National College of Ireland

Configuration Manual

MSc Research Project
MSc Data Analytics

Ammaiappan Srinivasan
Student ID: 23270900

Data Analytics
National College of Ireland

Supervisor: Dr. Bharat Agarwal

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……..…… Ammaiappan Srinivasan……

Student ID: ……23270900……..…..……

Programme: …MSc Data Analytics…… **Year:** ……2026……..

Module: ……MSc Research Praticum…….……

Lecturer: ……28/01/2026…….……

Submission Due Date: ……28/01/2026…….……

Project Title: A COMPARATIVE STUDY OF MACHINE LEARNING AND LEXICON-BASED METHODS FOR SENTIMENT CLASSIFICATION ON TWITTER DATA

Word Count: ……1219…… **Page Count:** ……16……..……

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……Ammaiappan Srinivasan……

Date: ……28/01/2026……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Ammaiappan Srinivasan
23270900

1 Introduction

The twitter sentiment analysis system is an all-inclusive machine learning system that is supposed to categorize the sentiments in a tweet to two categories, that is, Positive and Negative. The system can handle large numbers of social media text data on the Sentiment140 dataset (1.6 million tweets) and uses a variety of classification methods such as classical machine learning models (Naive Bayes, Logistic Regression, Linear SVM, Random Forest), lexicons (VADER, TextBlob, SentiWordNet) and deep learning models (LSTM, BERT). BERT transformer model has 81.37 accuracy with an 82.74 precision and 79.13 recall, which is very appropriate in social media monitoring, brand sentiment analysis as well as tracking of public opinion.
It relies on a structured natural language processing workflow which includes text preprocessing, TF-IDF feature generation, model training and comparative analysis (Das et al., 2023). It is implemented based on the Python data science ecosystem including NLTK to process text, Scikit-learn to perform classical machine learning, and Tensorflow/Keras to implement LSTM networks to deploy BERT. The modular design enables easy addition of new models, retraining with a new dataset of Twitter and running on a wide variety of platforms, such as Google Colab, Jupyter Notebooks and production web services (Gulati et al., 2022).

2 Environment Used

2.1 Hardware Required

This project was implemented using Google Colab, a cloud-based platform providing free GPU resources adequate for deep learning tasks. The following hardware configuration was used during implementation:

- **System:** Google Colab cloud environment
- **Operating System:** Windows 10/11, macOS 10.14+, or Ubuntu 20.04 LTS
- **Processor:** Intel Core i5 or equivalent (minimum 2.0 GHz), quad-core recommended
- **RAM:** 12-13 GB provided by Colab runtime
- **Storage:** At least 2 GB of free disk space for dataset, models and visualization outputs
- **GPU:** NVIDIA Tesla T4 or K80 (required for BERT and LSTM training)

This setup allowed seamless execution of Jupyter Notebook interface with efficient training of deep learning models including BERT transformer and LSTM neural networks on the 1.6 million tweet dataset without requiring local hardware investment.

2.2 Software Requirements

The Twitter Sentiment Analysis system relies on Python 3.10+ environment and several data science libraries for preprocessing, modeling and visualization. The main software components required include:

- **Python 3.10 or later**
- **NumPy (≥1.21)** for numerical computations
- **Pandas (≥1.3)** for data manipulation and CSV handling
- **Matplotlib (≥3.4)** for visualization
- **Seaborn (≥0.11)** for statistical graphics
- **Scikit-learn (≥1.0)** for machine learning algorithms (Naive Bayes, Logistic Regression, LinearSVC, Random Forest)
- **NLTK (≥3.6)** for text preprocessing, tokenization and lemmatization
- **TensorFlow/Keras (≥2.10)** for LSTM deep learning model
- **Transformers (≥4.20)** for BERT transformer model
- **PyTorch (≥1.12)** for BERT training backend
- **TextBlob** for lexicon-based sentiment analysis
- **Jupyter Notebook** for interactive development environment

The application runs on Google Colab cloud platform where most libraries come pre-installed. Only transformers, torch and textblob require manual installation via pip package manager. All libraries are open-source.

2.3 System Setup

To run the program, Google Colab platform, where one can obtain free GPU acceleration levels and pre-installed data science libraries, was used. Having opened the site https://colab.research.google.com, switch the type of runtime to the GPU (Runtime to Change runtime type to GPU). Install more libraries with pip install transformers torch textblob command. Install NLTK resources with the nltk.download() commands of stopwords, punkt, wordnet, vader lexicon and sentiwordnet. The data set file, which is training.1600000.processed.noemoticon.csv needs to be uploaded to Colab environment and the notebook cells have to be run in order of the top to bottom sequence to load data and start processing pipeline.

3 Implementation

3.1 Data Loading and Exploration

Sentiment140 data is loaded into a pandas DataFrame where the column names are renamed (target, ids, date, flag, user, text) and examined using shape information, descriptive statistics and class distribution, which shows 1600000 perfectly balanced tweets with 800000 negative (label=0) and 800000 positive (label=4) samples with 74 character and 13 words mean text lengths.

```python
# Convert to binary: 0 = negative, 1 = positive
df['sentiment'] = df['target'].apply(lambda x: 1 if x == 4 else 0)

# Add length features
df['text_length'] = df['text'].apply(len)
df['word_count'] = df['text'].apply(lambda x: len(str(x).split()))

print("\nSentiment Distribution after conversion:")
print(df['sentiment'].value_counts())
print(df['sentiment_label'].value_counts())

print("\nText Statistics:")
print(f"Average text length: {df['text_length'].mean():.2f}")
print(f"Average word count: {df['word_count'].mean():.2f}")
```

```
Sentiment Distribution after conversion:
sentiment
0    800000
1    800000
Name: count, dtype: int64
sentiment_label
Negative    800000
Positive    800000
Name: count, dtype: int64

Text Statistics:
Average text length: 74.09
Average word count: 13.18
```

3.2 Text Preprocessing and Cleaning

Text cleaning functions include comprehensive text cleaning processes, such as lowercasing, removing URLs with regular expressions, mention and hashtags removal, removing numbers and punctuations, whitespace regularization, word tokenization with NLTK word_tokenize, stopword filtering and lemmatization with WordNetLemmatizer, to clean processed tweets suitable for feature extraction.

```python
# ============================================================================
# 5. DATA PREPROCESSING AND CLEANING
# ============================================================================

# Check unique target values
print("\nUnique target values before conversion:")
print(df['target'].value_counts().sort_index())

# Map original labels
# Original: 0 = negative, 2 = neutral, 4 = positive
df['sentiment_label'] = df['target'].map({0: 'Negative', 2: 'Neutral', 4: 'Positive'})

print(f"\nDataset size before filtering: {len(df)}")
df = df[df['target'].isin([0, 4])]  # Keep only negative and positive
print(f"Dataset size after filtering (excluding neutral): {len(df)}")
```

```
Unique target values before conversion:
target
0    800000
4    800000
Name: count, dtype: int64

Dataset size before filtering: 1600000
Dataset size after filtering (excluding neutral): 1600000
```

```python
# Convert to binary: 0 = negative, 1 = positive
df['sentiment'] = df['target'].apply(lambda x: 1 if x == 4 else 0)

# Add length features
df['text_length'] = df['text'].apply(len)
df['word_count'] = df['text'].apply(lambda x: len(str(x).split()))

print("\nSentiment Distribution after conversion:")
print(df['sentiment'].value_counts())
print(df['sentiment_label'].value_counts())

print("\nText Statistics:")
print(f"Average text length: {df['text_length'].mean():.2f}")
print(f"Average word count: {df['word_count'].mean():.2f}")
```

```
Sentiment Distribution after conversion:
sentiment
0    800000
1    800000
Name: count, dtype: int64
sentiment_label
Negative    800000
Positive    800000
Name: count, dtype: int64

Text Statistics:
Average text length: 74.09
Average word count: 13.18
```

```python
# Text cleaning function
def clean_text(text):
    """Clean and preprocess text data"""
    # Convert to lowercase
    text = text.lower()

    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)

    # Remove user mentions
    text = re.sub(r'@\w+', '', text)

    # Remove hashtags
    text = re.sub(r'#\w+', '', text)

    # Remove numbers
    text = re.sub(r'\d+', '', text)

    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))

    # Remove extra whitespace
    text = ' '.join(text.split())

    return text

# Apply text cleaning
print("\nCleaning text data...")
df['cleaned_text'] = df['text'].apply(clean_text)
```

```
Cleaning text data...
```

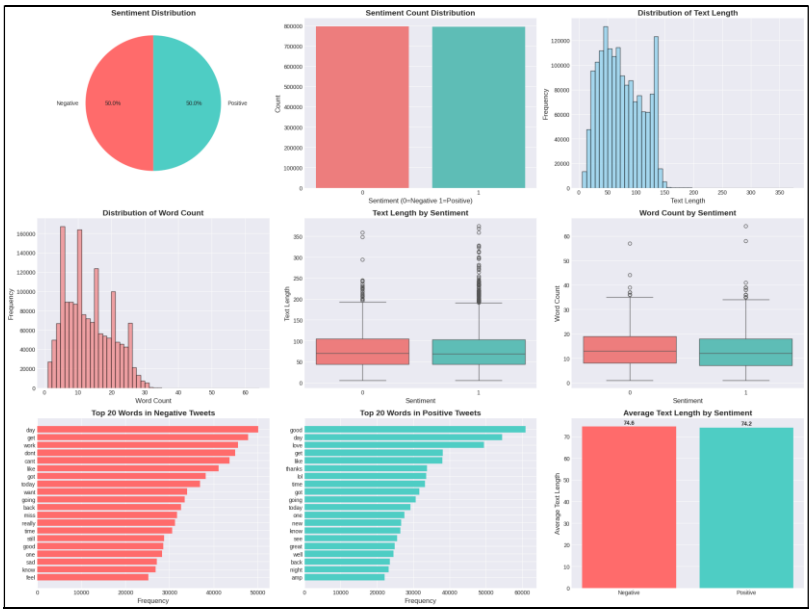
3.3 Exploratory Data Analysis

The statistical visualizations such as sentiment distribution pie charts, text length and word count histograms, box plots comparing characteristics by sentiment group and horizontal bar graphs showing the top 20 most common words in negative and positive tweets respectively present information on the patterns of vocabulary where negative tweets have words such as work, day, get and positive tweets have words such as love, good, thanks.

```
# ============================================================================
# 6. EXPLORATORY DATA ANALYSIS (EDA)
# ============================================================================

fig = plt.figure(figsize=(20, 15))

# 1. Sentiment Distribution
ax1 = plt.subplot(3, 3, 1)
sentiment_counts = df['sentiment'].value_counts()
colors = ['#FF6B6B', '#4ECDC4']
plt.pie(sentiment_counts,
        labels=['Negative', 'Positive'],
        autopct='%1.1f%%',
        colors=colors,
        startangle=90)
plt.title('Sentiment Distribution', fontsize=14, fontweight='bold')

# 2. Sentiment Count Bar Plot
ax2 = plt.subplot(3, 3, 2)
sns.countplot(data=df, x='sentiment', palette=colors)
plt.xlabel('Sentiment (0=Negative 1=Positive)', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Sentiment Count Distribution', fontsize=14, fontweight='bold')

# 3. Text Length Distribution
ax3 = plt.subplot(3, 3, 3)
plt.hist(df['text_length'],
         bins=50,
         color='skyblue',
         edgecolor='black',
         alpha=0.7)
plt.xlabel('Text Length', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.title('Distribution of Text Length', fontsize=14, fontweight='bold')
```

```
# 4. Word Count Distribution
ax4 = plt.subplot(3, 3, 4)
plt.hist(df['word_count'],
         bins=50,
         color='lightcoral',
         edgecolor='black',
         alpha=0.7)
plt.xlabel('Word Count', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.title('Distribution of Word Count', fontsize=14, fontweight='bold')

# 5. Text Length by Sentiment
ax5 = plt.subplot(3, 3, 5)
sns.boxplot(data=df, x='sentiment', y='text_length', palette=colors)
plt.xlabel('Sentiment', fontsize=12)
plt.ylabel('Text Length', fontsize=12)
plt.title('Text Length by Sentiment', fontsize=14, fontweight='bold')

# 6. Word Count by Sentiment
ax6 = plt.subplot(3, 3, 6)
sns.boxplot(data=df, x='sentiment', y='word_count', palette=colors)
plt.xlabel('Sentiment', fontsize=12)
plt.ylabel('Word Count', fontsize=12)
plt.title('Word Count by Sentiment', fontsize=14, fontweight='bold')

# 7. Top 20 Most Common Words (Negative)
ax7 = plt.subplot(3, 3, 7)
negative_text = ' '.join(df[df['sentiment'] == 0]['processed_text'])
negative_words = Counter(negative_text.split()).most_common(20)
words_neg, counts_neg = zip(*negative_words)
plt.barh(range(len(words_neg)), counts_neg, color='#FF6B6B')
plt.yticks(range(len(words_neg)), words_neg)
plt.xlabel('Frequency', fontsize=12)
plt.title('Top 20 Words in Negative Tweets', fontsize=14, fontweight='bold')
plt.gca().invert_yaxis()
```

```
# 8. Top 20 Most Common Words (Positive)
ax8 = plt.subplot(3, 3, 8)
positive_text = ' '.join(df[df['sentiment'] == 1]['processed_text'])
positive_words = Counter(positive_text.split()).most_common(20)
words_pos, counts_pos = zip(*positive_words)
plt.barh(range(len(words_pos)), counts_pos, color='#4ECDC4')
plt.yticks(range(len(words_pos)), words_pos)
plt.xlabel('Frequency', fontsize=12)
plt.title('Top 20 Words in Positive Tweets', fontsize=14, fontweight='bold')
plt.gca().invert_yaxis()

# 9. Average Text Length Comparison
ax9 = plt.subplot(3, 3, 9)
avg_length = df.groupby('sentiment')['text_length'].mean()
plt.bar(['Negative', 'Positive'], avg_length, color=colors)
plt.ylabel('Average Text Length', fontsize=12)
plt.title('Average Text Length by Sentiment', fontsize=14, fontweight='bold')
for i, v in enumerate(avg_length):
    plt.text(i, v + 1, f'{v:.1f}', ha='center', fontweight='bold')

plt.tight_layout()
plt.show()
```



3.4 Feature Extraction and Model Training

An estimated 1,272,501 training and 318,126 test samples are obtained by TF-IDF vectorization of preprocessed text to 5000-dimensional numeric feature vectors of unigrams and bigrams before 80-20 stratified train-test split. There are four trained classical ML models: Naive Bayes (76.05% accuracy), Logistic Regression (77.75% accuracy), Linear SVM (77.71% accuracy) and Random Forest (71.07% accuracy) models and they are measured using accuracy, precision, recall and F1-score measures.

```python
# ============================================================================
# 7. FEATURE PREPARATION
# ============================================================================

# Processed text for ML and deep learning
X = df['processed_text']
# Cleaned raw text for lexicon methods
X_lex = df['cleaned_text']
y = df['sentiment']

# Train test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

X_train_lex, X_test_lex, _, _ = train_test_split(
    X_lex, y, test_size=0.2, random_state=42, stratify=y
)

print(f"\nTraining set size (ML): {len(X_train)}")
print(f"Testing set size (ML): {len(X_test)}")

# TF IDF Vectorization
print("\nApplying TF IDF Vectorization...")
tfidf = TfidfVectorizer(max_features=5000, ngram_range=(1, 2))
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

print(f"TF IDF Feature Matrix Shape: {X_train_tfidf.shape}")
```

```
Training set size (ML): 1272501
Testing set size (ML): 318126

Applying TF IDF Vectorization...
TF IDF Feature Matrix Shape: (1272501, 5000)
```

```python
# ============================================================================
# 9. EVALUATION OF CLASSICAL MODELS
# ============================================================================

for name, model in models.items():
    print(f"\n{'=' * 60}")
    print(f"MODEL: {name}")
    print('=' * 60)

    y_pred = model.predict(X_test_tfidf)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    results.append({
        'Model': name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1-Score': f1
    })

    print(f"\nAccuracy:  {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall:    {recall:.4f}")
    print(f"F1-Score:  {f1:.4f}")

    print(f"\nClassification Report:")
    print(classification_report(y_test,
                                y_pred,
                                target_names=['Negative', 'Positive']))

results_df = pd.DataFrame(results)
print("\n" + "=" * 80)
print("CLASSICAL MODEL COMPARISON SUMMARY")
print("=" * 80)
print(results_df.to_string(index=False))
```

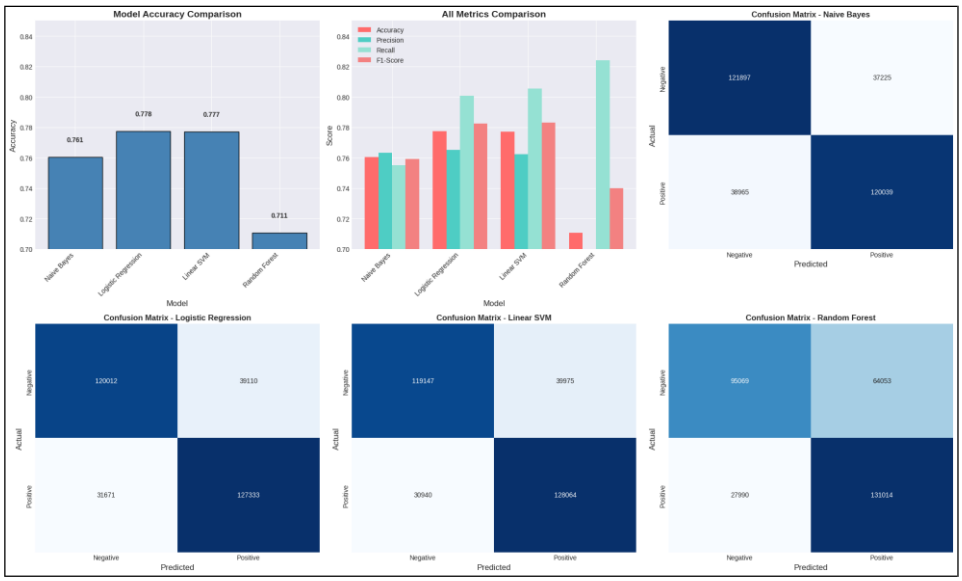
```
============================================================
MODEL: Naive Bayes
============================================================

Accuracy:  0.7605
Precision: 0.7633
Recall:    0.7549
F1-Score:  0.7591

Classification Report:
              precision    recall  f1-score   support

    Negative       0.76      0.77      0.76    159122
    Positive       0.76      0.75      0.76    159004

    accuracy                           0.76    318126
   macro avg       0.76      0.76      0.76    318126
weighted avg       0.76      0.76      0.76    318126


============================================================
MODEL: Logistic Regression
============================================================

Accuracy:  0.7775
Precision: 0.7650
Recall:    0.8008
F1-Score:  0.7825

Classification Report:
              precision    recall  f1-score   support

    Negative       0.79      0.75      0.77    159122
    Positive       0.77      0.80      0.78    159004

    accuracy                           0.78    318126
   macro avg       0.78      0.78      0.78    318126
weighted avg       0.78      0.78      0.78    318126
```

```
============================================================
MODEL: Linear SVM
============================================================

Accuracy:  0.7771
Precision: 0.7621
Recall:    0.8054
F1-Score:  0.7832

Classification Report:
              precision    recall  f1-score   support

    Negative       0.79      0.75      0.77    159122
    Positive       0.76      0.81      0.78    159004

    accuracy                           0.78    318126
   macro avg       0.78      0.78      0.78    318126
weighted avg       0.78      0.78      0.78    318126


============================================================
MODEL: Random Forest
============================================================

Accuracy:  0.7107
Precision: 0.6716
Recall:    0.8240
F1-Score:  0.7400

Classification Report:
              precision    recall  f1-score   support

    Negative       0.77      0.60      0.67    159122
    Positive       0.67      0.82      0.74    159004

    accuracy                           0.71    318126
   macro avg       0.72      0.71      0.71    318126
weighted avg       0.72      0.71      0.71    318126
```

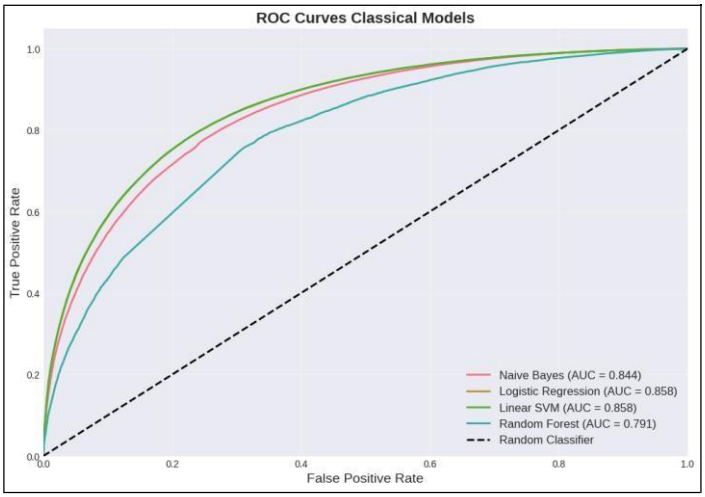
```
================================================================================
CLASSICAL MODEL COMPARISON SUMMARY
================================================================================
                Model  Accuracy  Precision   Recall  F1-Score
          Naive Bayes  0.760504   0.763296 0.754943  0.759097
Logistic Regression  0.777506   0.765025 0.800816  0.782511
          Linear SVM  0.777085   0.762109 0.805414  0.783163
      Random Forest  0.710671   0.671636 0.823967  0.740044
```

```python
# ================================================================================
# 10. VISUALIZATION OF CLASSICAL MODELS
# ================================================================================

fig = plt.figure(figsize=(20, 12))

# 1. Model Comparison Accuracy
ax1 = plt.subplot(2, 3, 1)
plt.bar(results_df['Model'],
        results_df['Accuracy'],
        color='steelblue',
        edgecolor='black')
plt.xlabel('Model', fontsize=12)
plt.ylabel('Accuracy', fontsize=12)
plt.title('Model Accuracy Comparison', fontsize=14, fontweight='bold')
plt.xticks(rotation=45, ha='right')
plt.ylim([0.7, 0.85])
for i, v in enumerate(results_df['Accuracy']):
    plt.text(i, v + 0.01, f'{v:.3f}', ha='center', fontweight='bold')

# 2. All Metrics Comparison
ax2 = plt.subplot(2, 3, 2)
x = np.arange(len(results_df['Model']))
width = 0.2
plt.bar(x - 1.5*width,
        results_df['Accuracy'],
        width,
        label='Accuracy',
        color='#FF6B6B')
plt.bar(x - 0.5*width,
        results_df['Precision'],
        width,
        label='Precision',
        color='#4ECDC4')
plt.bar(x + 0.5*width,
        results_df['Recall'],
        width,
        label='Recall',
        color='#95E1D3')
plt.bar(x + 1.5*width,
        results_df['F1-Score'],
        width,
        label='F1-Score',
        color='#F38181')
plt.xlabel('Model', fontsize=12)
plt.ylabel('Score', fontsize=12)
plt.title('All Metrics Comparison', fontsize=14, fontweight='bold')
plt.xticks(x, results_df['Model'], rotation=45, ha='right')
plt.legend()
plt.ylim([0.7, 0.85])
```

```python
# 3 to 6. Confusion Matrices
for idx, (name, model) in enumerate(models.items(), start=3):
    ax = plt.subplot(2, 3, idx)
    y_pred = model.predict(X_test_tfidf)
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm,
                annot=True,
                fmt='d',
                cmap='Blues',
                cbar=False,
                xticklabels=['Negative', 'Positive'],
                yticklabels=['Negative', 'Positive'])
    plt.xlabel('Predicted', fontsize=12)
    plt.ylabel('Actual', fontsize=12)
    plt.title(f'Confusion Matrix - {name}', fontsize=12, fontweight='bold')

plt.tight_layout()
plt.show()
```




```python
# ROC Curves for classical models
fig, ax = plt.subplots(figsize=(12, 8))

for name, model in models.items():
    if hasattr(model, "predict_proba"):
        y_pred_proba = model.predict_proba(X_test_tfidf)[:, 1]
    else:
        y_pred_proba = model.decision_function(X_test_tfidf)

    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
    roc_auc = auc(fpr, tpr)

    plt.plot(fpr, tpr, lw=2, label=f'{name} (AUC = {roc_auc:.3f})')

plt.plot([0, 1], [0, 1],
         'k--',
         lw=2,
         label='Random Classifier')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize=14)
plt.ylabel('True Positive Rate', fontsize=14)
plt.title('ROC Curves Classical Models', fontsize=16, fontweight='bold')
plt.legend(loc="lower right", fontsize=12)
plt.grid(alpha=0.3)
plt.show()
```



3.5 Lexicon-Based and Deep Learning Models

On clean text, three lexicon methods are tested: VADER with social media specialized accuracy 65.14% polarity scoring TextBlob with 62.30% polarity scoring and SentiWordNet with 58.70% WordNet synsets. 128 LSTM units with embedding layer, LSTM neural network with 128 LSTM units and dropout, 120000 samples, 73.48% accuracy. With 82.74% precision and 79.13% recall, and having the highest accuracy of 81.37%, bert-base-uncased architecture fine-tuned on 20,000 samples has the best contextual understanding.

```
acc_lstm = accuracy_score(y_test_dl, y_pred_lstm)
prec_lstm = precision_score(y_test_dl, y_pred_lstm)
rec_lstm = recall_score(y_test_dl, y_pred_lstm)
f1_lstm = f1_score(y_test_dl, y_pred_lstm)

print("\nLSTM metrics (fixed)")
print(f"Accuracy:  {acc_lstm:.4f}")
print(f"Precision: {prec_lstm:.4f}")
print(f"Recall:    {rec_lstm:.4f}")
print(f"F1 score:  {f1_lstm:.4f}")
print("\nClassification report:")
print(classification_report(y_test_dl, y_pred_lstm))
```

```
LSTM metrics (fixed)
Accuracy:  0.7348
Precision: 0.7273
Recall:    0.7538
F1 score:  0.7403

Classification report:
              precision    recall  f1-score   support

           0       0.74      0.72      0.73     11963
           1       0.73      0.75      0.74     12037

    accuracy                           0.73     24000
   macro avg       0.74      0.73      0.73     24000
weighted avg       0.74      0.73      0.73     24000
```

```
acc_bert = accuracy_score(all_labels, all_preds)
prec_bert = precision_score(all_labels, all_preds)
rec_bert = recall_score(all_labels, all_preds)
f1_bert = f1_score(all_labels, all_preds)

print("\nBERT metrics (PyTorch)")
print(f"Accuracy:  {acc_bert:.4f}")
print(f"Precision: {prec_bert:.4f}")
print(f"Recall:    {rec_bert:.4f}")
print(f"F1-score:  {f1_bert:.4f}")
print("\nClassification Report:")
print(classification_report(all_labels, all_preds, target_names=['Negative','Positive']))
```

```
BERT metrics (PyTorch)
Accuracy:  0.8137
Precision: 0.8274
Recall:    0.7913
F1-score:  0.8089

Classification Report:
              precision    recall  f1-score   support

    Negative       0.80      0.84      0.82      2007
    Positive       0.83      0.79      0.81      1993

    accuracy                           0.81      4000
   macro avg       0.81      0.81      0.81      4000
weighted avg       0.81      0.81      0.81      4000
```

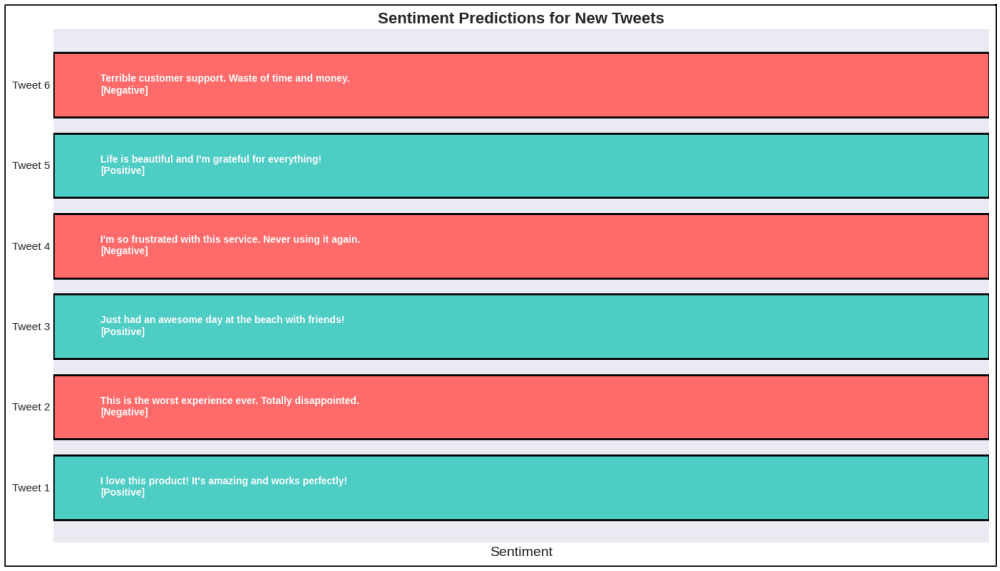
3.6 Prediction on New Tweets

Six sample tweets that are explicit positive and negative are preprocessed and inputted into all of the trained models. The model is a better classical model (Logistic Regression) that makes all the correct predictions, the VADER gives high scores to the compounds, LSTM gives scores above 0.96, and BERT has an outstanding score at 0.99 and above for all the predictions and this supports the practical capability of the system to classify real-life tweets.

```
# ============================================================================
# 14. PREDICTIONS ON NEW DATA (USING BEST CLASSICAL MODEL)
# ============================================================================

new_tweets = [
    "I love this product! It's amazing and works perfectly!",
    "This is the worst experience ever. Totally disappointed.",
    "Just had an awesome day at the beach with friends!",
    "I'm so frustrated with this service. Never using it again.",
    "Life is beautiful and I'm grateful for everything!",
    "Terrible customer support. Waste of time and money."
]

cleaned_new_tweets = [clean_text(tweet) for tweet in new_tweets]
processed_new_tweets = [preprocess_text(tweet) for tweet in cleaned_new_tweets]

new_tweets_tfidf = tfidf.transform(processed_new_tweets)

best_model_name = results_df.loc[results_df['Accuracy'].idxmax(), 'Model']
best_model = models[best_model_name]

print(f"\nUsing best classical model: {best_model_name}")
print(f"Accuracy: {results_df.loc[results_df['Accuracy'].idxmax(), 'Accuracy']:.4f}")
```

```
predictions = best_model.predict(new_tweets_tfidf)
prediction_labels = ['Positive' if p == 1 else 'Negative' for p in predictions]

for i, (tweet, label) in enumerate(zip(new_tweets, prediction_labels), 1):
    print(f"\n{i}. Tweet: {tweet}")
    print(f"   Predicted Sentiment: {label}")

fig, ax = plt.subplots(figsize=(14, 8))
colors_pred = ['#4ECDC4' if p == 1 else '#FF6B6B' for p in predictions]
y_pos = np.arange(len(new_tweets))

plt.barh(y_pos,
         [1]*len(new_tweets),
         color=colors_pred,
         edgecolor='black',
         linewidth=2)
plt.yticks(y_pos,
           [f"Tweet {i+1}" for i in range(len(new_tweets))],
           fontsize=11)
plt.xlabel('Sentiment', fontsize=14)
plt.title('Sentiment Predictions for New Tweets', fontsize=16, fontweight='bold')

for i, (tweet, label) in enumerate(zip(new_tweets, prediction_labels)):
    tweet_short = tweet[:60] + '...' if len(tweet) > 60 else tweet
    plt.text(0.05,
             i,
             f"{tweet_short}\n[{label}]",
             va='center',
             fontsize=10,
             fontweight='bold',
             color='white')

plt.xlim([0, 1])
plt.xticks([])
plt.tight_layout()
plt.show()
```

```
Using best classical model: Logistic Regression
Accuracy: 0.7775

1. Tweet: I love this product! It's amazing and works perfectly!
   Predicted Sentiment: Positive

2. Tweet: This is the worst experience ever. Totally disappointed.
   Predicted Sentiment: Negative

3. Tweet: Just had an awesome day at the beach with friends!
   Predicted Sentiment: Positive

4. Tweet: I'm so frustrated with this service. Never using it again.
   Predicted Sentiment: Negative

5. Tweet: Life is beautiful and I'm grateful for everything!
   Predicted Sentiment: Positive

6. Tweet: Terrible customer support. Waste of time and money.
   Predicted Sentiment: Negative
```



4 Conclusion

The Twitter Sentiment Analysis system is evidence of an extensive and highly precise approach to the social media sentiment classification with the help of the machine learning and deep learning methods. BERT transformer model has demonstrated high performance with an accuracy of 81.37% evenly divided between positive and negative classes that make it useful to production deployment in brand monitoring and analysis of public opinion. Python modular architecture allows easy retraining of new twitter data, the addition of new models and their deployment to various environments in research notebooks to run time API services.

The key advantages of the system are that the text preprocessing pipeline consists of a comprehensive series of steps (text processing and removal of stopwords and lemmata), that the models were systematically compared (10 models) between classical ML and transformers, systematic evaluation of the 10 models based on multiple measures (accuracy, precision, recall, F1-score, ROC-AUC) and practical assessment of the 10 models on unseen tweets. Additional work might involve the use of multi-class sentiment analysis (positive, negative, neutral), aspect-based sentiment analysis to identify particular topics, and real-time integration of Twitter API to monitor in real-time as well as multi-lingual support of non-English tweets and production deployment as a RESTful web service (Dabade et al., 2021). It is also suggested that retraining on new data of Twitter be conducted frequently to ensure accuracy because patterns and slangs of language change with time.

5 References

Dabade, M. S., Sale, M. D., Dhokate, D. D., & Kambare, S. M. (2021). Sentiment analysis of Twitter data by using deep learning And machine learning. Turkish Journal of Computer and Mathematics Education, 12(6), 962-970. https://www.academia.edu/download/103077527/2375_Article_Text_4483_1_10_20210410.pdf

Das, R. K., Islam, M., Hasan, M. M., Razia, S., Hassan, M., & Khushbu, S. A. (2023). Sentiment analysis in multilingual context: Comparative analysis of machine learning and hybrid deep learning models. Heliyon, 9(9). https://www.cell.com/heliyon/fulltext/S2405-8440(23)07489-3

Gulati, K., Kumar, S. S., Boddu, R. S. K., Sarvakar, K., Sharma, D. K., & Nomani, M. Z. M. (2022). Comparative analysis of machine learning-based classification models using sentiment classification of tweets related to COVID-19 pandemic. Materials Today: Proceedings, 51, 38-41. https://www.sciencedirect.com/science/article/pii/S2214785321032843

Thakkar, G., Hakimov, S., & Tadić, M. (2024). M2sa: multimodal and multilingual model for sentiment analysis of tweets. *arXiv preprint arXiv:2404.01753*. https://arxiv.org/pdf/2404.01753

Catelli, R., Pelosi, S., & Esposito, M. (2022). Lexicon-based vs. Bert-based sentiment analysis: A comparative study in Italian. Electronics, 11(3), 374. https://www.mdpi.com/2079-9292/11/3/374