IST 664 - Natural Language Processing

Final Project: Classification of Text

Cherngywh Lee

Kaggle competition movie review phrase data, labeled for sentiment
Table of contents

Step 1: Pre-processing or filtering - Alpha filter

Step 2: Create feature function for experiments
- *Trigram & POS tag*
- *Negation & LIWC*
- *Weka*

Step 3: Advance experiment
- *Negation & AFINN*

Baseline
- *5000 phrases & 5 folds*
- *30000 phrases & 10 folds*

Alpha Filter
- *5000 phrases & 5 folds*
- *15000 phrases & 10 folds*

Trigram & POS tag feature function
- *5000 phrases & 5 folds*
- *15000 phrases & 10 folds*
- *Compare with Weka*

Negation & LIWC feature function
- *5000 phrases & 5 folds*
- *15000 phrases & 10 folds*
- *Compare with Weka*

Negation & AFINN feature function
- *5000 phrases & 5 folds*
- *15000 phrases & 10 folds*
- *Compare with Weka*

Conclusion

Step 1: pre-processing or filtering

First of all, we need to clean the data. Because this is for sentiment analysis and I'm going to use sentiment lexicon which only contain alphabetic words. So I decided to remove punctuation by alpha_filter function.

```
def alpha_filter(w):
  # pattern to match word of non-alphabetical characters
  pattern = re.compile('^[^a-z]+$')
  if (pattern.match(w)):
    return True
  else:
    return False
```

Applied the function when tokenizing.

```
  for phrase in phraselist:
    tokens = nltk.word_tokenize(phrase[0])
    alphawords = [w for w in tokens if not alpha_filter(w)]
    phrasedocs.append((alphawords, int(phrase[1])))
```

Then I need to consider if I need to remove stop words or not. However I realized there will be a conflict between stop words and negation words when I was planning to do negation experiment.

```
 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at',
'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'du
ring', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'i
n', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once
', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', '
each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not',
 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'wil
l', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm',
```

We can find some critical negation words in the stop words list which I think very important for representing sentiments. And also, because these reviews are all short sentences, I think it's necessary to keep all words of them for more precise analysis.

Finally, I only chose alpha filter as a further filtering method.

Step 2: Create feature function for experiments

Experiment 1 - Trigram & POS tag

I would like to try some basic knowledge of NLP for experiment 1. So, I combined trigram and POS tag features.

```
def tri_features(document, word_features, trigram_features):
    document_words = set(document)
    document_trigrams = nltk.trigrams(document)
    tagged_words = nltk.pos_tag(document)
    features = {}
    for word in word_features:
        features['V_{}'.format(word)] = (word in document_words)
    for trigram in trigram_features:
        features['T_{}_{}_{}'.format(trigram[0], trigram[1], trigram[2])] = (trigram
in document_trigrams)
    numNoun = 0
    numVerb = 0
    numAdj = 0
    numAdverb = 0
    for (word, tag) in tagged_words:
        if tag.startswith('N'): numNoun += 1
        if tag.startswith('V'): numVerb += 1
        if tag.startswith('J'): numAdj += 1
        if tag.startswith('R'): numAdverb += 1
    features['nouns'] = numNoun
    features['verbs'] = numVerb
    features['adjectives'] = numAdj
    features['adverbs'] = numAdverb
    return features
```

Experiment 2 - Negation & LIWC

The 2nd experiment, I think it's time to borrow the wisdom from smart people and also take a couple of straightforward methods for the sentiment analysis. I think negation words are very important especially when they occurred in a simple sentence, only the word can determine the sentiment of the whole sentience.

```
def tri_features(document, word_features):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['V_{}'.format(word)] = False
        features['V_NOT{}'.format(word)] = False
    # go through document words in order
    for i in range(0, len(document)):
        word = document[i]
        if ((i + 1) < len(document)) and ((word in negationwords) or
(word.endswith("n't"))):
            i += 1
            features['V_NOT{}'.format(document[i])] = (document[i] in word_features)
        else:
            features['V_{}'.format(word)] = (word in word_features)
    # apply LIWC and count the score for Positive and Negative respectively
    Pos = 0
    Neg = 0
    poslist, neglist = sentiment_read_LIWC_pos_neg_words.read_words()
    for word in document_words:
        if isPresent(word, poslist):
            Pos += 1
        if isPresent(word, neglist):
            Neg += 1
    features['positivecount'] = Pos
    features['negativecount'] = Neg
    return features
```

Step 3: Advance experiment

Advance Experiment – Negation & AFINN

I googled some sentiment lexicon and found a popular lexicon – AFINN. Instead of classifying words, it directly gives words a score. I think it's interesting if we can get the score of the whole sentence by adding them up.

In order to compare AFINN with LIWC and I also thought negation is important, so I still keep Negation in the advance experiment.

```
def tri_features(document, word_features):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['V_{}'.format(word)] = False
        features['V_NOT{}'.format(word)] = False
    # go through document words in order
    for i in range(0, len(document)):
        word = document[i]
        if ((i + 1) < len(document)) and ((word in negationwords) or
(word.endswith("n't"))):
            i += 1
            features['V_NOT{}'.format(document[i])] = (document[i] in word_features)
        else:
            features['V_{}'.format(word)] = (word in word_features)
    # apply the AFINN and calculate the total score of words in a sentence
    afinn_list = []
    flexicon = open(afinn, encoding='latin1')
    wordlines = [line.strip() for line in flexicon]
    for line in wordlines:
      if not line == '':
        items = line.split("\t")
        afinn_list.append((items[0], int(items[1])))
    score = 0
    for word in document_words:
      for i in range(len(afinn_list)):
        if word == afinn_list[i][0]:
          score += afinn_list[i][1]
    features['scores'] = score
    return features
```

Baseline

5000 phrases & 5 folds

Average Precision Recall F1 Per Label

0	0.228	0.190	0.206
1	0.235	0.362	0.283
2	0.818	0.623	0.707
3	0.249	0.403	0.307
4	0.160	0.289	0.205

Macro Average Precision Recall F1 Over All Labels
0.338 0.374 0.342

Label Counts {'3': 1036, '4': 303, '2': 2566, '0': 227, '1': 868}
Micro Average Precision Recall F1 Over All Labels
0.532 0.492 0.498

15000 phrases & 10 folds

Average Precision Recall F1 Per Label

0	0.259	0.193	0.219
1	0.262	0.397	0.315
2	0.824	0.642	0.721
3	0.265	0.445	0.332
4	0.201	0.292	0.237

Macro Average Precision Recall F1 Over All Labels
0.362 0.394 0.365

Label Counts {'3': 3192, '1': 2578, '0': 665, '2': 7736, '4': 829}
Micro Average Precision Recall F1 Over All Labels
0.549 0.519 0.520

Comparison

We can see that more data and folds can raise the average result. We also can see labels which originally didn't have too many phrases significantly improve with larger dataset.

Alpha Filter

5000 phrases & 5 folds

Average Precision Recall F1 Per Label

0	0.117	0.126	0.120
1	0.223	0.363	0.274
2	0.831	0.621	0.711
3	0.248	0.391	0.303
4	0.128	0.216	0.160

Macro Average Precision Recall F1 Over All Labels
0.309 0.343 0.314

Label Counts {'2': 2575, '0': 210, '3': 1060, '1': 844, '4': 311}
Micro Average Precision Recall F1 Over All Labels
0.531 0.483 0.492

15000 phrases & 10 folds

Average Precision Recall F1 Per Label

0	0.240	0.221	0.229
1	0.247	0.369	0.296
2	0.819	0.629	0.711
3	0.277	0.460	0.345
4	0.238	0.349	0.282

Macro Average Precision Recall F1 Over All Labels
0.364 0.406 0.373

Label Counts {'0': 685, '4': 920, '3': 3156, '2': 7702, '1': 2537}
Micro Average Precision Recall F1 Over All Labels
0.546 0.513 0.516

Comparison

We only keep alphabetic words after this alpha filter. Looks like it doesn't have a significant improvement. Still keep this filter for further experiments.

Trigram & POS tags

5000 phrases & 5 folds

Average Precision	Recall	F1	Per Label
0	0.211	0.178	0.193
1	0.277	0.363	0.313
2	0.806	0.638	0.712
3	0.243	0.401	0.302
4	0.152	0.281	0.194

Macro Average Precision	Recall	F1	Over All Labels
0.338	0.372	0.343	

Label Counts {'3': 1031, '1': 843, '4': 285, '0': 232, '2': 2609}

Micro Average Precision	Recall	F1	Over All Labels
0.536	0.501	0.507	

15000 phrases & 10 folds

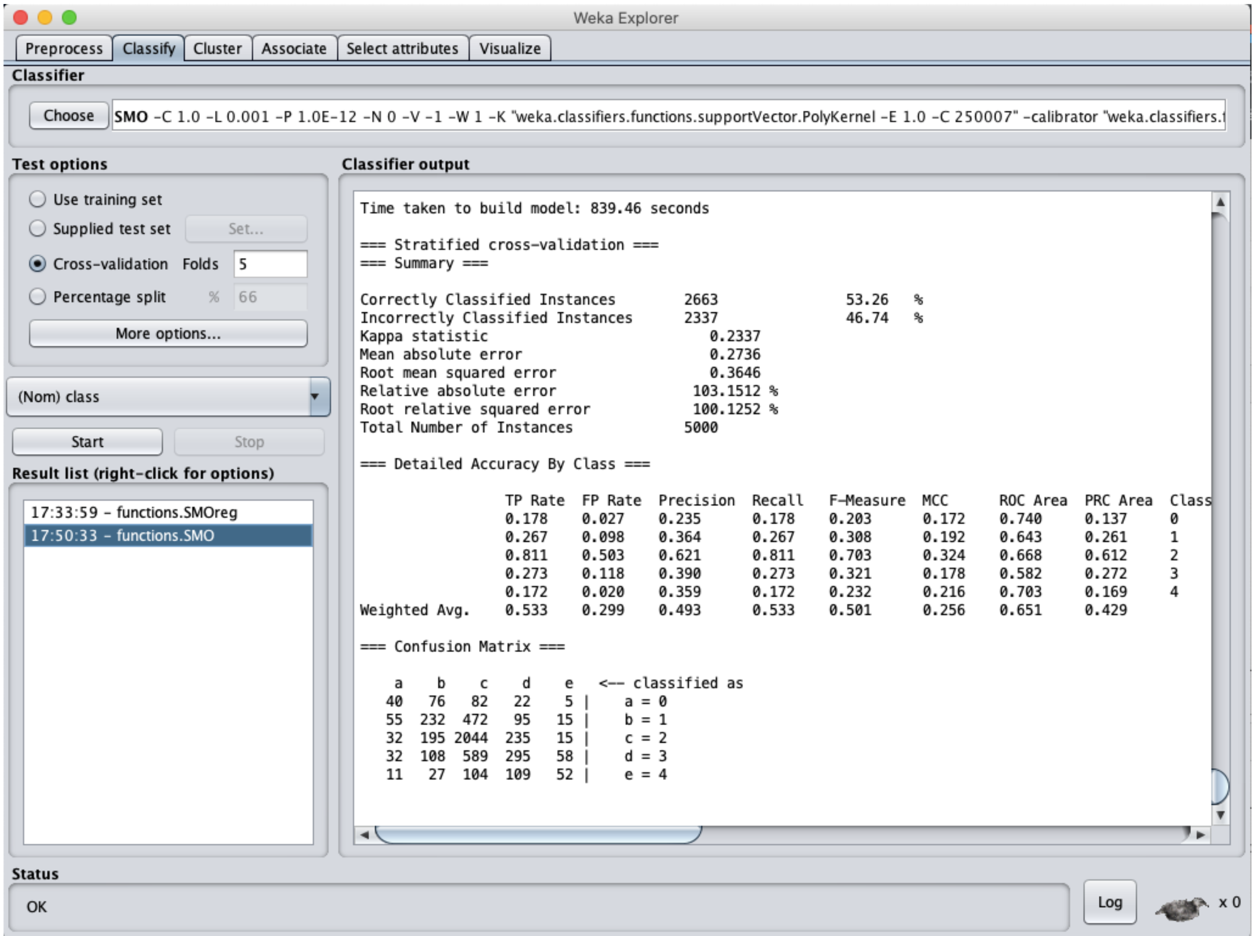
Average Precision	Recall	F1	Per Label
0	0.304	0.203	0.242
1	0.262	0.397	0.315
2	0.800	0.635	0.708
3	0.250	0.430	0.316
4	0.250	0.276	0.261

Macro Average Precision	Recall	F1	Over All Labels
0.373	0.388	0.368	

Label Counts {'2': 7566, '1': 2704, '0': 680, '3': 3150, '4': 900}

Micro Average Precision	Recall	F1	Over All Labels
0.532	0.508	0.507	

Weka SMO classifier 5000 phrases & 5 folds



Comparison

Weighted Average	Precision	Recall	F1
5000 phrases & 5 folds	0.536	0.501	0.507
15000 phrases & 10 folds	0.532	0.508	0.507
Weka (5000 & 5folds)	0.493	0.533	0.501

We got the best result with 5000 phrases & 5 folds, didn't see a significant improvement after applying Trigram & POS.

Negation & LIWC

5000 phrases & 5 folds

Average Precision Recall F1 Per Label

0	0.225	0.173	0.194
1	0.283	0.392	0.328
2	0.787	0.655	0.715
3	0.284	0.422	0.339
4	0.222	0.218	0.218

Macro Average Precision Recall F1 Over All Labels

0.360 0.372 0.359

Label Counts {'1': 886, '4': 292, '2': 2555, '3': 1047, '0': 220}

Micro Average Precision Recall F1 Over All Labels

0.535 0.513 0.516

15000 phrases & 10 folds

Average Precision Recall F1 Per Label

0	0.426	0.177	0.250
1	0.280	0.415	0.334
2	0.733	0.690	0.711
3	0.323	0.456	0.378
4	0.395	0.289	0.332

Macro Average Precision Recall F1 Over All Labels

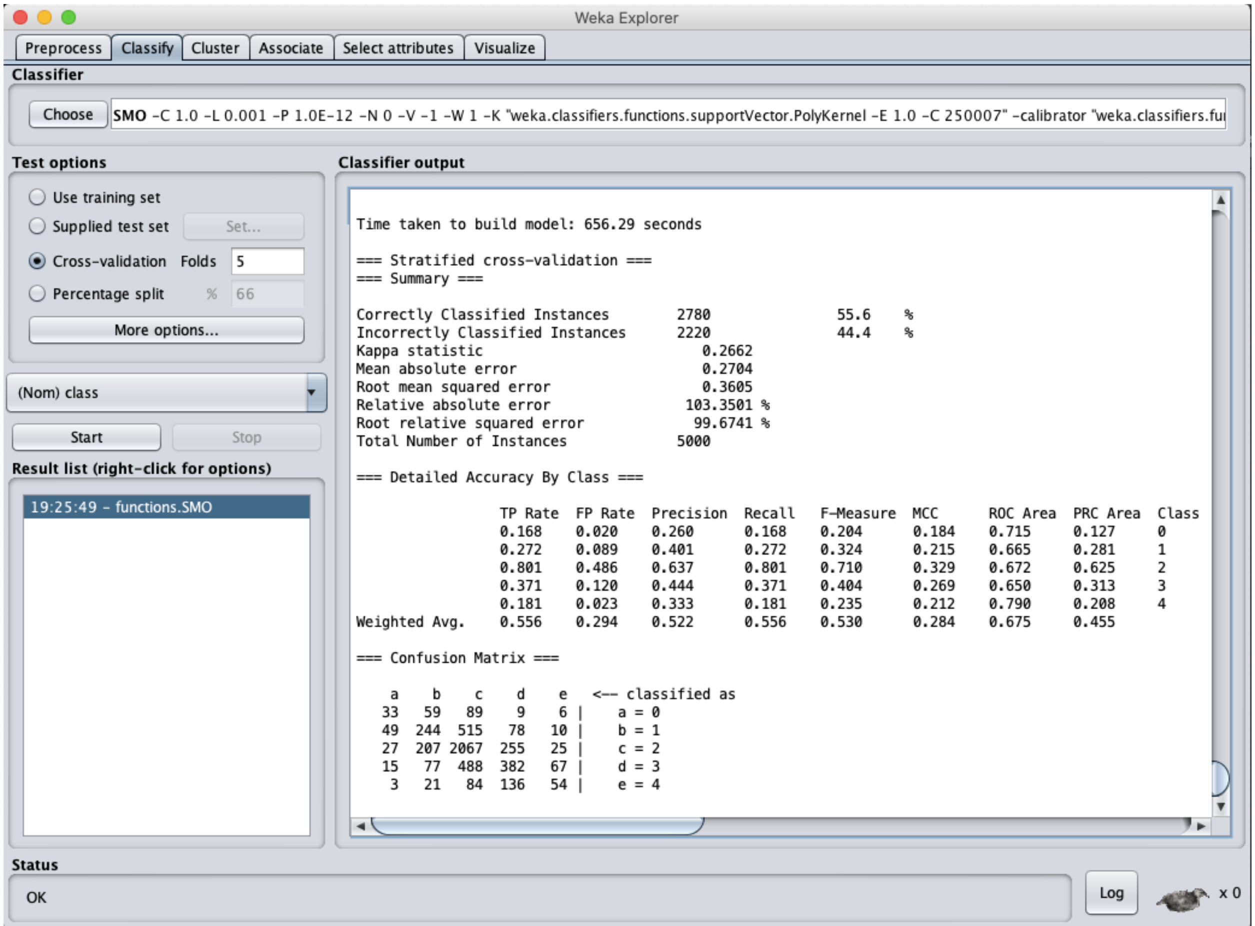
0.431 0.405 0.401

Label Counts {'0': 687, '4': 880, '2': 7594, '3': 3226, '1': 2613}

Micro Average Precision Recall F1 Over All Labels

0.532 0.545 0.530

Weka SMO classifier 5000 phrases & 5 folds



Comparison

Weighted Average	Precision	Recall	F1
5000 phrases & 5 folds	0.535	0.513	0.516
15000 phrases & 10 folds	0.532	0.545	0.530
Weka (5000 & 5folds)	0.522	0.556	0.530

Although we still got the best result with 5000 phrases & 5 folds, we can see some phenomena:

1. The precision of label 2 dropped, it even dropped more with more data.
2. Recall and F1 improved a little

3. The precision of labels with fewer samples improved significantly, and it even improved more when we have more data (15000 phrases) which we didn’t see too much difference in former experiments.
4. Weka has a significant improvement in this experiment.

Negation & AFINN

5000 phrases & 5 folds

Average Precision Recall F1 Per Label

0	0.251	0.184	0.211
1	0.250	0.382	0.302
2	0.766	0.640	0.697
3	0.314	0.419	0.358
4	0.302	0.296	0.297

Macro Average Precision Recall F1 Over All Labels
0.377 0.384 0.373

Label Counts {'0': 226, '4': 296, '2': 2490, '3': 1071, '1': 917}
Micro Average Precision Recall F1 Over All Labels
0.524 0.504 0.506

15000 phrases & 10 folds

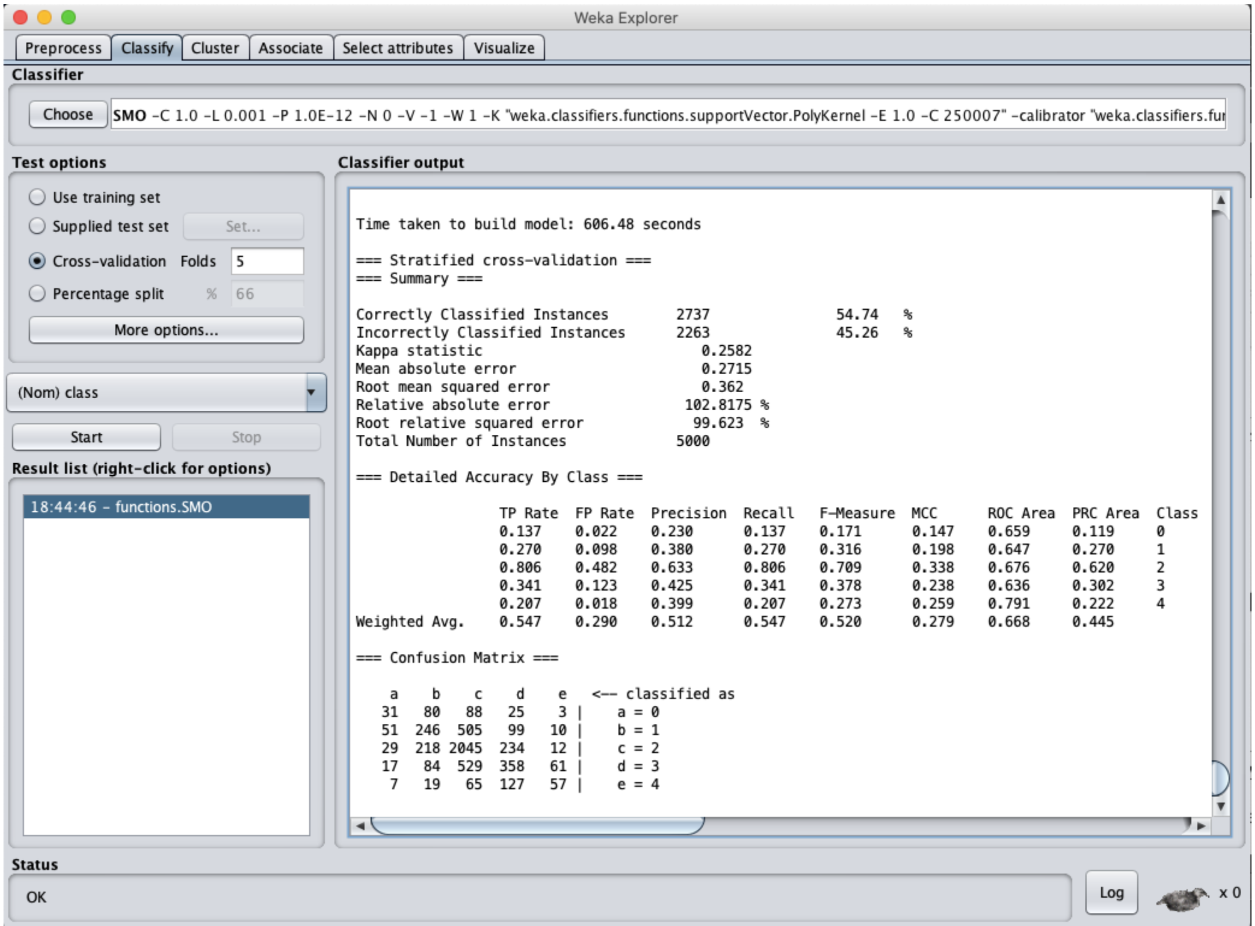
Average Precision Recall F1 Per Label

0	0.422	0.175	0.247
1	0.292	0.419	0.343
2	0.728	0.685	0.706
3	0.319	0.470	0.380
4	0.373	0.260	0.306

```
Macro Average Precision Recall          F1      Over All Labels
                   0.427      0.402      0.396

Label Counts {'1': 2696, '4': 866, '0': 685, '3': 3188, '2': 7565}
Micro Average Precision Recall          F1      Over All Labels
                   0.529      0.543      0.527
```

Weka SMO classifier 5000 phrases & 5 folds



Comparison

Weighted Average	Precision	Recall	F1
5000 phrases & 5 folds	0.524	0.504	0.506
15000 phrases & 10 folds	0.529	0.543	0.527
Weka (5000 & 5folds)	0.512	0.547	0.520

This time we got the best result with 15000 phrases & 10 folds, we can see some phenomena:

1. The precision of label 2 dropped, it even dropped more with more data.
2. Recall and F1 improved a little

3. The precision of labels with few samples improved significantly, and it even improved more when we have more data (15000 phrases) which we didn’t see too much difference in former experiments.
4. Weka has a significant improvement in this experiment.

Conclusion

The experiment with the best Precision score

Macro Average – Negation & LIWC (15000 phrases & 10 folds)

Precision	Recall	F1
0.431	0.405	0.401

Micro Average – Baseline (15000 phrases & 10 folds)

Precision	Recall	F1
0.549	0.519	0.520

The experiment with the best Recall score

Macro Average – Alpha Filter (15000 phrases & 10 folds)

Precision	Recall	F1
0.364	0.406	0.373

Micro Average – Negation & LIWC (15000 phrases & 10 folds)

Precision	Recall	F1
0.532	0.545	0.530

In general:

- Actually, I had pretty similar result like 15000 phrases & 10 folds by 10000 phrases & 5 folds. I think more samples for each label does improve the Precision when a fold size is under 1500 samples.
- Still chose 15000 phrases & 10 folds to make a distinction between 5 folds.
- We only ran 5000 phrases 5 folds on Weka duo to the computation limitation. But we can see Weka always has the highest Recall score.

After applying sentiment lexicons:

- The label "2" represents "Neutral", which is the major category of the original dataset. We can see Precision of label 2 dropped whereas Recall seems didn't change that much. It means False Positive (Type I) decreased and False Negative (Type II) increased.

- The Precision of labels without many samples but have strong sentiment significantly improved, whereas Recall didn't also improve that much. It means False Positive (Type I) Increased and False Negative (Type II) decreased.

After checking the precision of label "2", we can see a limitation about 0.82. I think this is a fundamental limitation of the methods I used. I need to try more complex and profound methods, **otherwise it may be just kind of a trade-off between "Neutral" and "Non-Neutral".**

However, because this is about movie review, So **I think we actually care about the review with strong sentiments more.**

It's obvious that sentiment lexicons successfully made it.

Because the sentiment lexicon analyzes the sentence by words. It helps to figure more sentences with stronger sentiment. **But maybe it's too detail so sometimes it may misclassify some natural sentences like "I like the book version more" as positive. Hence it creates more type I error (Neutral but classified as Strong sentiment).**

And because of it, type II error increased on "Neutral".

But again, I still think it's a good result because we care about the reviews with strong sentiment more.

In conclusion, the performance of sentiment lexicons like LIWC and AFINN did a better job on finding out more sentences with sentiment. Whereas Alpha Filter and Trigram & POS tags are doing better on the general purpose.