CENG 463

Natural Language Processing

Fall 2023-2024

Assignment 1

Text Classification

Full Name: **Mert Uludoğan**

Student ID: **2380996**

1 Task 1

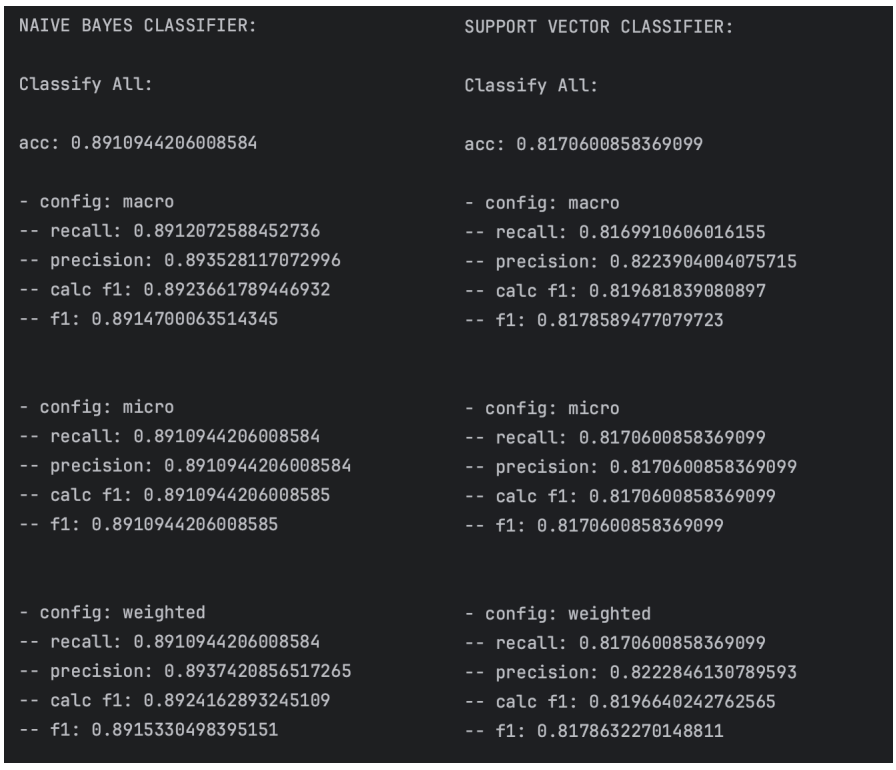

Figure 1: Overall Accuracy, F1 score, Recall and Precision for NB and SVM

- In my implementation I have a "Module" class.
- runNaiveBayes() and runSVC() to print out import features about results
- The data is the descriptions, name of the film is not used in training set. Before tokenizing to $P(Desc|C) = P(W_1) * ...$, I "lower all characters" and apply regex to "eliminate numbers".

- Then I tokenized description to words by splitting with blanks: " ".
- I filtered words by:
  - Stopwords: Examples of stopwords in English include words like "the", "and", "is", "in", "of" etc. These words are generally ignored or filtered out when processing text data to focus on the more meaningful words.
  - string.punction: to filter punctions
  - PREPROCESS-CUSTOM-FILTER-LST: is my eye observed filter to not choose some words. Not patterns but definite words.
- Tokenized Data is description represented with frequency count of words as features.
- This data is fit in "Multinominal Naive Bayes" and "Linear SVC" classifiers of "scikit-learn" module.
- resulted y test, predicted classes by data.
- It is tested for "f1 score", "accuracy score", "precision score", and "recall score" (Figure 1)
- All observations and tunnings computed on "development set".
- "Test set" is used just for final result of my configurations.
- comments:
  - Naive Bayes Classifier dominates Support Vector Classifier on all metrics.
  - "Weighted averages" as metrics' parameter dominates on both classifiers.

```
horror: 66848
mystery: 76077
philosophy: 68004
religion: 80689
romance: 75814
science-fiction: 70430
science: 90836
sports: 87604
```

Figure 2: Sample sizes

  - Samples sizes are not perfectly balanced but not so many differed. It shows the reason why macro, micro and weighted averages differs very slightly.
  - We can see true positives, false negatives, false positives and true negatives on Figure 3 but it can be more easy to look at the features calculated with confusion matrix (Figure 4).
  - Religion has relatively poor accuracy even it has sufficient amount of data. It indicates the low efficiency raise from feature extraction: Most commons are mostly same but at different frequencies. For example, while "faith" is appeared 366 times with 3rd most frequent in train sample religion class, it is appeared 71 times at 7th most frequent. Again, the most 6-8 terms are so similar but the frequency indexing very differs, it may occur cause the test samples may be about significantly "different context" about religion class (It might affect the most frequent token indexing). The difference is not so much significant actually cause the accuracy is not so low, but lower than the other classes.

SUPPORT VECTOR CLASSIFIER:

```
[[190  18   6   1   4   1  14   0]
 [ 26 197   3   0   5   1   5   3]
 [  0   0 199  14   0  10   4   1]
 [  4   0  33 178   1   6   8   0]
 [  7  10   1   1 170   0  14  25]
 [  5   0  21   1   0 197   6   0]
 [ 26  10   0   1   4   3 196   0]
 [  4   2   3   0  29   0   0 196]]
```

NAIVE BAYES CLASSIFIER:

```
[[203  13   0   1   8   0   9   0]
 [ 15 213   0   0  10   0   2   0]
 [  0   0 209  10   0   9   0   0]
 [  5   0  26 192   1   2   4   0]
 [  2   4   0   2 208   0   6   6]
 [  1   0  12   2   0 210   5   0]
 [ 15   6   0   0   5   2 212   0]
 [  0   0   0   0  19   1   0 214]]
```

Figure 3: Confusion Matrix for SVC and NVC

	Num of Data	Accuracy	Recall	Precision	F1 Score
Overall	636302	0.89	0.89	0.89	0.89
Horror	66848	0.87	0.87	1.0	0.93
Mystery	76077	0.89	0.89	1.0	0.94
Philosophy	68004	0.92	0.92	1.0	0.96
Religion	80689	0.83	0.83	1.0	0.91
Romance	75814	0.91	0.91	1.0	0.95
Sci-Fi	70430	0.88	0.88	1.0	0.94
Science	90836	0.91	0.91	1.0	0.95
Sports	87604	0.91	0.91	1.0	0.96

Figure 4: Naive Bayes Metrics by classes (weighted averaged)

– Naive Bayes Classifier gives better performance, probably there is no enough number of class type to extract the power of SVC, also data have small sentences so after filtering, the data transforms into a successful frequencies. These frequencies are used in probabilistic features of Naive Bayes Classifier.

2 Task 2

1 The first column of the dataset includes tokens as words, the second column represents "Universal POS-Tags" and last column represents "Penn Treebank POS-Tags". Penn Treebank tags are used for classification in this experiment, but with a modification. There is no tag for punctuation in Penn Treebank tagset as I see, so we are about label it. Label pattern is "¡punctuation symbol¿P.

```
feature_set_1:

accuracy: 0.8571677711893966
f1: 0.8571677711893966
precision: 0.8571677711893966
recall: 0.8571677711893966

feature_set_2:

accuracy: 0.8730380188350192
f1: 0.8730380188350192
precision: 0.8730380188350192
recall: 0.8730380188350192

feature_set_3:

accuracy: 0.8667596791070806
f1: 0.8667596791070806
precision: 0.8667596791070806
recall: 0.8667596791070806
```

Figure 5: Metric results of sets

2 Figure 5 shows the accuracies of sets, most successful one is feature set 2. Its features can be seen in Figure 6.

```
dict = {
    "wrd" + pos: wordlow,  # the token itself
    "prefix" + pos: wordlow[:3],
    "suffix" + pos: wordlow[-3:],
    "length" + pos: len(word),
    "is_alpha" + pos: word.isalpha(),
    "is_alnum" + pos: word.isalnum(),

}
```

Figure 6: Features of set 2

features of set2:

- "prefix" + pos: Include the first few letters of the word as a prefix feature.
- "suffix" + pos: Include the last few letters of the word as a suffix feature.
- "length" + pos: The length of the word.
- "is alpha" + pos: Check if the word consists only of alphabetic characters.
- "is alnum" + pos: Check if the word consists of alphanumeric characters.

The second best one, feature set 3, has features in Figure7.

```
dict = {
    "wrd" + pos: wordlow,  # the token itself
    "is_title" + pos: word.istitle(),  # title case
    "prev_word" + pos: sentence[index - 1].lower() if index > 0 else "",  # previous word
    "next_word" + pos: sentence[index + 1].lower() if index < len(sentence) - 1 else "",  # next word
    "has_hyphen" + pos: '-' in word,  # hyphen presence
    "is_upper_sentence" + pos: all(w.isupper() for w in sentence),  # entire sentence in uppercase
    "has_punctuation" + pos: any(char in string.punctuation for char in word),  # punctuation presence
    "is_numeric" + pos: word.isdigit(),  # numeric content
    "word_pattern" + pos: ''.join('X' if char.isupper() else 'x' for char in word) if word else "",  # word pattern
    "is_stopword" + pos: wordlow in stopwords.words(),  # stopword detection
    "word_shape" + pos: ''.join('X' if char.isupper() else 'x' for char in word) if word else "",  # word shape
}
```

Figure 7: Features of set 3

features of set3:

- "is title" + pos: Check if the word is in title case.
- "prev word" + pos: Include the lowercase version of the previous word as a feature.
- "next word" + pos: Include the lowercase version of the next word as a feature.
- "has hyphen" + pos: Check if the word contains a hyphen.
- "is upper sentence" + pos: Check if the entire sentence is in uppercase.
- "has punctuation" + pos: Check if the word contains any punctuation.
- "is numeric" + pos: Check if the word is a numeric value.
- "word pattern" + pos: Represent the word using a pattern of its capitalization.
- "is stopword" + pos: Check if the word is a common stopword.
- "word shape" + pos: Represent the general shape of the word (e.g., all lowercase, all uppercase, initial capital).

3 If "Logistic Regression" is used, accuracy would decrease probably cause "Conditional Random Fields" is powerful on neighbour likelihood, and the texts are in the order of a contextual sequence. We fit our classifier with sentences and they are close each other in a medium concept, so CRF is better on this task.