Midterm Literature Review: Document vs. Sentence vs. Aspect Level Sentiment Analysis

Emily Parrish emparrish93@gwu.edu
Shiva Omrani somrani@gwu.edu
Vedant Shrivastava vedant_s@gwu.edu
Marcelino Velasquez mav20@gwu.edu

Abstract

In this past decade, sentiment analysis/opinion mining has become an increasingly popular technique to assess and categorize the opinion of users over a product, topic, etc. via computational linguistics. Its approaches vary and can be divided into three main categories: document level, sentence level, and aspect level. This paper seeks to give an overview on Sentiment Analysis by exploring these three levels in detail, discussing different implementations and current applications of the technique, and reviewing its limitations and future possible works. The code for an example implementation is included in the appendix.

1 Introduction

Sentiment analysis, or opinion mining, is an area of Natural Language Processing (NLP) that moves beyond looking at patterns of language or inferring document context. It aims to truly decipher the emotional backbone of a specific context in an effort to personalize that document in a more subjective way. Not only can one approach sentiment analysis by looking at an entire document and aim to understand the sentiment at large, but also break down this analysis into sentence and even sub-sentence elements to get further understanding into the nuances of the opinion being conveyed in the text. Furthermore, one can combine the broader, document level applications of sentiment analysis and techniques with more granularity to accomplish larger tasks and compare polarities between a variety of different corpora types. In this survey, we will break down these approaches and discuss the benefits and limitations the techniques used to accomplish three levels of sentiment analysis: document, sentence, and aspect level (both categorical and term).

2 Background

Sentiment classification/opinion analysis can be broken down into two primary approaches, which are also sometimes used in conjunction with one another. A lexicon-based approach pre-compiles the opinion lexicon and uses statistical and semantic methods to identify opinion terms. Machine learning methods often prove more generalizable, as they utilize linguistic and syntactic features. This can be accomplished in both a supervised and unsupervised manner, but typically acts as a classic text classification problem (Medhat et al., 2014).

There are also multiple types of metrics and tasks that can be evaluated to infer sentiment of a corpus or entity (sentence, phrase, word, etc.). Some of the early approaches implemented a binary form of polarity detection, categorizing an entity as positive or negative. Over time a neutral class has been introduced and even finer grained and more continuous polarity intensity metrics by using Convolutional Neural Networks (CNN) and Long-short term memory (LSTM) techniques (Cambria et al., 2017). One can also assign opinion/sentiment to specific aspects of a sentence, for sentences that contain multiple sentiments towards specific objects. Finally, a common NLP sentiment analysis task is agreement detection, which assigns a binary label to whether two entities agree with one another in terms of their sentiment polarity (Cambria et al., 2013).

Some general challenges exist outside of the specific technique or models used often boil down to the semantics of a language. If a text lacks an explicit opinion, one must employ subjectivity detection to aid in sentiment classification (Cambria et al., 2013). Sentiment analysis models also struggle to detect elements like sarcasm and judging context (Bakshi et al., 2016). Polarity evaluation and mood classification often rely on a set of emotions and might limit the range of analysis to those moods. Also, because all of these contextual elements – subjective phrases, sarcasm, metaphor, etc. are subject to a specific linguistic structure and many corpora that have been developed have been done so in the English language, sentiment analysis can often be very language sensitive. (Cambria et al., 2013).

3 Methods and Approaches

Sentiment classification is usually conducted at three levels of granularity: document level, sentence level and aspect level (Liu). At the document level, the goal is to determine the overall sentiment of an opinion document such as the overall positive or negative opinion regarding an item in a product review (Liu).

At the sentence level, the task involves distinguishing opinionated sentences from factual sentences, and assessing whether they express a positive or a negative sentiment. Aspect level analysis is more fine-grained as it involves extracting sentiments expressed on different aspects/features of entities (Zhang et al., 2018). For example, aspect level sentiment analysis on a movie review could be identifying that people liked the casting and costume design in the movie, but disliked sound mixing and the story.

Lately, the outstanding performance of deep learning models in various fields such as computer vision led to their widespread adoption in the NLP community and specially for sentiment classification tasks. Deep learning's ability to learn multiple layers of features from the data makes it an ideal technique for learning word vector representations, as they can extract some of the semantic and syntactic information about words and encode them in vector space. This is especially remarkable for sentiment classification at the document and sentence levels, since the success of these tasks depends greatly upon an accurate representation of the underlying text.

3.1 Document Level Sentiment Classification

Although traditional methods such as using bag of words and its TFDIF for representing documents are effective to some extent, they fail to capture semantics and are also unable to preserve word order. Therefore, regardless of their content, two documents will have the exact same representation as long as they share the same words under the bag of words method (Zhang et al., 2018). Today's state of the art methods use word embeddings along with neural network models such as gated recurrent neural networks (GRU) and s and attention mechanisms to construct more accurate representation and achieve better results in sentiment classification.

One such technique is proposed by Yang et al. who suggest a hierarchical attention network (HAN) for document classification. They base their argument on the intuition that documents have a hierarchical structure, and that different words and sentences in a document are differentially informative (Yang et al., 2016). By using GRU-based sequence encoder, along with two levels of attention mechanisms both at word and sentence level they construct better document representation. Their proposed model outperforms many state of the art techniques in sentiment classification tasks.

Another creative approach suggested by Zhang et al. is to use Character-level CNNs, instead of word-based CNNs, for text classification. They show that when trained on a large scale dataset, CNNs do not require knowledge about the syntactic or semantic structure of a language and therefore show that their method can be extended to languages other than English. Although Zhang et al. obtain competetive results for many tasks including sentiment classification, they acknowledge the fact that the success of character-level CNNs depends on many factors such as dataset size, whether the text is curated and the choice of alphabet (Zhang et al., 2015).

3.2 Sentence Level Sentiment Classification

Similar to document level, representation plays an important role in sentiment classification at the sentence level. However, since the text is much shorter, additional information about the sentence such as part of the speech tags and other syntactic properties could be leveraged to help (Zhang et al., 2018). CNN and Recurrent Neural Networks' (RNN) ability to learn intrinsic relationships between words in a sentence have made them popular for sentiment classification at sentence level tasks (Zhang et al., 2018).

Kim showed that CNN's trained on top of pre-trained word2vec word embeddings produce competetive results for sentence level sentiment classification tasks and suggested that further improvement in performance could be achieved by learning task-specific vectors (Kim, 2014).

Despite their ability to extract higher level features that are invariant to local translation, CNNs require stacking multiple convolutional layers in order to capture long-term dependencies (Hassan and Mahmood, 2018). To handle this shortcoming, Hassan and Mahmood described a joint CNN and RNN framework which captures long term dependencies with fewer parameters while reducing the loss of details in local information. Their method achieves excellent results in several sentiment analysis benchmarks (Hassan and Mahmood, 2018).

3.3 Aspect Level Sentiment Classification

In general, aspect-level sentiment analysis can provide more details than a document level analysis, because rather than characterizing the sentiment polarities of a corpus at large, it attempts to classify polarity toward a single phrase or word. Aspect-level models can boil down to two main tasks: aspect-category sentiment analysis (ACSA) and aspect-term sentiment analysis (ATSA). ACSA looks to gain insight into polarity of a particular aspect classified under a particular list of categories, at a high level. ATSA looks more at the target entities that actually exist within the sentence.

Many novel techniques have been introduced in recent years to address the challenges and improve performance of models associated with aspect-level sentiment analysis. For ACSA, researchers implemented a convolutional neural network (CNN) with two separate convolutional layers atop an embedding layer. The outputs of these layers are pulled together by gating units that can extract features at many granularities such that aspect-specific sentiment information is inferred by the model. To generalize to ATSA, the model is extended with an additional convolutional layer to accommodate terms which contain more than one word (Xue and Li, 2018).

In 2014, Lau, Li, and Liao came up with a methodology for aspect-oriented sentiment analysis. They implemented a way to effectively mine a fuzzy ontology of a product via a Latent Dirichlet allocation i.e. LDA-based topic modeling. Their proposed solution could accurately predict the polarities of aspect-level sentiments without the requirement of any manual labeling of training examples. They also made use of a semi-supervised statistical learning method to extract context-sensitive sentiments from the user comments (Lau et al., 2014).

However as recent as early 2020, new models have been introduced and argued methods superior to CNNs. They claim that a CNN could fail in many cases for ATSA, when there are multiple terms in a single sentence where those terms differ in polarity. They also discuss RNNs, that sequentially encode a sentence, but are limited by long-term dependency because the gradients in a gradient decent methodology often disappear while training. Also, when encoding aspects separately, one loses contextual information (Xue and Li, 2018).

Xue et al. compare their new model with a variety of techniques including baseline methods, support vector machines (SVM) models what they consider the state-of-the-art currently, MGAN4 (which leverages both fine-grained attention and bidirectional coarse-grained attention mechanisms). The model itself is based on a multi-layer perceptron and "stacked layers of self attention". This deep self-attention layers are pre-trained and are used as an encoder to avoid problems in considering long-distance terms. Also, the multi-granularity better captures sentences and aspects so they can be used for sentiment polarity prediction (Xue and Li, 2018).

4 Discussion

There is a diverse range of applications when it comes to Sentiment Analysis and Opinion Mining. Firms can apply social analytics using sentiment analysis and opinion mining methodologies to tap into the collective social intelligence of something on the internet. This would help improve and innovate their product design and strategies for marketing. Similar applications can be made on social media to analyze an account's influence on that particular platform and even predict sales for accounts running businesses on such platforms. Besides this, the applications also enter into the world of online music streaming services such as Spotify, Apple Music, etc. where sentiment analysis and opinion mining would help make better music recommendations, generate playlists, analyse popular music trends in specific geographical areas, and more. Majority of the revenue generated by companies like Netflix, YouTube, and Amazon depend upon their state-of-the-art recommender systems that operate heavily using sentiment analysis and opinion mining methodologies.

When it comes to different methods and approaches, there are various benefits and drawbacks of document-level, sentence-level, and aspect-level sentiment classification that are dependent majorly on their use case. The simplest form among these is considered to be document-level as it classifies the entire textual review within a document as positive or negative. While it considers the whole matter to express a single opinion, sentence-level analysis considers each sentence as a separate unit usually assuming that it must be the opinion of one author about one topic only. On the other hand, aspect-level based analysis would take in consideration, every aspect and associate it with a sentiment expression. However, this may cause conflicting sentiments and assign dual opinions about something with respect to the aspects of its entities. When it lies too much between positive and negative so as to reach an impasse of categorizing it in either of the two, it causes neutrality which is another major challenge to the world of sentiment analysis and opinion mining.

Although aspect-level sentiment classification is the most trending and growing type among all, document-level and sentence-level sentiment classification are still being used for many contemporary applications from movie/product review monitoring to resume parsing and filtering bots to brand monitoring and social analytics. Besides, all these techniques can also be used in conjuction for various applications and such models can be applied to get the same metrics.

5 Conclusion

In this paper, we reviewed different sentiment analysis methods along with today's state of the art models. We have concluded that sentiment analysis is primarily conducted at the document, sentence and aspect levels and that deep learning is the popular technique for all three levels of granularity. Word embeddings obtained from neural models are used as the starting point for text representation and different neural network models such as CNNs and RNNs in conjunction with attention mechanisms are used to produce state of the art results in sentiment classification benchmarks. However, one limitation of the current research is that most techniques are focused on sentiment analysis within the English language. An interesting future direction for research in this literature would be to build a universal system that can perform sentiment classification for many languages despite their differences in syntactic structure.

References

R. K. Bakshi, N. Kaur, R. Kaur, and G. Kaur. 2016. Opinion mining and sentiment analysis. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 452–455.

E. Cambria, S. Poria, A. Gelbukh, and M. Thelwall. 2017. Sentiment analysis is a big suitcase. *IEEE Intelligent Systems*, 32(6):74–80.

E. Cambria, B. Schuller, Y. Xia, and C. Havasi. 2013. New avenues in opinion mining and sentiment analysis. *IEEE Intelligent Systems*, 28(2):15–21.

A. Hassan and A. Mahmood. 2018. Convolutional recurrent deep learning model for sentence classification. *IEEE Access*, 6:13949–13957.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar. Association for Computational Linguistics.

Raymond Y.K. Lau, Chunping Li, and Stephen S.Y.Liao. 2014. Social analytics: Learning fuzzy product ontologies for aspect-oriented sentiment analysis. *Decision Support Systems Vol. 65, Elsevier, pages 80-94.*

Bing Liu. *Sentiment analysis and opinion mining*. Synthesis lectures on human language technologies, 16. Morgan Claypool, San Rafael, Calif. (1537 Fourth Street, San Rafael, CA 94901 USA).

Walaa Medhat, Ahmed Hassan, and Hoda Korashy. 2014. Sentiment analysis algorithms and applications: A survey. *Ain Shams Engineering Journal*, 5(4):1093 – 1113.

Wei Xue and Tao Li. 2018. Aspect based sentiment analysis with gated convolutional networks. *CoRR*, abs/1805.07043.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, San Diego, California. Association for Computational Linguistics.

Lei Zhang, Shuai Wang, and Bing Liu. 2018. Deep learning for sentiment analysis : A survey. *CoRR*, abs/1801.07883.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Proceedings of the 28th International*

Conference on Neural Information Processing Systems - Volume 1, NIPS'15, page 649–657, Cambridge, MA, USA. MIT Press.

A Appendices

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import os
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix


import nltk
#nltk.download()
import string
import re
from collections import Counter


data = pd.read_csv('imdb_labelled.tsv',
                    header=None,
                   delimiter='\t')

data.columns = ['text', 'label']

data.head()


data['label'].unique()


data.shape


pos = []
neg = []

for i in data['label']:
    if i == 0:
        neg.append(1)
        pos.append(0)
    elif i == 1:
        pos.append(1)
        neg.append(0)

print(len(pos))
print(len(neg))


data['pos'] = pos
data['neg'] = neg

data.head()


# # Data Cleaning


# Convert to lowercase

data['text'] = [text.lower() for text in data['text']]

data['text'].head()


# word tokenization

from nltk.tokenize import word_tokenize, sent_tokenize
```

```python
token_text = [word_tokenize(text) for text in data['text']]

print(token_text)


# remove punctuation


no_punc = []

for filt in token_text:
    review = []
    for token in filt:
        new_token = re.sub('['+string.punctuation+']', u'', token)
        if not new_token == u'':
            review.append(new_token)
    no_punc.append(review)

print(no_punc)


# Removing the stopwords

from nltk.corpus import stopwords

no_stop = []

for text in no_punc:
    new_term_vector = []
    for word in text:
        if not word in stopwords.words('english'):
            new_term_vector.append(word)

    no_stop.append(new_term_vector)

print(no_stop)


# Stemming and Lemmatizing

from nltk.stem.porter import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

pstem = PorterStemmer()
wlem = WordNetLemmatizer()

preproc_text = []

for text in no_stop:
    final_text = []
    for word in text:
        pstem.stem(word)
        final_text.append(wlem.lemmatize(word))

    preproc_text.append(final_text)

print(preproc_text)


result = ['␣'.join(sen) for sen in preproc_text]

data['text_final'] = result
data['tokens'] = preproc_text
data = data[['text','text_final', 'tokens', 'label', 'pos', 'neg']]

data.to_csv('imdb.csv', index=False)

data.head()
```

```
# Model Building


# Split Data into Training and Testing

train_data, test_data = train_test_split(data, test_size=0.10, random_state=42)

train_data.to_csv(r'train_data.csv', index=False)
test_data.to_csv(r'test_data.csv', index=False)

'''
training vocabulary,
maximum training sentence length,
and total number of words in training data
'''

training_words = [word for tokens in train_data['tokens'] for word in tokens]

sent_lengths = [len(tokens) for tokens in train_data['tokens']]

train_vocab = sorted(list(set(training_words)))

print('{}␣words␣total,␣vocab␣size:␣{}'.format(len(training_words),
                                              len(train_vocab)))
print('Max␣sentence␣length:␣{}'.format(max(sent_lengths)))


# check for nulls

test_data.isnull().sum()


# Same as previous step except with testing data

test_words = [word for tokens in test_data['tokens'] for word in tokens]

sent_lengths = [len(tokens) for tokens in test_data['tokens']]

test_vocab = sorted(list(set(test_words)))

print('{}␣words␣total,␣vocab␣size:␣{}'.format(len(test_words),
                                              len(test_vocab)))
print('Max␣sentence␣length:␣{}'.format(max(sent_lengths)))


# Loading Google Word2Vec as pretrained word embeddings
import sys
#!{sys.executable} -m pip install gensim

from gensim import models

word2vec_path = 'https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz
word2vec = models.KeyedVectors.load_word2vec_format(word2vec_path, binary=True)


def get_average_word2vec(tokens_list, vector, generate_missing=False, k=300):
    if len(tokens_list)<1:
        return np.zeros(k)
    if generate_missing:
        vectorized =
        [vector[word] if word in vector else np.random.rand(k) for word in tokens_list]
    else:
        vectorized = [vector[word] if word in vector else np.zeros(k) for word in tokens_list]
    length = len(vectorized)
    summed = np.sum(vectorized, axis=0)
    averaged = np.divide(summed, length)
    return averaged
```

```
def get_word2vec_embeddings(vectors, clean_comments, generate_missing=False):
    embeddings = clean_comments['tokens'].apply(lambda x: get_average_word2vec(x, vectors,
                                                                             generate_missing=
                                                                             generate_missing)
    return list(embeddings)


# Get Embeddings

train_embeddings = get_word2vec_embeddings(word2vec, train_data, generate_missing=True)

max_seq_len = 50
emb_dim = 300


# import tensor

#!pip install tensorflow
import tensorflow as tf
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.layers import Dense, Dropout, Reshape, Flatten, concatenate, Input, Conv1D,
GlobalMaxPooling1D, Embedding
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model


# Tokenize & Pad Sequences

tokenizer = Tokenizer(num_words=len(train_vocab), lower=True, char_level=False)
tokenizer.fit_on_texts(train_data['text_final'].tolist())

train_seq = tokenizer.texts_to_sequences(train_data['text_final'].tolist())

train_word_index = tokenizer.word_index
print('{}␣unique␣tokens␣found.'.format(len(train_word_index)))


cnn_train_data = pad_sequences(train_seq, maxlen=max_seq_len)


train_embedding_weights = np.zeros((len(train_word_index)+1, emb_dim))
for word,index in train_word_index.items():
    train_embedding_weights[index,:] =
    word2vec[word] if word in word2vec else np.random.rand(emb_dim)
print(train_embedding_weights.shape)


test_seq = tokenizer.texts_to_sequences(test_data["text_final"].tolist())
cnn_test_data = pad_sequences(test_seq, maxlen=max_seq_len)


# Define CNN

def CNN(embeddings, max_seq_len, num_words, emb_dim, labels_index):

    embedding_layer = Embedding(num_words,
                                emb_dim,
                                weights=[embeddings],
                                input_length=max_seq_len,
                                trainable=False)

    sequence_input = Input(shape=(max_seq_len,), dtype='int32')
    embedded_sequences = embedding_layer(sequence_input)

    convs = []
    filter_sizes = [2,3,4,5,6]

    for filter_size in filter_sizes:
        l_conv =
        Conv1D(filters=200, kernel_size=filter_size, activation='relu')(embedded_sequences)
        l_pool = GlobalMaxPooling1D()(l_conv)
        convs.append(l_pool)


    l_merge = concatenate(convs, axis=1)

    x = Dropout(0.1)(l_merge)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.2)(x)
    preds = Dense(labels_index, activation='sigmoid')(x)

    model = Model(sequence_input, preds)
    model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=['acc'])
    model.summary() # brief summary of all layers
    return model


label_names = ['pos', 'neg']

y_train = train_data[label_names].values

x_train = cnn_train_data
y_tr = y_train

model = CNN(train_embedding_weights, max_seq_len, len(train_word_index)+1, emb_dim,
            len(list(label_names)))

# train cnn

num_epochs = 5 # num of iterations
batch_size = 48 # data at a time

hist = model.fit(x_train,
                 y_tr,
                 epochs=num_epochs,
                 validation_split=0.1,
                 shuffle=True,
                 batch_size=batch_size)


# Model Testing


# test cnn

predictions = model.predict(cnn_test_data, batch_size=1024, verbose=1)


labels = [1, 0]

prediction_labels=[]
for p in predictions:
    prediction_labels.append(labels[np.argmax(p)])

print('Model␣Accuracy:', sum(test_data['label']==prediction_labels)/len(prediction_labels))


test_data['label'].value_counts()
```