International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 13 Issue: 11 | Nov 2026 www.irjet.net p-ISSN: 2395-0072

Sentiment Analysis of Social Media Text Using NLP

Sneha Pravin Shinde[1], Snehil Pramod Kamble[2], Soham Sarang Bapat[3], Dr. Saloni Bhushan[4]

[1,2,3,4]*D. Y. Patil School of Humanities and Sciences, India*

Abstract - *Classifying the sentiment of social-media posts is difficult: tweets are brief, informal and noisy, and polarity can hinge on word choice, negation, stretched spellings, hashtags and @-mentions. This paper reports a reproducible comparison on Sentiment140, a collection of 1.6 million tweets whose negative and positive labels were assigned automatically. From it we drew a class-balanced sample of 400,000 tweets (200,000 per class) and divided it 80:20 with stratification. A conventional cleaning routine was contrasted with a context-preserving one, and TF-IDF unigram and bigram features were passed to three classifiers: Logistic Regression, a linear Support Vector Machine and Multinomial Naive Bayes. On the 80,000 held-out tweets, Logistic Regression scored best, with 81.12% accuracy and 81.12% macro-F1, ahead of Linear SVM (79.93%) and Multinomial Naive Bayes (79.55%). Switching Logistic Regression from standard to context-preserving cleaning raised its accuracy from 77.43% to 81.12%, a gain of 3.69 percentage points. For classical sentiment classifiers, preprocessing choices therefore belong in the reported method and should not be dismissed as an implementation detail. Because Sentiment140 relies on distant supervision and dates from 2009, these scores should not be taken as a promise of equal performance on present-day social-media language.*

Key Words: Sentiment analysis, social media, Sentiment140, TF-IDF, Logistic Regression, Support Vector Machine, natural language processing

1. INTRODUCTION

Platforms such as Twitter generate enormous volumes of brief text in which people voice opinions, reactions and moods. Sentiment analysis assigns such messages to polarity categories, and it underpins opinion mining, analysis of customer feedback and social-media monitoring. In an early line of Twitter research, emoticons served as cheap but noisy labels for supervised learning, which made it possible to assemble very large sentiment corpora through distant supervision [1].

Tweets are hard for NLP for several reasons: non-standard spelling, abbreviations, links, user mentions, stretched characters, hashtags, and meaning that depends on missing context. A cleaning routine that strips everything non-alphabetic can throw away signals that carry sentiment, whereas keeping every character inflates the vocabulary and makes the features sparse. We examine this trade-off in a controlled setting with one dataset and one shared TF-IDF representation.

The work is framed as an empirical, reproducible study and not as a proposal. It runs three standard classifiers on the dataset supplied with the project, scores them on a held-out test set and adds a preprocessing ablation. Our aim is not to argue that classical methods beat transformers; it is to measure how they perform under a protocol that is fully specified and can be repeated.

2. PROBLEM STATEMENT

Reported results for social-media sentiment classification are hard to compare, since studies differ in dataset, cleaning rules, feature space, sampling and evaluation setup. As a result it is unclear how much of a reported score comes from the classifier and how much from the way the data were prepared. We tackle this by holding the dataset fixed, giving every classifier the same TF-IDF features, retaining selected sentiment-bearing cues during cleaning, and scoring all models on one shared stratified test set.

3. RESEARCH QUESTIONS

This study sets out to answer eight research questions:

RQ1. How accurately can TF-IDF-based classical classifiers label the supplied Sentiment140 tweets when all of them follow one experimental protocol?

RQ2. Among Logistic Regression, Linear SVM and Multinomial Naive Bayes, which performs best at binary sentiment classification on the held-out test set?

RQ3. Does cleaning that preserves context give better sentiment classification than conventional text cleaning?

RQ4. Which of the two sentiment classes accounts for more misclassifications across the chosen models?

RQ5. Do the classifiers err evenly on negative and positive tweets, or does any of them lean toward one class?

RQ6. How does the impact of the preprocessing choice compare with the impact of picking one classifier over another?

RQ7. Are there material performance differences among the three classifiers when they are trained and tested identically?

RQ8. Is the whole pipeline, including sampling, preprocessing, features, hyperparameters and code, described in enough detail to be reproduced?

4. RESEARCH OBJECTIVES

The study pursues the following objectives:

1. Build a reproducible binary sentiment-analysis pipeline on the supplied Sentiment140 data.
2. Apply a preprocessing procedure that removes structural noise but keeps selected cues that carry sentiment.
3. Encode each tweet as TF-IDF unigram and bigram features.
4. Compare Logistic Regression, Linear SVM and Multinomial Naive Bayes on a common test set.
5. Measure accuracy, precision, recall, macro-F1 and confusion matrices.
6. Gauge the effect of preprocessing with a controlled Logistic Regression comparison.
7. Examine how each model's errors divide between the negative and positive classes.
8. Document every step of the pipeline with runnable code so that the results can be regenerated.

5. RELATED WORK

5.1 Distant Supervision and Twitter Sentiment

The Sentiment140 corpus originates with Go, Bhayani and Huang [1], who labelled 1.6 million tweets automatically from the emoticons they contained, so its labels are noisy. Paul et al. [4] paired TF-IDF features with Logistic Regression, SVM and Random Forest models on this benchmark, which shows that conventional classifiers are still in use. Sharaf [5] reported a TF-IDF and Logistic Regression approach on Twitter data; it supports the idea of a reproducible classical baseline while also noting how results can shift across domains. Gupta and Gautam [6] normalised Sentiment140 text before TF-IDF and compared Logistic Regression, SVM, Naive Bayes and ensemble models, a design that speaks directly to the value of careful normalisation and side-by-side evaluation.

5.2 Classifiers, Representations and Preprocessing

On the modelling side, Joachims [2] showed that Support Vector Machines handle high-dimensional text categorisation well, and Manning, Raghavan and Schütze [3] supply the formal background for TF-IDF weighting. Pang, Lee and Vaithyanathan [8] applied Naive Bayes, maximum entropy and SVM classifiers to movie-review polarity, an early demonstration of machine learning for sentiment, and Wang and Manning [9] found that plain linear models over unigram and bigram features remain hard-to-beat baselines. Haddi, Liu and Shi [10] observed that the choice of cleaning and features can shift SVM sentiment accuracy considerably, which is precisely the issue our preprocessing ablation probes on a single, fixed Twitter dataset.

5.3 Transformer-Based Models

Recent work has turned to pre-trained transformers. BERT, from Devlin et al. [11], is a general bidirectional model, whereas BERTweet, from Nguyen, Vu and Nguyen [12], is pre-trained on English tweets and is reported to beat comparable models at text classification. Barbieri et al. [13] released TweetEval, a shared benchmark covering seven Twitter classification tasks, and their results indicate that extra training on tweet text helps. None of these models is evaluated here, because our focus is on reproducible classical baselines; they are left for future work.

5.4 Overall Findings

Taken together, the literature indicates that Sentiment140 is still a practical benchmark for comparisons and that both preprocessing and model choice noticeably change reported scores. Yet many implementations treat cleaning as a single undifferentiated step. We instead make preprocessing an explicit experimental factor and contrast standard cleaning with a context-preserving variant.

6. RESEARCH GAP AND STUDY CONTRIBUTION

The gap we address is not a shortage of classifiers: Logistic Regression, SVM and Naive Bayes are well established, and Sentiment140 has been studied many times. What is missing is a controlled, reproducible comparison in which preprocessing is measured as a component in its own right. Our study departs from common practice as follows:

1. **Dataset:** studies often differ in the data or sample they use, whereas we work with one fixed copy of Sentiment140.
2. **Task definition:** the task is sometimes described loosely; ours is strictly binary, negative versus positive.
3. **Preprocessing:** generic cleaning is typical; we spell out a context-preserving pipeline in full.
4. **Feature extraction:** feature spaces differ from study to study; we use one shared TF-IDF representation of unigrams and bigrams.
5. **Classifier comparison:** splits and settings frequently differ; every classifier here is scored on the same 80,000 test tweets.

6. **Preprocessing effect:** its contribution is seldom isolated; we compare standard and context-preserving cleaning under controlled conditions.
7. **Reproducibility:** implementation details are often left vague; we report the seed, sample size, split and TF-IDF settings.

7. DATASET DESCRIPTION

The supplied ZIP archive holds a single file, training.1600000.processed.noemoticon.csv, with 1,600,000 rows and six columns: sentiment, id, date, query, user and text. Only the labels 0 and 4 occur, standing for negative and positive polarity. The file has no neutral class, so we make no three-class claims.

Table -1: Properties of the Sentiment140 dataset and the experimental split

Property	Value
Dataset	Sentiment140
Total records	1,600,000
Negative	label 0; 800,000 records
Positive	label 4; 800,000 records
Classes	2 (negative, positive)
Fields	sentiment, id, date, query, user, text
Observed date range	6 April 2009 to 16 June 2009
Experimental subset	400,000 tweets; 200,000 per class
Train / test	320,000 / 80,000

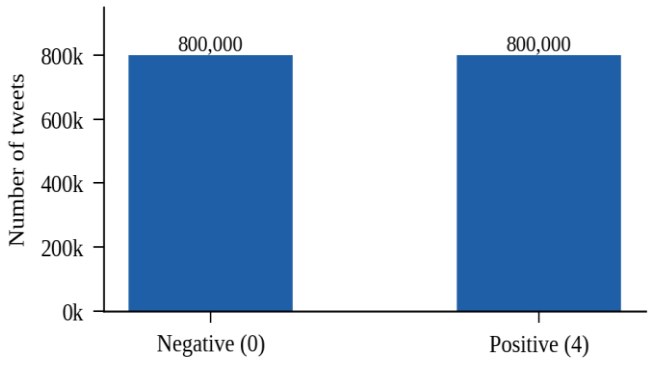


Fig -1: Class distribution in the supplied 1.6-million-tweet dataset

Sentiment140 was built with distant supervision: emoticons in the tweets acted as noisy sentiment labels in place of full manual annotation [1]. This matters for interpretation, because our experiment measures agreement with the dataset's labels and not necessarily with what a human annotator would say about each tweet.

8. METHODOLOGY

Fig. 2 outlines the experimental workflow.

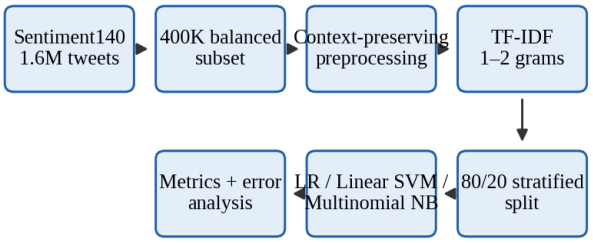


Fig -2: Experimental methodology

8.1 Experimental Sampling

The source data contain 1.6 million tweets, so we drew a balanced subset of 400,000 for the reported experiment: exactly 200,000 negative and 200,000 positive tweets, sampled with random_state = 42. The subset was shuffled and divided 80/20 into training and test portions with stratification, which leaves 80,000 test tweets, 40,000 per class.

8.2 Context-Preserving Preprocessing

The context-preserving pipeline consists of the operations listed below:

Table -2: Context-preserving preprocessing steps

Step	Operation	Rationale
Lowercasing	Lowercase all text	Merge vocabulary entries that differ only in letter case.
URL normalization	Substitute URLTOKEN for each URL	Signal that a link was present without keeping the address.
Mention normalization	Substitute USERTOKEN for each @username	Avoid sparse features tied to individual usernames.
Repeated-character normalization	Shorten runs of three or more identical characters to two	Keep the emphasis but curb vocabulary growth.
Whitespace normalization	Collapse repeated whitespace	Give uniform token boundaries.
Negation preservation	Keep negation words such as not, no and never	Retain words that can flip polarity.

Step	Operation	Rationale
Hashtag preservation	Keep the text of hashtags	Keep possibly informative word and topic cues.

The pipeline deliberately applies no stemming, lemmatization or stop-word removal; the purpose is to test whether keeping more of the original wording helps classification.

8.3 Standard-Cleaning Baseline

The preprocessing ablation uses a conventional baseline that lowercases text, strips URLs and mentions, deletes non-alphabetic characters and removes English stop words. It was run with the same TF-IDF settings and the same Logistic Regression model.

8.4 TF-IDF Feature Extraction

Features are word-level TF-IDF over unigrams and bigrams, built with scikit-learn [7]. The vectorizer was fitted on the training partition alone and then used to transform the test partition, which avoids information leakage. TF-IDF lowers the weight of terms that are common across the corpus and favours those that characterise individual documents [3]. With sublinear term frequency and smoothed inverse document frequency, the weight of term t in document d is

$$\mathrm{tfidf}(t,d) = (1 + \ln \mathrm{tf}(t,d)) \cdot \mathrm{idf}(t) \quad (1)$$

$$\mathrm{idf}(t) = \ln \frac{1+n}{1+\mathrm{df}(t)} + 1 \quad (2)$$

Here tf(t,d) is how often t occurs in d, df(t) is the number of training documents that contain t, and n is the number of training documents. Each document vector is then L2-normalised, which is the scikit-learn default. The settings are listed below.

Table -3: TF-IDF settings

Parameter	Setting
Representation	Word-level TF-IDF
N-grams	Unigram + bigram
Minimum document frequency	2
Maximum features	150,000
Maximum document frequency	0.95
Sublinear TF	True
Normalization	Default scikit-learn TF-IDF normalization

8.5 Classification Models

Table -4: Classifier configurations

Model	Configuration	Purpose
Logistic Regression	C=2.0; solver=liblinear; max_iter=1000	Linear probabilistic baseline for sparse text features.
Linear SVM	C=1.0	Margin-based model well suited to high-dimensional text.
Multinomial Naive Bayes	alpha=1.0	Quick probabilistic baseline for count-like or TF-IDF features.

8.6 Evaluation Measures

Accuracy is the share of test instances that are predicted correctly:

$$\mathrm{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \quad (3)$$

For each class c, precision is the fraction of that class's predictions that are correct, recall is the fraction of its true instances that are found, and F1 is the harmonic mean of the two:

$$P_c = \frac{TP_c}{TP_c+FP_c} \quad (4)$$

$$R_c = \frac{TP_c}{TP_c+FN_c} \quad (5)$$

$$F1_c = \frac{2P_cR_c}{P_c+R_c} \quad (6)$$

Macro-F1 takes the unweighted average of the F1 scores of the negative and positive classes:

$$\text{Macro-F1} = \frac{1}{2} \sum_{c \in \{neg,\, pos\}} F1_c \quad (7)$$

Where: TP, TN, FP and FN denote true positives, true negatives, false positives and false negatives, and c is a sentiment class. Confusion matrices also report, for each class, the counts of correct and incorrect predictions.

8.7 Reproducibility

Table -5: Reproducibility record

Item	Recorded value
Random seed	42
Dataset	Supplied Sentiment140 CSV

Item	Recorded value
Experimental sample	400,000 balanced tweets
Split	80% train / 20% test, stratified
TF-IDF vocabulary	150,000 maximum features
Python	3.13.5
scikit-learn [7]	1.8.0
Reported test size	80,000

8.8 Step-by-Step Implementation with Code

This subsection lists, in order, the steps and Python code that produced the reported results. The snippets follow execution order and share variables, so together they form one runnable script. All random operations use seed 42. The required libraries are pandas, numpy, scikit-learn and matplotlib.

8.8.1 Steps 1 and 2: load the dataset and draw the balanced sample

The CSV has no header row, so column names are supplied by hand and the file is read with latin-1 encoding. Labels 0 and 4 (800,000 records each) are recoded as 0 (negative) and 1 (positive). A balanced sample of 200,000 tweets per class is then drawn with random_state = 42 and shuffled.

```
# Steps 1-2: load Sentiment140, draw balanced sample
import pandas as pd
COLS = ["sentiment", "id", "date", "query", "user", "text"]
df = pd.read_csv("training.1600000.processed.noemoticon.csv",
                 encoding="latin-1", names=COLS)
print(df.shape)                  # (1600000, 6)
print(df["sentiment"].value_counts())  # 0: 800000, 4: 800000

# map labels 0 -> 0 (negative), 4 -> 1 (positive)
df["label"] = (df["sentiment"] == 4).astype(int)

# balanced subset: 200,000 per class, seed 42
sample = (df.groupby("label", group_keys=False)
  .apply(lambda g: g.sample(n=200_000, random_state=42))
  .sample(frac=1, random_state=42)   # shuffle
  .reset_index(drop=True))
print(sample["label"].value_counts()) # 200000 / 200000
```

8.8.2 Steps 3 and 4: text preprocessing

Two cleaning functions are applied to the same sample. The context-preserving one lowercases the text, substitutes URLTOKEN for URLs and USERTOKEN for mentions, shortens runs of three or more repeated characters to two, and collapses whitespace, while keeping negations, hashtags, punctuation and stop words. The standard one lowercases, removes URLs and mentions, deletes every non-alphabetic character and drops English stop words; it is used only in the Step 10 ablation.

```
# Steps 3-4: context-preserving and standard preprocessing
import re
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS

URL_RE = re.compile(r"(https?://\S+|www\.\S+)")
MENTION_RE = re.compile(r"@\w+")
REPEAT_RE = re.compile(r"(.)\1{2,}")   # 3+ repeated chars
SPACE_RE = re.compile(r"\s+")

def clean_context(text: str) -> str:
  """Context-preserving pipeline (all model comparisons)."""
  text = text.lower()
  text = URL_RE.sub(" URLTOKEN ", text)
  text = MENTION_RE.sub(" USERTOKEN ", text)
  text = REPEAT_RE.sub(r"\1\1", text)  # soooo -> soo
  text = SPACE_RE.sub(" ", text).strip()
  # negations, hashtags, punctuation, stop words are kept
  return text

def clean_standard(text: str) -> str:
  """Conventional baseline (preprocessing ablation only)."""
  text = text.lower()
  text = URL_RE.sub(" ", text)
  text = MENTION_RE.sub(" ", text)
  text = re.sub(r"[^a-z\s]", " ", text)  # drop non-alphabetic
  tokens = [t for t in text.split()
        if t not in ENGLISH_STOP_WORDS]
  return " ".join(tokens)

sample["text_context"] = sample["text"].map(clean_context)
sample["text_standard"] = sample["text"].map(clean_standard)
```

8.8.3 Steps 5 and 6: stratified split and TF-IDF features

The sample is split 80:20 with stratification, giving 320,000 training and 80,000 test tweets. The TF-IDF vectorizer uses the settings from Section 8.4 and is fitted on the training tweets only before being applied to the test tweets, which prevents information leakage.

```
# Steps 5-6: stratified 80/20 split, TF-IDF (fit on train only)
from sklearn.model_selection import train_test_split
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

def make_split(text_col):
  return train_test_split(
    sample[text_col], sample["label"], test_size=0.20,
    stratify=sample["label"], random_state=42)

X_tr, X_te, y_tr, y_te = make_split("text_context")
print(len(X_tr), len(X_te))    # 320000 80000

def make_tfidf():
  return TfidfVectorizer(ngram_range=(1, 2), min_df=2,
    max_df=0.95, max_features=150_000,
sublinear_tf=True)

tfidf = make_tfidf()
Xtr = tfidf.fit_transform(X_tr)    # fit ONLY on training tweets
Xte = tfidf.transform(X_te)    # transform test tweets
```

8.8.4 Steps 7 to 9: training, evaluation and confusion matrices

All three classifiers are trained on one training matrix and evaluated on one test matrix. For each model the code prints accuracy, per-class precision, recall and F1, and macro-F1, and it saves a separate confusion-matrix figure.

```
# Steps 7-9: train, evaluate, save confusion matrices
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import (accuracy_score, f1_score,
  classification_report, confusion_matrix)

models = {
 "Logistic Regression": LogisticRegression(
   C=2.0, solver="liblinear", max_iter=1000),
 "Linear SVM": LinearSVC(C=1.0),
 "Multinomial NB": MultinomialNB(alpha=1.0),
}
results = {}
for name, clf in models.items():
  clf.fit(Xtr, y_tr)
  pred = clf.predict(Xte)
  results[name] = dict(
    acc=accuracy_score(y_te, pred),
    macro_f1=f1_score(y_te, pred, average="macro"),
    cm=confusion_matrix(y_te, pred))
  print(name)
  print(classification_report(y_te, pred,
    target_names=["Negative", "Positive"], digits=4))
  # one separate confusion-matrix figure per model
  cm = results[name]["cm"]
  fig, ax = plt.subplots(figsize=(4.6, 4))
  ax.imshow(cm, cmap="Blues")
  for i in range(2):
    for j in range(2):
      ax.text(j, i, f"{cm[i, j]:,}",
          ha="center", va="center")
  ax.set_xticks([0, 1])
  ax.set_xticklabels(["Negative", "Positive"])
  ax.set_yticks([0, 1])
  ax.set_yticklabels(["Negative", "Positive"])
  ax.set_xlabel("Predicted label")
  ax.set_ylabel("Actual label")
  ax.set_title(f"{name}: confusion matrix")
  fig.tight_layout()
  fig.savefig(f"cm_{name.replace(' ', '_').lower()}.png",
        dpi=200)
  plt.close(fig)
```

8.8.5 Step 10: preprocessing ablation

The standard-cleaned text is split with the same seed and stratification, vectorized with identical TF-IDF settings and classified by Logistic Regression. Its accuracy is set against the context-preserving result (77.43% versus 81.12%).

```
# Step 10: preprocessing ablation (Logistic Regression only)
Xs_tr, Xs_te, ys_tr, ys_te = make_split("text_standard")
tf_std = make_tfidf()
A_tr = tf_std.fit_transform(Xs_tr)
A_te = tf_std.transform(Xs_te)
lr = LogisticRegression(C=2.0, solver="liblinear",
            max_iter=1000).fit(A_tr, ys_tr)
p = lr.predict(A_te)
print("Standard cleaning accuracy :", accuracy_score(ys_te, p))
# 0.7743
print("Context-preserving accuracy:",
   results["Logistic Regression"]["acc"])  # 0.8112
```

9. EXPERIMENTAL RESULTS

Every result reported here comes from the supplied dataset and follows the protocol described above. The 80,000-tweet test set played no part in model fitting.

9.1 Overall Performance

Table -6: Overall test-set performance

Model	Accuracy	Macro Precision	Macro Recall	Macro-F1
Linear SVM	79.93%	0.7994	0.7993	0.7993
Logistic Regression	81.12%	0.8113	0.8112	0.8112

Model	Accuracy	Macro Precision	Macro Recall	Macro-F1
Multinomial NB	79.55%	0.7957	0.7955	0.7955

Logistic Regression achieved the best test accuracy (81.12%) and macro-F1 (0.8112). Linear SVM obtained 79.93% accuracy with a macro-F1 of 0.7993, and Multinomial Naive Bayes obtained 79.55% accuracy with a macro-F1 of 0.7955. Logistic Regression's lead over Linear SVM was 1.19 percentage points in accuracy (Fig. 3).



Fig -3: Accuracy and macro-F1 comparison across models

9.2 Class-wise Performance

Table -7: Class-wise performance on the test set

Model	Class	Precision	Recall	F1-score	Support
Linear SVM	Negative	0.8031	0.7932	0.7981	40,000
	Positive	0.7957	0.8055	0.8006	40,000
Logistic Regression	Negative	0.8157	0.8040	0.8098	40,000
	Positive	0.8068	0.8184	0.8125	40,000
Multinomial NB	Negative	0.7881	0.8085	0.7982	40,000
	Positive	0.8034	0.7825	0.7928	40,000

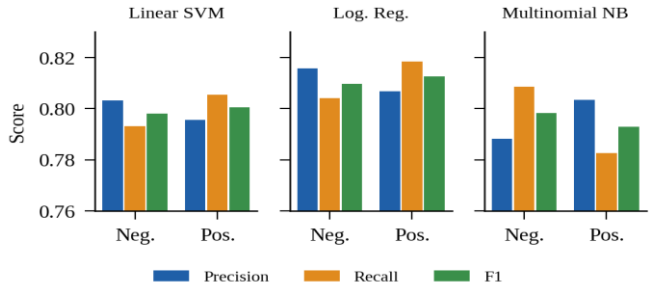


Fig -4: Class-wise precision, recall and F1-score for each classifier

9.3 Confusion Matrices

Figs. 5–7 show, for each classifier, the confusion matrix in counts and its row-normalised version, all computed on the same 80,000-tweet test set.

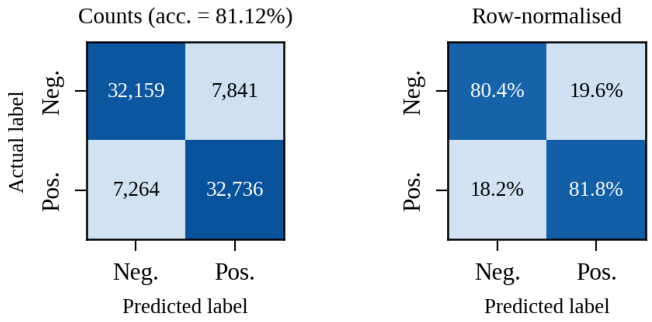


Fig -5: Logistic Regression: confusion matrix (left, counts) and row-normalised confusion matrix (right)

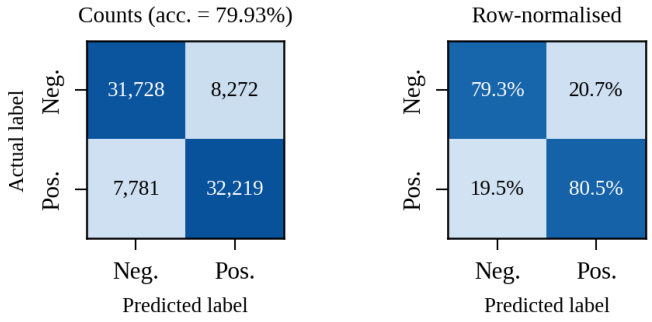


Fig -6: Linear SVM: confusion matrix (left, counts) and row-normalised confusion matrix (right)

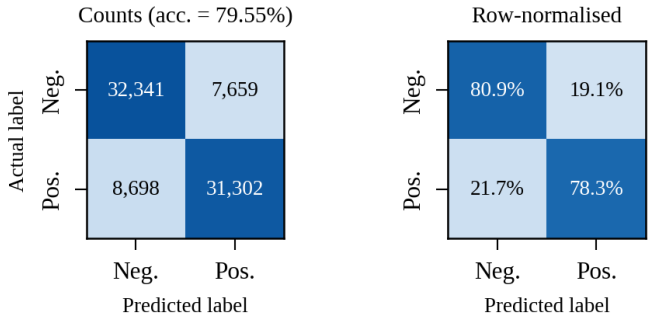


Fig -7: Multinomial NB: confusion matrix (left, counts) and row-normalised confusion matrix (right)

9.4 Preprocessing Ablation

With Logistic Regression, standard cleaning reached 77.43% accuracy and the context-preserving pipeline reached 81.12%, an absolute gain of 3.69 percentage points.

Table -8: Preprocessing ablation (Logistic Regression)

Preprocessing strategy	Accuracy	Macro-F1
Standard cleaning	77.43%	0.7743
Context-preserving cleaning	81.12%	0.8112
Absolute improvement	3.69 pp	0.0369

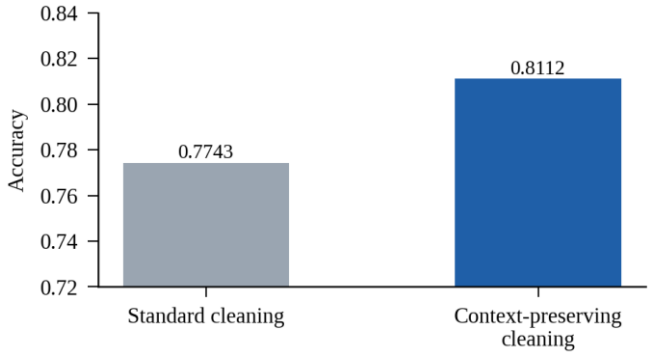


Fig -8: Controlled preprocessing comparison (Logistic Regression)

10. DISCUSSION

Regarding RQ1, conventional TF-IDF classifiers reach roughly 79.55% to 81.12% accuracy on the 80,000-tweet test set. For RQ2, Logistic Regression performed best of the three. For RQ3, the ablation favours the context-preserving approach: retaining contextual information lifted Logistic Regression accuracy by 3.69 percentage points over the standard-cleaning baseline, with identical TF-IDF features and test data.

RQ4 can be read off the confusion matrices (Section 9.3). Logistic Regression misclassified 7,841 of 40,000 negative tweets (19.60%) and 7,264 of 40,000 positive tweets (18.16%). The corresponding Linear SVM figures were 8,272 (20.68%) and 7,781 (19.45%), so for both models the negative class had more errors. Multinomial Naive Bayes reversed this, with 7,659 negative errors (19.15%) against 8,698 positive errors (21.75%). Which class is harder therefore depends on the classifier, and the gap between classes never exceeds about 2.6 percentage points.

RQ5 follows from the same results. No model collapsed onto a single class; each erred in both directions. The gap between negative and positive recall was 1.44 points for Logistic Regression (0.8040 versus 0.8184), 1.23 points for Linear SVM (0.7932 versus 0.8055) and 2.60 points for Multinomial Naive Bayes (0.8085 versus 0.7825). Logistic Regression and Linear SVM leaned slightly toward the positive class, while Multinomial Naive Bayes leaned toward the negative class and was the most asymmetric.

RQ6 compares two effects. With context-preserving preprocessing, accuracy varied by 1.57 points across the classifiers (81.12% for Logistic Regression versus 79.55% for Multinomial Naive Bayes), whereas moving from standard to context-preserving cleaning changed Logistic Regression accuracy by 3.69 points. In this experiment the cleaning choice thus mattered more than the choice of classifier, although the ablation was run for Logistic Regression only and the comparison should be read with that in mind.

For RQ7, Logistic Regression was ahead of Linear SVM by 1.19 points and of Multinomial Naive Bayes by 1.57 points in accuracy, while Linear SVM led Multinomial Naive Bayes by just 0.38 points. Logistic Regression is clearly in front, but the other two are close. Since only one seed and one split were used and no significance test or confidence interval was computed, these differences describe this experiment and are not statistically established.

RQ8 is addressed in Sections 8.1 to 8.8 and Section 15. The seed, sample size, class balance, split ratio, preprocessing rules, TF-IDF settings, hyperparameters and software versions are all recorded, and Section 8.8 provides the code for every step. Archiving the sampled row identifiers and the prediction files, as Section 15 notes, would complete the reproducibility package.

Class-wise results are fairly balanced. Negative and positive recall were 0.8040 and 0.8184 for Logistic Regression, 0.7932 and 0.8055 for Linear SVM, and 0.8085 and 0.7825 for Multinomial Naive Bayes, the one model with higher negative recall. In every case the confusion matrices show errors in both directions and no collapse toward one sentiment class.

The ablation matters especially for social-media NLP. Standard cleaning strips punctuation and stop words, which may remove short function words and polarity cues, whereas the context-preserving pipeline keeps the lexical context and normalises only selected structural noise. Because the two conditions differ in several operations at once, the 3.69-point gain cannot be assigned to any single operation without further one-factor-at-a-time ablations.

These findings must also be read within the limits of Sentiment140: its labels come from automatic distant supervision, its text is Twitter language from 2009, and the supplied training file has only positive and negative classes. The results are therefore no evidence that the models handle neutral sentiment, sarcasm or present-day social-media language reliably.

11. ERROR ANALYSIS

The confusion matrices show both false-positive and false-negative errors. Probable linguistic causes are negation,

sarcasm, ambiguous short messages, informal spelling and context that a single tweet does not provide. A publishable extension should manually review a fixed random sample of each model's errors and sort them with a predefined error taxonomy.

Table -9: Error categories and expected issues

Error category	Example pattern	Expected issue
Negation	not good / never works	Polarity may reverse or weaken.
Sarcasm	great, another broken update	Literal positive words conflict with intended meaning.
Ambiguity	fine / okay / sure	Sentiment depends strongly on context.
Informal spelling	soooo happy / luv this	Character variation increases sparsity.
Short message	lol / wow / hmm	Insufficient lexical context.
Mixed sentiment	love the idea, hate the execution	A single binary label may oversimplify the message.

12. THREATS TO VALIDITY AND LIMITATIONS

1. The experiment uses a 400,000-tweet subset of the 1.6-million-tweet source, so the results say nothing about training on all 1.6 million records.
2. Sentiment140 labels come from distant supervision and may contain label noise [1].
3. The data reflect Twitter language from 2009 and may not match current vocabulary or platform behavior.
4. Only binary sentiment is evaluated, since the supplied data carry negative and positive labels only; neutral sentiment lies outside the task.
5. The preprocessing ablation changes several cleaning decisions at once, so attributing the effect to a single operation needs further controlled experiments.
6. Only classical TF-IDF-based classifiers were run; transformer models are not part of the reported experiment.
7. The results depend on the chosen random seed and experimental subset, even though both the seed and the sampling procedure are documented.

13. FUTURE WORK

1. Rerun the experiment on the full 1.6-million-tweet training set where computing resources allow.
2. Carry out one-factor-at-a-time ablations for negation, hashtags, mentions, URLs and repeated-character normalization.
3. Evaluate transformer models such as BERT or RoBERTa under the same train/test protocol.
4. Add manually annotated datasets to measure performance against human sentiment labels.
5. Extend the task to neutral and mixed sentiment with datasets that truly include those categories.
6. Study sarcasm, irony and domain adaptation for present-day social-media text.
7. Report confidence intervals and repeated-seed runs to quantify variability.

14. CONCLUSION

We presented a reproducible sentiment-analysis experiment on the supplied 1.6-million-tweet Sentiment140 dataset, using a balanced 400,000-tweet subset and an 80,000-tweet stratified test set. Of the TF-IDF classifiers evaluated, Logistic Regression scored highest, with 81.12% accuracy and 0.8112 macro-F1, followed by Linear SVM and Multinomial Naive Bayes. The ablation showed that context-preserving cleaning raised Logistic Regression accuracy by 3.69 percentage points over standard cleaning, with the same features and evaluation settings. Preprocessing is therefore a measurable contributor to sentiment-analysis performance on social media and should be documented with care. Still, because Sentiment140 is historical, automatically labelled and binary, the findings may not transfer to current or neutral/multiclass settings. The study offers a reproducible empirical baseline and a starting point for future transformer-based, multilingual and human-annotated work.

15. DATA AND REPRODUCIBILITY STATEMENT

The experiment ran directly on the file provided by the authors, training.1600000.processed.noemoticon.csv. This paper documents the preprocessing, sampling seed, split ratio, TF-IDF settings and hyperparameters, and Section 8.8 gives the code step by step. For a journal or conference submission, the authors should additionally archive the final source code, the environment specification, the exact sampled row identifiers or the sampling script, and the generated prediction files, so that others can reproduce the results independently.

16. REFERENCES

[1] A. Go, R. Bhayani, and L. Huang, "Twitter Sentiment Classification using Distant Supervision," Stanford University, CS224N Project Report, 2009.

[2] T. Joachims, "Text Categorization with Support Vector Machines: Learning with Many Relevant Features," in Proc. 10th Eur. Conf. Mach. Learn. (ECML), Springer, 1998, pp. 137–142, doi: 10.1007/BFB0026683.

[3] C. D. Manning, P. Raghavan, and H. Schütze, Introduction to Information Retrieval. Cambridge University Press, 2008.

[4] Y. Paul, S. ul Nisa, H. Hussain, and R. Singh, "Twitter Sentiment Analysis Using TF-IDF and Machine Learning Classifiers," Int. J. Recent Innov. Trends Comput. Commun., vol. 11, no. 3, 2023, doi: 10.17762/ijritcc.v11i3.11515.

[5] L. Sharaf, "Sentiment Analysis of Twitter Data Using TF-IDF and Logistic Regression," SSRN, 2025, doi: 10.2139/ssrn.5505839.

[6] D. Gupta and H. Gautam, "Analysis of an Advanced Sentiment Classification Model Using Sentiment140 Dataset with Semantic Text Normalization and Comparative Machine Learning," 2026.

[7] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," J. Mach. Learn. Res., vol. 12, pp. 2825–2830, 2011.

[8] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up? Sentiment Classification using Machine Learning Techniques," in Proc. 2002 Conf. Empirical Methods Natural Lang. Process. (EMNLP), 2002, pp. 79–86, doi: 10.3115/1118693.1118704.

[9] S. Wang and C. D. Manning, "Baselines and Bigrams: Simple, Good Sentiment and Topic Classification," in Proc. 50th Annu. Meeting Assoc. Comput. Linguistics (Vol. 2: Short Papers), Jeju Island, Korea, 2012, pp. 90–94.

[10] E. Haddi, X. Liu, and Y. Shi, "The Role of Text Pre-processing in Sentiment Analysis," Procedia Comput. Sci., vol. 17, pp. 26–32, 2013, doi: 10.1016/j.procs.2013.05.005.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in Proc. NAACL-HLT, Minneapolis, MN, 2019, pp. 4171–4186, doi: 10.18653/v1/N19-1423.

[12] D. Q. Nguyen, T. Vu, and A. T. Nguyen, "BERTweet: A pre-trained language model for English Tweets," in Proc. 2020 Conf. Empirical Methods Natural Lang. Process.: System Demonstrations, 2020, pp. 9–14, doi: 10.18653/v1/2020.emnlp-demos.2.

[13] F. Barbieri, J. Camacho-Collados, L. Espinosa Anke, and L. Neves, "TweetEval: Unified Benchmark and Comparative Evaluation for Tweet Classification," in Findings Assoc. Comput. Linguistics: EMNLP 2020, 2020, pp. 1644–1650, doi: 10.18653/v1/2020.findings-emnlp.148.