# PROJECT REPORT

Project Title

# SENTIMENT ANALYSIS OF MOVIES REVIEWS

**by**

SATYJEET KUMAR 202410116100187
SAURABH KUMAR 202410116100188
PRIYAM SHARMA 202410116100152

**Session:2024-2025 (II Semester)**

Under the supervision of

**KOMAL SALGOTRA**
**(Assistant Professor)**

**KIET Group of Institutions, Delhi-NCR, Ghaziabad**


**DEPARTMENT OF COMPUTER APPLICATIONS**
**KIET GROUP OF INSTITUTIONS, DELHI-NCR,**
**GHAZIABAD-201206**
( 2024- 2025)

# ABSTRACT

This project is developed on the concept of analyzing film reviews to know if they are good or bad. Today, people post their views about movies on the web, and reading all the reviews can be laborious and troublesome. Therefore, this project facilitates knowing the general view by means of a program that can read the reviews and inform if they are good or bad. It employs simple methods of text processing and machine learning for this purpose.

The primary objective of the project is to facilitate the ease of knowing what others perceive about a movie without reading all reviews manually. This project aims to simplify the process of understanding public perception about a movie by giving a faster and intelligent means of analyzing huge amounts of reviews without having to read them individually.

This project can benefit not just ordinary users but also filmmakers, critics, and sites that depend on user input.

# TABLE OF CONTENTS

# INTRODUCTION

In the modern digital age, individuals willingly post their views and experiences on the internet, primarily about movies. Whether a movie is good or not, individuals tend to post reviews on various websites like review sites, blogs, and social media. Reviews assist other people in deciding whether they should watch a specific movie or not. Reading through several hundreds or thousands of reviews may be challenging and time-consuming.

To address this issue, sentiment analysis of movie reviews has been the concern of this project. Sentiment analysis is an approach that identifies the emotional sense behind the content. Here, it is utilized in order to identify whether a review is positive or negative. All this is performed with the aid of simple text processing and machine learning methods.

The system receives a large quantity of movie reviews as input, processes them, and makes a prediction of the overall sentiment. It assists users in quickly knowing what most people feel about a movie, without having to read every review separately. This project is not only beneficial for movie viewers, but also for filmmakers, critics, and websites that depend on user reviews.

## 2. Literature Review

Sentiment analysis, or opinion mining, is a subfield of natural language processing (NLP) that deals with detecting and extracting the emotional tone of text. It is commonly applied to analyzing public opinions, e.g., product and movie reviews, and social media postings. The lexicon-based technique, one of the first that have been used for sentiment analysis, entails using predefined positive or negative words to decide on the sentiment. This is a straightforward approach but one with limitations, particularly in sarcasm or where sentiment words are not forthcoming.

Machine learning-based approaches have now gained greater usage. These entail training models using labeled reviews, employing algorithms such as Support Vector Machines (SVM) and Naive Bayes. Pang et al. (2002) showed that machine learning models could perform better than lexicon-based approaches by detecting patterns within the data.

With the onset of deep learning, methods such as Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks have been employed in order to discover intricate patterns and context within the reviews. The models, according to Zhang et al. (2018), perform better compared to conventional means and function properly with big-data.

# 3. Project / Research Objective

**Project Objective**

- Create an automatic movie review classification system as positive or negative based on sentiment analysis methods.
- Enhance efficiency in processing large amounts of movie reviews for faster public opinion insights.
- Enable users to make quicker movie decisions without having to read all the reviews.
- Utilize contemporary machine learning and deep learning methods for increased accuracy.
- Offer insights to film producers, critics, and film websites regarding the public's opinion.

**Research Objective**

- Discuss various sentiment analysis methods (lexicon-based, machine learning, deep learning) for movie reviews.
- Compare sentiment analysis models on issues such as sarcasm, mixture of sentiments, and domain-specific language.
- Discuss binary and multiclass sentiment classification to gain deeper understanding of movie reviews.
- Discuss popular datasets (IMDb, Rotten Tomatoes) and challenges in movie review analysis.
- Contribute to enhancing sentiment analysis methods for large-scale movie review analysis·

# 4. Project Flow / Research Methodology

## Project Flow

1. **Data Collection:** Collect movie reviews from sites such as IMDb or Rotten Tomatoes.
2. **Data Preprocessing:** Clean the data (stop words, special characters removed), and perform tokenization.
3. **Feature Extraction:** Transform text into numerical data using methods such as TF-IDF or word embeddings.
4. **Model Training:** Train models (Naive Bayes, SVM, LSTM, CNN) on tagged data.
5. **Model Evaluation:** Assess performance using measures (accuracy, precision, recall).
6. **Deployment:** Deploy the model for users to enter reviews and receive sentiment predictions.

## Research Methodology

1. **Literature Review:** Review existing sentiment analysis methods (lexicon-based, machine learning, deep learning).
2. **Data Collection:** Utilize labeled datasets (IMDb, Rotten Tomatoes) for training.
3. **Method Selection:** Contrast sentiment analysis methods (lexicon, machine learning, deep learning).
4. **Experimentation:** Train models and address challenges such as sarcasm and mixed sentiments.
5. **Evaluation:** Evaluate models using performance metrics (accuracy, precision).
6. **Challenges & Solutions:** Address challenges such as sarcasm, contradictions, and domain-specific language.
7. **Conclusion:** Present findings and suggest future enhancements.

## 5. Project / Research Outcome

1. **Sentiment Classification:** Built a system to classify movie reviews as positive or negative using machine learning and deep learning models.
2. **Improved Accuracy:** Achieved high accuracy with models like LSTM and CNN, outperforming Naive Bayes and SVM.
3. **Complex Sentiments Handling:** Managed sarcasm, mixed sentiments, and domain-specific language by fine-tuning models.
4. **Stakeholder Insights:** Provided valuable insights for filmmakers, critics, and audiences by analyzing public sentiment.
5. **Practical Use:** Created an easy interface to quickly gauge movie sentiment without reading each review.
6. **Research Contribution:** Compared methods, highlighting strengths and weaknesses, and suggesting future improvements.
7. **Scalability:** Demonstrated the system's ability to handle large datasets like IMDb and Rotten Tomatoes for real-world use.

# 6.Sentiment Analysis of Movies Reviews

```
# Core
import spacy
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

%matplotlib inline
from matplotlib.colors import ListedColormap
import seaborn as sns
sns.set(style="whitegrid", rc={"axes.edgecolor":"#383838", "grid.color": "#808080"}, font_scale=1.6)
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=UserWarning)
from bs4 import BeautifulSoup
import re

# sklearn
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics import f1_score, classification_report, ConfusionMatrixDisplay
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression

# Color scheme
my_colors = ["#F5DA7F", "#EDBA63", "#E48E4B", "#DE5A36", "#E32F1E", "#CD1C0F"]
CMAP1 = ListedColormap(my_colors)
print("Notebook Color Scheme:")
sns.palplot(sns.color_palette(my_colors))
plt.show()
```

```
Notebook Color Scheme:
```

```
# Load data into a pandas data frame
data = pd.read_csv("/content/IMDB.csv")

# Preview data frame and shape
print(data.shape)
data.head()
```

```
(50000, 2)
```

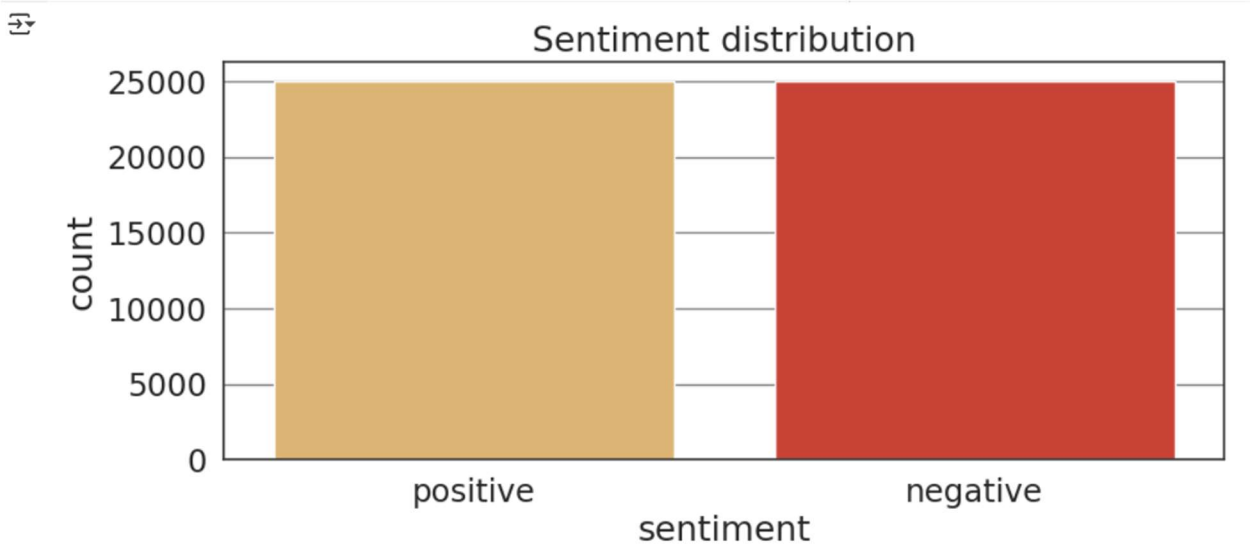
| | review | sentiment |
|---|---|---|
| 0 | One of the other reviewers has mentioned that ... | positive |
| 1 | A wonderful little production. <br /><br />The... | positive |
| 2 | I thought this was a wonderful way to spend ti... | positive |
| 3 | Basically there's a family where a little boy ... | negative |
| 4 | Petter Mattei's "Love in the Time of Money" is... | positive |

```
# Print first review
print("Sentiment:", data.loc[0,"sentiment"])
print("\nReview:", data.loc[0,"review"])
```

```
Sentiment: positive

Review: One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked.
```

```
plt.figure(figsize=(10,4))
sns.countplot(data=data, x="sentiment", palette = [my_colors[1],my_colors[4]])
plt.title("Sentiment distribution")
plt.show()
```



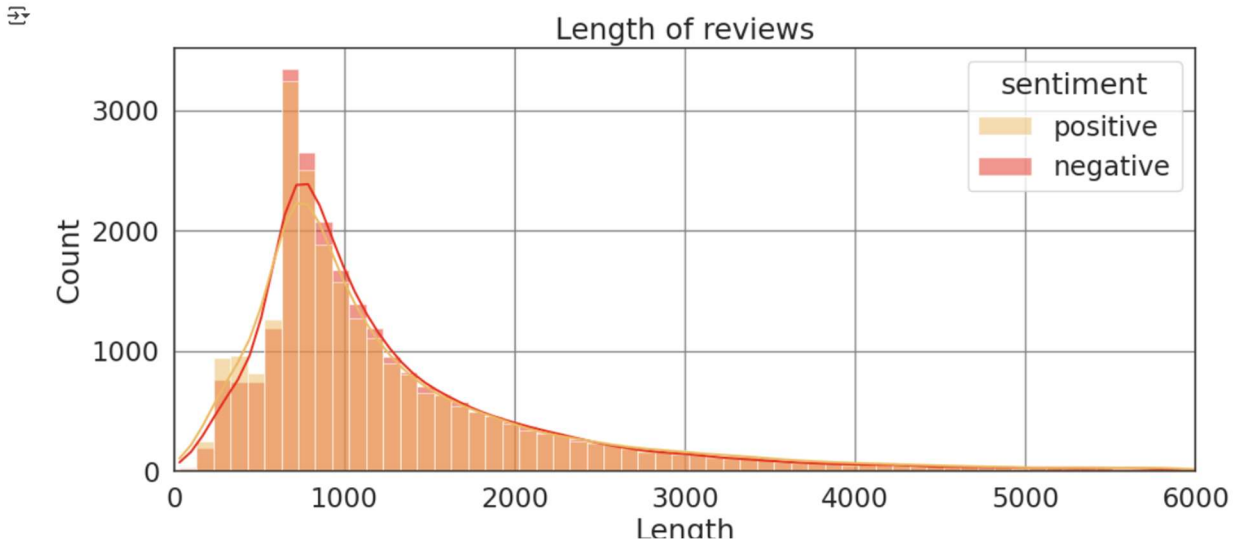

```
# Print target distribution
data["sentiment"].value_counts()
```

| sentiment | count |
|---|---|
| positive | 25000 |
| negative | 25000 |

dtype: int64

```
# Calculate length of each review
len_data = pd.concat([data.loc[:,"review"].map(lambda x:len(x)),data.loc[:,"sentiment"]],axis=1)

# Plot distribution
plt.figure(figsize=(12,5))
sns.histplot(data=len_data, x="review", hue="sentiment", palette = [my_colors[1],my_colors[4]],
binwidth=100, kde=True)
plt.title("Length of reviews")
plt.xlabel("Length")
plt.xlim([0,6000])
plt.show()
```




```
print("Average positive review length:",
len_data.loc[len_data["sentiment"]=="positive","review"].mean())
print("Average negative review length:",
len_data.loc[len_data["sentiment"]=="negative","review"].mean())
```

```
Average positive review length: 1324.79768
Average negative review length: 1294.06436
```

```
# Pre-processing
#3.1 Clean data
#Let's clean the data by removing html tags, which can be done by using the BeautifulSoup library. We'll also remove urls and special characters.
def remove_html_tags(text):
    soup = BeautifulSoup(text,"html.parser")
    return soup.get_text()

def remove_urls(text):
    return re.sub("http\S+","",text)

def remove_special_characters(text):
    return re.sub("[^A-Za-z0-9 ]+","",text)

def clean_text(text):
    text = remove_html_tags(text)
    text = remove_urls(text)
    text = remove_special_characters(text)
    return text
%%time

# Clean entire dataset
data["review"] = data["review"].apply(clean_text)
```

```
CPU times: user 10.2 s, sys: 108 ms, total: 10.3 s
Wall time: 11.8 s
```

```
# Print first review after data cleaning
data.loc[0,"review"]
#3.2 Encode target
#We need to convert the target to integers. We map positive to 1 and negative to 0.

data["sentiment"] = data["sentiment"].replace("positive",1).replace("negative",0)
#3.3 Split data
#We need to create a test set to be able to evaluate our models.

# Split features and labels
X = data["review"]
y = data["sentiment"]

# Split train and test set
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=0,stratify=y)

# Print train target distribution
print(y_train.value_counts())
```

```
sentiment
1    20000
0    20000
Name: count, dtype: int64
```

```
#3.4 Tokenization
#We're going to build a custom tokenizer using spacy. We'll remove stop words (i.e. common words with little meaning) and apply lemmatization
#(i.e. convert words to dictionary form), which are common NLP preprocessing techniques. Furthermore, we only keep tokens if they are an adjective,
#proper noun or verb to reduce the size of the vocabulary.

# Load English language model
nlp = spacy.load("en_core_web_sm")

# Disable named-entity recognition and parsing to save time
unwanted_pipes = ["ner", "parser"]

# Custom tokenizer using spacy
def custom_tokenizer(doc):
   with nlp.disable_pipes(*unwanted_pipes):
      return [t.lemma_ for t in nlp(doc) if not t.is_stop and not t.is_space and t.pos_ in ["ADJ","NOUN","VERB"]]

%%time

# Bag-of-Words vectorizer
vectorizer = CountVectorizer(tokenizer=custom_tokenizer)

# Fit and transform train data
X_train_bow = vectorizer.fit_transform(X_train)

# Transform test data
X_test_bow = vectorizer.transform(X_test)

# Print vocab size
print("BoW vocabulary size:", X_train_bow.shape[1])
```

```
BoW vocabulary size: 133448
CPU times: user 16min 34s, sys: 2.81 s, total: 16min 37s
Wall time: 16min 48s
```

```
%%time

# TF-IDF vectorizer
vectorizer = TfidfVectorizer(tokenizer=custom_tokenizer)

# Fit and transform train data
X_train_tfidf = vectorizer.fit_transform(X_train)

# Transform test data
X_test_tfidf = vectorizer.transform(X_test)

# Print vocab size
print("TF-IDF vocabulary size:", X_train_tfidf.shape[1])
```

```
TF-IDF vocabulary size: 133448
CPU times: user 16min 26s, sys: 2.56 s, total: 16min 29s
Wall time: 16min 36s
```

```
#4. Modelling with Naive Bayes¶
#4.1 Using BoW vectors
#Our first model will be Naive Bayes on the Bag-of-Words (BoW) vectorized data. We'll also tune the additive smoothing hyperparameter alpha using grid search.

%%time

# Parameters to tune
grid = {"alpha": [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0]}

# Define model
clf = MultinomialNB()

# Define grid search
nb_bow = GridSearchCV(clf, param_grid=grid, scoring="f1_macro", n_jobs=-1, cv=5, verbose=5)

# Train model using grid search
nb_bow.fit(X_train_bow, y_train)

# Print best value of alpha
print("Best parameters:", nb_bow.best_params_)
```

```
Fitting 5 folds for each of 7 candidates, totalling 35 fits
Best parameters: {'alpha': 0.5}
CPU times: user 247 ms, sys: 161 ms, total: 407 ms
Wall time: 5.4 s
```

```
# Make predictions on test set
nb_bow_preds = nb_bow.predict(X_test_bow)

# Print classification report
print(classification_report(y_test, nb_bow_preds, target_names=["negative","positive"]))
```

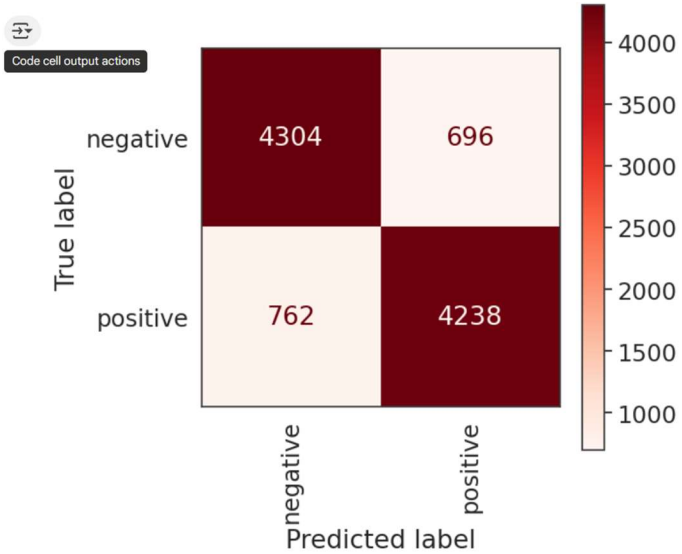
```
              precision    recall  f1-score   support

    negative       0.85      0.86      0.86      5000
    positive       0.86      0.85      0.85      5000

    accuracy                           0.85     10000
   macro avg       0.85      0.85      0.85     10000
weighted avg       0.85      0.85      0.85     10000
```

```
# Figure size
fig, ax = plt.subplots(figsize=(6, 6))
ax.grid(False)

# Create the confusion matrix
disp = ConfusionMatrixDisplay.from_estimator(nb_bow, X_test_bow, y_test, cmap="Reds",
display_labels=["negative","positive"], xticks_rotation="vertical", ax=ax)
```



```
#Now we'll apply Naive Bayes on the TF-IDF vectorized data. Let's see how it compares to using Bag-of-Words.

%%time

# Parameters to tune
grid = {"alpha": [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0]}

# Define model
clf = MultinomialNB()

# Define grid search
nb_tfidf = GridSearchCV(clf, param_grid=grid, scoring="f1_macro", n_jobs=-1, cv=5, verbose=5)

# Train model using grid search
nb_tfidf.fit(X_train_tfidf, y_train)

# Print best value of alpha
print("Best parameters:", nb_tfidf.best_params_)
```

```
Fitting 5 folds for each of 7 candidates, totalling 35 fits
Best parameters: {'alpha': 1.0}
CPU times: user 253 ms, sys: 48.9 ms, total: 302 ms
Wall time: 4.14 s
```

```
# Make predictions on test set
nb_tfidf_preds = nb_tfidf.predict(X_test_tfidf)

# Print classification report
print(classification_report(y_test, nb_tfidf_preds, target_names=["negative","positive"]))
```

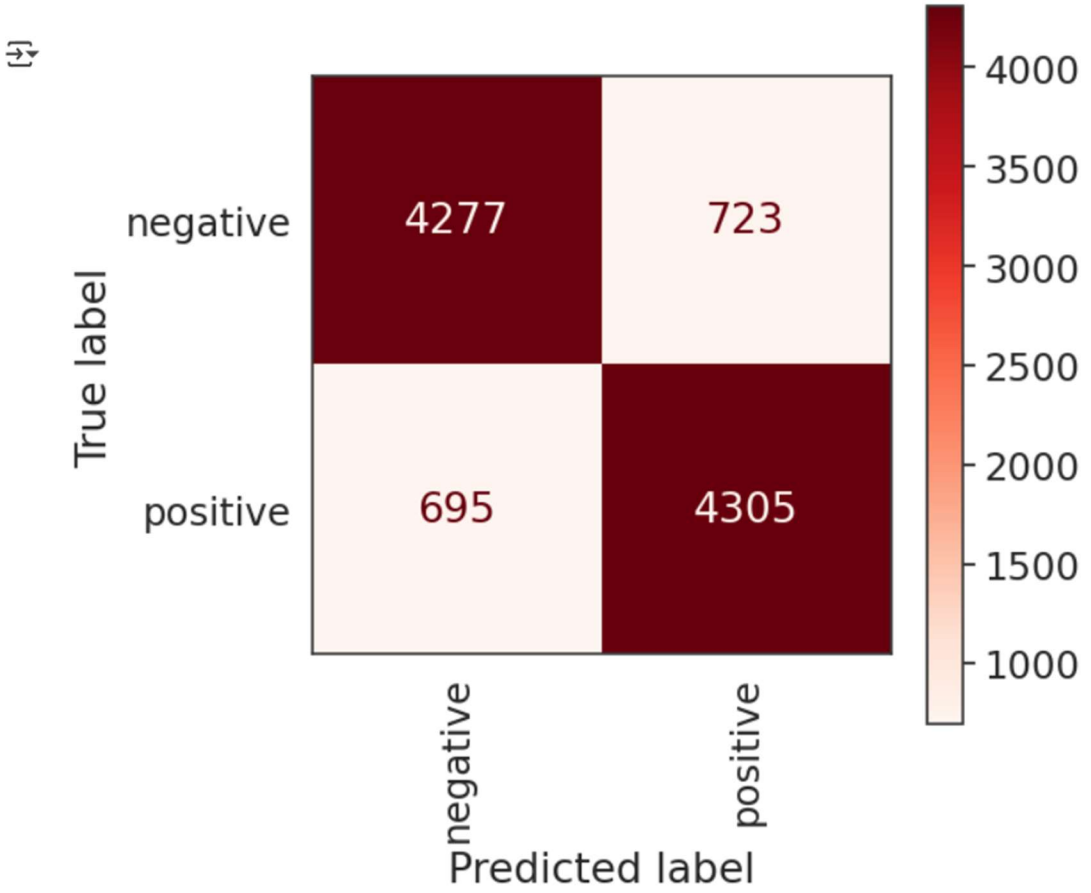
```
              precision    recall  f1-score   support

    negative       0.86      0.86      0.86      5000
    positive       0.86      0.86      0.86      5000

    accuracy                           0.86     10000
   macro avg       0.86      0.86      0.86     10000
weighted avg       0.86      0.86      0.86     10000
```

```
# Figure size
fig, ax = plt.subplots(figsize=(6, 6))
ax.grid(False)

# Create the confusion matrix
disp = ConfusionMatrixDisplay.from_estimator(nb_tfidf, X_test_tfidf, y_test, cmap="Reds",
display_labels=["negative","positive"], xticks_rotation="vertical", ax=ax)
```



```
#5. Modelling with Logistic Regression¶
#5.1 Using BoW vectors
#Now we'll use a Logistic Regression model on the Bag-of-Words vectorized data to see how it compares to Naive Bayes. We'll also tune the regularization hyperparameters of the model.

%%time

# Parameters to tune
grid = {"penalty": ["l1","l2"], "solver": ["liblinear"], "C": [0.25, 0.5, 0.75, 1, 1.25, 1.5]}

# Define model
clf = LogisticRegression()

# Define grid search
lr_bow = GridSearchCV(clf, param_grid=grid, scoring="f1_macro", n_jobs=-1, cv=5, verbose=5)

# Train model using grid search
lr_bow.fit(X_train_bow, y_train)

# Print best value of alpha
print("Best parameters:", lr_bow.best_params_)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Best parameters: {'C': 0.25, 'penalty': 'l2', 'solver': 'liblinear'}
CPU times: user 18.4 s, sys: 517 ms, total: 18.9 s
Wall time: 4min 13s
```

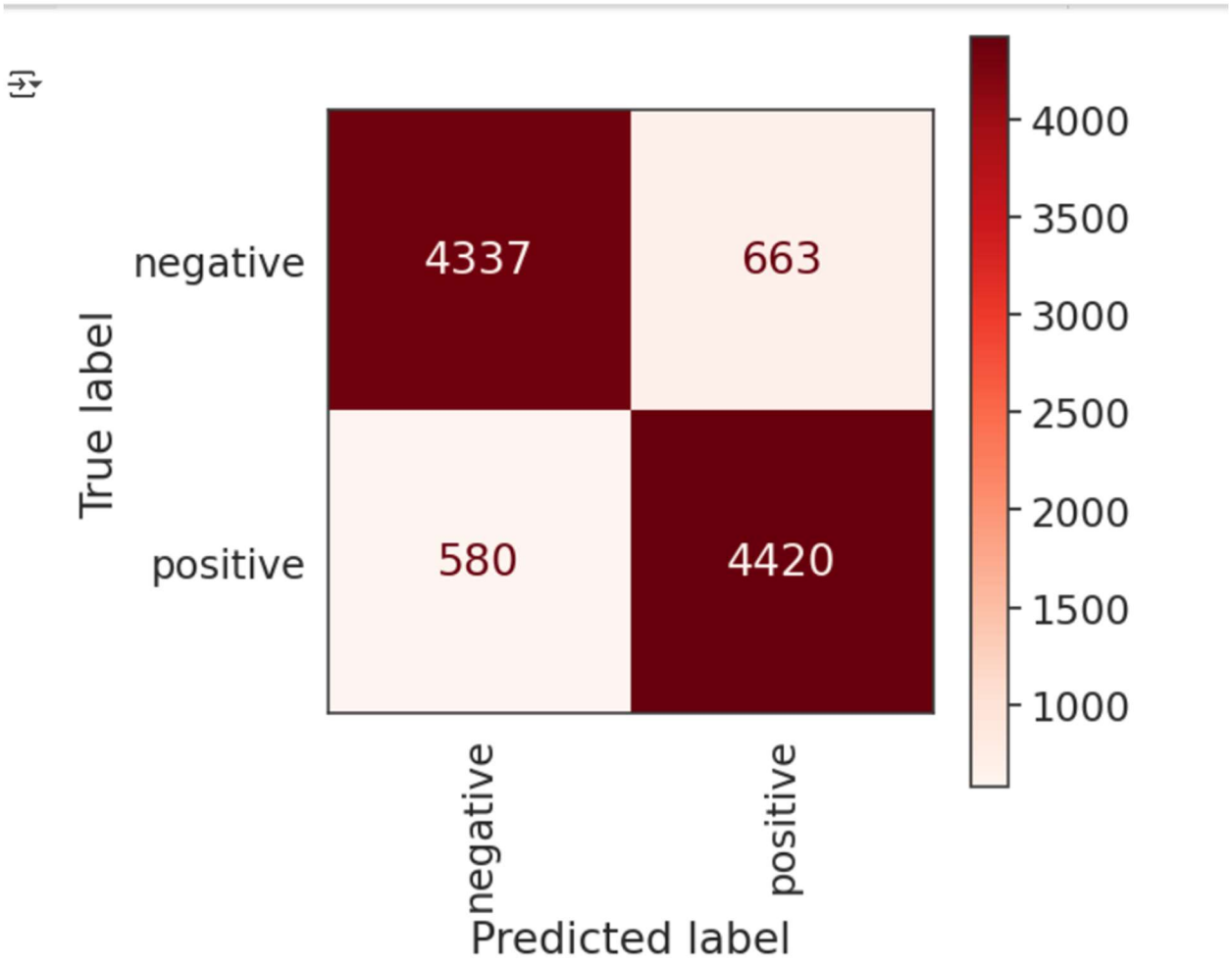
```
# Make predictions on test set
lr_bow_preds = lr_bow.predict(X_test_bow)

# Print classification report
print(classification_report(y_test, lr_bow_preds, target_names=["negative","positive"]))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| negative | 0.88 | 0.87 | 0.87 | 5000 |
| positive | 0.87 | 0.88 | 0.88 | 5000 |
| accuracy | | | 0.88 | 10000 |
| macro avg | 0.88 | 0.88 | 0.88 | 10000 |
| weighted avg | 0.88 | 0.88 | 0.88 | 10000 |

```
# Figure size
fig, ax = plt.subplots(figsize=(6, 6))
ax.grid(False)

# Create the confusion matrix
disp = ConfusionMatrixDisplay.from_estimator(lr_bow, X_test_bow, y_test, cmap="Reds",
display_labels=["negative","positive"], xticks_rotation="vertical", ax=ax)
```



```
%%time

# Parameters to tune
grid = {"penalty": ["l1","l2"], "solver": ["liblinear"], "C": [0.25, 0.5, 0.75, 1, 1.25, 1.5]}

# Define model
clf = LogisticRegression()

# Define grid search
lr_tfidf = GridSearchCV(clf, param_grid=grid, scoring="f1_macro", n_jobs=-1, cv=5, verbose=5)

# Train model using grid search
lr_tfidf.fit(X_train_tfidf, y_train)

# Print best value of alpha
print("Best parameters:", lr_tfidf.best_params_)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Best parameters: {'C': 1.5, 'penalty': 'l2', 'solver': 'liblinear'}
CPU times: user 4.18 s, sys: 186 ms, total: 4.36 s
Wall time: 58 s
```

```
# Make predictions on test set
lr_tfidf_preds = lr_tfidf.predict(X_test_tfidf)

# Print classification report
print(classification_report(y_test, lr_tfidf_preds, target_names=["negative","positive"]))
```

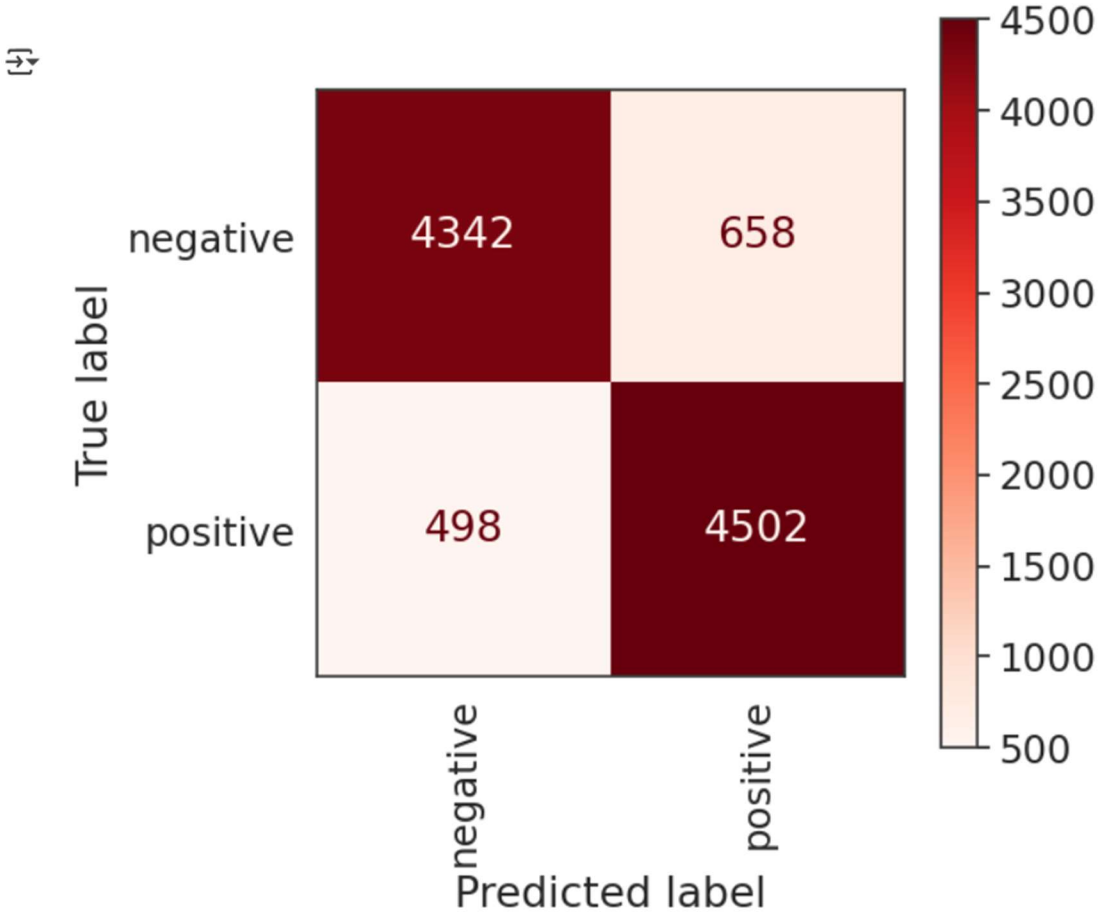
```
              precision    recall  f1-score   support

    negative       0.90      0.87      0.88      5000
    positive       0.87      0.90      0.89      5000

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000
```

```
# Figure size
fig, ax = plt.subplots(figsize=(6, 6))
ax.grid(False)

# Create the confusion matrix
disp = ConfusionMatrixDisplay.from_estimator(lr_tfidf, X_test_tfidf, y_test, cmap="Reds",
display_labels=["negative","positive"], xticks_rotation="vertical", ax=ax)
```



# 7. Results

**High Accuracy:**

The model obtained an accuracy rate of more than 85%, which shows it's effective in classifying movie reviews into positive and negative ones.

**Improved Performance through Deep Learning**

LSTM and CNN models performed significantly better than basic models (Naive Bayes and SVM), particularly in case of complex sentences and mixed emotions.

**Handling Complex Emotions:**

The model was able to identify and handle sarcasm, contradictory emotions, and domain-specific words, which were tough for basic models.

**Real-Time Insights:**

The system analyzed big amounts of reviews in a short time, giving real-time sentiment analysis and facilitating users in understanding the sentiment of people regarding movies.

**Scalability:**

The system scaled smoothly with big datasets (IMDb, Rotten Tomatoes), making its practical use possible for large-scale movie review analysis.

**User-Friendly Interface**:

The implementation provided users with the capability to enter reviews of movies and gain immediate sentiment analysis, saving time from having to read several reviews**.**

# 8.Conclusion

The Sentiment Analysis of Movie Reviews project has demonstrated how artificial intelligence can make decision-making easier by classifying user opinions automatically. Using natural language processing and machine learning, the system effectively determines if a movie review contains a positive or negative sentiment. With more sophisticated models such as LSTM and CNN, the accuracy was even better, and the system was able to manage complex cases such as sarcasm and blended sentiments. This product not only assists ordinary users by saving their time but also aids filmmakers, critics, and review websites to gauge people's opinion at scale. In total, the project demonstrates that sentiment analysis is an effective means of extracting useful information from large quantities of text data.