Aspect Based Sentiment Analysis - Restaurant Reviews

Brandeis University

193LING-131A-1

Introduction to Natural Language Processing with Python

Group Members:

Danling Ma

Shengchen Fu

Tianyi Zhou

Xuyuan Zhang

Table of Content

Introduction

Imagine you are starting your business, what would be the most important thing to consider in order to achieve success? The answer is definitely your customers. Nowadays customers are less sensitive about price but instead care more about the service quality. If they did not find the thing they want, they would simply look elsewhere for an alternative. Therefore, it is so important to understand your customers and deal with their demands as quickly as possible.

One way to communicate with your customers is to obtain customer feedback, either reviews or complaints. Customer feedback helps you understand their voice and know their expectations for products or services, so that managers are able to make key decisions based on insights from customers. Customer complaints also allow you to know where the drawbacks are and how to improve your products or services.

Despite the perceived benefits, it's still a great challenge for businesses to parse and organize this large amount of unstructured data into more digestible and actionable insights. Reviews are coming from different sources (e.g. Twitter, Yelp, etc.) and analyzing these unstructured data manually could be especially difficult and time-consuming. However, machine learning-based opinion mining techniques have the potential to enable automatic extraction of opinions and their corresponding polarities from such user-generated content.

Sentiment Analysis is the most common text classification tool. It uses natural language processing knowledge to identify whether the incoming message is positive, negative, neutral or conflict. It allows you to automatically analyze large amounts of data, which saves money, time and means teams can focus on more important tasks. In addition, this real-time analysis can help you immediately identify complaints and take action in time.

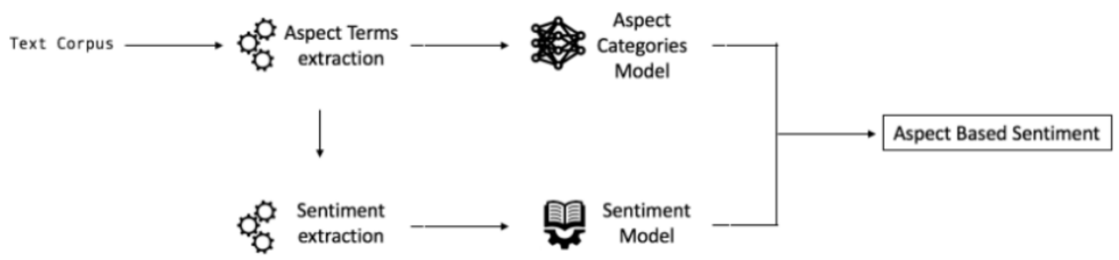
While sentiment analysis provides fantastic insights and has a wide range of real-world applications, the overall sentiment of a piece of text won't always pinpoint the root cause of an author's opinion. Customer reviews may contain sentiment about different aspects (e.g. food, service, etc.) and this information can be highly valuable for understanding customers' opinion about a particular service. Aspect-based Sentiment Analysis (ABSA) makes it easier to identify the sentiment towards specific aspects in the given text.

Instead of classifying the overall sentiment of a text into positive or negative, Aspect-based Sentiment Analysis (ABSA) analyzes each text to identify various aspects and determine the corresponding sentiment for each one, revealing more detailed and accurate information behind the text.

In this article, we will be focusing on the restaurant area by using restaurant reviews data and predict sentiment for various aspects such as food and service.

Overview

Our Aspect-based Sentiment Analysis (ABSA) is based on machine learning models and the ABSA model is composed of two distinct models: ***The Aspect Categories Model and the Sentiment Model***. The Aspect Categories Model is used to predict sentence category based on features generated by aspect terms. The Sentiment Model is used to predict the sentiment of the sentence based on features generated by polarity terms.



How to Run the Code

1. Download Stanford Parser files to get access via NLTK
   Requirements[1]
   Download two .jar files from the website:
    1. Stanford CoreNLP parser (3.9.2 version)
    2. English language model (3.9.2 English version)
2. Unzip file 1(Stanford CoreNLP parser) and put both of the files under the /config/.
   E.g.
   File 1: 'config/stanford-parser-full-2018-10-17/stanford-parser-3.9.2-models.jar'
   File 2: 'config/stanford-english-corenlp-2018-10-05-models.jar'
3. In the command line: $ python aspect_main.py:

   --extract (to extract aspect terms),

   --train (to train models)

   --test (to use a holdout data set to test model)

[1] NLTK StanfordNLP API Instruction, Stack Overflow. https://www.google.com/url?q=https://stackoverflow.com/a/33808164/10297662&sa=D&ust=1576830974136000&usg=AFQjCNFdrN4BSvkXS9Lg3q9Uo0XWEtw31Q

4. In the command line: `$ python senti_main.py`:
   --train_test (to train models and report the accuracy and elapsed time on the testing data)
5. In the command line: `$ python absa_combined.py`

Data

SemEval-2014 Restaurant Review dataset

- Aspect Term Extract
  - Training (Restaurants_Train_v2.xml): 700 reviews with 1 aspect category
  - Validation(Restaurants_Train_v2.xml): 300 reviews with 1 aspect category
  - Testing (Restaurants_Test_Gold.xml): 100 reviews with 1 aspect category
- Sentiment Classify
  - Training (Restaurants_Train_v2.xml): 2465 reviews with 1 aspect category
  - Testing (Restaurants_Test_Gold.xml): 800 reviews with 1 aspect category

Tools Used

- NLTK
- XML: to read the reviews in an xml file
- StanfordNLP: to get the dependency of words
- Sklearn: to learn and predict the aspect category and the sentiment
- Argparse

Challenges

- We assumed all sentences are grammatically correct, which is not always the real-life scenario.
- Aspect term extract:
  - Under what rules should we extract terms?
  - We used the rule-based method, and some of the aspect terms weren't extracted correctly by the rules.
- Sentiment extract:
  - What features and classifiers are optimal to generate higher accuracy of predicting sentiment with good time efficiency
  - How can we improve and generalize the existing work from assignment 5?
- How to design the user interface so the ABSA analyzer can generate real-time analysis and help user immediately identify complaints and take action in time

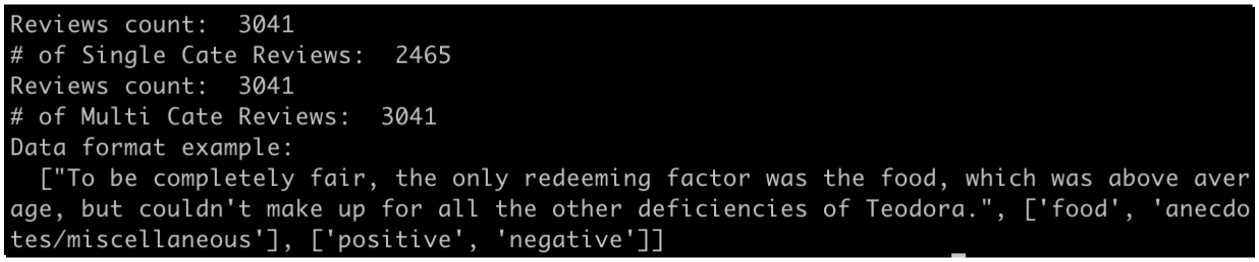
The Phase of the Project

Phase 1 Data Preprocessing

read_xml.py

This file is to read an XML format data. The SamEval reviews is in XML format. We need to parse out the ***reviews, aspect category, and polarity.***

$ python read_xml.py



```
Reviews count:  3041
# of Single Cate Reviews:  2465
Reviews count:  3041
# of Multi Cate Reviews:  3041
Data format example:
  ["To be completely fair, the only redeeming factor was the food, which was above aver
age, but couldn't make up for all the other deficiencies of Teodora.", ['food', 'anecdo
tes/miscellaneous'], ['positive', 'negative']]
```

Normalization: all the reviews are converted into lower case.

Phase 2 Aspect Category Model *(Danling Ma, Xuyuan Zhang)*

Task 1 Extract Aspect Terms based on Aspect Categories

stanfordNLP.py

This file is to adjust the format of the Stanford NLP parser for the preparation of the extractor.
The original output is:

```
[(('kitchen', 'NN'), 'nsubj', ('food', 'NN')),
 (('food', 'NN'), 'det', ('The', 'DT')),
 (('kitchen', 'NN'), 'cop', ('is', 'VBZ')),
 (('kitchen', 'NN'), 'advmod', ('exceptional', 'RB')),
 (('exceptional', 'RB'), 'advmod', ('uniformly', 'RB')),
 (('kitchen', 'NN'), 'case', ('with', 'IN')),
 (('kitchen', 'NN'), 'det', ('a', 'DT')),
 (('kitchen', 'NN'), 'amod', ('capable', 'JJ')),
 (('capable', 'JJ'), 'advmod', ('very', 'RB')),
 (('kitchen', 'NN'), 'acl:relcl', ('whip', 'VB')),
 (('whip', 'VB'), 'nsubj', ('which', 'WDT')),
 (('whip', 'VB'), 'aux', ('will', 'MD')),
...]
```

As we could see, each word has multiple dependency with other words in the sentence. Therefore, we need to combine multiple dependency words (kitchen ~ food, kitchen ~ is, kitchen ~ exceptional ...) together and affiliate them under the governor word, e.g. kitchen.

After adjusting:

```
sent = "The food is uniformly exceptional, with a very capable kitchen which will
proudly whip up whatever you feel like eating, whether it's on the menu or not."
    print("Dependency: ", sent_to_dep(sent))
```

```
>>> Dependency:  {
'feel': {'pos_tag': 'VBP', 'nsubj': 'you', 'nmod': 'eating', 'dobj': 'whatever',
'ccomp': 'menu'},
'whip': {'pos_tag': 'VB', 'compound:prt': 'up', 'aux': 'will', 'nsubj': 'which',
'advmod': 'proudly', 'ccomp': 'feel'},
'menu': {'pos_tag': 'NN', 'det': 'the', 'cop': "'s", 'conj': 'not', 'cc': 'or',
'mark': 'whether', 'case': 'on', 'nsubj': 'it'},
'kitchen': {'pos_tag': 'NN', 'cop': 'is', 'acl:relcl': 'whip', 'advmod':
'exceptional', 'case': 'with', 'amod': 'capable', 'det': 'a', 'nsubj': 'food'},
'exceptional': {'pos_tag': 'RB', 'advmod': 'uniformly'},
'eating': {'pos_tag': 'NN', 'case': 'like'},
'capable': {'pos_tag': 'JJ', 'advmod': 'very'}, 'food': {'pos_tag': 'NN', 'det':
'The'}}
```

After the adjustment, the word/key "kitchen" is related to its value (*{'pos_tag': 'NN', 'cop': 'is', 'acl:relcl': 'whip', 'advmod': 'exceptional', 'case': 'with', 'amod': 'capable', 'det': 'a', 'nsubj': 'food'}*)

Aspect_extractor.py

This file is to extract aspect terms based on rules below.

Extract rules:

Our system is based on 2 general rules.

1. For sentences having a subject-verb. (7 sub-rules)
2. For sentences without a subject-verb. (3 sub-rules)

Rules:

For sentences with subject-verb.

1. if t has any adverbial or adjective modifier and the modifier exists in SentiWordNet, then t is extracted.

```
        if not words[word].__contains__("nsubj"):
                continue
                # Point 1
                if words[word]["pos_tag"] in verb and checkModifiers(words, word):
                    aspect_terms.append(word)
```

2. If the sentence does not have an auxiliary verb and t modified by adjective or adverb then both h & t are aspects.

```
        # Point 2
                elif not has_auxiliary and getAdverbOrAdjective(words, word):
                    try:
                        if (words[word].__contains__("nsubj") and \
                        not ("DT" in words[words[word]["nsubj"]]["pos_tag"])):
                            aspect_terms.append(words[word]["nsubj"])
                            aspect_terms.append(word)
                    except:
                        print("** Rule2 Fail: ", word, words[word])
```

3. t is in direct object relation with noun n,
    a. if n not in SentiWordNet -> n is aspect term.

```
    # Point 3
                elif not has_auxiliary and words[word].__contains__("dobj") and
isNoun(words, words[word]["dobj"]):
                    if not wordInSentic(words[word]["dobj"]):
                        aspect_terms.append(words[word]["dobj"])
```

    b. Else n is in SentiWordNet > n is connected to n1(Noun) using any dependency relation then n1 is aspect term

```
                    else:
                        word1 = getNounConnectedByAny(words, words[word]["dobj"])
                        if (word1):
                            aspect_terms.append(word1)
```

4. if t has any direct object relation with a token n and the POS of the token n is Noun and n exists in SentiWordNet, then the token n extracted as aspect term. In the dependency parse tree of the sentence, if another token n1 is connected to n using any dependency relation and the POS of n1 is Noun, then n1 is extracted as an aspect.

```
    # Point 4
                elif not has_auxiliary and (words[word].__contains__("xcomp")):
                    xcomp = words[word]["xcomp"]
                    word1 = getNounConnectedByAny(words, xcomp)
                    if (word1):
                        aspect_terms.append(word1)
```

5. t connected to t1 using open clausal complement relation and t-t1 should exist in opinion lexicon.
    a. Then t-t1 is extracted as aspect.
    b. If n (Noun) connected to t1 then n is aspect term
6. If t is in relation with copular verb then t is an aspect.
7. If h is noun then h is also aspect.

```
    elif "cop" in words[word]:
            dep = getDependency(words, word)
                if wordInSentic(word) and words[word]["pos_tag"] in noun:
                    # print("In 5", word)
                    aspect_terms.append(word)
                try:
                    # print("In 6", words[word]["nsubj"])
                    if "nsubj" in words[word] \
                    and not ("DT" in words[words[word]["nsubj"]]["pos_tag"] \
                    or "PRP" in words[words[word]["nsubj"]]["pos_tag"]):
                        aspect_terms.append(words[word]["nsubj"])
                except KeyError:
                    print("** Rule6 Fail: ", words[words[word]["nsubj"]],
                          words[word]["nsubj"], word)
                if dep:
                    # print("In 7", dep)
                    aspect_terms.append(dep)
```

For sentences without subject verb:

1. Adverb h is in clausal complement relation with t, then h is extracted as adverb

```
    else:
            for word in words.keys():
                prepN = hasPropositionalNoun(words, word)
                tmp = getVmodorXcomp(words, word)
                if tmp and wordInSentic(tmp):
                    # print("In 8", word)
                    aspect_terms.append(word)
```

2. if a token h is connected to a noun t using a prepositional relation, then both h and t are aspects

```
                elif prepN:
                    # print("In 9")
                    if "appos" in words[word]:
                        # print(words[word]["appos"])
                        aspect_terms.append(words[word]["appos"])
                    # print(prepN)
                    aspect_terms.append(prepN)
```

3. If h is in direct object relation with t , then t is aspect term

```
                elif "dobj" in words[word]:
                    # print("In 10")
                    tmp1 = words[words[word]["dobj"]]["pos_tag"]
                    if not ("DT" in tmp1) or ("PRP" in tmp1):
                        # print(words[word]["dobj"])
                        aspect_terms.append(words[word]["dobj"])
```

Using these set of rules we get aspect terms from reviews.

Task 2 Modeling Process

aspect_main.py

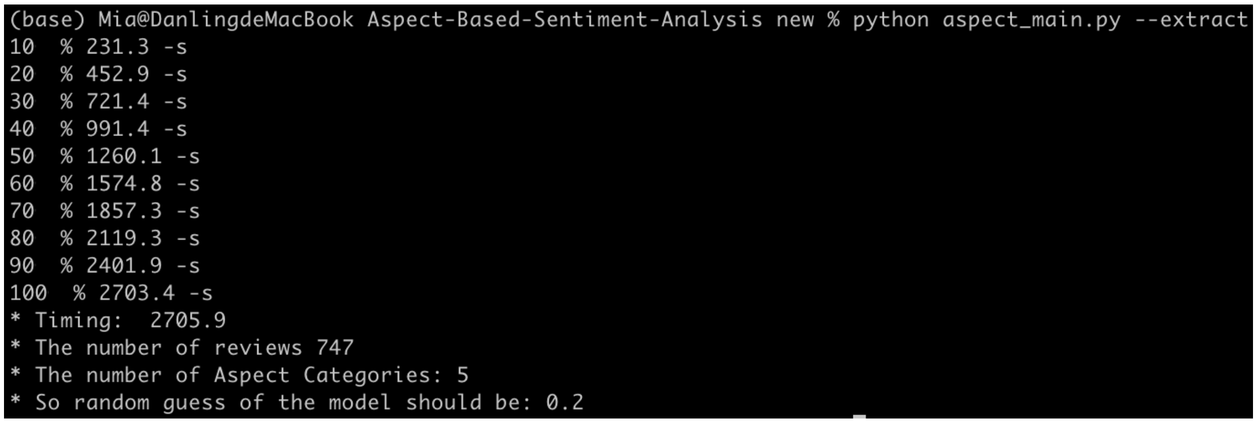
This file is to generate features based on extracted aspect terms, train models, and generate model performance.

Data:

```
# read xml file, only include reviews with one category (single_cate)
f_path = "./data/Restaurants_Train_v2.xml"
reviews = read_xml(f_path, single_cate=True, print_len=False)
test_path = './data/Restaurants_Test_Gold.xml'
test_reviews = read_xml(test_path, single_cate=True, print_len=False)
```

Feature Generation:

```
$ python aspect_main.py --extract
```



```
(base) Mia@DanlingdeMacBook Aspect-Based-Sentiment-Analysis new % python aspect_main.py --extract
10  % 231.3 -s
20  % 452.9 -s
30  % 721.4 -s
40  % 991.4 -s
50  % 1260.1 -s
60  % 1574.8 -s
70  % 1857.3 -s
80  % 2119.3 -s
90  % 2401.9 -s
100  % 2703.4 -s
* Timing:  2705.9
* The number of reviews 747
* The number of Aspect Categories: 5
* So random guess of the model should be: 0.2
```

After running aspect_main.py, we got the above result.
With 1000 reviews, we got 747 reviews in total after dropping the reviews without any aspect terms extracted.

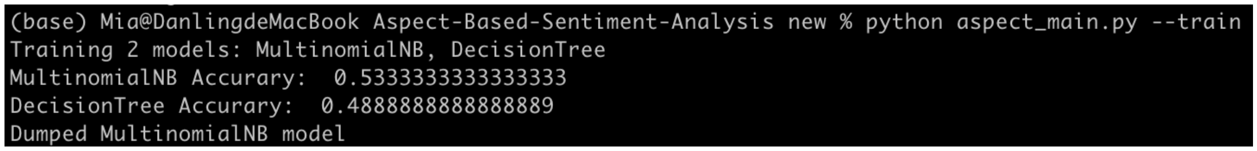
We are learning and predicting 5 classes. Therefore, the random guess should be 0.2, which is a score our model should beat later.

```python
if args.extract:
   # term generation
   start = time.time()
   X = []
   y = []
   cnt = 0
   for row in reviews:
       # try:
       # if cnt < 50:
       cnt += 1
       if cnt % 10 == 0:
           print(cnt,"%", round(time.time()- start,1), "-s")
       review = row[0].lower()
       asp_cat = row[1]
       dep = sent_to_dep(review)
       aspects = extractor(dep)
       # reviews that doesn't have terms won't be in the training set
       if len(aspects) > 0:
           X.append(nltk.FreqDist(aspects))
           y.append(asp_cat[0])

   # transform aspect terms to raw count
   v = DictVectorizer()
   Xs = v.fit_transform(X)

   # dump data
   dump(Xs, open('data/processed_Xs.jbl', 'wb'))
   dump(y, open('data/processed_y.jbl', 'wb'))
   file = open('data/vectorizer.pkl', 'wb')
   dump(v, file)
   file.close()
```

For each (review, category) item, we first normalizing the reviews by lowering case. Then use `sent_to_dep()` to get the dependency of each review and extracts aspect terms from `dep` using `extractor()`.

After getting X and y, we use `DictVectorizer()` to transform and get raw count and dump the processed data into `/data/` for later use.

Models:



```
(base) Mia@DanlingdeMacBook Aspect-Based-Sentiment-Analysis new % python aspect_main.py --train
Training 2 models: MultinomialNB, DecisionTree
MultinomialNB Accuracy:  0.5333333333333333
DecisionTree Accurary:  0.4888888888888889
Dumped MultinomialNB model
```

We adopted two models: MultinomialNB and DecisionTree. And we use the train/validation method to get the accuracy.
From the above, the MultinomialNB model beats both the random guess(0.2) and DecisionTree model. Therefore, we choose to dump the model with higher accuracy.

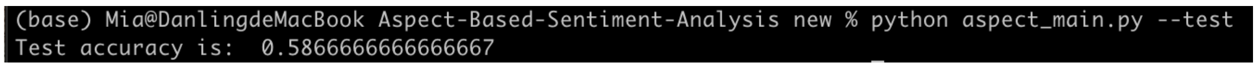
```python
if args.train:
   # load data
   Xs = load(open('data/processed_Xs.jbl', 'rb'))
   y = load(open('data/processed_y.jbl', 'rb'))

   # data split
   X_train, X_val, y_train, y_val = train_test_split(Xs, y, train_size=0.7,
random_state=0)

   # train/valid model
   print("Training 2 models: MultinomialNB, DecisionTree")
   clf1 = MultinomialNB().fit(X_train, y_train)
   clf2 = DecisionTreeClassifier().fit(X_train, y_train)
   acc1 = clf1.score(X_val, y_val)
   acc2 = clf2.score(X_val, y_val)
   print("MultinomialNB Accurary: ", acc1)
   print("DecisionTree Accurary: ", acc2)

   # dump the model with highest accuracy
   if acc1 > acc2:
       print('Dumped MultinomialNB model')
       dump(clf1, open('clf.jbl', 'wb'))
   else:
       print('Dumped DecisionTree model')
       dump(clf2, open('clf.jbl', 'wb'))
```

Model Performance:

Baseline:

```
set(y)

{'ambience', 'anecdotes/miscellaneous', 'food', 'price', 'service'}
```

From above, we know that we are predicting 5 classes, which means our random guess is 0.2. And our model should beat this score.



```
(base) Mia@DanlingdeMacBook Aspect-Based-Sentiment-Analysis new % python aspect_main.py --test
Test accuracy is:  0.5866666666666667
```

As we had expected, the model gives an accuracy much higher than 0.2, indicating the model has a good performance.

```
if args.test:
    # load mdl
    mdl = load(open('clf.jbl','rb'))
    v = load(open('data/vectorizer.pkl','rb'))

    # read test data
    test = read_xml('data/Restaurants_Test_Gold.xml', single_cate=True,
print_len=False)
    test = test[:100]

    #process data...
    X_test = []
    y_test = []
    cnt = 0
    for row in test:
        review = row[0].lower()
        asp_cat = row[1]
        dep = sent_to_dep(review)
        aspects = extractor(dep)
        # reviews that doesn't have terms won't be in the training set
        if len(aspects) > 0:
            X_test.append(nltk.FreqDist(aspects))
            y_test.append(asp_cat[0])

    # transform aspect terms to raw count
    X_test = v.transform(X_test)
    acc = mdl.score(X_test, y_test)
    print("Test accuracy is: ", acc)
```

The test data includes 100 reviews that were unseen by the model. We first loaded the model and vectorizer we dumped before and then the accuracy on test data is generated after we apply the model onto the test data. The reason why we are using a test data in addition to a validation data is that the test data can give us a more accurate and reliable result as the model has not seen this data before.

Phase 3 Sentiment Model *(Shengchen Fu, Tianyi Zhou)*

Task 1 Extract Polarity Terms

senti_extractor.py

This file is for generating vocabulary of polarity terms and using these vocabularies to generate features of the given data. The vocabulary decides the size of a feature, which is the number of columns in an array. We applied 5 rules without or with negation and generated 10 vocabularies based on training data and other wordnets such as. There are 2 methods we used to generate features: count and binary. Thus, in the later Task2, for each classifier we have 20 features to create a model and compare the performance.

Before we applied the rules to generate the vocabularies, we preprocessed the data by lowering and tokenizing all the words.

- Five rules and ten vocabularies(Class **`Vocabularies()`**)
    - Raw words

      All words from training data excluding stopwords.
    - SentiWordNet

      Words both in training data and SentiWordNet with positive or negative score over 0.5
    - MPQA

      Words both in training data and MPQA Subjectivity Lexicon
    - Part of Speech tagging

      Words where part of speech are adjective, adverb or verb

```
def getPOS(self):
       # Rule 1: Part of Speech is adj, adv or verb
       # Return a set of 2163/2159 words
       self.sentiment_terms = set()
       for i in self.pos_tags:
           for j in i:
               if j[1] in ['ADJ', 'ADV', 'VERB']:
                   self.sentiment_terms.add(j[0])
       return self.sentiment_terms
```

-
    - POS and Distributional Thesaurus

      Words where part of speech are adjective, adverb or verb and their top 2 synonyms in WordNet

```
def getPOSnDT(self):
       # Rule 2: add Distributional Thesaurus into POS vocabularies
       # Use the top 5 DT expansions of current token
       # Return a set of 8157 words
       self.sentiment_terms_DT = self.sentiment_terms.copy()
       for i in self.sentiment_terms:
           synonyms = set()
           for syn in wordnet.synsets(i)[:2]:
               for w in syn.lemmas():
                   synonyms.add(w.name())
           self.sentiment_terms_DT.update(synonyms)
       return self.sentiment_terms_DT
```

-
    - Additional process - Negation

After using the previous 5 rules to generate 5 vocabularies, we double the size of the 5 vocabularies by adding the negated form of each word in the vocabulary. These 5 negated vocabularies were only used to generate features when the sentences were also negated.

```
def getPOS_neg(self):
        # Add negated form words to previous sentiment terms
        s_terms_neg = self.sentiment_terms.copy()
        for w in self.sentiment_terms:
            negated_w = 'not_' + w
            s_terms_neg.update(negated_w)
        return s_terms_neg
```

- Two methods to generate features (Class **`Features()`**)
    - Preprocess - Negation

      By assigning True to parameter **`neg`** in class **`Features`**, we preprocess the training data sentences by adding 'not_' before words that follow a negative word such as *don't, never, nothing* and so on. The result of negated sentences was only used when vocabulary was also negated to generate features.
    - Method 1 - Count

      The value in the feature array is the count of appearance of each word in the vocabulary.
    - Method 2 - Binary

      The value in the feature array is 1 when the word appeared at least once otherwise 0.

Since we only created not-negated pairs and both-negated pairs of vocabulary and feature generating method, we prepared 5*2 + 5*2 = 20 features to fit classifiers and calculate accuracy and elapsed time to figure out the optimal model in the Task 2 Sentiment Model. We used the function **`getFeatureDict()`** and stored the 20 features in a dictionary where key is the vocabulary with the first value is the array from counting and the second value is the array from binarizing.

Table: feature combinations

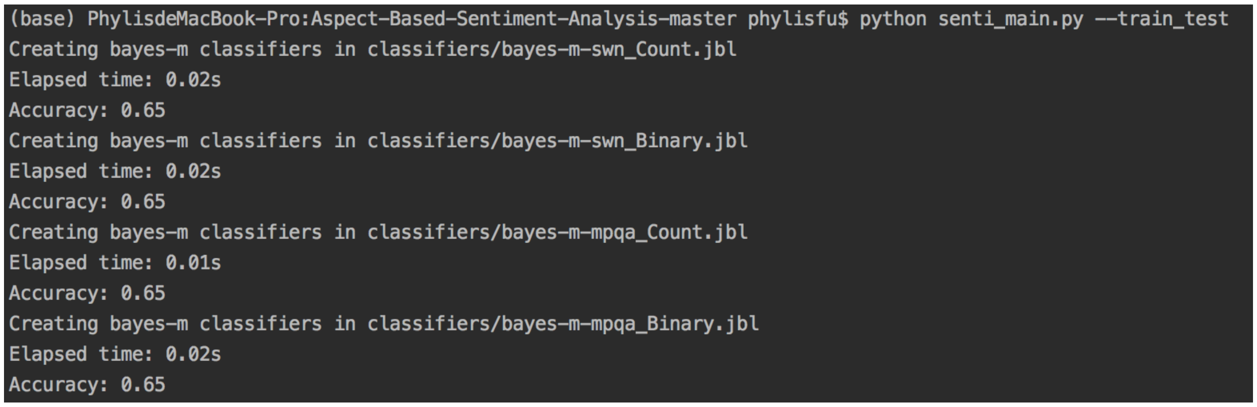
vocabulary	**method**	
Raw	count	binary
swn	count	binary
mpqa	count	binary
POS	count	binary
POS_DT	count	binary

Raw_neg	count_neg	binary_neg
swn_neg	count_neg	binary_neg
mpqa_neg	count_neg	binary_neg
POS_neg	count_neg	binary_neg
POS_DT_neg	count_neg	binary_neg

Task 2 Modeling Process

senti_extractor.py: def model()
senti_main.py

This file is for training the models by applying 20 features in the previous task to different classifiers. We used 6 classifiers, which are multinomial naive bayes, bernoulli naive bayes, decision trees, support vector machine, linear support vector machine and neural network. As a result, we trained 120 models and stored them into `/classifiers/` using **joblib** package. Accuracy on testing data and model training elapsed time were also calculated and printed by running `$ python senti_main.py --train_test` in the command line.



```
(base) PhylisdeMacBook-Pro:Aspect-Based-Sentiment-Analysis-master phylisfu$ python senti_main.py --train_test
Creating bayes-m classifiers in classifiers/bayes-m-swn_Count.jbl
Elapsed time: 0.02s
Accuracy: 0.65
Creating bayes-m classifiers in classifiers/bayes-m-swn_Binary.jbl
Elapsed time: 0.02s
Accuracy: 0.65
Creating bayes-m classifiers in classifiers/bayes-m-mpqa_Count.jbl
Elapsed time: 0.01s
Accuracy: 0.65
Creating bayes-m classifiers in classifiers/bayes-m-mpqa_Binary.jbl
Elapsed time: 0.02s
Accuracy: 0.65
```

Below is the table of multinomial naive bayes classifier that shows the evaluation metrics. The table is generated by running the script **senti_extractor.py**. Please see the appendix for tables of the rest classifiers.

Table: Accuracy and Elapsed Time of Multinomial Naive Bayes Classifier

	acc_count	**acc_binary**	**time_count**	**time_binary**
swn	0.65	0.65	0.03	0.02

swn_neg	0.66	0.66	0.03	0.03
mpqa	0.65	0.65	0.02	0.02
mpqa_neg	0.66	0.66	0.03	0.03
Raw	0.7	0.7	0.08	0.05
Raw_neg	0.69	0.69	0.15	0.16
POS	0.7	0.71	0.03	0.03
POS_neg	0.7	0.7	0.03	0.03
POS_DT	0.68	0.68	0.12	0.08
POS_DT_neg	0.67	0.68	0.12	0.07

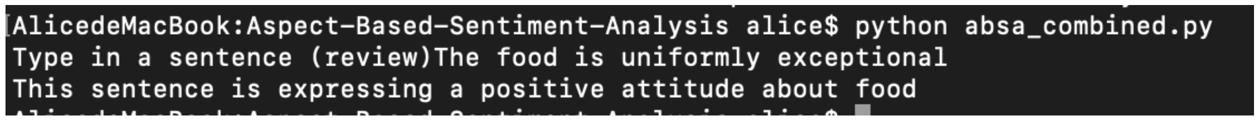
We compared the evaluation metrics of 6 classifiers with 20 features. For classifiers, in terms of the accuracy, Multinomial Naive Bayes reached the highest score, followed by Support Vector Machine and Neural Network. While considering the elapsed time, Multinomial Naive Bayes reached the fastest speed , followed by Bernoulli Naive Bayes and Support Vector Machine. However, Neural Network required 200 times of the time compared to the fastest models. Thus, we considered Multinomial Naive Bayes and Support Vector Machine as the top 2 optimal classifiers.

For features, based on all models, two different methods generated similar accuracy while binarizing required less time. Raw vocabulary and part of speech vocabulary are generally the 2 optimal ones. **Thus, our best model is the one using POS vocabulary to generate binary feature for Multinomial Naive Bayes classifier.**

Phase 4 Combining the Two model

absa_combined.py

By running this file, the user will be asked to type in a sentence (review) and the aspect-based sentiment will be given as output. In other words, the user will be able to know which aspect the sentence is talking about and what the sentiment is.



```
AlicedeMacBook:Aspect-Based-Sentiment-Analysis alice$ python absa_combined.py
Type in a sentence (review)The food is uniformly exceptional
This sentence is expressing a positive attitude about food
```

```
def sent_sentiment(sent):

   # predict category
   dep = sent_to_dep(sent)
   aspects = extractor(dep)
   raw_count = nltk.FreqDist(aspects)
   v = load(open('data/vectorizer.pkl', 'rb'))
   X = v.transform(raw_count)
   mdl = load(open('clf.jbl', 'rb'))
   category = mdl.predict(X)

   # predict sentiment

   f_path = "./data/Restaurants_Train_v2.xml"
   reviews = read_xml(f_path, single_cate=True, print_len=False)
   voc = Vocabularies(reviews)
   POS = voc.getPOS()
   feature = Features(data=sent, vocabulary=POS, neg=False, file=True)
   feature_set = feature.binary()
   mdl_senti = load(open('classifiers/bayes-m-POS_Binary.jbl', 'rb'))
   sentiment = mdl_senti.predict(feature_set)

   return ("This sentence is expressing a " + str(list(sentiment)[0]) + " attitude
about " + str(list(category)[0]))
```

We used the best-performed category model and sentiment model to predict both category and sentiment for the given sentence. And the result is printed as:
E.g. This sentence is expressing a positive attitude about food.

Final Review

Aspect-based sentiment analysis (ABSA) can help businesses become customer-centric and place their customers at the heart of everything they do. It's about listening to customers, understanding their voices, analyzing their feedback, and learning more about customer experiences, as well as their expectations for products or services.

While Aspect-based sentiment analysis (ABSA) has been proved to be very useful in reading user-generated context, there are still challenges we need to face in order to improve the model performance. In this article, we assume that all the reviews are grammatically correct. But what if there are typos in the review? And what if the review does not follow general English grammar? There are still many more we need to consider.

Also, our models are trained on single aspect data, while often in a real business case, customers make comments on multiple aspects. To link each aspect with the sentiment, we need to involve dependency method. Considering the problem complexity and the time limit, we focused on single aspect ABSA model in this project. To improve the result, the multiple aspects based sentiment analysis can be the next stage.

Reference

AspectExtractRules_Reference.pdf (https://github.com/TianyiZhou1/Aspect-Based-Sentiment-Analysis/blob/master/AspectExtractRules_Reference.pdf)

Appendix

Table: Accuracy and Elapsed Time of Bernoulli Naive Bayes Classifier

	acc_count	acc_binary	time_count	time_binary
swn	0.66	0.66	0.04	0.02
swn_neg	0.64	0.64	0.07	0.05
mpqa	0.65	0.65	0.02	0.02
mpqa_neg	0.64	0.64	0.05	0.04
Raw	0.64	0.64	0.12	0.1
Raw_neg	0.6	0.6	0.3	0.3
POS	0.67	0.67	0.08	0.05
POS_neg	0.67	0.67	0.08	0.05
POS_DT	0.61	0.61	0.19	0.15
POS_DT_neg	0.61	0.61	0.26	0.15

Table: Accuracy and Elapsed Time of Decision Trees

	acc_count	acc_binary	time_count	time_binary
swn	0.59	0.59	0.58	0.61
swn_neg	0.6	0.59	1	1.09
mpqa	0.56	0.57	0.49	0.55
mpqa_neg	0.6	0.6	1.07	1.11
Raw	0.58	0.59	7.53	5.95
Raw_neg	0.57	0.58	12.17	8.71
POS	0.57	0.58	1.17	1.24
POS_neg	0.56	0.55	1	1
POS_DT	0.58	0.58	2.28	1.96
POS_DT_neg	0.58	0.58	1.49	1.46

Table: Accuracy and Elapsed Time of Support Vector Machine

	acc_count	acc_binary	time_count	time_binary
swn	0.6	0.6	7.87	7.94
swn_neg	0.6	0.6	13.87	13.6
mpqa	0.6	0.6	6.14	6.35
mpqa_neg	0.6	0.6	13.63	13.64
Raw	0.6	0.6	32.8	33.57
Raw_neg	0.6	0.6	61.28	61.35
POS	0.6	0.6	18.26	18.02
POS_neg	0.6	0.6	17.35	17.38
POS_DT	0.6	0.6	47.24	46.42
POS_DT_neg	0.6	0.6	48.6	46.73

Table: Accuracy and Elapsed Time of Linear Support Vector Machine

	acc_count	acc_binary	time_count	time_binary
swn	0.63	0.63	0.07	0.05
swn_neg	0.63	0.63	0.06	0.05
mpqa	0.64	0.64	0.1	0.05
mpqa_neg	0.63	0.63	0.07	0.05
Raw	0.66	0.67	0.1	0.07
Raw_neg	0.68	0.69	0.18	0.18
POS	0.66	0.65	0.1	0.06
POS_neg	0.66	0.66	0.1	0.06
POS_DT	0.67	0.68	0.15	0.11
POS_DT_neg	0.67	0.67	0.13	0.1

Table: Accuracy and Elapsed Time of Neural Network

	acc_count	acc_binary	time_count	time_binary

swn	0.63	0.64	50.14	50.44
swn_neg	0.64	0.64	92.25	94.83
mpqa	0.62	0.62	37.11	33.43
mpqa_neg	0.64	0.64	94.03	94.29
Raw	0.67	0.67	140.64	147.6
Raw_neg	0.68	0.68	231.95	227.57
POS	0.65	0.64	109.28	114.65
POS_neg	0.64	0.64	100.22	99.6
POS_DT	0.68	0.68	282.78	283.18
POS_DT_neg	0.66	0.65	218.36	226.3