Amirkabir University of Technology
(Tehran Polytechnic)

Report of the Project about classification in NLP

Professor:
Dr. M.Akbari (ma.akbari@aut.ac.ir)
Teaching Assistants:
A.Malekzade(malekzadeh@ieee.org) , Y.Ommi(yassi.ommi@gmail.com)

Student:
Mohammadreza Ardestani (ardestani.zr@gmail.com)

5,Nov,2020

0) Introduction

This report will be concise but thorough. If you need more detail, please contact me.

How to run:

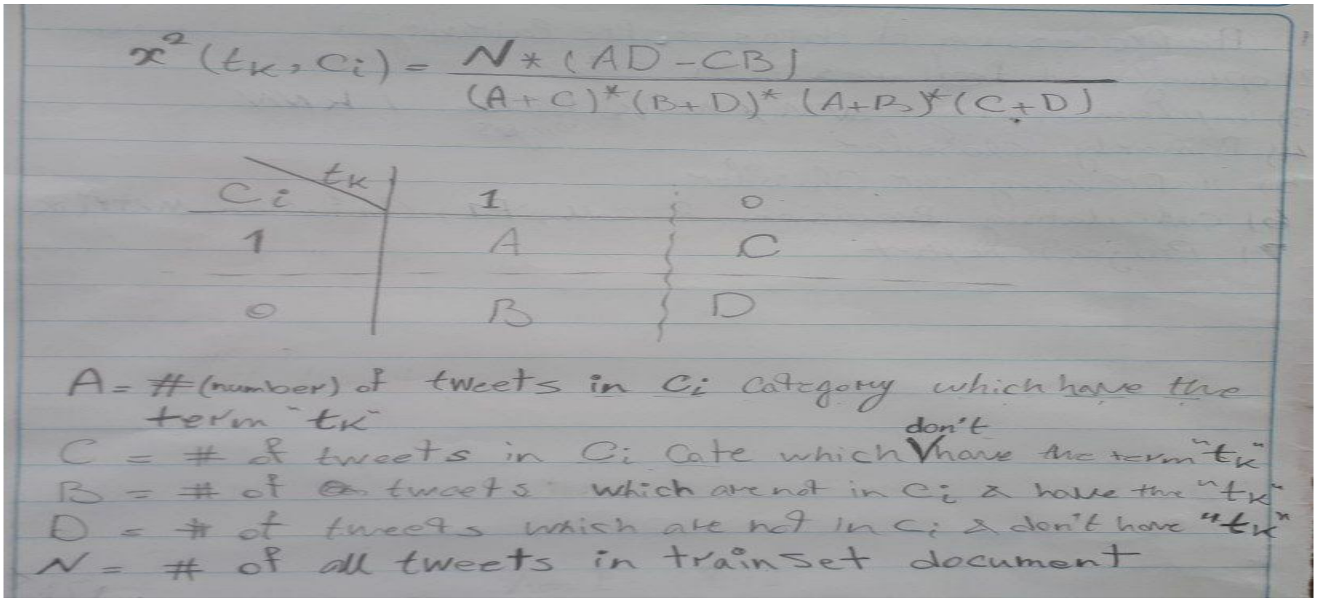
I. At first you should have been installed all required libraries and Jupyter notebook.
II. Create a folder & paste 'airline-train.csv' and 'airline-test.csv' and 'airline-dev.csv' and “FinalSourceCode.ipynb” files.
III. At the address bar write “cmd” then click “Enter”.
IV. At the Cmd console type “jupyter notebook” and then “Enter”.
V. Open Jupyter sourceCode and run each cell in order
VI. At the last panel(cell) you can write your query and run it and give the result.
VII. **Notice some cells take a few second to run completely**

Phase 1) **Preprocessing the data and preparing it**

After I open csv file and extract whole tweets’ text and Save it in the list “All_texts”, I’m going to clean it with the following functions:
UserNameRemover(list)
URLremover(list)
punctuationRemover(list)
LowerCaseAll(list)
LowerCaseAll(list)
StopWordsRemover(list)
stemmer(list) and removing words with less than 3 characters and removing redundant spase

Phase 2) **Opting the best features with CHI-SQUARE method**

Following terminology describes CHI-Square method and we how to calculate it.



$$x^2(t_K, C_i) = \frac{N*(AD-CB)}{(A+C)*(B+D)*(A+B)*(C+D)}$$

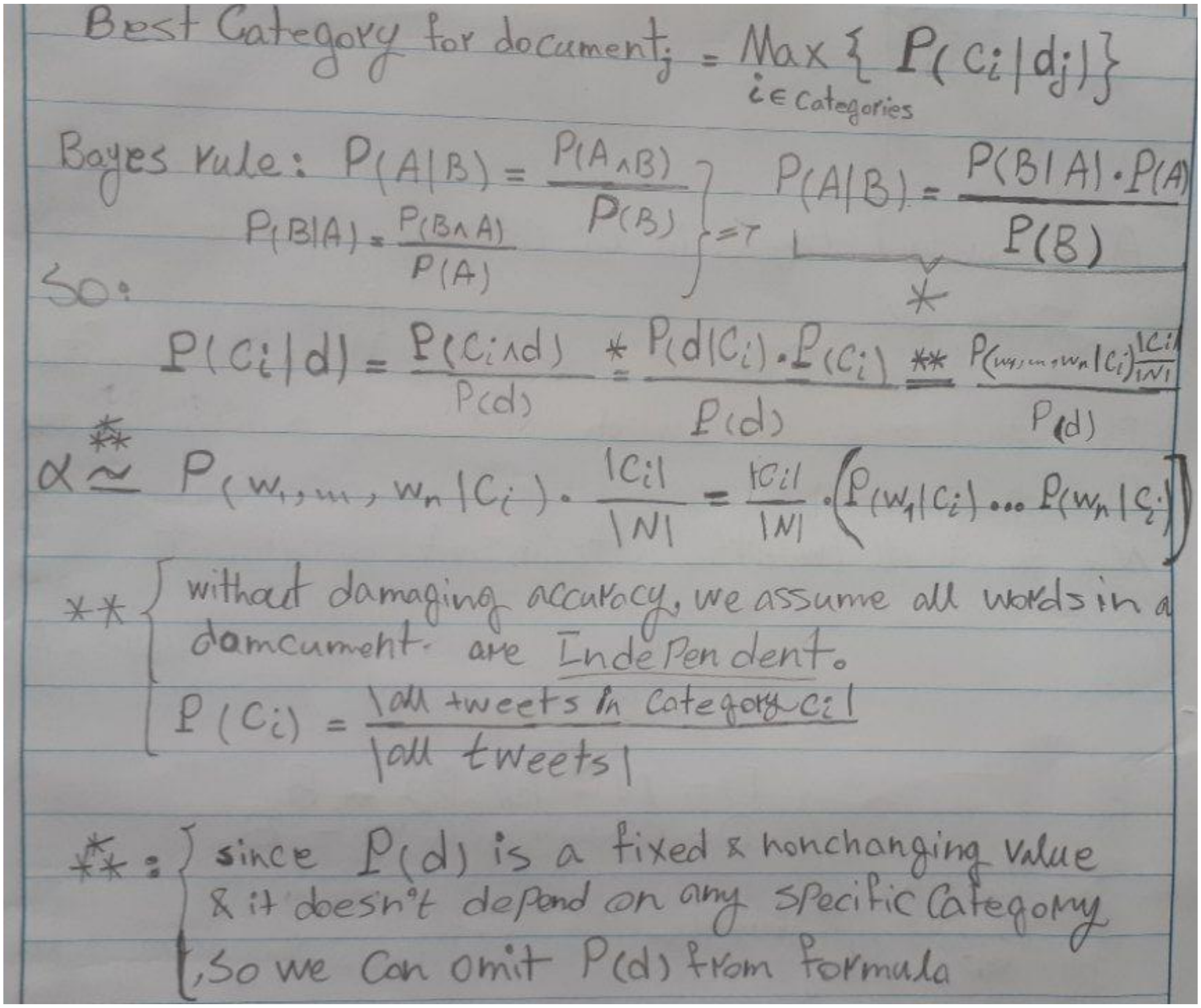
C_i \ t_K	1	0
1	A	C
0	B	D

A = #(number) of tweets in C_i category which have the term "t_K"
C = # of tweets in C_i Cate which don't have the term "t_K"
B = # of tweets which are not in C_i & have the "t_K"
D = # of tweets which are not in C_i & don't have "t_K"
N = # of all tweets in trainSet document

After we calculate Chi-square for all words, we need to pick up some of them that have more value (or more correlation).
In Final_Bag of features we select those words which have more than 0.1 Chi-square value.

Phase 3) **Implementing SVM/ Naïve Bayes/ KNN from scratch**

I’ve selected “Naïve Bayes” method for implementing by my own (from scratch).



Best Category for document$_j$ = $\underset{i \in \text{Categories}}{\text{Max}} \{ P(C_i|d_j) \}$

Bayes rule: $P(A|B) = \frac{P(A \wedge B)}{P(B)}$
$P(B|A) = \frac{P(B \wedge A)}{P(A)}$ $\Big\} \Rightarrow \underbrace{P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}}_{*}$

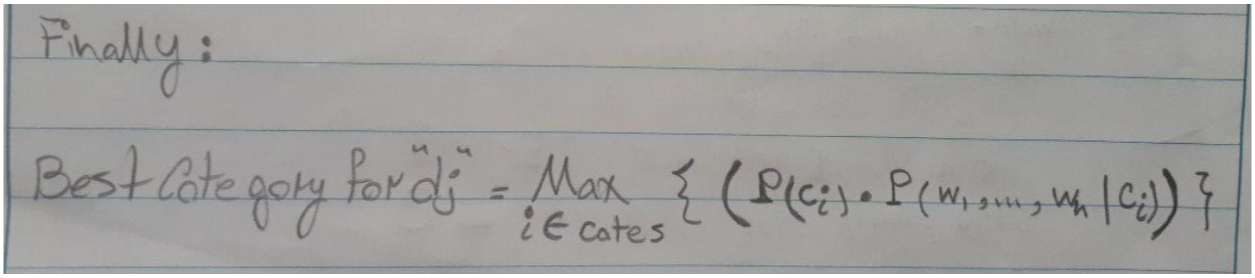
So:

$$P(C_i|d) = \frac{P(C_i \wedge d)}{P(d)} \overset{*}{=} \frac{P(d|C_i) \cdot P(C_i)}{P(d)} \overset{**}{=} \frac{P(w_1, ..., w_n|C_i) \frac{|C_i|}{|N|}}{P(d)}$$

$$\overset{***}{\alpha \simeq} P(w_1, ..., w_n|C_i) \cdot \frac{|C_i|}{|N|} = \frac{|C_i|}{|N|} \cdot \left(P(w_1|C_i) \dots P(w_n|C_i) \right)$$

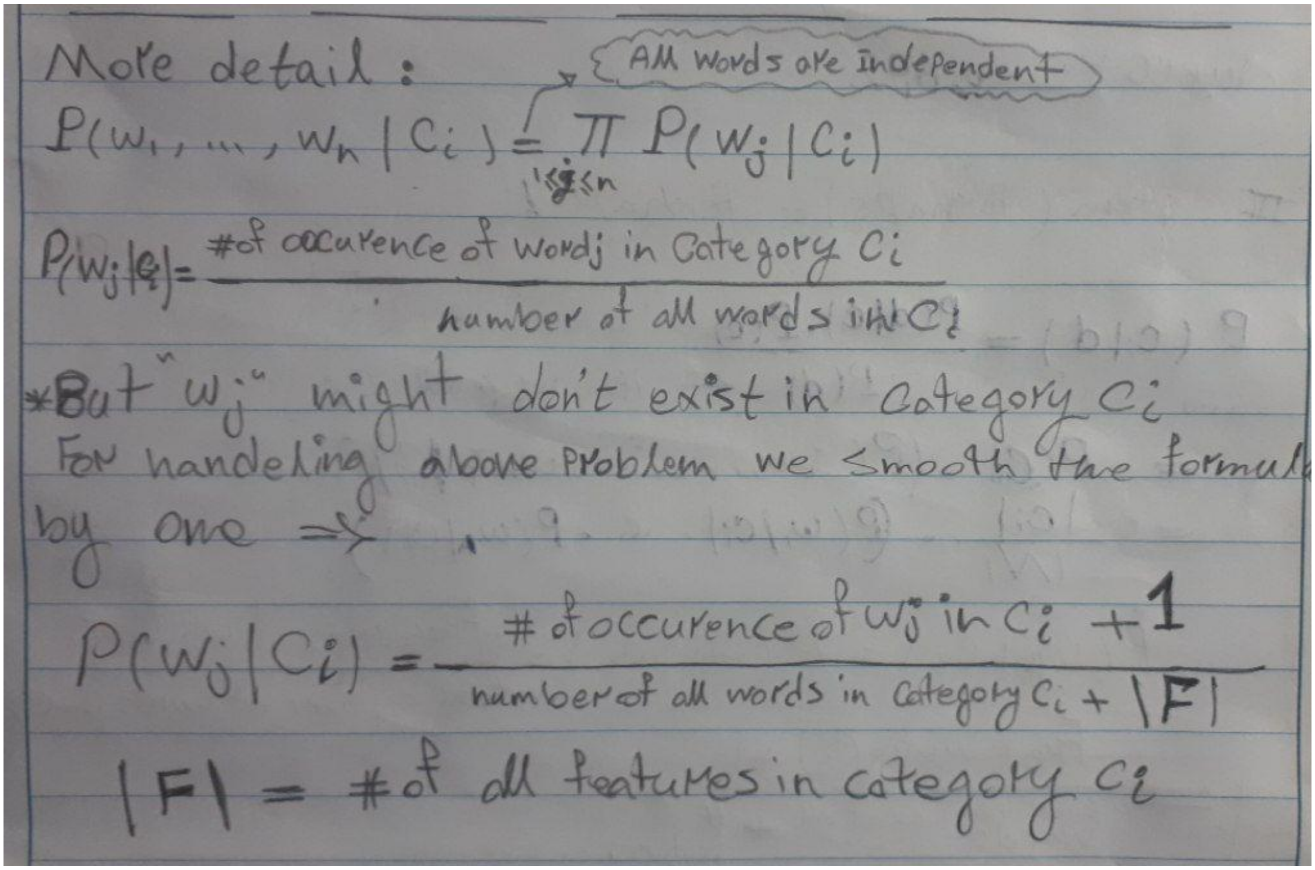
** { without damaging accuracy, we assume all words in a damcument are Independent.
$P(C_i) = \frac{|\text{all tweets in category } C_i|}{|\text{all tweets}|}$

*** : { since $P(d)$ is a fixed & nonchanging value & it doesn't depend on any specific category, so we can omit $P(d)$ from formula



Finally:

Best Category for "dj" $= \underset{i \in \text{cates}}{\text{Max}} \{ (P(C_i) \cdot P(w_1, ..., w_n|C_i)) \}$



More detail :

$$P(w_1, \dots, w_n \mid C_i) \overset{\text{All words are independent}}{=} \prod_{1 \le j \le n} P(w_j \mid C_i)$$

$$P(w_j \mid C_i) = \frac{\text{\#of occurence of word}_j \text{ in Category } C_i}{\text{number of all words in } C_i}$$

*But "w_j" might don't exist in Category C_i
For handeling above Problem we Smooth the formul
by one →

$$P(w_j \mid C_i) = \frac{\text{\# of occurence of } w_j \text{ in } C_i + 1}{\text{number of all words in Category } C_i + |F|}$$

$$|F| = \text{\# of all features in category } C_i$$

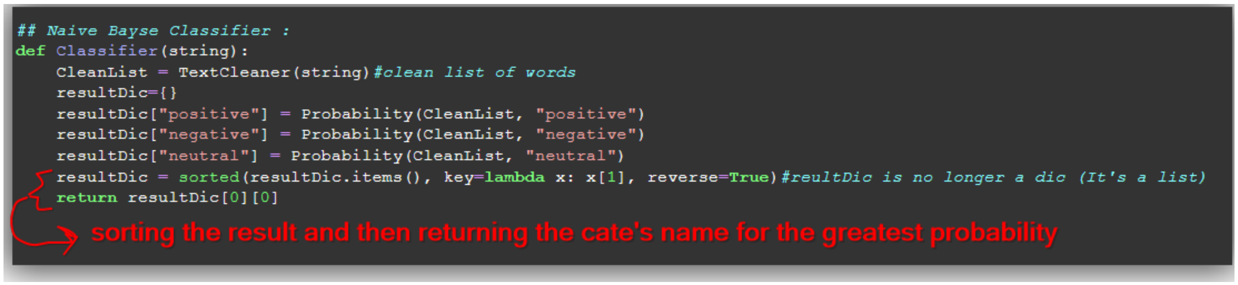
Phase 4) **Primarily classifier**

Based on the knowledge that we have gotten from last phase, now we can design a classifier. In order to do that, at first we find the value of all words in our bag of words given corresponding category ($P_{w_j \in C_i}(w_j \mid c_i)$).

We store the result in Prob_P , Prob_N , Prob dictionaries.

Prob_P is dictionary for selected features in category positive and its probability given class "positive". And same description for Prob_N that is a dictionary for class "negative" and Prob is related to class "neutral".

Classifier function gives a string and then it calculate its probability regarding to each category that we have and then return the most likely category.



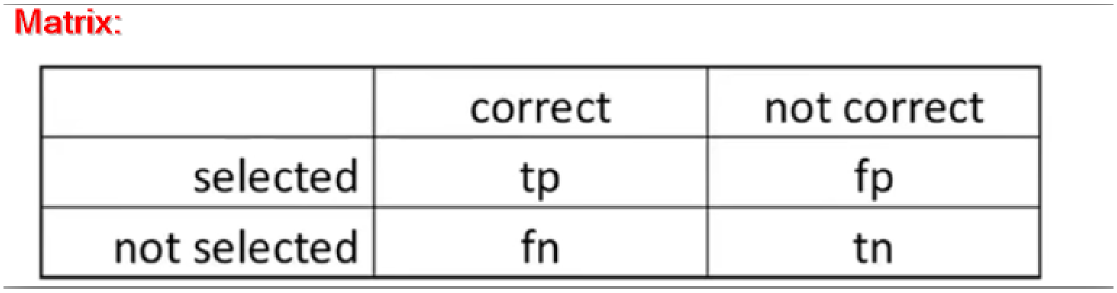
```
## Naive Bayse Classifier :
def Classifier(string):
    CleanList = TextCleaner(string)#clean list of words
    resultDic={}
    resultDic["positive"] = Probability(CleanList, "positive")
    resultDic["negative"] = Probability(CleanList, "negative")
    resultDic["neutral"] = Probability(CleanList, "neutral")
    resultDic = sorted(resultDic.items(), key=lambda x: x[1], reverse=True)#reultDic is no longer a dic (It's a list)
    return resultDic[0][0]
```

sorting the result and then returning the cate's name for the greatest probability

Phase 5) **Improving the classifier**

1. **I have implemented all required functions for preprocessing separately to increase readability and modularity of the code**
2. **I have added more function than we are supposed to add for preprocessing like: Removing URL, Numbers, Usernames, Punctuations, makeLowerCaseFuncition.**
3. **I've written the most efficient code for Chi-square and Naïve Bayes method in order to decrease the running time**
4. **Based on the differences between probability of final result of Classifier method I've increased the confidence interval to 5. I've done this improvement based on this fact that the average word in all tweets is 6 and if we multiply 6 times the greatest Chi-square value of the corresponding category, we can get the apex of the confidence probability. And then we map all probabilities between lowest probability(which its confident value is 0) and the greatest probability (That its confident level is 5)**
5. **I find out that if we assume, after cleaning the data we don't have any word that has occurred in one tweet more than one time and then we calculate Chi-Square, it doesn't change it conspicuously. SO in order to decrease running time we can use this assumption.**

Phase 6) **Calculating Precision, Recall, F1, Confusion Matrix**



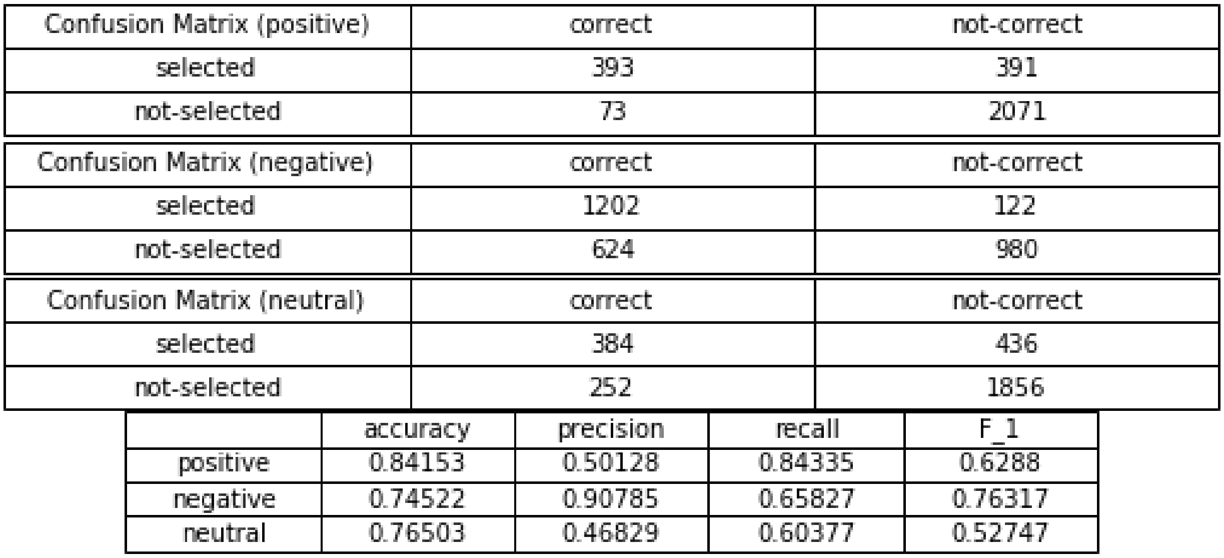
Matrix:

	correct	not correct
selected	tp	fp
not selected	fn	tn

$\mathrm{P} = \mathrm{Precision} \ = \frac{tp}{tp+fp}$, $\mathrm{R} = \mathrm{Recall} \ = \frac{tp}{tp+fn}$, $F_1 = \frac{2*PR}{P+R}$

We have 2 file and each file contains 3 sentiment class.
So at the end, the code will show the result in 6 plot as following:

TEST File:



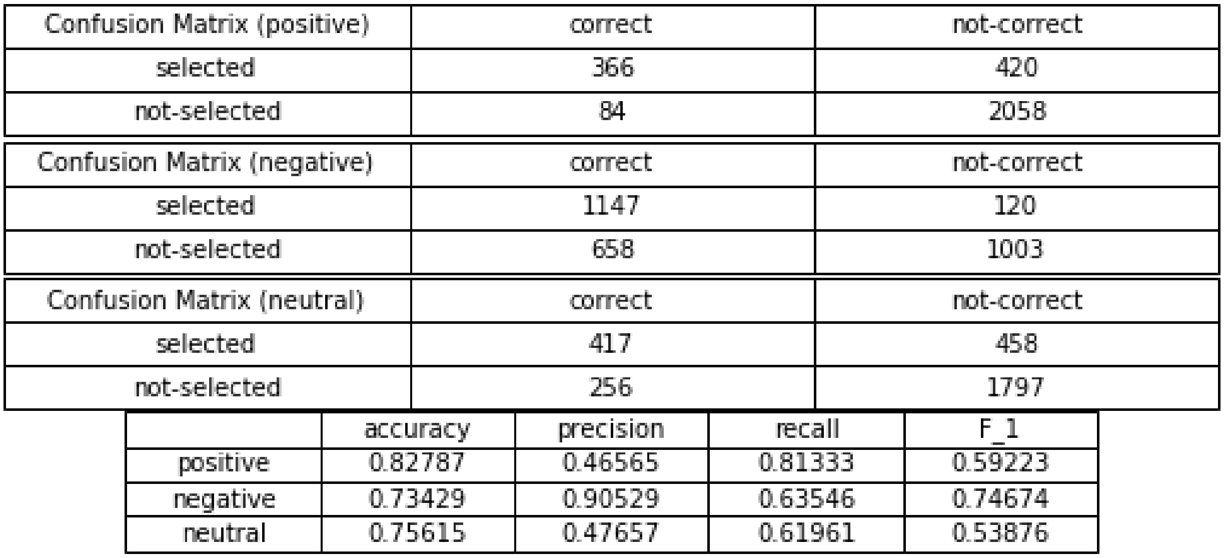
Confusion Matrix (positive)	correct	not-correct
selected	393	391
not-selected	73	2071

Confusion Matrix (negative)	correct	not-correct
selected	1202	122
not-selected	624	980

Confusion Matrix (neutral)	correct	not-correct
selected	384	436
not-selected	252	1856

	accuracy	precision	recall	F_1
positive	0.84153	0.50128	0.84335	0.6288
negative	0.74522	0.90785	0.65827	0.76317
neutral	0.76503	0.46829	0.60377	0.52747

Dev File:



Confusion Matrix (positive)	correct	not-correct
selected	366	420
not-selected	84	2058

Confusion Matrix (negative)	correct	not-correct
selected	1147	120
not-selected	658	1003

Confusion Matrix (neutral)	correct	not-correct
selected	417	458
not-selected	256	1797

	accuracy	precision	recall	F_1
positive	0.82787	0.46565	0.81333	0.59223
negative	0.73429	0.90529	0.63546	0.74674
neutral	0.75615	0.47657	0.61961	0.53876

Thanks for your time.