



Comparing Support Vector Machines and Decision Trees for Text Classification

Last Updated : 23 Jul, 2025

Support Vector Machines (SVMs) and Decision Trees are two popular machine learning algorithms for text classification, but they have different strengths and are suitable for different types of problems.

[Suggest changes](#)[Like](#)[Report](#)

Why is model selection important in Text Classification?

Selecting the ideal model for text classification resembles selecting the ideal tool for a task – it's crucial to weigh accuracy against interpretability. Accuracy guarantees our model can correctly identify, for example, spam emails, while interpretability enables us to comprehend the reasoning behind those identifications. A model that is highly accurate but opaque might be confusing, whereas a transparent yet less accurate model could result in missed chances. Finding the right equilibrium ensures our model performs effectively and provides us with insights into its operations, enabling us to make informed choices and establish trust in its outcomes.

Using Machine Learning for Text Classification

- Text classification is a powerful tool in the world of [Natural Language Processing](#) (NLP), allowing computers to understand and categorize text data automatically.
- It's like teaching a computer to read and organize vast amounts of written information, such as emails, news articles, or social media posts.

- Imagine you have thousands of emails and you want to sort them into different folders based on their topics, like work, personal, or spam.
- Instead of manually going through each email, you can use machine learning algorithms to do this automatically. These algorithms learn from examples to recognize patterns and make predictions about new, unseen text.

Building a Text Classifier in Python

We will use the 20 Newsgroups dataset, it is a classic dataset commonly used for text classification tasks. It consists of approximately 20,000 newsgroup documents, partitioned across 20 different newsgroups. Each document belongs to one of these newsgroups, making it suitable for multi-class classification tasks.

Importing necessary libraries

```
# Import necessary libraries
from sklearn.datasets import fetch_20newsgroups
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score,
classification_report
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
```

Loading and splitting the dataset

1. **Dataset Loading:** The code uses `fetch_20newsgroups` from `sklearn.datasets` to load the 20 Newsgroups dataset. This dataset is a collection of approximately 20,000 newsgroup documents, partitioned across 20 different newsgroups. The `subset='all'` argument indicates that it should load all available documents.
2. **Data Splitting:** The `train_test_split` function from `sklearn.model_selection` is used to split the dataset into training and testing sets. The `test_size=0.2` argument specifies that 20% of the data should be used for testing, while the rest is used for training. The

`random_state=42` argument ensures reproducibility by fixing the random seed for the data splitting process.

3. **Data Structure:** `X_train` and `X_test` are arrays containing the text content of the newsgroup documents, while `y_train` and `y_test` are arrays containing the corresponding target labels (the index of the newsgroup category) for the training and testing sets, respectively. These arrays are used for training and evaluating machine learning models on the dataset.

```
# Load the 20 Newsgroups dataset
newsgroups_data = fetch_20newsgroups(subset='all')

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(newsgroups_data,
                                                    newsgroups_data,
                                                    test_size=0.2,
                                                    random_state=42)
```

Training Support Vector Machines (SVMs)

- Training [Support Vector Machines \(SVMs\)](#) involves transforming textual data into a numerical format through a process called vectorization.
- This conversion enables SVMs to understand and process the text. Once the dataset is vectorized, the SVM classifier is trained on the transformed data to learn patterns and relationships between different categories.
- After training, the model's performance is evaluated using a classification report and a confusion matrix. The classification report provides metrics such as accuracy, precision, recall, and F1-score, offering insights into the classifier's performance across various categories.

Vectorizing the dataset

```
# TF-IDF VECTORIZATION
```

```
tfidf_vectorizer = TfidfVectorizer()
```

```
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
```

```
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

Evaluating the SVM classifier

- **Making Predictions:** The `predict` method of the `svm_classifier` object is used to make predictions on the test set `x_test_tfidf`, which likely contains [TF-IDF](#) transformed text data.
- **Evaluating Performance:** The `accuracy_score` function from `sklearn.metrics` is used to calculate the accuracy of the predictions by comparing them to the true labels `y_test`. The `classification_report` function is also used to generate a detailed report containing precision, recall, F1-score, and support for each class in the dataset.

```
# Make predictions
```

```
svm_predictions = svm_classifier.predict(X_test_tfidf)
```

```
# Evaluate performance
```

```
svm_accuracy = accuracy_score(y_test, svm_predictions)
```

```
svm_classification_report = classification_report(y_test,
svm_predictions)
```

```
print("Support Vector Machines (SVM)")
```

```
print("Accuracy:", svm_accuracy)
```

```
print("Classification Report:")
```

```
print(svm_classification_report)
```

Output:

Support Vector Machines (SVM)

Accuracy: 0.9143236074270557

Classification Report:

	precision	recall	f1-score	support
0	0.94	0.91	0.92	151
1	0.78	0.87	0.82	202
2	0.88	0.84	0.86	195

3	0.74	0.83	0.78	183
4	0.91	0.87	0.89	205
5	0.88	0.88	0.88	215
6	0.82	0.85	0.83	193
7	0.92	0.96	0.94	196
8	0.98	0.95	0.96	168
9	0.97	0.98	0.97	211
10	0.97	0.95	0.96	198
11	0.99	0.95	0.97	201
12	0.90	0.87	0.88	202
13	0.95	0.94	0.95	194
14	0.97	0.97	0.97	189
15	0.96	0.98	0.97	202
16	0.96	0.95	0.95	188
17	0.99	0.98	0.99	182
18	0.95	0.91	0.93	159
19	0.89	0.85	0.87	136
accuracy			0.91	3770
macro avg		0.92	0.91	3770
weighted avg		0.92	0.91	3770

- The Support Vector Machines (SVM) model achieved an accuracy of 91.43% on the test set, indicating that it correctly predicted the newsgroup category for the majority of the documents.
- The classification report provides a detailed breakdown of the model's performance across different categories.
- Precision measures the proportion of correctly predicted instances among all instances predicted as belonging to a particular category, while recall measures the proportion of correctly predicted instances among all instances that actually belong to a particular category.
- The F1-score offers a fair assessment of a model's performance as it is the harmonic mean of accuracy and recall. Overall, the SVM model performed well across most categories, with particularly high scores for categories 9, 10, 11, 14, 15, 17, and 18, indicating strong predictive performance.

Training Decision Trees

- The [Decision Tree classifier](#) is trained on the text dataset, learning to classify documents into different categories based on the features present in the data.
- Following training, the classifier's performance is evaluated using a classification report and a confusion matrix.
- The classification report provides metrics such as accuracy, precision, recall, and F1-score, offering insights into the classifier's performance across various categories.

Training a Decision Tree classifier

```
# Train Decision Tree classifier
dt_classifier = DecisionTreeClassifier()
dt_classifier.fit(X_train_tfidf, y_train)
```

Evaluating the model

```
# Make predictions
dt_predictions = dt_classifier.predict(X_test_tfidf)

# Evaluate performance
dt_accuracy = accuracy_score(y_test, dt_predictions)
dt_classification_report = classification_report(y_test,
dt_predictions)

print("\nDecision Trees")
print("Accuracy:", dt_accuracy)
print("Classification Report:")
print(dt_classification_report)
```

Output:

Decision Trees

Accuracy: 0.616710875331565

Classification Report:

	precision	recall	f1-score	support
0	0.49	0.45	0.47	151
1	0.45	0.45	0.45	202
2	0.59	0.65	0.62	195
3	0.39	0.44	0.41	183
4	0.61	0.55	0.58	205
5	0.62	0.61	0.62	215
6	0.69	0.66	0.68	193
7	0.61	0.61	0.61	196
8	0.76	0.76	0.76	168
9	0.65	0.63	0.64	211
10	0.70	0.72	0.71	198
11	0.81	0.80	0.80	201
12	0.43	0.45	0.44	202
13	0.60	0.64	0.62	194
14	0.72	0.74	0.73	189
15	0.72	0.71	0.71	202
16	0.68	0.64	0.66	188
17	0.80	0.75	0.78	182
18	0.52	0.60	0.56	159
19	0.48	0.42	0.45	136
accuracy				0.62
macro avg				0.62
weighted avg				0.62

The decision tree model achieved an accuracy of 61.67% on the test set, which is lower than the accuracy achieved by the SVM model. The classification report shows that the decision tree model's performance varies across different categories. Some categories, such as 8, 10, 11, 14, 15, and 17, have relatively high precision, recall, and F1-score, indicating that the model performed well in predicting these categories. However, other categories, such as 0, 1, 3, 12, 18, and 19, have lower scores, suggesting that the model struggled to accurately predict these

categories. Overall, the decision tree model's performance is decent but not as strong as the SVM model, particularly in categories where precision, recall, and F1-score are lower.

Comparing the results of SVM and Decision Trees

In this specific comparison on the 20 Newsgroups dataset, the Support Vector Machines (SVM) model outperforms the Decision Trees model across all metrics, including accuracy, precision, recall, and F1-score. SVMs are often preferred for text classification tasks due to their ability to handle high-dimensional data like text features and their effectiveness in dealing with non-linear boundaries between classes. However, the choice of model can depend on various factors, including the specific characteristics of the dataset and the computational resources available. In some cases, Decision Trees may be preferred for their simplicity and interpretability, especially when the dataset is not as complex or when interpretability is important

Key Differences Between SVM and Decision Trees

Keyword	Support Vector Machines (SVM)	Decision Trees
Model Complexity	More complex	Simpler
Handling Non-linearity	Efficient through kernel trick	Can capture non-linear relationships
Robustness to Noise	More robust	Susceptible to noise

Keyword	Support Vector Machines (SVM)	Decision Trees
Training Time	Computationally expensive	Faster
Interpretability	Less interpretable	More interpretable
Handling Imbalanced Data	Can handle well with class weights or SMOTE	May require additional techniques
Generalization Performance	Tends to generalize well	May suffer from overfitting
Handling High- dimensional Data	Efficient	May struggle, especially with irrelevant features
Parameter Sensitivity	Sensitive to kernel and regularization parameters	Less sensitive, easier to train

Comment

C

cipher...

+ Follow

0

Article Tags : [Machine Learning](#) [AI-ML-DS](#) [AI-ML-DS With Python](#)

Explore

Machine Learning Basics

Python for Machine Learning

Feature Engineering

Supervised Learning

Unsupervised Learning

Model Evaluation and Tuning

Advanced Techniques

Machine Learning Practice



Corporate & Communications Address:

A-143, 7th Floor, Sovereign Corporate
Tower, Sector- 136, Noida, Uttar Pradesh
(201305)

Registered Address:

K 061, Tower K, Gulshan Vivante
Apartment, Sector 137, Noida, Gautam
Buddh Nagar, Uttar Pradesh, 201305



Company

[About Us](#)
[Legal](#)
[Privacy Policy](#)
[Contact Us](#)
[Advertise with us](#)
[GFG Corporate Solution](#)

Explore

[POTD](#)
[Job-A-Thon](#)
[Blogs](#)
[Nation Skill Up](#)

Campus Training Program

Tutorials

Programming Languages

DSA

Web Technology

AI, ML & Data Science

DevOps

CS Core Subjects

Interview Preparation

Software and Tools

Videos

DSA

Python

Java

C++

Web Development

Data Science

CS Subjects

Courses

IBM Certification

DSA and Placements

Web Development

Programming Languages

DevOps & Cloud

GATE

Trending Technologies

Preparation Corner

Interview Corner

Aptitude

Puzzles

GfG 160

System Design

@GeeksforGeeks, Sanchhaya Education Private Limited, All rights reserved