*Ruchira Nirmal Ekanayaka*
*ID: 24065156*

# SVM for High-Dimensional Text Classification: A Teaching-Focused Tutorial

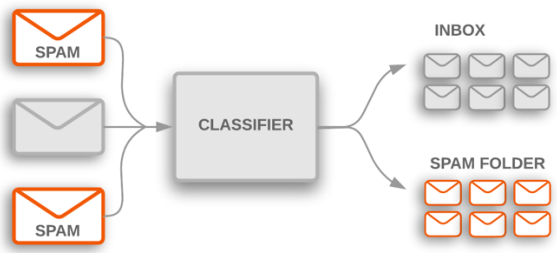
## Why Text Is Hard for Machines

Text classification is an essential machine-learning task that appears in many real-world applications such as spam detection, sentiment analysis, customer-support routing, and hatred-comment filtering. Although the idea is simple—assign a category to a piece of text—computers face a challenge: text is not numerical, yet machine-learning models require numerical inputs.

Text data naturally produces very high-dimensional feature spaces. Even short messages contain words that, when considered across thousands of documents, form large vocabularies. For example, the SMS Spam dataset contains thousands of unique words, while larger datasets like 20 Newsgroups contain tens of thousands. Each word becomes a feature, meaning each document becomes a long vector with mostly zero values.

This combination of high dimensionality and sparsity makes text difficult for many algorithms. However, Support Vector Machines (SVMs) handle these challenges remarkably well, making them one of the strongest choices for text classification.

Before building the model, let's understand the core concepts behind these methods, starting with the basics of text classification.

*Ruchira Nirmal Ekanayaka*
*ID: 24065156*

## ○ What Is Text Classification?

Text classification classifies text into predetermined categories, such as emails, chats, reviews, or articles. In machine learning, this task involves learning a mapping between documents and labels by analysing patterns in language.

A key characteristic of this task is that documents must be transformed into numerical feature vectors. Methods like "TF-IDF" produce high-dimensional representations, often with tens of thousands of features. Most entries in these vectors are zero, making them highly sparse**.**

Joachims (1998) notes that text classification problems have three unique properties that distinguish them from other machine-learning tasks:

1. ***Very high dimensionality****:* Vocabulary sizes often reach 40,000–60,000 words.
   ex**:**
   Vocabulary = 50,000 → document = 50,000-dimensional vector.
2. ***Sparse representations:*** Each document uses only a small number of words, so most entries in the vector are zero.
   ex**:**
   "This product is terrible" → 4 non-zero values, 49,996 zeros.
3. ***Few irrelevant features:*** Even rare words carry meaning and help classification.
   ex**:**
   Words like "horrendous", "masterpiece", "refund", "virus" appear rarely but strongly indicate their category.

These properties explain why Support Vector Machines are so effective for text classification tasks and while we step forward you will see why.

## ○ Turning Text into Numbers

Machines do not understand words; they understand numerical patterns. The first major step in text classification is to transform human language into numerical vectors that algorithms can process. There are several methods.

### *Bag-of-Words (BoW): The Basic Idea*

Bag-of-Words is the simplest way to turn text into numbers.
It ignores grammar, order of words, and meaning — it only cares about which words appear and how many times.

*Ruchira Nirmal Ekanayaka*
*ID: 24065156*

1. Build a vocabulary

   Go through the entire dataset and collect *every unique word*.

   Example vocabulary:
   ["*win", "free", "prize", "now", "hello", "are*", … ]

2. Turn each document into a count vector

   For the message: **"***Win a free prize now***"**

   Vector becomes, [1, 1, 1, 1, 0, 0, 0, ...]

   Each number shows how many times the word appears.

However, BoW treats all words equally, causing common words to dominate.

## *TF-IDF: A Smarter Representation*

TF-IDF improves BoW by weighting words based on how informative they are.

It answers two questions:

1. **TF — Term Frequency:**
   How often does this word appear in *this* document?
2. **IDF — Inverse Document Frequency:**
   How rare or unique is this word across *all* documents?

### 1. Term Frequency (TF)

TF measures how many times a word appears in a document.

$$TF(t,d) = \frac{\text{Number of times word t appears in document d}}{\text{Total words in document d}}$$

Ex: Document: "win a free prize now"

- Total words = 6
- TF("now") = 2 / 6 = 0.33
- TF("win") = 1 / 6 = 0.16

TF helps measure importance inside one document.

*Ruchira Nirmal Ekanayaka*
*ID: 24065156*

**2. Inverse Document Frequency (IDF)**

IDF measures how rare a word is across the dataset.

$$IDF(t) = \ln\left(\frac{N}{df_t}\right)$$

Where:

- **N** = total number of documents
- **$df_t$** = number of documents containing the word

Why?

- Rare words → **high IDF**
- Common words → **low IDF**

Example with 5 documents:

| **Word** | **Appears in how many docs** | **IDF** |
|---|---|---|
| "free" | 1 | ln(5/1) = 1.60 |
| "the" | 5 | ln(5/5) = 0 |

So:

- "free" is **useful** → high weight
- "the" is **useless** → low weight

---

3. Putting It Together: TF × IDF

$$TF\text{-}IDF(t,d) = TF(t,d) \times IDF(t)$$

ex: TF-IDF for "free"

Document: "win a free prize now"
TF("free") = 1/5 = 0.20
IDF("free") across dataset = 1.60
So:

$$TF\text{-}IDF(free) = 0.20 \times 1.60 = 0.32$$

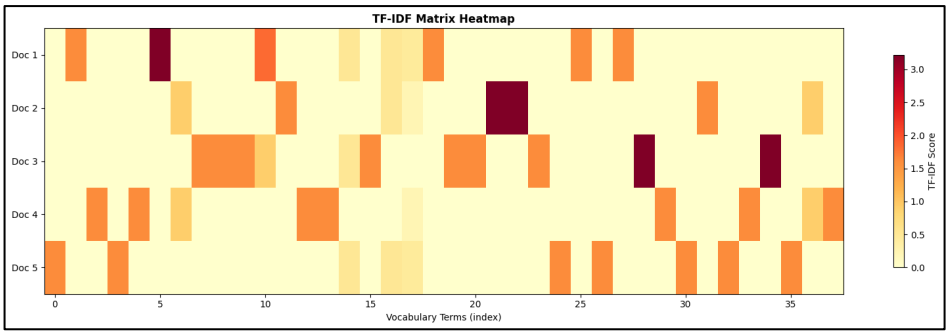
*Ruchira Nirmal Ekanayaka*
*ID: 24065156*

This value **represents how important "free" is** for this specific document.

(5572 documents, 9000 features)

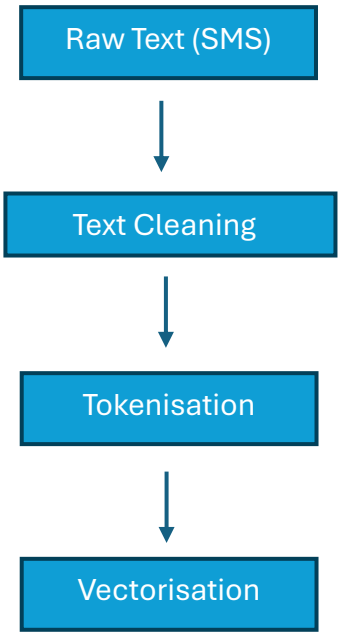
Meaning:

- 5572 rows = one per SMS
- 9000 columns = vocabulary size
- Most values = 0 (sparse)




*An example, mostly empty TF-IDF matrix heatmap showing sparse non-zero entries, illustrating the high-dimensional but sparse nature of text data.*

So wrapping up everything we done so far here's simple visual representation of what happened so far with our example.




In the accompanying Jupyter notebook, you will implement all the steps shown in this section using the SMS Spam dataset. The notebook mirrors this tutorial and includes TF-IDF extraction, Linear SVM training, evaluation metrics, and visualisations.

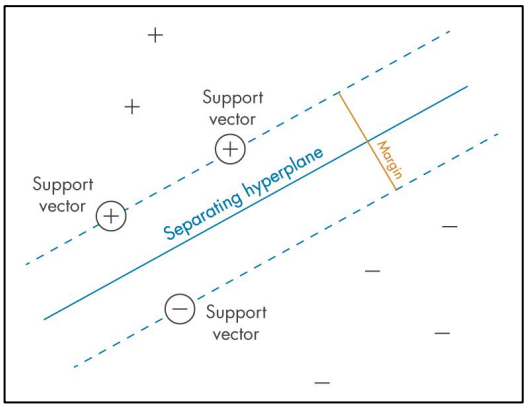
*Ruchira Nirmal Ekanayaka*
*ID: 24065156*

- ## Understanding SVM in Simple Terms

Support Vector Machines classify data by drawing a boundary between classes**.** But unlike many algorithms that simply try to separate points, SVM searches for the *best possible* boundary — the one that maximises the margin, which is the distance between the decision line (or hyperplane) and the closest training examples from each class.

### *Margins and Support Vectors*

The closest data points to the margin are called **support vectors**. Only these points influence where the boundary is placed. Points far away from the boundary do not affect the model, which is one reason SVMs are robust and resistant to noise.




- ## Why SVM Works Better than Many Algorithms for High-Dimensional Text

SVM has properties that give it a major advantage over algorithms like Naive Bayes, Logistic Regression, Decision Trees, and K-Nearest Neighbours — especially when the number of features is very large, as in text classification:

- **High-dimensional capability:**
  Models like logistic regression or neural networks may struggle when feature counts reach tens of thousands, but SVM performance actually *improves* with dimensionality. SVM complexity depends on the **margin**, not the number of features. This makes SVM ideal for TF-IDF vectors with 10,000+ dimensions.
- **Works extremely well with sparse data:**
  Decision Trees or k-NN slow down drastically with sparse matrices. SVM's optimisation handles sparsity naturally and efficiently.
- **Resistant to overfitting:**
  Unlike models that try to memorise all training examples (e.g., k-NN), SVM uses only a handful of support vectors to define the decision boundary, reducing the chance of overfitting.

*Ruchira Nirmal Ekanayaka*
*ID: 24065156*

- **Fast training in linear mode:**
  LinearSVC is optimised for large feature spaces and can train on tens of thousands of documents quickly, unlike kernelised models or neural networks that require GPUs.

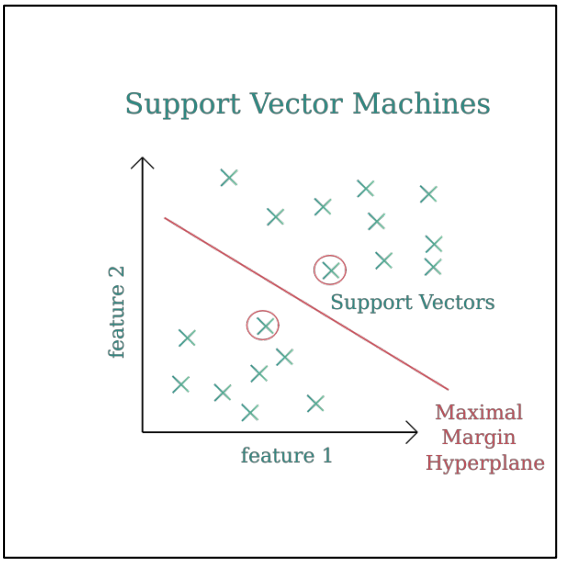
### *Linear SVM Works Best for Text*

SVM supports nonlinear kernels, but text classification almost always uses a **linear SVM** because:

- Text is already high-dimensional
- Many text datasets are **linearly separable or nearly so(**Joachims -1998)
- Linear models are faster and scale better
- LinearSVC is optimised for sparse data

It is also important to consider the C parameter when training an SVM, because it controls how strictly the model tries to separate the classes during text classification. Text data often contains noise and rare words, so C influences whether the model focuses on keeping a smooth, generalisable margin or tries to fit every training example closely, which can lead to overfitting.

- Low C → softer margin → better generalisation
- High C → tighter fit → possible overfitting




The following implementation steps correspond directly to Sections 1–5 in your Python notebook, where you load the SMS Spam dataset and train the Linear SVM model.

*Ruchira Nirmal Ekanayaka*
*ID: 24065156*

- ## Building the Classifier Step by Step

This section now connects your understanding to a real implementation using the SMS Spam Collection dataset.

### Step 1: Load and Inspect the Data

The dataset typically comes in a two-column format:

| label | text |
|---|---|
| ham | Hello, how are you? |
| spam | FREE entry to win cash! |

A snippet of loading code:

```
import pandas as pd
df = pd.read_csv("spam.csv", encoding='latin-1')[['v1','v2']]
df.columns = ['label','text']
```

---

### Step 2: Split Data for Training and Testing

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
   df['text'], df['label'], test_size=0.2, random_state=42)
```

Training data teaches the model; test data checks how well it generalises.

---

### Step 3: Build a TF-IDF + SVM Pipeline

This is the most elegant way to combine preprocessing and modelling:

Note that the stop_words='english' parameter removes common, low-information words (like “the” and “and”) so they don’t weaken the TF-IDF features or increase unnecessary dimensionality.

```
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
   ('tfidf', TfidfVectorizer(stop_words='english')),
   ('clf', LinearSVC(C=1.0))
])
pipeline.fit(X_train, y_train)
```

This pipeline handles everything:

*Ruchira Nirmal Ekanayaka*
*ID: 24065156*

1. Clean text
2. Transform into TF-IDF
3. Train SVM classifier

### Step 4: Make Predictions

```
predictions = pipeline.predict(X_test)
```

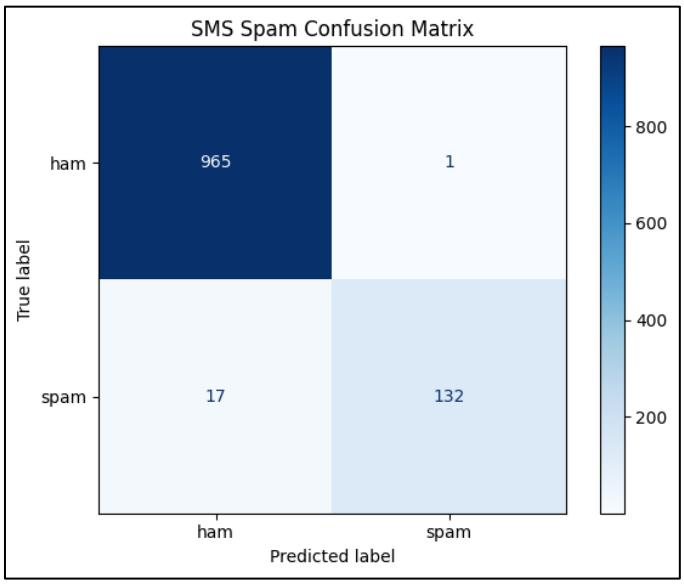
### Step 5: Evaluate the Model

SVMs typically achieve **95%+ accuracy** on SMS Spam classification.

```
from sklearn.metrics import classification_report, confusion_matrix

print(classification_report(y_test, predictions))
```

here's the resaulted confusion matrix.




# How Well Does It Perform?

### Classification Metrics Explained Simply

- **Accuracy** = overall correctness
- **Precision** = when the model says "spam," how often is it right?
- **Recall** = how many actual spam messages did it successfully catch?
- **F1-score** = the balance between precision and recall

*Ruchira Nirmal Ekanayaka*
*ID: 24065156*

In spam detection, **recall matters**, because missing a spam message is worse than accidentally marking ham as spam. You can get the relevant values for these metrics while you are working through the python file like in below.

Ex:

```
Classification report:
              precision    recall  f1-score   support

         ham       0.98      1.00      0.99       966
        spam       0.99      0.89      0.94       149

    accuracy                           0.98      1115

   macro avg       0.99      0.94      0.96      1115
```

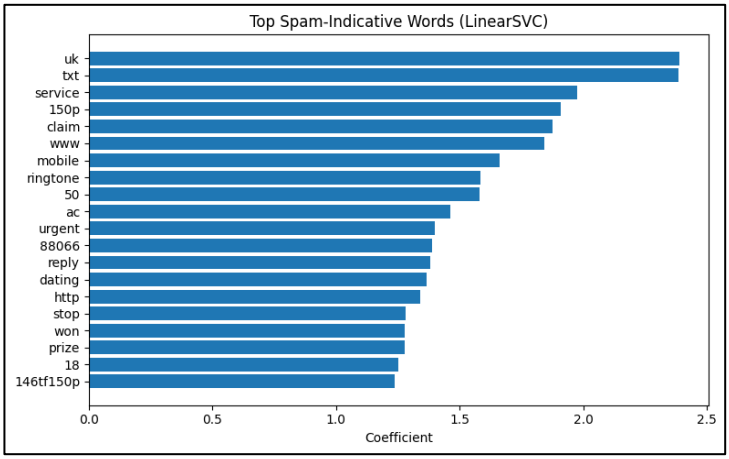
## What Words Does the Model Think Are Important?

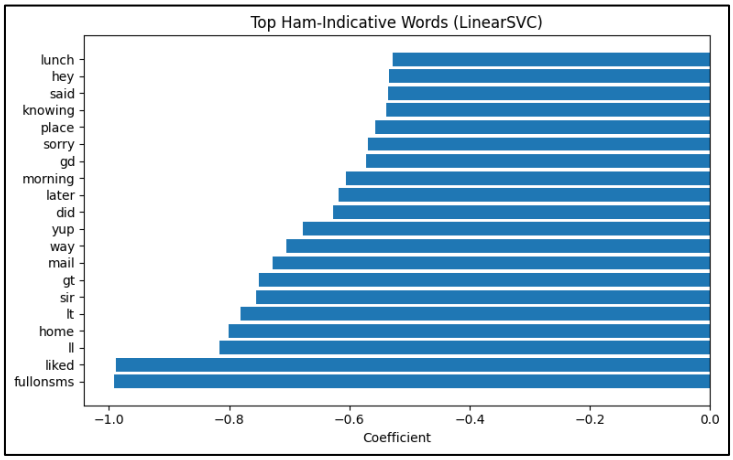
Linear SVMs provide access to their internal weights, allowing us to identify which words push predictions toward “spam” or “ham”. (In the python file please refer to section 2.7)

```
clf = pipeline.named_steps['clf']
tfidf = pipeline.named_steps['tfidf']
feature_names = tfidf.get_feature_names_out()
```

Words with large positive weights may include: “free”, “win”, “txt”

Words with negative weights may include: “hey”, “okay”, “tomorrow”




*Bar charts showing the top words most strongly associated with spam and ham, based on the highest positive and negative SVM feature weights.*

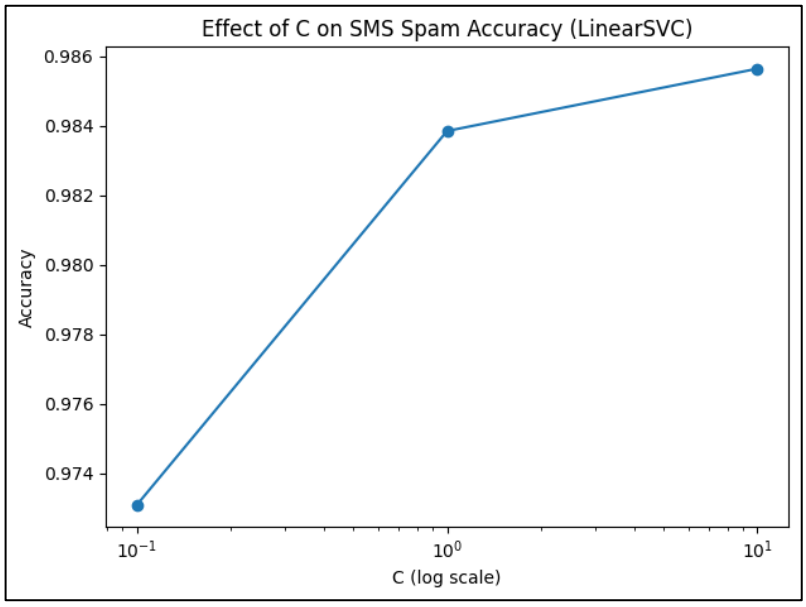
*Ruchira Nirmal Ekanayaka*
*ID: 24065156*

# Exploring SVM Behaviour: The C Parameter

SVM's main hyperparameter, **C**, influences the model's behaviour. Trying multiple values demonstrates its impact:

```
for c in [0.1, 1, 10]:
   model = LinearSVC(C=c)
   model.fit(X_train_tfidf, y_train)
   print("C =", c, "Accuracy:", model.score(X_test_tfidf, y_test))
```

A higher C usually increases training accuracy but may reduce generalisation. According to our example.




# Extending the Concepts to Larger Datasets

While the tutorial uses the SMS dataset, the same pipeline scales effortlessly to larger datasets like

**20 Newsgroups** (Scikit-learn inbuilt)

- ~20,000 documents
- ~100,000 features
- Multi-class (20 categories)

This second dataset will be demonstrated in your **Python notebook**, showing the strength of SVMs on truly high-dimensional data. Linear SVM remains efficient even here because its complexity depends on the margin, not the number of features.

*Ruchira Nirmal Ekanayaka*
*ID: 24065156*

# What Can Go Wrong?

- **Overfitting**

High C values or an excessive number of rare n-grams can push the SVM to focus too heavily on unusual or noise-based patterns in the training data. Instead of learning general decision boundaries, the model begins to "memorise" specific examples, which improves training accuracy but reduces its ability to generalise to new messages.

- **Data leakage**

If preprocessing is done incorrectly before splitting data, performance becomes unrealistic.

- **Biased dataset**

Spam datasets often have more ham than spam or vice versa. Evaluation must consider imbalance.

- **Ethical implications**

Models must avoid automatically deleting legitimate messages.

# Final Thoughts

Support Vector Machines are one of the strongest & efficient classical machine-learning methods for text classification because they thrive in high-dimensional, sparse, and nearly linearly separable environments. Beyond binary spam detection, SVMs can also be extended to **multiclass classification** using strategies such as **one-vs-rest** or **one-vs-one**, making them effective for tasks like topic categorisation, news sorting, intent detection, and document tagging. Through TF-IDF representation, margin-based learning, and efficient optimisation, SVMs remain a highly competitive choice for many NLP problems.

Although modern NLP with deep learning and artificial neural networks—such as LSTMs, Transformers, and BERT—SVMs still perform remarkably well in scenarios with limited data, high feature sparsity, or when computational efficiency matters. In many real-world datasets, a well-tuned Linear SVM can match or even outperform more complex neural architectures while remaining easier to interpret and faster to train.

Using the SMS Spam dataset makes the workflow intuitive for beginners, while extending the same pipeline to a multiclass dataset like 20 Newsgroups demonstrates the scalability and technical sophistication of SVMs in larger NLP tasks. With research-backed explanations, clear visuals, reproducible code, and accessibility features, this tutorial provides both a

practical guide and a deeper understanding of why SVMs continue to be a reliable and widely used technique in text classification.

For readers interested in further comparison and deeper theoretical insights, Joachims' research offers an excellent review of why SVMs excel in text categorisation and how they compare to other learning methods.

## References

- Joachims, T. (1998). *Text Categorization with Support Vector Machines*.
- Cortes, C. & Vapnik, V. (1995). *Support-Vector Networks*.
- Scikit-Learn Documentation: https://scikit-learn.org

The complete Jupyter notebook, figures, and tutorial files are available in the GitHub repository:

https://github.com/ruchira0011/SVM-for-High-Dimensional-Text-Classification