A Study Using One-R Feature Selection Strategy and Naive Bayes Training on Twitt Data Based on Weight of Binary Attributes to Discover Potential Adverse Drug Reactions

Abstract — **This research intends at predicting patients' Adverse Drug Reactions (ADR) based on the study of their Twitter records. It uses Naive Bayes to classify user Twitter records aiming at both improving accuracy and recall of the prediction based on the weight analysis of all possible binary combination of words of each twit. Then one-R supervised learning strategy is applied to train the model and the evaluation result is compared to which unigram attributes are used. The results suggest that applying binary term analysis on ADR tends to gain an improvement on accuracy compared to applying unigram terms only. However, the recall of applying ADR predictions based on binary terms still needs improvement.**

Keywords – One-R; Naive Bayes; TF-IDF; Adverse Drug Reactions; Supervised Machine Learning;

I. INTRODUCTION

Adverse Drug Reaction (ADR) is a harm which patients tend to get caused by inappropriate use of drugs or the combination use of drugs under prescribed medical care processes. Studying ADR related EHR (electronic health records) is a significant topic during the recent research trends. [9][11] This research intends to use twitter records to predict Adverse Drug Reaction (ADR) of patients using Twitter. It is a researched based on using unigrams to predict ADR and the Twitter data is fetched according to Twitter data term of use. [1]

The research took three steps, firstly a model built according to TF-IDF analysis of the ADR tokens was built, using frequency of the combination of binary terms in each sentences as attributes. Secondly, the built model was trained using the Naive Bayes training approach. Then, labels in developing data were predicted using one-R strategy using the trained model, namely, the ADR label "Y" or "N" indicating the occurrence of ADR and no evidence of ADR correspondingly are predicted according to one variable which is the weight of the sentences. Finally, the results were evaluated and the accuracy and recall of the model was compared with which using unigram attributes.

One-R is a supervised machine learning strategy which selects one possible attribute which is most correlated to the label among all the possible attributes to predict the corresponding output of given unknown attributes. [4] Naive Bayes is a conditional-based classifying strategy which predicts the probability of event A happening under condition B giving probability of event B happening under condition A. [6]

The supervised learning effects can be evaluated using basic classifying evaluation techniques based on the relationship between actual instance and the prediction of instances. [10] This research mainly focuses on accuracy of the model and the recall of the model. Because accuracy measures the overall prediction ability of the learning model given "Y" and "N" attributes. The improvement of this attribute means it is more confident to diagnose whether an ADR happened or not. While recall is another important variable in the context of this research because the improvement of which means that more people already suffering from ADR can be detected out.

Term Frequency – Inversed Document Frequency (TF-IDF) model is a model used in information retrieval which searches document according to the evaluation of weights of user queries. Because each binary term under the context of this research is considered as a "query" to the overall twitter data, this model is then modified and applied in this research to evaluate the weight of each binary term. [7]

The rest of the paper is organized as following: In part II some terms related to this research are defined. In part III I stated the processes used to develop, train and evaluate the model. In part IV I discussed the possible reasons leading to the evaluations. I will end up with the paper in part V by briefly conclude the research and stated some future works.

II. DEFINE OF TERMS

- Development data: Data used to analyze and to develop the model for later training. Also, training model is used to predict dataset from development data to evaluate the accuracy and recall.
- Train data: Data used to train the model developed, intending at finding a specific "threshold" as the one-R attribute to predict the "Y" or "N" label.
- Unigram attributes: Attributes formed by single words.
- Binary term attributes: Every possible combination of two words from a given sentences, without the consideration of the order of the terms. More details are explained in the experiment design phage.
- Weka system: A standard system used for machine learning and data mining purposes usually. In this research, it is directed produce the result of applying unigram attributes in this research, the result is then compared with using binary words as proposed by this research. The system is an open-source system available at: https://www.cs.waikato.ac.nz/ml/weka/

III. EXPERIMENT DESIGN

A. *Develop of Model:*

In order to classify twitter records using one-R supervised learning approach, to successfully identify this feature which tends to have correlation with label thus becomes an important issue. [4] This research decides to study the correlation between weights of binary terms in a given sentence and the actual label indicating whether or not an ADR occurs or not. Binary terms (twitter tokens consist of 2 words) are chosen as attributes to be studied because unigram attributes have already been studied and would be used as a comparison in this research. [1] Because higher dimension of terms, such as ternary terms (tokens consist of 3 words) can be further divided into binary terms. For each sentence, every word of the sentence forms a binary term with any other term in the sentence and thus an array of binary terms can be formed from each sentence. Terms are directly processed to smaller cases due to the reason that the capitalization of tokens is not a consideration by the content of this research. The algorithm is shown as below:

```
#Input: Sentences, output: every possible binary term combination of the sentence
def substractBinTerms(tokens):
    binTerms = []
    size = len(tokens)
    for i in xrange(0, size):
        j = i + 1
        for j in xrange(j, size):
            binTerms.append(tokens[i] + " " + tokens[j])

    return binTerms
```

Fig 2A-1: Algorithm used to extract binary terms as arrays for each given sentences need to be predicted

As an example, twit:

```
Horcoff? Give her a lozenge.
```

would be taken apart as binary terms' array as:

```
["horcoff give", "horcoff her", "horcoff a", "horcoff lozenge",
"give her", "give a", "give lozenge",
"her a", "her lozenge",
"a lozenge"]
```

Then the development data has been processes based on the frequency of binary terms as a feature. Therefore, TF-IDF model is applied, each binary terms are viewed as terms which queries the document, while sentences with labels "Y" are considered as documents. The formula for calculating the "weight" of each binary "query" is calculated as:

$$W_{d,t} = f_{d,t} \times \frac{N}{f_t}$$

Where $W_{d,t}$ is the weight of a certain given term in a certain document, $f_{d,t}$ is the frequency of the term in each document, N is total number of documents and f_t is the total amount of occurrence of the 'Y' labelled sentences in each given data source needs to be predicted. [5]

This formula is used to process the term frequency related to "Y" labelled twits as documents under several considerations:

- In each "Y" twit, more frequently occurred binary terms tend to suggest that the term is more correlated to the label.
- For each document, more general occurrence across documents of each might suggests the term might be a general term in sentences and thus might have less correlation with the label.

The implementation of this uses accumulators and N is measured before applying the algorithm to improve the efficiency and applicability of the algorithm in practical situations, the algorithm is presented as below:

```
def __calculateAccu(self, term):
    if term not in self.accumulator:
        self.accumulator[term] = self.__applyAccuFormula(term)
    else:
        self.accumulator[term] += self.__applyAccuFormula(term)
def __applyAccuFormula(self, term):
    return self.totalDocs * 1.0 / self.wordFreqInFile[term]
```

Fig 2A-2: Algorithm used to calculate the weight of each binary term attribute in develop document

For each binary term in the sentences labelled "Y", their weights are calculated and stored as term-weight pairs in a dictionary data structure and later stored into files for further processing.

Then, for each sentences given in the development data, the weight of each sentence is calculated according to the following algorithm:

```
def weightForSentence(self, sentenceData):
    length = len(sentenceData)

    if length == 0:
        return 0.0

    terms = substractBinTerms(sentenceData)
    weight = 0.0

    for eachTerm in terms:
        if eachTerm in self.termWeights.keys():
            weight += self.termWeights[eachTerm]

    return weight * 1.0 / length
```

Fig 2A-3: Algorithm calculating the weight for each sentences, with weights of terms stored in file. After the calculation, the results are output to a file recording the weight together with the label of each sentences, which will be used for training the model later.

Weight of each sentence is calculated by sum of each binary terms of the sentences divided by the length of the sentence. The length is divided under the consideration that it is not related to the occurrence of ADR and longer sentences contributes more binary terms and thus might contribute more weights. Thus, in order to make the occurrence of ADR holy depends on the binary terms, the length of the sentences are divided.

B. Training of the Prediction Model

In the context of this experiment, as stated from part A and the observation of experimental results of part A, the goal of training the model is actually to find a "threshold" of weights of sentences above which all sentences could be predicted as "Y" while below which all the predictions shall be "N", with the goal of reaching expected recall and accuracy. In the case of improving accuracy, both recall and specificity are important and specificity tends to obtain a heavier portion, due to the reason that "N" labels take majority of the amount of actual labels and prediction labels. The algorithm giving certain expected recall and specificity and returns the suggested threshold is shown as below in Fig 2B -1.

Because the goal of the training is to improve the accuracy and recall, however, from the observation of experiments from part A, the improvement of accuracy might be related to the decrement of recall and vise-versa. Thus in this case, a limitation of lowest accuracy and lowest recall shall be set. If denote event A as ensuring accuracy at a certain level, denote event B as ensuring recall at a certain level, then Naive Bayes learning strategy can be applied, namely, to ensure the recall given expected accuracy can be calculated as:

$$P(B|A) = \frac{P(AB)}{P(A)}$$

Because P(A) is given and P(AB) can be counted using number of true positive labels divided by total number of labels, since a label must be accuracy if it is a true positive one. Then, the probability of given expected recall and ensure accuracy can be calculated as:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

```
"""
    Input: Training goals as precision and specificity,
    Output: possible range of threshold (as hiper plane) and accuracy
"""
def trainDevThreshold(self, expectedRecall, expectedSpecificity):

    self.threshold = 0.0
    self.getExtremeValues()
    #Lower than this value, prediction to be no
    noThreshold = self.highNo
    #Higher than this value, prediction to be yes
    yesThreshold = self.lowYes

    allowedFalseNegative = int(self.positive * (1.0 - expectedRecall))
    #print "Allowed false negative:", allowedFalseNegative
    allowedFalsePositive = int(self.negative * (1.0 - expectedSpecificity))
    #print "Allowed false positive:", allowedFalsePositive

    self.actualFalseNegative = self.actualFalsePositive = 0

    while(self.actualFalseNegative <= allowedFalseNegative):
        self.actualFalseNegative = self.getNumFalseNeg(yesThreshold)
        yesThreshold = self.getNextHigherYesThreshold(yesThreshold)

    while(self.actualFalsePositive <= allowedFalsePositive):
        self.actualFalsePositive = self.getNumFalsePos(noThreshold)
        noThreshold = self.getNextLowerNoThreshold(noThreshold)

    #print "Final suggested threashold predicted as yes:", yesThreshold
    #print "Final suggested threashold predicted as no:", noThreshold
    #Ideal condition: higher than this must be yes, lower than this must be no, very
clear.
    if yesThreshold >= noThreshold:
        #Though noThreashold able to detect ADR is more important
        #yesThreashold dominates accuracy.
        self.threshold = yesThreshold
    else:
        self.threshold = (yesThreshold + noThreshold) * 1.0 / 2.0

    #print "Final suggested threashold:", self.threshold
    return (yesThreshold, noThreshold)
```

Fig 2B-1: Training algorithm of giving expected recall and specificity, return suggested threshold

However, still what value of least accuracy and least recall to be expected in order to achieve the goal under this context is not clear. Thus, 3 models are trained in this case. One is to ensure accuracy to a certain value and improve recall as much as possible, the second one doing vise-versa, while the third one is based on the thought that accuracy and recall are equally important. The algorithms for 3 models are shown below in Fig 2B – 2.

```
def trainAccurateModel(self):

    specification = 0.99
    possibleAccuracies = {}

    while specification >= 0.75:
        recall = 0.99
        while recall >= 0.58:
            accuracy = self.getAccuracy(recall, specification)
            #print "Accuracy and expected precision, specification: ", accuracy, precision, specification
            possibleAccuracies[(recall, specification)] = accuracy
            recall -= 0.01
        specification -= 0.01

    #print sorted(possibleAccuracies.keys())
    #Highest accuracy, precision and most proper specification can be achieved:
    highestAccuracy = highestPrecision = mostProperSpecification = 0.0

    for eachKey in sorted(possibleAccuracies, key=possibleAccuracies.get, reverse=True):
        recall, specification = eachKey
        accuracy = possibleAccuracies[eachKey]

        if recall > highestPrecision and accuracy >= highestAccuracy:
            highestPrecision = recall
            highestAccuracy = accuracy
            mostProperSpecification = specification

    return self.threshold
```

```
def trainPreciseModel(self):
    recall = 0.99
    possibleFalseNegatives = {}

    while recall >= 0.75:
        specification = 0.99
        while specification >= 0.80:
            accuracy = self.getAccuracy(recall, specification)
            #print "Actual accuracy and expected precision, specification: ",
accuracy, precision, specification
            possibleFalseNegatives[(recall, accuracy, specification)] =
self.falseNegative
            specification -= 0.01
        recall -= 0.01

    #print sorted(possibleAccuracies.keys())
    # Highest accuracy, precision and most proper specification can be achieved:
    highestAccuracy = highestPrecision = mostProperSpecification = 0.0
    lowestFalseNegative = 100

    for eachKey in sorted(possibleFalseNegatives, key=possibleFalseNegatives.get):
        recall, accuracy, specification = eachKey
        falseNegative = possibleFalseNegatives[eachKey]

        if accuracy > highestAccuracy and falseNegative <= lowestFalseNegative:
            highestAccuracy = accuracy
            highestPrecision = recall
            lowestFalseNegative = falseNegative
            mostProperSpecification = specification

    self.trainDevThreshold(highestPrecision, mostProperSpecification)

    return self.threshold
#Both accuracy and precision shall be high:
def trainBalancedModel(self):
    recall = 0.99
    actualTruePositives = {}

    while recall >= 0.80:
        specification = 0.99
        while specification >= 0.80:
            accuracy = self.getAccuracy(recall, specification)
            #print "Actual accuracy and expected recall, specification: ",
accuracy, recall, specification
            actualTruePositives[(recall, accuracy, specification)] = self.positive -
self.falseNegative

            specification -= 0.01
        recall -= 0.01

    # print sorted(possibleAccuracies.keys())
    # Highest accuracy, recall and most proper specification can be achieved:
    highestAccuracy = highestActualPrecision = mostProperSpecification = 0.0

    for eachKey in sorted(actualTruePositives, key=actualTruePositives.get):
        recall, accuracy, specification = eachKey
        truePositives = actualTruePositives[eachKey]
        actualPrecision = truePositives * 1.0 / self.positive

        if accuracy + actualPrecision >= highestAccuracy + highestActualPrecision:
            highestAccuracy = accuracy
            highestActualPrecision = actualPrecision
            mostProperSpecification = specification

    self.trainDevThreshold(highestActualPrecision, mostProperSpecification)

    return self.threshold
```

Fig 2B-2: Different models trying different pairs of P(A) and P(B) having different requirements of accuracy and recall

C. *Evaluation on Trained Data*

To evaluate the trained model from part B, twit data in developing data was used. Also, the dictionary file containing weights of each binary attributes from part B was used. The evaluation algorithms developed in this case is similar to which is presented in Fig 4B-1.

Then the two main focuses, namely accuracy and recall, were compared to the prediction according to unigram attributes using Weka.

IV. EXPERIMENTAL RESULTS

A. *Observation of the Sentences' Weight Calculating Results from Developing Data:*

The result of applying TF-IDF on binary terms and weighting algorithm for sentences output a document mainly consists of three hyper-plains, as shown below:

ClassificationResults.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

wow	if	only	more	doctors	thought	like	you	lost	my	entire	
life	to	twelve	cipro	pills		7098.98472222	Y				
take	fluoxetine		and	it	doesn	do	shit	it	just	makes	
me	dizzy	and	tired		7077.26190476	Y					
have	any	of	you	developed		knee	pain	since	starting		
tysabri	what	helps	with	the	problem		6647.28888889	Y			
re	victoza	was	prescribed		this	in	for	obesity	caused		
pancreatitis		and	digestive		problems		ever	since			
6635.33333333	Y										
but	switched		to	trazodone		was	doing	some	crazy	ish	
on	ambien	smh	not	good	lol		6581.53333333	Y			
little	did	know	that	would	go	through	withdrawals		when	coming	
off	of	effexor	xr		6071.71428571	Y					
take	olanzapine		and	have	permanent		dry	mouth	what	can	
do	to	help	it		6065.21098901	Y					
but	have	realised		that	it	the	lamotrigine		which	causes	
the	joint	pain		5918.0	Y						
thank	god	was	born	with	so	much	hair	wouldn	been	bald	
by	now		5661.41538462	Y							
when	you	only	have	vyvanse	to	last	you	the	rest	of	
the	week	and	you	have	two	shows	and	teaching		day	
what	the	fuck		5622.43964646	N						
alain	used	to	do	nothing	but	suck	my	lozenge	and	regret	
it	for	the	rest	of	the	day	now	will	do	the	
opposite		and	do	something			5607.78110223	N			
that	11	be	the	quetiapine		making	you	drowsy	it	common	
side	effect	xx		5531.74358974	Y						
ciprofloxacin	how	do	you	expect	to	sleep	when	your	stomach		
is	cement	mixer		5531.74358974	Y						
got	offered	seat	on	the	tube	this	morning	quetiapine		like	
pregnancy		but	hungrier			5531.74358974	Y				
am	way	too	hyper	to	go	home	and	do	nothing	tonight	
damn	prozac		5531.74358974	Y							
certainly		weird	but	it	quetiapine		instead of		crack	can	
get	that	on	prescription			5504.15384615	Y				

737.259259259	N									
alcohol	and	prozac	bwah	ha	ha	ha	and	the	some	tranks
725.484848485	N									
bananas	contain	natural	chemical		that	can	make	person	happy	the
same	chemical		found	in	prozac	be	happy		717.333333334	
N										
it	the	power	of	paxil	baby		717.333333333	N		
lilly	pleads	ignorance		aware	of	any	confirmed		cases	mt
fda	probes	zyprexa	deaths	http	www	pharmalive	com	fda		
investigates	two	deaths	in	zyprexa	patients			717.333333333		
N										
does	taking	cipro	with	rum	have	side	effects	ill	report	back
later		717.333333333	N							
what	is	humira		717.333333333	N					
any	day	can	be	improved		by	quetiapine		or	heroin
717.333333333	N									
bet	pierre	is	sitting	between	the	benches	crying	and	eating	prozac
like	its	candy		710.928571429	N					
bananas	contain	natural	chemical		that	can	make	person	happy	the
same	chemical		found	in	prozac	let	do	this		
707.370370371	N									
ah	good	thought	you	suggested		dabigatran		patients		need
asa	threw	me	for	loop		703.538461538	N			
drove	to	hospital		for	in	blue	left	early	traffic	only
took	mins	blue	badge	space	next	to	entrance			
694.916666667	N									
debating		on	taking	seroquel		tonight	or	not		
691.714285714	N									
vyvanse	adderall		seriously		man	get	it	together		
691.714285714	N									
well	there	nicotine		in	it	just	not	as	much	as
fucking	cigarettes		he	dumb	pic	twitter	com		689.905882353	
N										
lets	play	the	how	long	til	the	seroquel		kicks	in
game		684.727272727	N							

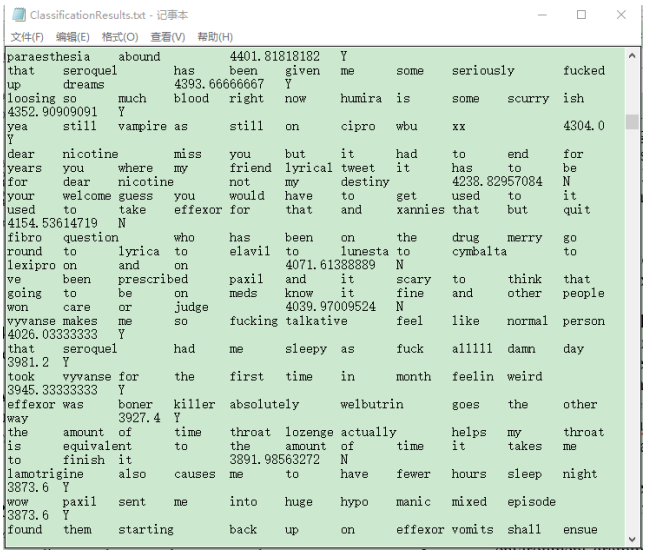

Fig 4A-1: Three different hyper-plains achieved from applying the sentences' weight calculation approach

For those three hyper-plains, one of them (with sentences having high weights) are all labelled as "Y", while another one all labelled as "N", for sentences having low weights, together with the third one having some sentences labelled as "Y" and some other labelled as "N".

Further investigation of the documents suggests that treating some of the labels in the third hyper-plain as anomalies might still help to predict the label, due to statistics of sentences higher than a weight (defined as "threshold") to be labelled as "Y" and "N" (shall be predicted as "Y" in later training) and also statistics of sentences lower than that weight and labelled as "Y" and "N" (shall be predicted as "N" in later training) are evaluated correspondingly, the three threshold in this content is randomly chosen. It is also worth pointing out that the "threshold" here is also regarded as a margin between "Y" and "N" labels, however with some anomalies exist. The result is shown as below:

```
Total twits, true positives, false negatives, false positives and true negatives:
1076 101 13 15 947

Accuracy, recall and specificity:
(0.9739776951672863, 0.8859649122807017, 0.9844074844074844)


Total twits, true positives, false negatives, false positives and true negatives:
1076 107 7 47 915

Accuracy, recall and specificity:
(0.949814126394052, 0.9385964912280702, 0.9511434511434511)


Total twits, true positives, false negatives, false positives and true negatives:
1076 110 4 106 856

Accuracy, recall and specificity:
(0.8977695167286245, 0.9649122807017544, 0.8898128898128899)
```

Fig 4A-2: Results of evaluating the sentence data document in developing model using randomly chosen three "thresholds", first value 3753, second 2824.5 and the third 2228.8

It can be also concluded that when improving this threshold, accuracy and specificity of the prediction is increased, while recall is decreased. Thus, the set of the "threshold" mainly depends on user contends, namely, if users want to develop a system of high accuracy, a high "threshold" might be chosen, while if the user wants to diagnose more people whose ADR already happened, a corresponding lower "threshold" might be set.

B. Observation of Training Effects on Training Data:

Based on the observations in part A and the training algorithm designed in part B of experimental design, the values of accuracy and recall before and after applying the training data set are both evaluated. Before training, the dictionary and from develop phase was used to predict the sentences in training data while during the training process, the dictionary was updated according to the yes-labelled binary term combinations. The evaluating algorithm is shown as below:

```
def predictTwittData(self):
    f = open("train.txt", "r")
    predictResult = open("PredictRet" + str(self.threashold) + ".txt", "w")
    for eachLine in f:
        data = eachLine.split("\t")
        id = data[0]
        actualLabel = data[1]
        sentence = data[2]
        weight = self.weightForSentence(sentence)

        if weight >= self.threashold:
            prediction = 'Y'
        else:
            prediction = 'N'

        self.evaluatePrediction(actualLabel, prediction)
        #print id, sentence, actualLabel, prediction, weight
        predictResult.write(id + "\t" + sentence + "\t" + str(weight) + "\t" + actualLabel +
                            "\t" + prediction + "\n")

    f.close()
    predictResult.close()
```

```
def evaluatePrediction(self, label, prediction):
    self.total += 1
    if label == 'Y':
        if prediction == 'Y': self.truePos += 1
        else: self.falseNeg += 1
    else:
        if prediction == 'Y': self.falsePos += 1
        else: self.trueNeg += 1

def evaluateModel(self):
    print "Threashold: ", self.threashold
    print self.total, self.truePos, self.falseNeg, self.falsePos, self.trueNeg
    accuracy = (self.truePos + self.trueNeg) * 1.0 / self.total
    recall = self.truePos * 1.0 / (self.truePos + self.falseNeg)
    specificity = self.trueNeg * 1.0 / (self.trueNeg + self.falsePos)

    return (accuracy, recall, specificity)
```

Fig 4B-1: Evaluation algorithm used to evaluate the effect of training the data set

The results of the Naive Bayes training algorithm developed corresponding for models ensure accuracy, ensure recall and balance the two are shown in Fig 4B-2 to Fig 4B-3. As can be shown from those figures, this training algorithm has a significant improvement on accuracy and recall based on most of the different contexts, only the accuracy drops when trying to ensure the recall as shown in the second model.

```
Output format:
Threashold
Total, True positive, False Negative, False Positive, True Negative
Accuracy, Recall, Specificity

Threashold:  3753.038
3166 8 365 46 2747
(0.8701831964624132, 0.021447721179624665, 0.983530254206946)


Threashold:  2824.5
3166 44 329 125 2668
(0.8566013897662665, 0.11796246648793565, 0.9552452559971357)


Threashold:  2228.8
3166 85 288 276 2517
(0.8218572331017057, 0.22788203753351208, 0.9011815252416756)
```

Fig 4B-2: Results of three corresponding models before training

```
Output format:
Threashold
Total, True positive, False Negative, False Positive, True Negative
Accuracy, Recall, Specificity

Threashold:  18875.3904762
3166 263 110 13 2780
(0.9611497157296273, 0.7050938337801609, 0.9953455066237021)

Threashold:  6144.56840278
3166 357 16 895 1898
(0.7122552116234997, 0.9571045576407506, 0.6795560329394916)

Threashold:  12845.8445626
3166 311 62 130 2663
(0.9393556538218573, 0.8337801608579088, 0.9534550662370211)
```

Fig 4B-3: Results of three corresponding models after training

C. *Results of Evaluation on Trained Data:*

All of the three models are evaluated using developing data according to the evaluating strategy stated in part C of experimental design. The result is shown below:

```
Threashold: 18875.3904762
1076 2 112 8 954
(0.8884758364312267, 0.017543859649122806, 0.9916839916839917)


Threashold: 6144.56840278
1076 67 47 285 677
(0.6914498141263941, 0.5877192982456141, 0.7037422037422038)


Threashold: 12845.8445626
1076 14 100 44 918
(0.8661710037174721, 0.12280701754385964, 0.9542619542619543)
```

Fig 4C-1: Result of using trained model to predict developing data. The number of second row of each model are organized as total predicted twits, true positive predicted twits, false negative predicted twits, false positive predicted twits and true negative predicted twits. The last line states the accuracy, recall and specificity correspondingly.

Also, the results using Weka system on unigram data are also evaluated, as shown in Fig 4C-2.

It can be concluded that 2 models out of 3 shows a significant improvement on the overall accuracy of predicting the attributes. However, the expected recall is relatively low using weight binary terms in one-R in this case.

```
=== Summary ===

Correctly Classified Instances         884               82.1561 %
Incorrectly Classified Instances       192               17.8439 %
Kappa statistic                          0.2676
Mean absolute error                      0.2067
Root mean squared error                  0.3873
Relative absolute error                103.8938 %
Root relative squared error            125.7529 %
Total Number of Instances             1076

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0.862     0.518     0.934     0.862     0.896     0.279     0.758     0.961     N
               0.482     0.138     0.293     0.482     0.364     0.279     0.757     0.228     Y
Weighted Avg.  0.822     0.477     0.866     0.822     0.840     0.279     0.758     0.884
```

Fig 4C-2: Prediction evaluation on developing data of unigram attributes using Weka.

The results suggest that using one-R strategy, namely by predicting the label of given twits according to the weights of each possible binary combinations of each sentences, implied a significant improvement on accuracy. But in terms of successfully selecting out Twitter users who actually suffers ADR, unigram words tends to outperform than binary attributes.

V. CONCLUSIONS AND FUTURE WORKS

This research studied a novel new method of predicting the ADR reaction based on twitter records. It proposed the idea of using TF-IDF model to calculate weights of binary terms from twitter sentences. Based on the weights being calculated, Naive Bayes supervised machine learning technique is used to train the model which can itself calculate a threshold which identifies the happening of ADR labelled as "Y". This threshold is used as the one attribute which is considered as related to the prediction of model.

Followed by the trained model, the training result was evaluated using the developing data. Also, similar approach was performed to unigram attributes. The result of this research suggests that using binary terms as attributes outperforms using unigrams by about 5% in terms of accuracy. However, when mentioning about recall, the unigram prediction performs much better than the binary term model, by about 70%.

Therefore, which model to apply is a case which really depends on user's context. If users seek for a model of high accuracy, for example, if users want to know in a general group of people, whether or not ADR occurs, then the binary term feature is suggested to be applied. In other cases which requires high recall, however, unigram words as attributes is the suggested solution. For example, it is better to use unigram feature to predict for a group of high risk of ADR, whether it happens or not.

In the future, if given chances to continue this research, there are several points worth considering. Firstly, the binary term weights are fixed after developed and trained. However, during the training processes, evolution algorithms can be applied such that the weights can change according to the predicting results to make which more accurate and have higher recall. Secondly, this research only found out that using

binary terms as attributes tend to have a higher accuracy while applying unigram words tend to have a higher recall, however, the reasons behind this phenomenon is not clear enough, and it could also be a future work. A brief guess could be because binary terms have a more thorough analysis on each token, thus it provides higher accuracy. While unigrams set a lower "threshold" for an attribute to be predicted as "Y", such that its recall becomes correspondingly lower.

REFERENCES

[1] Abeed Sarker and Graciela Gonzalez. (2015) Portable automatic text classification for adverse drug reaction detection via multi-corpus training. Journal of Biomedical Informatics, 53: 196-207.

[2] "17-project2.pdf." .

[3] "2017S2-90049P2-spec.pdf." .

[4] G. James, D. Witten, T. Hastie, and R. Tibshirani, An Introduction to Statistical Learning, vol. 103. New York, NY: Springer New York, 2013.

[5] M. S. Islam, F. E. M. Jubayer, and S. I. Ahmed, "A support vector machine mixed with TF-IDF algorithm to categorize Bengali document," in 2017 International Conference on Electrical, Computer and Communication Engineering (ECCE), 2017, pp. 191–196.

[6] M. Braović, D. Stipaničev, D. Gotovac, and D. Krstinić, "Comparison of cogent confabulation based classifier and Naive Bayes classifier in the detection of lens flares in wildfire smoke detection," in 2017 2nd International Multidisciplinary Conference on Computer and Energy Science (SpliTech), 2017, pp. 1–4.

[7] J. Zobel and A. Moffat, "Inverted files for text search engines," ACM computing surveys (CSUR), vol. 38, no. 2, p. 6, 2006.

[8] A. Rajaraman and J. D. Ullman, "Jure Leskovec," 2010.

[9] S. Viveka and B. Kalaavathi, "Review on clinical data mining with psychiatric adverse drug reaction," in 2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare (Startup Conclave), 2016, pp. 1–3.

[10] M. Makary, M. Oakes, R. Mitkov, and F. Yammout, "Using Supervised Machine Learning to Automatically Build Relevance Judgments for a Test Collection," in 2017 28th International Workshop on Database and Expert Systems Applications (DEXA), 2017, pp. 108–112.

[11] E. H. Shortliffe and J. J. Cimino, Eds., Biomedical Informatics. London: Springer London, 2014.