Sentiment Analysis
(Logistic Regression, Naive Bayes)

Christopher Ho
University of Oregon
cho10@uoregon.edu

Abstract

In this paper, we present an approach to Twitter sentiment analysis using a text classification pipeline that integrates traditional machine learning techniques and advanced natural language processing (NLP) features. We leverage a variety of feature extraction methods, including Term Frequency-Inverse Document Frequency (TF-IDF), to capture the semantic richness and sentiment nuances of tweets.

Our methodology involves preprocessing raw tweet data to remove noise, followed by feature extraction through a FeatureUnion that combines multiple representations of the text. This includes TF-IDF vectors to account for the significance of words, and features such as text length, and part-of-speech tagging.

We implement and compare several classifiers, including Multinomial Naive Bayes (MNB) and Logistic Regression (LR), each optimized through hyperparameter tuning using GridSearchCV. Our experiments demonstrate that incorporating diverse features enhances model performance. Specifically, our best models achieve notable improvements in accuracy and robustness over baselines that rely solely on traditional text representations.

1 Introduction

In the age of digital communication, social media platforms like Twitter have become powerful tools for expressing public opinion on a myriad of topics, ranging from politics and global events to consumer preferences and brand perception. Analyzing the sentiment expressed in tweets can provide valuable insights for businesses, researchers, and policymakers, enabling them to understand public sentiment and react accordingly. However, the inherent noise, brevity, and informal language in tweets present significant challenges for accurate sentiment analysis.

The primary challenge addressed is the accurate classification of sentiment in tweets. Sentiment analysis of Twitter data is complicated by the platform's unique characteristics, such as the 280-character limit, widespread use of slang, abbreviations, emojis, and the rapid evolution of language. Traditional text classification methods often struggle to capture the nuanced sentiment expressed in such diverse and dynamic text. Additionally, tweets can express sentiment through various linguistic constructs, including word choice, syntax, and even the use of hashtags and mentions, making it difficult to rely solely on basic text representations.

As such we will utilize preprocessing of raw tweet data tor remove noise and ready data for feature extraction. We will then employ a FeatureUnion to integrate multiple text representations to pass to separate models for evaluation. We will be utilizing a Logistic Regression model and a Naive Bayes model.

2 Related Work

The field of sentiment analysis, particularly in the context of social media data like Twitter, has seen significant research and development over the years. This section reviews the key aspects of previous work related to models, features, datasets, and applications, highlighting their contributions and limitations, and situating our current work within this body of knowledge.

Sentiment analysis models primarily relied on classical machine learning algorithms such as Naive Bayes, Support Vector Machines (SVM), and Logistic Regression. These models have been effective due to their simplicity and interpretability. Pang et al. (2002) demonstrated the effectiveness of these models in sentiment classification by employing unigram and bigram features, achieving notable accuracy on movie reviews .

3 Methods

Data Preprocessing: Cleaning and preparing the tweet data to ensure it is suitable for feature extraction and modeling.

Feature Extraction: Utilizing a diverse set of features to capture the richness and nuances of the text.

Modeling: Applying and optimizing machine learning classifiers to accurately predict sentiment.

Evaluation: Assessing model performance using appropriate metrics to ensure robustness and accuracy.

The preprocessing step involves several tasks to clean and standardize the tweet data:

Tokenization: Splitting tweets into individual tokens or words.

Stopword Removal: Eliminating common stopwords that do not contribute significantly to sentiment.

Noise Removal: Removing URLs, mentions, hashtags, special characters, and excessive punctuation.

Lowercasing: Converting all text to lowercase to ensure uniformity.

Normalization: Handling repeated characters and correcting common slang and abbreviations. Feature Extraction

We employ a FeatureUnion to combine multiple feature extraction techniques, ensuring a comprehensive representation of the text:

TF-IDF Vectors: Term Frequency-Inverse Document Frequency (TF-IDF) is used to represent the importance of words in the corpus. This helps in emphasizing significant words while downplaying common ones.

Text Length and Structure: Features such as text length, the presence of exclamation marks, and capitalization patterns. These features capture additional signals of sentiment, such as emphasis and excitement.

Part-of-Speech (POS) Tagging: Counting the occurrences of specific POS tags (e.g., adjectives) that are indicative of sentiment. Certain POS tags, such as adjectives, are strongly associated with sentiment expression.

We apply and compare several machine learning classifiers, optimizing them through hyperparameter tuning using GridSearchCV:

Multinomial Naive Bayes (MNB): A probabilistic classifier based on Bayes' theorem, suitable for discrete features like TF-IDF.

Key Parameters: Smoothing parameter alpha.

Logistic Regression (LR): A linear model for binary classification, extended to handle multi-class classification.

Key Parameters: Regularization strength C, solver type, and penalty. Deep Learning Models: For comparison, we include a neural network model with embeddings.

Key Parameters: Number of layers, units per layer, activation functions, and dropout rates. Evaluation

We evaluate our models using standard metrics such as accuracy, precision, recall, and F1-score. These metrics provide a comprehensive understanding of model performance, capturing both the correctness of the predictions and the balance between precision and recall.

Integrated Feature Extraction Pipeline: Our method combines multiple feature extraction techniques, including TF-IDF, parts of speech, and text length to capture the richness of tweet data. We implement a detailed preprocessing pipeline tailored to the unique characteristics of Twitter data, ensuring the quality and consistency of the input text. Through extensive hyperparameter tuning, we optimize several classifiers, demonstrating the effectiveness of traditional machine learning algorithms in conjunction with advanced features. Our approach provides a scalable and adaptable framework for sentiment analysis, capable of being applied to various domains beyond Twitter.

4 Experiments

To ensure that our approach can be replicated by other researchers, we provide detailed descriptions of our experimental setup, hyperparameters, and the values used in our methods. We also compare our method to several baselines and simplified versions to evaluate its performance.

We utilize the train.csv and testdata.manual.2009.06.14.csv from https://www.kaggle.com/datasets/abhi8923shriv/sentiment-analysis-dataset for our experiment, which contains tweets labeled as positive, negative, or neutral. The data is preprocessed to remove noise and prepare it for feature extraction and modeling.

Training Set: 70% of the data

Validation Set: 10% of the data

Test Set: 20% of the data

Preprocessing:

Model	Accuracy	Precision	Recall	F1
LR Base	0.78	0.80	0.77	0.77
MNB Base	0.77	0.81	0.75	0.75
LR Preprocessing/Stop Words	0.80	0.80	0.79	0.80
MNB Preprocessing/Stop Words	0.78	0.82	0.75	0.77
LR POS/ Length of Text	0.81	0.81	0.80	0.81
MNB POS/ Length of Text	0.76	0.84	0.73	0.75
LR Base Tune	0.82954	0.83	0.82	0.82
MNB Base Tune	0.79989	0.82	0.78	0.79
LR Tune	0.82972	0.83	0.82	0.83
MNB Tune	0.80116	0.83	0.78	0.80

Table 1: Results from several rounds of testing.

Tokenization: Splitting tweets into individual tokens using NLTK's word_tokenize.

Stopword Removal: Using NLTK's predefined list of stopwords.

Noise Removal: Removing URLs, mentions,special characters, and excessive punctuation.

Lowercasing: Converting all text to lowercase.

Normalization: Handling repeated characters by limiting them to a maximum of two consecutive occurrences.

We employ a combination of feature extraction techniques:

TF-IDF Vectors: Using TfidfVectorizer from scikit-learn.

Text Length: Length of the tweet.

Modeling We compare multiple machine learning classifiers:

Multinomial Naive Bayes (MNB):

Hyperparameters: alpha values nb__alpha: [0.01, 0.04, 0.07, 0.1, 0.14, 0.16, 0.19, 0.22, 0.25, 0.28, 0.3, 0.34, 0.38, 0.4, 1, 10, 100] => 0.16

Logistic Regression (LR):

Hyperparameters: C values 'log_reg__C': [0.01, 0.04, 0.1, 0.7, 1, 1.1, 1.2, 1.3, 1.32, 1.36, 1.37, 1.38, 1.39, 1.40, 1.41, 1.42, 1.43, 1.44, 1.48, 1.5, 1.6, 10, 100] => 1.41

solver (liblinear) log_reg__solver: [newton-cg, lbfgs, liblinear, sag, saga] => liblinear

Selection: GridSearchCV with 5-fold cross-validation.

We use the following metrics to evaluate model performance:

Accuracy, Precision, Recall, F1-Score, Results, Baseline Comparison

Baseline 1: TF-IDF with Multinomial Naive Bayes.

Naive Bayes Accuracy: 0.77

Baseline 2: Word2Vec embeddings with Logistic Regression.

Logistic Regression Accuracy: 0.78

(Results on Table 1)

Key Findings Our Method vs. Baselines: Our method significantly outperforms the baselines, demonstrating the effectiveness of combining multiple feature extraction techniques. Logistic Regression vs. MNB: Logistic Regression consistently outperforms Multinomial Naive Bayes across different feature sets, likely due to its ability to handle complex relationships between features.

5 Conclusion

Custom features such as stopwords, POS, and text length significantly contribute to model performance, highlighting the importance of feature engineering. Traditional models like Logistic Regression and MNB provide a good balance of accuracy and computational efficiency. Finally hyperparameter tuning using GridSearchCV is helpful for optimizing model performance.

Our experimental evaluation demonstrates that combining traditional machine learning techniques with advanced feature extraction methods enhances the accuracy and robustness of Twitter sentiment analysis. By integrating TF-IDF vectors, and custom features, our approach outperforms standard baselines and offers a scalable solution for sentiment classification in social media. Future work could explore more sophisticated deep learning architectures and further refinement of feature engineering techniques to address the remaining challenges.

6 References

CS 410/510 - Natural Language Processing, Spring 2024. Available at: https://classes.cs.uoregon.edu/24S/cs410nlp/

Shrivastava, A. (2021) Sentiment Analysis Dataset, Kaggle. Available at: https://www.kaggle.com/datasets/abhi8923shriv/sentiment-analysis-dataset (Accessed: 11 June 2024).

7 Extra

We had intended to implement vaderSentiment and Word2Vec as features as well but could not generate results for MNB due to output being incompatible. The results for Logistic Regression look promising though. Combined they had achieved:

Accuracy = 0.84 : Precision = 0.84 : Recall = 0.83 : F1 = 0.84

All prior to hyperparameter tuning.

On the Following page is the associated code with implementing Word2Vec and vaderSentiment.

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
import gensim.downloader as api
from gensim.models import Word2Vec

word2vec_model = api.load('word2vec-google-news-300')

class Word2VecVectorizer(BaseEstimator, TransformerMixin):
  def __init__(self, model, vector_size=300):
      self.model = model
      self.vector_size = vector_size

   def fit(self, X, y=None):
      return self

   def transform(self, X):
    return np.array([self._document_vector(doc) for doc in X])

   def _document_vector(self, doc):
    # Remove words not in the Word2Vec vocabulary
    words = [word for word in doc.split() if word in self.model]
    if not words:  # If none of the words are in the vocabulary, return a zero vector
      return np.zeros(self.vector_size)
    # Average the vectors of the words in the document
    return np.mean([self.model[word] for word in words], axis=0)

class SentimentLexiconExtractor(BaseEstimator, TransformerMixin):
   def fit(self, X, y=None):
      return self

   def transform(self, X):
    vader = SentimentIntensityAnalyzer()
    return np.array([[vader.polarity_scores(doc)['compound']] for doc in X])

('word2vec', Word2VecVectorizer(model=word2vec_model)),
('sentiment_lexicon', SentimentLexiconExtractor()),
VADER
```