ISSN 2722-2039

Sarcasm Detection on Reddit Using Classical Machine Learning and Feature Engineering

Subrata Karmaker [a,1,*]

[a] Department of Mathematics, Technische Universität Chemnitz, Germany
[1] skarmaker.tuc@gmail.com
* corresponding author

ARTICLE INFO

Article history
Received October 5, 2025
Revised November 17, 2025
Accepted December 22, 2025

Keywords
Sarcasm detection
Feature engineering
Text classification
Classical machine learning
Reddit

ABSTRACT

Sarcasm is common on social media, yet difficult for machines to interpret. Its meaning often relies on conversational tone, speaker intent or situational contrast—signals not directly visible in plain text. This study investigates how far one can go in sarcasm detection using only classical machine learning techniques and hand-crafted feature engineering, without relying on neural architecture or contextual information. Using a 100,000-comment stratified subsample of the Self-Annotated Reddit Corpus (SARC 2.0), I combine word-level and character-level TF–IDF representations with simple stylistic features such as length, punctuation use, and uppercase ratios. Four classical classifiers are evaluated: logistic regression, linear support vector machines, multinomial Naive Bayes, and random forests. Despite the context-free design, logistic regression and Naive Bayes reach F1-scores of approximately 0.57 on sarcastic comments, demonstrating that classical approaches capture part of the underlying signal. The full code is included for reproducibility.

1. Introduction

Sarcasm is pervasive on platforms such as Reddit, where people use irony to express humor, criticism or subtle disagreement. Humans typically recognize sarcasm with little effort, drawing on tone, background knowledge, and the surrounding conversation. Automated systems, however, see only the raw text, and a sentence such as “Great job.” may be interpreted as sincere or sarcastic depending on the unseen context. This ambiguity poses a substantial challenge for natural language processing (NLP) models. While modern sarcasm detection research frequently relies on deep learning and transformer-based models, classical machine learning methods remain relevant. They offer transparency, lower computational cost and straightforward interpretability. This study examines the extent to which predictive performance can be achieved using only classical models and feature engineering on reply-only comments from Reddit. The objective is not to compete with state-of-the-art neural models, but rather to establish a clear, reproducible baseline that highlights what can (and cannot) be learned from simple representations of text.

Early work on sarcasm detection focused heavily on lexical signals and supervised learning methods using TF–IDF, sentiment cues, or handcrafted linguistic features. Joshi et al. [1] provide a comprehensive survey of these approaches, noting the inherent difficulty of detecting sarcasm from surface text alone. Later studies incorporated conversational context, showing clear improvements in predictive accuracy. Ghosh et al. [2] demonstrate that using both the reply and the parent comment enables models to resolve ambiguities that would otherwise be impossible. More recent research has explored multimodal sarcasm detection, combining text with images, metadata, or social interactions [3]. Large pre-trained transformers also dominate modern NLP work. The Self-Annotated Reddit Corpus (SARC) introduced by Khodak et al. [4] has become a standard dataset for sarcasm research. The present study intentionally adopts a conservative constraint—using only the reply text—to evaluate the capabilities of classical baselines.

The experiments are founded on the carefully curated, balanced subset of SARC 2.0, an intriguing dataset that showcases a diverse array of sarcastic and non-sarcastic replies, all marked by the insightful contributions of Reddit users. Each entry in this rich collection is comprised of a label denoting the nature of the reply, along with pertinent metadata, the original parent comment, and the reply itself. To ensure a controlled and conducive environment for analysis, only the reply text and the accompanying binary label are utilized in the experiments. From this extensive dataset, a subsample of 100,000 instances is meticulously drawn, stratified by label to maintain a balanced representation of both sarcastic and non-sarcastic comments. Any empty entries or those containing only whitespace are systematically excluded to improve data quality. The final refined dataset is then divided into two parts: an 80% training set used to train the models and a 20% test set reserved for model evaluation.

2. Materials and Method

The feature representation is meticulously crafted using a custom transformer, TextFeatures. This innovative approach seamlessly integrates several key components to enhance the analysis of text data. First, it employs word-level TF-IDF calculations that analyze both unigrams and bigrams, capturing the fundamental building blocks of language usage. Additionally, it incorporates character-level TF-IDF, focusing on 3- to 5-character sequences to capture subtle linguistic nuances. To further enrich the feature set, five distinct stylistic numeric indicators are included: reply length, total word count, frequency of exclamation marks, prevalence of question marks, and the ratio of uppercase letters. These metrics provide valuable insights into the text's stylistic characteristics.

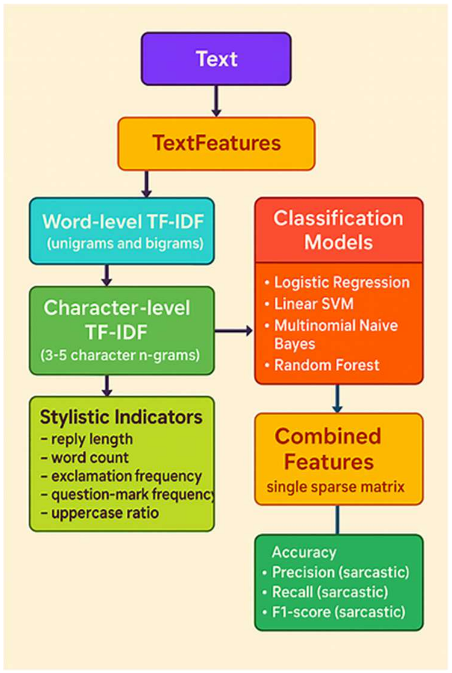

Figure 1. The Architecture of a Sarcasm-detection Pipeline

All these components are skillfully concatenated into a single sparse matrix, forming a comprehensive representation of the data. Leveraging this unified feature set, four classical machine learning models—logistic regression, a linear Support Vector Machine (SVM), multinomial Naive

Bayes, and a random forest—are trained. To evaluate model performance, key metrics such as accuracy, precision, recall, and F1-score are employed, with a focus on the detection of the sarcastic class. This multifaceted approach not only enhances model robustness but also improves the quality of insights derived from the analysis.

2.1 Feature Engineering

Text inputs were transformed using a custom TextFeatures preprocessing module that integrates lexical, subword, and stylistic information into a unified representation. The module produced three complementary feature groups.

2.1.1 Word-Level TF–IDF Features

Word-based term frequency–inverse document frequency (TF–IDF) vectors were computed using unigrams and bigrams. These features capture semantic and contextual patterns associated with sarcastic expression, including common phrase-level constructions and sentiment inversions.

2.1.2 Character-Level TF–IDF Features

To model subword structure and orthographic variation, character n-gram TF–IDF features were extracted using 3–5 character sequences. This representation is effective for capturing elongated spellings, punctuation patterns, and other stylistic markers frequently present in sarcastic text.

2.1.3 Stylistic Indicators

Five handcrafted numeric features were included to represent surface-level stylistic tendencies:

- Total reply length,
- Word count,
- Frequency of exclamation marks,
- Frequency of question marks, and
- Ratio of uppercase characters to total characters.

These indicators provide additional cues related to emphasis, rhetorical tone, and expressive intensity. All feature groups were concatenated into a single high-dimensional sparse matrix, which served as the shared input representation for all downstream models.

2.2 Classification Models

Four classical machine-learning algorithms were trained on the unified feature representation:

- Logistic Regression (L2-regularized),
- Linear Support Vector Machine (SVM),
- Multinomial Naive Bayes, and
- Random Forest Classifier.

These models were selected to provide a diverse set of linear and nonlinear baselines commonly used in text classification tasks. Hyperparameters were tuned using grid search on the training set.

2.3 Evaluation

Model performance was assessed on a held-out test set. Standard classification metrics were computed, including overall accuracy and class-specific precision, recall, and F1-score. Because sarcasm detection is often class-imbalanced, with the sarcastic class typically more challenging to identify, the evaluation emphasized precision, recall, and F1-score for the sarcastic category.

3. Results and Discussion

Table 1 summarizes the performance. Logistic regression and Naive Bayes perform similarly well, each reaching an F1-score of approximately 0.57. The linear SVM performs slightly worse, while the random forest shows no significant advantage over the linear models.

Table 1. Performance of classical models on the SARC 2.0 balanced subset (100k sample).

Model	Accuracy	Precision	Recall	F_1
Logistic Regression	0.564	0.564	0.574	0.569
Linear SVM	0.541	0.542	0.534	0.538
Naive Bayes	**0.565**	0.566	**0.574**	**0.569**
Random Forest	0.558	0.558	0.568	0.563

Figures 1 and 2 display the confusion matrix and ROC curve for the Naive Bayes classifier. The ROC curve achieves an AUC of 0.59.

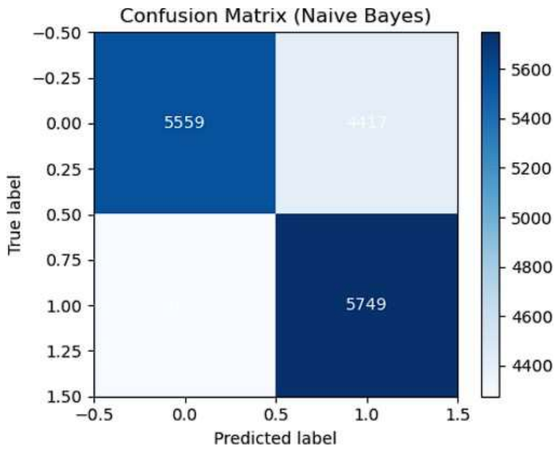

Figure 2. Confusion matrix for the Naive Bayes classifier

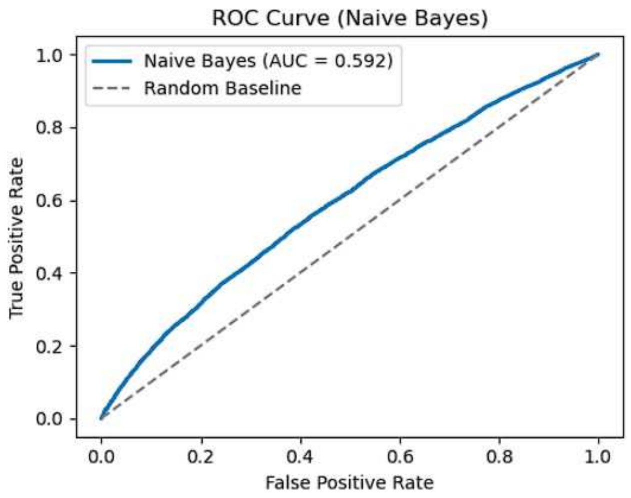

Figure 3. ROC curve for Naive Bayes (solid line) and random baseline (dashed).

The findings reveal that classical machine learning methods, even in the absence of contextual information, can effectively identify distinct stylistic patterns linked to sarcasm. While the performance may be somewhat modest, this is to be anticipated, as many sarcastic remarks hinge on contrasts with earlier communications—contextual nuances that the model lacks access to. Nonetheless, the approach's reproducibility and transparency provide a significant benchmark for foundational work. The TF–IDF representations encapsulate lexical tendencies, while the numeric features highlight exaggerated or atypical stylistic patterns frequently observed in sarcastic exchanges. This interplay between lexical and numeric dimensions enriches our understanding of sarcasm detection in text.

4. Conclusion

This research examines the challenge of sarcasm detection by employing traditional machine learning techniques and carefully engineered features. Using a substantial dataset from Reddit, the study employs a context-free framework, enabling a focused examination of the problem. The findings reveal that both logistic regression and Naive Bayes models yield F1 scores hovering around 0.57. While these results fall short of providing a definitive solution to the complexities of sarcasm detection, they establish a significant baseline for future investigations. Researchers can build upon these findings by integrating contextual elements or exploring advanced neural embeddings. Additionally, to promote transparency and facilitate further exploration, the complete implementation of this study is made available for reproducibility.

Acknowledgements

I thank the authors of the SARC dataset for making their data available to the research community.

References

[1] A. Joshi, P. Bhattacharyya, and M. J. Carman, "Automatic sarcasm detection: A survey," *ACM Comput. Surv.*, vol. 50, no. 5, pp. 1–22, Sep. 2017, doi: 10.1145/3124420.

[2] D. Ghosh, A. Guo, and S. Muresan, "Analyzing sarcasm in conversation context," *Comput. Linguistics*, vol. 43, no. 4, pp. 761–794, Dec. 2017, doi: 10.1162/coli_a_00336.

[3] S. Farabi, T. Ranasinghe, D. Kanojia, Y. Kong, and M. Zampieri, "A survey of multimodal sarcasm detection," in *Proc. 33rd Int. Joint Conf. Artif. Intell. (IJCAI)*, Macao, China, Aug. 2024, pp. 8020–8028, doi:10.24963/ijcai.2024/887.

[4] M. Khodak, N. Saunshi, and K. Vodrahalli, "A large self-annotated corpus for sarcasm," *arXiv:1704.05579*, 2018. [Online]. Available: https://arxiv.org/abs/1704.05579.

Appendix

Python Code

```
from typing import List, Optional
from pathlib import Path
import numpy as np
import pandas as pd
from scipy.sparse import hstack, csr_matrix
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
accuracy_score,
classification_report,
confusion_matrix,
roc_auc_score,
roc_curve,
)
import matplotlib.pyplot as plt
class TextFeatures(BaseEstimator, TransformerMixin):
"""
Custom feature engineering transformer for sarcasm detection.
It combines:- word-level TF-IDF features- character-level TF-IDF features- simple
numeric stylistic features
"""
def __init__(
self,
max_features_word: int = 20000,
max_features_char: int = 10000,
ngram_range_word=(1, 2),
ngram_range_char=(3, 5),
lowercase: bool = True,
stop_words: Optional[str] = "english",
):
self.max_features_word = max_features_word
self.max_features_char = max_features_char
self.ngram_range_word = ngram_range_word
self.ngram_range_char = ngram_range_char
self.lowercase = lowercase
self.stop_words = stop_words
self.word_vectorizer_: Optional[TfidfVectorizer] = None
self.char_vectorizer_: Optional[TfidfVectorizer] = None
def _basic_numeric_features(self, texts: List[str])-> np.
ndarray:
"""
Compute  simple  numeric  features:-  length  in  characters-  number  of  words-
exclamation marks per word- question marks per word- uppercase ratio
"""
lengths = np.array([len(t) for t in texts], dtype=float)
num_words = np.array([len(t.split()) for t in texts], dtype=
float) + 1.0
num_exclam = np.array([t.count("!") for t in texts], dtype=
float)
num_question = np.array([t.count("?") for t in texts], dtype
=float)
num_upper = np.array(
[sum(1 for ch in t if ch.isupper()) for t in texts],
dtype=float,
)

exclam_per_word = num_exclam / num_words
question_per_word = num_question / num_words
upper_ratio = num_upper / lengths.clip(min=1.0)
features = np.vstack(
[lengths, num_words, exclam_per_word, question_per_word,
upper_ratio]
).T
return features
def fit(self, X, y=None):
"""
Fit word and character TF-IDF vectorizers on the training
texts.
"""
if isinstance(X, pd.Series):
texts = X.astype(str).tolist()
elif isinstance(X, (list, np.ndarray)):
texts = [str(t) for t in X]
else:
raise ValueError(f"Unsupported input type for
TextFeatures: {type(X)}")
self.word_vectorizer_ = TfidfVectorizer(
max_features=self.max_features_word,
ngram_range=self.ngram_range_word,
lowercase=self.lowercase,
stop_words=self.stop_words,
sublinear_tf=True,
)
self.char_vectorizer_ = TfidfVectorizer(
max_features=self.max_features_char,
ngram_range=self.ngram_range_char,
lowercase=self.lowercase,
analyzer="char",
sublinear_tf=True,
)
```

```
self.word_vectorizer_.fit(texts)
self.char_vectorizer_.fit(texts)
return self
def transform(self, X):
"""
Transform texts into a combined sparse feature matrix.
"""
if isinstance(X, pd.Series):
texts = X.astype(str).tolist()
elif isinstance(X, (list, np.ndarray)):
texts = [str(t) for t in X]
else:
raise ValueError(f"Unsupported input type for
TextFeatures: {type(X)}")
word_tfidf = self.word_vectorizer_.transform(texts)
char_tfidf = self.char_vectorizer_.transform(texts)
dense_feats = self._basic_numeric_features(texts)
dense_sparse = csr_matrix(dense_feats.astype(float))
return hstack([word_tfidf, char_tfidf, dense_sparse])
def make_logreg_model()-> Pipeline:
"""
Logistic regression pipeline.
"""
return Pipeline(
[
("features", TextFeatures()),
("clf", LogisticRegression(max_iter=500, n_jobs=-1)),
]
)
def make_svm_model()-> Pipeline:
"""
Linear SVM pipeline.
"""
return Pipeline(
[
("features", TextFeatures()),
("clf", LinearSVC(max_iter=5000, dual="auto")),
]
)
def make_nb_model()-> Pipeline:
"""
Multinomial Naive Bayes pipeline.
"""
return Pipeline(
[
("features", TextFeatures()),
("clf", MultinomialNB()),
]
)
def make_rf_model()-> Pipeline:
"""
Random Forest pipeline.
"""
return Pipeline(
[
("features", TextFeatures()),
("clf", RandomForestClassifier(
n_estimators=150,
max_depth=None,
n_jobs=-1,
random_state=42,
)),
]
)
def get_all_models():
"""
Return a dictionary of all models to be evaluated.
```

```
"""
return {
"logreg": make_logreg_model(),
"svm": make_svm_model(),
"nb": make_nb_model(),
"rf": make_rf_model(),
}
# =======================
# Main experiment script
# =======================
RANDOM_STATE = 42
data_path = Path("train-balanced.csv.bz2")
# Load the balanced SARC subset
df = pd.read_csv(
data_path,
compression="bz2",
sep="\t",
header=None,
engine="python",
quoting=3,
on_bad_lines="skip",
)
# Subsample for computational efficiency
df_small = df.sample(n=100000, random_state=RANDOM_STATE)
# Label and reply text
y = df_small.iloc[:, 0].astype(int)
X = df_small.iloc[:, 9].astype(str)
# Remove empty / whitespace-only texts
mask = X.str.strip().astype(bool)
X = X[mask]
y = y[mask]
# Stratified train-test split
X_train, X_test, y_train, y_test = train_test_split(
X,
y,
test_size=0.2,
random_state=RANDOM_STATE,
stratify=y,
)
models = get_all_models()
metrics_summary = {}
# Train and evaluate each model
for name, model in models.items():
print("=" * 60)
print(f"Training model: {name}")
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
acc = accuracy_score(y_test, y_pred)
report_dict = classification_report(
y_test, y_pred, digits=3, output_dict=True
)
# Print detailed classification report
print("Accuracy:", acc)
print(classification_report(y_test, y_pred, digits=3))
# Store key metrics for the sarcastic class (label "1")
metrics_summary[name] = {
"accuracy": acc,
"precision_sarcastic": report_dict["1"]["precision"],
"recall_sarcastic": report_dict["1"]["recall"],
"f1_sarcastic": report_dict["1"]["f1-score"],
}
# Summary table for all models
metrics_df = pd.DataFrame(metrics_summary).T
print("\nSummary metrics:")
print(metrics_df)
# ========== Confusion Matrix & ROC Curve for Naive Bayes ==========
nb_model = models["nb"]
```

```
y_pred_nb = nb_model.predict(X_test)
# Confusion matrix
cm = confusion_matrix(y_test, y_pred_nb)
print("\nConfusion matrix (Naive Bayes):")
print(cm)
plt.figure(figsize=(5, 4))
plt.imshow(cm, cmap="Blues")
plt.title("Confusion Matrix (Naive Bayes)")
plt.xlabel("Predicted label")
plt.ylabel("True label")
plt.colorbar()
# Add counts inside the cells
for i in range(cm.shape[0]):
for j in range(cm.shape[1]):
plt.text(
j, i, cm[i, j],
ha="center",
va="center",
color="white" if cm[i, j] > cm.max() / 2.0 else "black",
)
plt.tight_layout()
plt.show()
plt.savefig("confusion_matrix_nb.png", dpi=300, bbox_inches="tight")
plt.close()
# ROC curve for Naive Bayes
y_score_nb = nb_model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_score_nb)
auc = roc_auc_score(y_test, y_score_nb)
print(f"\nAUC (Naive Bayes): {auc:.3f}")
plt.figure(figsize=(5, 4))
# Model ROC line
plt.plot(fpr, tpr, label=f"Naive Bayes (AUC = {auc:.3f})", linewidth
=2)
# Random baseline line
plt.plot([0, 1], [0, 1], "--", color="gray", label="Random baseline"
)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve (Naive Bayes)")
plt.legend()
plt.tight_layout()
plt.show()
plt.savefig("roc_curve_nb.png", dpi=300, bbox_inches="tight")
plt.close()
```