CV Mean Accuracy
0	Naive Bayes	0.650558	0.693301
1	SVM	0.627058	0.694008
2	Decision Tree	0.437201	0.473485

Figure 14. Comparison of Naive Bayes, Support Vector Machine, and Decision Tree model performance

YouTube link

<https://www.youtube.com/@kellycasey4650>

Code

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

train_data = fetch_20newsgroups(subset='train', remove=('headers', 'footers',
'quotes'))
test_data = fetch_20newsgroups(subset='test', remove=('headers', 'footers',
'quotes'))

print("Number of training documents:", len(train_data.data))
print("Number of test documents:", len(test_data.data))
print("Number of categories:", len(train_data.target_names))

print("First category names:", train_data.target_names[:5])

print("\nExample document:\n")
print(train_data.data[0][:500])

vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)

X_train = vectorizer.fit_transform(train_data.data)
X_test = vectorizer.transform(test_data.data)

y_train = train_data.target
y_test = test_data.target

print("Training feature matrix shape:", X_train.shape)
print("Test feature matrix shape:", X_test.shape)
```

```
nb_model = MultinomialNB()

nb_model.fit(X_train, y_train)

y_pred_nb = nb_model.predict(X_test)

print("Naive Bayes Accuracy:", accuracy_score(y_test, y_pred_nb))
```

```
print(classification_report(y_test, y_pred_nb,
                             target_names=test_data.target_names))

cm_nb = confusion_matrix(y_test, y_pred_nb)

plt.figure(figsize=(10, 8))
sns.heatmap(cm_nb, cmap="Blues")
plt.title("Naive Bayes Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
cv_scores_nb = cross_val_score(nb_model, X_train, y_train, cv=5,
                                 scoring='accuracy')

print("Naive Bayes CV scores:", cv_scores_nb)
print("Naive Bayes CV mean:", cv_scores_nb.mean())
print("Naive Bayes CV standard deviation:", cv_scores_nb.std())
```

```
svm_model = LinearSVC()

svm_model.fit(X_train, y_train)

y_pred_svm = svm_model.predict(X_test)

print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))

cv_scores_svm = cross_val_score(svm_model, X_train, y_train, cv=5,
                                 scoring='accuracy')
print("SVM CV mean:", cv_scores_svm.mean())
```

```

results = pd.DataFrame({
    'Model': ['Naive Bayes', 'SVM'],
    'Test Accuracy': [
        accuracy_score(y_test, y_pred_nb),
        accuracy_score(y_test, y_pred_svm)
    ],
    'CV Mean Accuracy': [
        cv_scores_nb.mean(),
        cv_scores_svm.mean()
    ]
})

print(results)

```

```

dt_model = DecisionTreeClassifier(random_state=42)

dt_model.fit(X_train, y_train)

y_pred_dt = dt_model.predict(X_test)

print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_dt))

cv_scores_dt = cross_val_score(dt_model, X_train, y_train, cv=5,
                                scoring='accuracy')
print("Decision Tree CV mean:", cv_scores_dt.mean())

```

```

sample_texts = [
    "This product is amazing and I love it",
    "Terrible experience, very disappointed",
    "It was okay, nothing special"
]

sample_features = vectorizer.transform(sample_texts)
predictions = nb_model.predict(sample_features)

for text, pred in zip(sample_texts, predictions):
    print("Text:", text)
    print("Predicted category:", train_data.target_names[pred])
    print()

```

```

results = pd.DataFrame({
    'Model': ['Naive Bayes', 'SVM', 'Decision Tree'],
    'Test Accuracy': [
        accuracy_score(y_test, y_pred_nb),
        accuracy_score(y_test, y_pred_svm),
        accuracy_score(y_test, y_pred_dt)
    ]
})

```

```
    ],  
    'CV Mean Accuracy': [  
        cv_scores_nb.mean(),  
        cv_scores_svm.mean(),  
        cv_scores_dt.mean()  
    ]  
})  
  
print(results)
```

FINAL GRADE

GENERAL COMMENTS

75/100

Generic Assessment Criteria

Ma Ma
rks rks
ava
ilab aw
le ard
ed

1. Engagement
with Literature Skills

.

A good number and reasonable quality of sources in reference.

17

I wonder if there were any more up-to-date papers available in this field.

20

Good referencing throughout the text backing up and motivating the points raised.

2. Knowledge
and Understanding Skills

Some good discussion demonstrating reasonable knowledge from research.

28

40

A little more detailed critical analysis would have been advantageous although reasonable comparisons presented.

3. Cognitive and Intellectual Skills

Model choice and workflow is motivated well by practice, although finding the sentiment datasets would have proved more prudent.

22

30

A reasonable discussion of workflow including the feature extraction shown however some more detailed discussion of fine tuning would have been beneficial (albeit that it is mentioned in passing later in the report).

4. Practical Application Skills

0

5. Transferable Skills for Life and Professional Practice

A well presented and nicely written report.

10

8

PAGE 1
PAGE 2
PAGE 3
PAGE 4
PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27