Analysis of Customer Satisfaction using Sentiment Analysis

Roghieh Farajialamooti

Seda Sensoy

Sraboni Bhuiyan

Group X

Topics

- Data Gathering
- Preprocessing
- Analysis using Naïve Bayes
- SVM Machine Learning Model
- LSTM Deep Learning Model using Word Embedding Vectorization
- LSTM Deep Learning Model using TF-IDF Vectorization
- Results

Data Gathering

- Dataset
  - How to Download
- Specification
  - Excel File of Size 1500 with Two Columns: Comment, Sentiment
  - Separate 1290 of Data

```
youtube = build('youtube', 'v3', developerKey='KEY')

video_id = '78IJdhvY1zg'
comments = []

request = youtube.commentThreads().list(
    part='snippet',
    videoId=video_id,
    textFormat='plainText')

while request:
    response = request.execute()
    for item in response['items']:
        comment =
item['snippet']['topLevelComment']['snippet']['textDispla
y']
        comments.append(comment)

    request = youtube.commentThreads().list_next(request,
response)
```

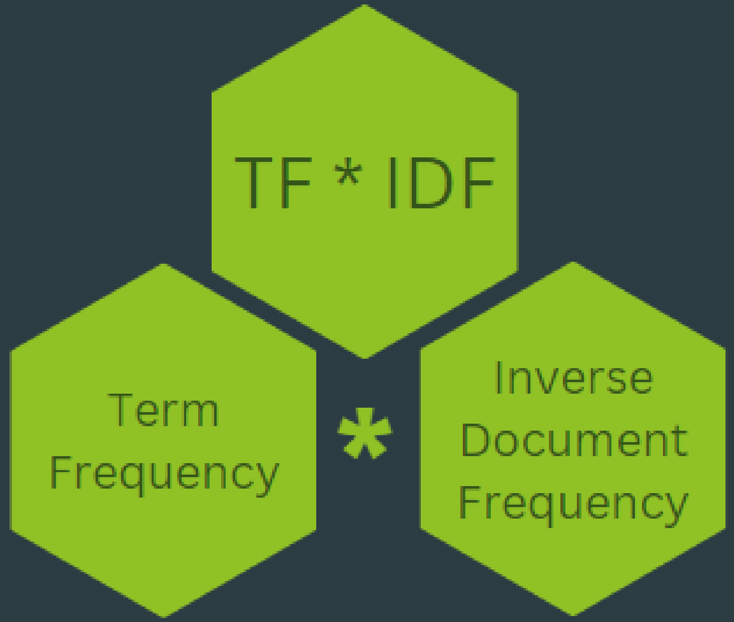
Preprocessing

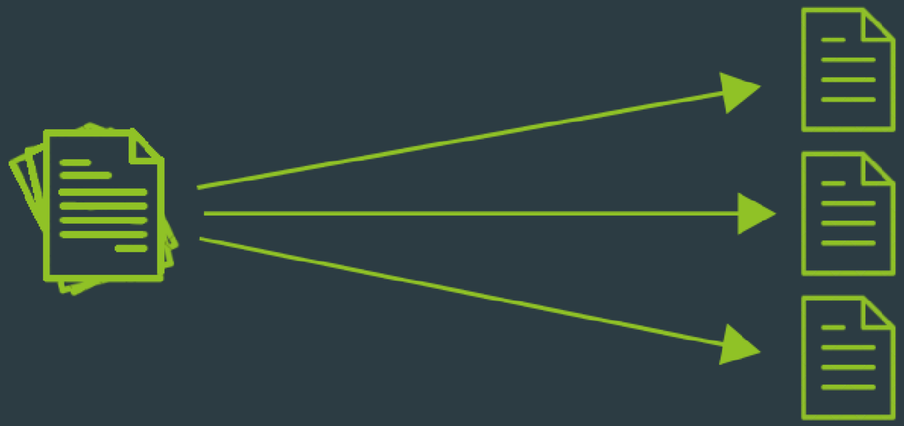
- Tokenization
  - Regular Expression: I'm --> "I", ",", "m"
  - NLTK: I'm ---> "I", "'m"
  - Digits stick to Words -> Manually Separated

- POS Tagging
  - Challenge: Role of Word
  - Solution: Add Subject to the sentences starting with verb

- Not Relevant
  - NER
  - Lemmatization

TF-IDF





```
ngram_range = (1, 3)
num = data['comment'].str.split().explode().nunique()
print('Unique Words in Data: ', num)
tfidf_vectorizer = TfidfVectorizer(ngram_range=ngram_range, min_df=5,
max_features=num, stop_words='english')
```

Sentiment Analysis using TF-IDF Vectorization + Tokens + POS Tags

```
X_train_new, X_test_new, y_train_new, y_test_new = 
train_test_split(
    data_tfidf, data['sentiment_numeric'], test_size=0.2, 
random_state=20, stratify=data['sentiment_numeric'])

classifier = MultinomialNB()
classifier.fit(X_train_new, y_train_new)
y_pred = classifier.predict(X_test_new)
accuracy = accuracy_score(y_test_new, y_pred)
classification_rep = classification_report(y_test_new, 
y_pred, target_names=sentiment_mapping.keys())

print(f'Accuracy: {accuracy:.2f}')
print(classification_rep)
```

Sentiment Analysis using TF-IDF Vectorization + Tokens + POS Tags

Accuracy: 0.81

	Precision	recall	f1-score	support
positive	0.89	0.86	0.88	86
Neutral	0.79	0.74	0.77	86
negative	0.76	0.83	0.79	86
accuracy			0.81	258
macro avg	0.81	0.81	0.81	258
weighted avg	0.81	0.81	0.81	258

Sentiment Analysis using TF-IDF Vectorization + Tokens + POS Tags

	Predicted Positive	Predicted Neutral	Predicted Negative
Actual Positive	74	10	2
Actual Neutral	18	63	5
Actual Negative	10	7	69

SVM

```
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

param_grid_svm = {'C': [0.1, 1, 10,100], 'gamma': ['scale', 'auto'],  'kernel': ['linear', 'rbf']}

svm_model = SVC(random_state=1)
svm_model_grid = GridSearchCV(estimator=svm_model, param_grid=param_grid_svm, verbose=10, cv=5, n_jobs=-1)
svm_model_grid.fit(X_train_tfidf, y_train)

results_df = pd.DataFrame(svm_model_grid.cv_results_)
print('Grid Search Results:')
print(results_df[['params', 'mean_test_score', 'rank_test_score']])

best_estimator_svm = svm_model_grid.best_estimator_
y_pred = best_estimator_svm.predict(X_test_tfidf)
```

SVM

Best Estimator: SVC(C=10, random_state=1)

Accuracy: 0.88

Classification Report:

	precision	recall	f1-score	support
positive	0.92	0.93	0.92	86
neutral	0.80	0.87	0.83	86
negative	0.92	0.83	0.87	86
accuracy			0.88	258
macro avg	0.88	0.88	0.88	258
weighted avg	0.88	0.88	0.88	258

	Predicted Positive	Predicted Neutral	Predicted Negative
Actual Positive	79	7	0
Actual Neutral	11	75	0
Actual Negative	0	14	72

Lowest Misclassification

LSTM with Word Embedding Vectorization

```
X_train = X_train.astype(str)
X_test = X_test.astype(str)

tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)
tokenizer.fit_on_texts(X_test)

X_train_sequences = tokenizer.texts_to_sequences(X_train)
X_test_sequences = tokenizer.texts_to_sequences(X_test)

max_sequence_length = num
X_train_padded = pad_sequences(X_train_sequences, maxlen=max_sequence_length, 
padding='post')
X_test_padded = pad_sequences(X_test_sequences, maxlen=max_sequence_length, 
padding='post')
```

LSTM with Word Embedding Vectorization

```
embedding_dim = 50
embedding_matrix = {}

with open('glove.6B.50d.txt', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embedding_matrix[word] = coefs

vocab_size = len(tokenizer.word_index) + 1
embedding_matrix_for_model = np.zeros((vocab_size, embedding_dim))

for word, i in tokenizer.word_index.items():
    embedding_vector = embedding_matrix.get(word)
    if embedding_vector is not None:
        embedding_matrix_for_model[i] = embedding_vector
```

LSTM with Word Embedding Vectorization

Layer (type)	Output Shape	Param #
embedding_4 (Embedding)	(None, 2791, 50)	101800
bidirectional_8 (Bidirectional)	(None, 2791, 100)	40400
dropout_4 (Dropout)	(None, 2791, 100)	0
bidirectional_9 (Bidirectional)	(None, 100)	60400
dense_3 (Dense)	(None, 3)	303

Total params: 202903 (792.59 KB)

Trainable params: 101103 (394.93 KB)

Non-trainable params: 101800 (397.66 KB)

LSTM with Word Embedding Vectorization

	precision	recall	f1-score	support
positive	0.89	0.86	0.88	86
neutral	0.76	0.88	0.82	86
negative	0.91	0.79	0.84	86
accuracy			0.84	258
macro avg	0.85	0.84	0.85	258
weighted avg	0.85	0.84	0.85	258

	Predicted Positive	Predicted Neutral	Predicted Negative
Actual Positive	74	10	2
Actual Neutral	10	76	0
Actual Negative	12	16	68

LSTM with TF-IDF Vectorization

- **TF-IDF Vectorization**:
  1. Converts text to numerical form, preserving important word frequencies
  2. Balances word significance across documents
- **Model Structure**:
  1. **Embedding Layer**: Projects words into a dense vector space
  2. **Bidirectional LSTM Layers**: Captures context from both past and future data points
  3. **Dropout Layer**: Prevents overfitting by randomly dropping units
  4. **Dense Output Layer**: Classifies text into sentiment categories
- **Training Details**:
  1. **Epochs**: 10
  2. **Batch Size**: 16
  3. **Validation Split**: 10%
- **Performance Metrics**:
  1. Tested accuracy and loss
  2. Improvement across epochs
- **Challenges & Insights**:
  1. Complexity and training time
  2. Overfitting concerns
  3. Effectiveness in capturing sentiment nuances

LSTM with TF-IDF Vectorization

```
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

model_lstm = Sequential()
max_sequence_length = max(X_train_tfidf.shape[1], X_test_tfidf.shape[1])
model_lstm.add(Embedding(input_dim=X_train_tfidf.shape[1], output_dim=50,
input_length=max_sequence_length))
model_lstm.add(Bidirectional(LSTM(50, return_sequences=True)))
model_lstm.add(Dropout(0.2))
model_lstm.add(Bidirectional(LSTM(50)))
model_lstm.add(Dense(3, activation='softmax'))

model_lstm.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
model_lstm.fit(X_train_tfidf.toarray(), y_train, epochs=10, batch_size=16, validation_split =
0.1)

test_loss, test_acc = model_lstm.evaluate(X_test_tfidf.toarray())
```

LSTM with TF-IDF Vectorization

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, 401, 50)	20050
bidirectional_6 (Bidirectional)	(None, 401, 100)	40400
dropout_3 (Dropout)	(None, 401, 100)	0
bidirectional_7 (Bidirectional)	(None, 100)	60400
dense_2 (Dense)	(None, 3)	303

Total params: 121153 (473.25 KB)

Trainable params: 121153 (473.25 KB)

Non-trainable params: 0 (0.00 Byte)

LSTM with TF-IDF Vectorization

	precision	recall	f1-score	support
positive	0.00	0.00	0.00	86
neutral	0.00	0.00	0.00	86
negative	0.33	1.00	0.50	86
accuracy			0.33	258
macro avg	0.11	0.33	0.17	258
weighted avg	0.11	0.33	0.17	258

LSTM with TF-IDF Vectorization

	Predicted Positive	Predicted Neutral	Predicted Negative
Actual Positive	0	86	0
Actual Neutral	0	86	0
Actual Negative	0	0	86

Prediction

```
def predict_sentiment(comment):

    comment_sequence = tokenizer.texts_to_sequences([comment])
    comment_padded = pad_sequences(comment_sequence, maxlen=50, padding='post')

    lstm_predictions = model_lstm.predict(comment_vectorized)
    lstm_pretrained_prediction = model_pretrained.predict(comment_padded)

    svm_predicted_class_index = np.argmax(svm_prediction)
    nb_predicted_class_index = np.argmax(nb_prediction)
    lstm_predicted_class_index = np.argmax(lstm_predictions)
    lstm_pretrained_prediction_class_index = np.argmax(lstm_pretrained_prediction)

    svm_sentiment = sentiment_classes[svm_predicted_class_index]
    nb_sentiment = sentiment_classes[nb_predicted_class_index]
    lstm_sentiment = sentiment_classes[lstm_predicted_class_index]
    lstm_pretrained_sentiment=sentiment_classes[lstm_pretrained_prediction_class_index]

    return svm_sentiment, nb_sentiment, lstm_sentiment, lstm_pretrained_sentiment, tokens, 
pos_tags
```

Prediction

```
def predict_from_command_line():
    comment = input('Enter your comment: ')
    if comment:
        svm_sentiment, nb_sentiment,
lstm_sentiment, lstm_pretrained_sentiment,
tokens, pos_tags= predict_sentiment(comment)
        print(f'Tokens: {tokens}')
        print(f'POS Tags: {pos_tags}')
        print(f'SVM Model Prediction:
{svm_sentiment}')
        print(f'Naive Bayes Model Prediction:
{nb_sentiment}')
        print(f'LSTM Model Prediction:
{lstm_sentiment}')
        print(f'LSTM Pretrained Model
Prediction: {lstm_pretrained_sentiment}')
    else:
        print('Please enter a comment.')

predict_from_command_line()
```

```
Enter your comment: I love the movie
1/1 [==============================] - 0s 69ms/step
1/1 [==============================] - 0s 28ms/step
Tokens: ['I', 'love', 'the', 'movie']
POS Tags: ['PRP', 'VBP', 'DT', 'NN']
SVM Model Prediction: negative
Naive Bayes Model Prediction: negative
LSTM Model Prediction: positive
LSTM Pretrained Model Prediction: positive
```

```
Enter your comment: I hate the movie
1/1 [==============================] - 0s 67ms/step
1/1 [==============================] - 0s 29ms/step
Tokens: ['I', 'hate', 'the', 'movie']
POS Tags: ['PRP', 'VBP', 'DT', 'NN']
SVM Model Prediction: negative
Naive Bayes Model Prediction: negative
LSTM Model Prediction: positive
LSTM Pretrained Model Prediction: negative
```

```
Enter your comment: I have no idea about the movie
1/1 [==============================] - 0s 70ms/step
1/1 [==============================] - 0s 28ms/step
Tokens: ['I', 'have', 'no', 'idea', 'about', 'the', 'movie']
POS Tags: ['PRP', 'VBP', 'DT', 'NN', 'IN', 'DT', 'NN']
SVM Model Prediction: negative
Naive Bayes Model Prediction: negative
LSTM Model Prediction: positive
LSTM Pretrained Model Prediction: neutral
```

Results & Comparison

Compare Results of Different Model:

Model	*Accuracy*
Naïve Bayes	0.81
SVM	0.88
LSTM with Word Embedding Vectorization	0.84
LSTM with TF-IDF Vectorization	0.33

Suggestion:

Best Model: LSTM

Suggestion on Improvement: More Data for Vectorization & Training