Sentimental Analysis

Sucharitha Pinnu
Venkata Ratnam
Abidemi Akinade

Overview

Sentimental analysis involves examining text data to determine the underlying sentiment or opinion expressed. For reviewing comments, it classifies the text as positive or negative.

This process typically involves cleaning and preprocessing the text data, converting it to numerical features, and training a machine learning model on labeled data. The trained model can then predict the sentiment of new comments as positive or negative.

Analyzing these sentiment predictions helps businesses understand customer opinions and make informed decisions to improve their offerings.

What is in the Dataset?

- Dataset consists of 50000 rows and two columns..
- 1st column consists of text data, which are comments that are reviews of the movies.
- 2nd column is classification of the text sentence in the 1st column, which says whether the comment is either a positive comment or a negative comment.

```
df = pd.read_csv("imdbdataset.csv")
print(df.head())
```

```
                                              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
```

Preprocessing Steps

1.**Lowercasing**: Convert all text to lowercase for standardization.

```
df['review'] = df['review'].str.lower()
print(df.head())
```

2.**Removing HTML Tags**: Remove any HTML tags present in the text to extract only the textual content.

```
def remove_html_tags(text):
    pattern = re.compile('<[^<]+?>')
    return pattern.sub(r'', text)
```

3.**Removing Punctuation Marks**: Eliminate punctuation marks as they often do not carry sentiment information

```
def remove_punctuation(text):
    return re.sub(r'[^\w\s]', '', text)
```

4.**Removing URLs**: Remove any URLs from the text as they are irrelevant for sentiment analysis.

```
def remove_url(text):
    return re.sub(r'http[s]?://\S+|www\.\S+', '', text)
```

5.**Removing Extra White Spaces**: Remove any extra white spaces to standardize the text.

```
def remove_extra_whitespaces(text):
    return re.sub(' +', ' ', text).strip()
```

6.**Removing Stop Words:** Eliminate commonly occurring words that do not contribute to sentiment analysis.

```
stop_words = set(stopwords.words('english'))

def remove_stopwords(text):
    lst = []
    for word in text.split():
        if word not in stop_words:
            lst.append(word)
    return " ".join(lst)
```

7.**Removing Numbers**: Remove numerical values unless they are important for sentiment analysis in your specific context.

```
def remove_numbers(text):
    lst = []
    for word in text.split():
      if not word.isdigit():
        lst.append(word)
    return " ".join(lst)
```

8.**Word Tokenization**: Break down the text into individual words for further analysis.

```
def tokenize(text):
    return word_tokenize(text)

df['review'] = df['review'].apply(tokenize)
```

9.**Word Lemmatization/Stemming**: Reduce words to their base or root form for normalization and dimensionality reduction.

```
lemmatizer = WordNetLemmatizer()

def lemmatize_text(text):
    return [lemmatizer.lemmatize(word) for word in text]

df['review'] = df['review'].apply(lemmatize_text)
```

10.**Converting Sentiment to Numeric Values**: Convert sentiment labels (e.g., positive,negative) to numeric values for training machine learning models.

Implementation of ML Algorithms

For this Project, we have implemented following three Machine learning Algorithms.

- **Logistic Regression**
  - Linear and Discriminative model.
- **Naive Bayes Algorithm**
  - Probabilistic and Generative model.
- **Random Forest Classifier**
  - Ensemble Learning and Bagging.

Step 1 - Defining Hyperparameters

- Hyperparameters are defined using a list named param_grid and different parameters are defined in three different algorithms.

```
param_grid = {
    'tfidf__max_features': [1000, 5000, 10000],
    'lr__C': [0.1, 1.0, 10.0]
}
```

```
param_grid = {
    'tfidf__max_features': [1000, 5000, 10000],
    'nb__alpha': [0.1, 0.5, 1.0, 2.0, 5.0, 10.0],
    'nb__fit_prior': [True, False]
}
```

```
param_grid = {
    'tfidf__max_features': [1000, 5000, 10000],
    'rf__n_estimators': [100, 200, 300],
    'rf__max_depth': [10, 20, 30]
}
```

TFIDF Vectorizer

- TFIDF is a vectoriser that helps in representing text data numerically while giving more weight to the most important words and less weight to common words.
- TF-IDF vectorizer can convert a document into a numerical vector, where each dimension represents the importance of a term in the document.
- These vectors can then be used as input to machine learning algorithms for sentiment analysis.

Pipeline Creation:

- A pipeline has been created with two steps.

```
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('lr', LogisticRegression())
])
```

- First step of the pipeline is implementing TFIDF Vectorizer on the data and three different algorithms has implemented after that.

Performing GridSearchCV:

- It is defined as follows:

```
grid_search_lr = GridSearchCV(pipeline, param_grid, cv=3, n_jobs=-1, verbose=2)
```

- Fitted the model on the training data
- Extracted best hyperparameters and best model.

```
best_hyper_params = grid_search_lr.best_params_
best_lr_model = grid_search_lr.best_estimator_
```

Preparing input for Embedding layer:

- In this step of preprocessing, sentences are converted into the sequences into 2D NumPy arrays of fixed length (maxlen). Sequences shorter than maxlen are padded with zeros at the end.

```
maxlen = 100
X_train_pad = pad_sequences(sequences, maxlen=maxlen)
X_val_pad = pad_sequences(tokenizer.texts_to_sequences(X_val), maxlen=maxlen)
X_test_pad = pad_sequences(tokenizer.texts_to_sequences(X_test), maxlen=maxlen)
word_index = tokenizer.word_index
reverse_word_index = {v: k for k, v in word_index.items()}
```

Creation of an embedding matrix

- In this step an embedding matrix is created using a GloVe embeddings file named glove.6B.100d.txt.
- This file consists of words followed by the corresponding words vector.

```
embedding_dim = 100
embedding_matrix = np.zeros((max_words, embedding_dim))
with open('glove.6B.100d.txt', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        if word in word_index and word_index[word] < max_words:
            embedding_matrix[word_index[word]] = np.array(values[1:], dtype='float32')
```

Implementation of Neural Network Algorithms

- Three different Neural Network Algorithms are implemented in this project. They are:

  - Simple Neural Network

  - Convolutional Neural Network

  - LSTM (Recurrent Neural Network)

Simple Neural Network

- A simple neural network model is defined and created as below:

```
def create_model(embedding_matrix, maxlen, embedding_dim=100, max_words=10000, dropout_rate=0.2):
    model = Sequential()
    model.add(Embedding(embedding_matrix.shape[0],
                        embedding_matrix.shape[1],
                        input_length=maxlen,
                        weights=[embedding_matrix],
                        trainable=False))
    model.add(Flatten())
    model.add(Dense(32, activation='relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

Fitting of the model

- After the model is created, the model is then fitted on the padded sentences training data for different dropout rate values.
- Best Hyperparameter value and the best model is then fetched

```
for dropout_rate in param_grid['dropout_rate']:
    model = create_model(embedding_matrix=embedding_matrix, maxlen=maxlen, embedding_dim=embedding_dim, max_words=max_words, drop
    early_stopping = EarlyStopping(monitor='val_loss', patience=3)
    model.fit(X_train_pad, y_train, epochs=10, batch_size=32, verbose=1,
              validation_data=(X_val_pad, y_val), callbacks=[early_stopping])
    val_loss, val_acc = model.evaluate(X_val_pad, y_val)
    if val_acc > best_val_acc_nn:
        best_val_acc_nn = val_acc
        best_model_nn = model
        best_hyperparams = {'dropout_rate': dropout_rate}
```

Convolution Neural Network

- In this Convolution Neural Network model, the model is created similar to Simple Neural Network. The network is defined and created as follows:

```
def create_cnn_model(embedding_matrix, maxlen, embedding_dim=100, max_words=10000, dropout_rate=0.2):
    model = Sequential()
    model.add(Embedding(embedding_matrix.shape[0],
                        embedding_matrix.shape[1],
                        input_length=maxlen,
                        weights=[embedding_matrix],
                        trainable=False))
    model.add(Conv1D(32, 5, activation='relu'))
    model.add(MaxPooling1D(5))
    model.add(Conv1D(32, 5, activation='relu'))
    model.add(GlobalMaxPooling1D())
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

LSTM

- LSTM model is defined and created as follows:

```
def create_lstm_model(embedding_matrix, maxlen, embedding_dim=100, max_words=10000, dropout_rate=0.2):
    model = Sequential()
    model.add(Embedding(embedding_matrix.shape[0],
                        embedding_matrix.shape[1],
                        input_length=maxlen,
                        weights=[embedding_matrix],
                        trainable=False))
    model.add(LSTM(32))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

Performance Evaluation

- Performance of the model is evaluated using different factors for each algorithm.They are:

  - Best Hyperparameters
  - Training Accuracy
  - Testing Accuracy
  - Log Loss
  - Matthews Coefficient
  - Classification Report

```
print("Best Hyperparameters of CNN:", best_hyperparams_cnn)
print("Best Training Accuracy:", best_val_acc_cnn)
print("Testing Accuracy with best CNN model:", test_acc_cnn)
print("Log Loss of CNN :", logloss_cnn)
print("Matthews Correlation Coefficient of CNN:", mcc_cnn)
print("Best Model CNN Summary:")
print(best_model_cnn.summary())
```

Performance Representations and Curves

- Confusion Matrix
- Precision-Recall Curve
- ROC Curve
- Learning Curve
- Validation Curve