Analyzing Differences in Indonesian Spontaneous and Dictated Speech

Cil Hardianto Satriawan*, Dessi Puji Lestarti†
School of Electrical Engineering and Informatics
Bandung Institute of Technology
Bandung, Indonesia 40132
*23515053@std.stei.itb.ac.id, †dessi@informatika.org

***Abstract*—The recognition of spontaneous, conversational speech is crucial in achieving practical speech recognition. Due to the difficulties associated with the collection and labelling of spontaneous speech, a larger amount of dictated speech recorded from prepared transcripts is commonly utilized in the training of statistical-based speech recognition systems. Unfortunately, such systems often suffer from poor spontaneous speech recognition performance. In an effort to improve spontaneous recognition performance, we attempt to pinpoint the differences between spontaneous and dictated Indonesian speech. At the phone level, we find that there are differences in the distribution and pronunciation of phones between conversational and written Indonesian. More generally, conversational/spontaneous Indonesian is generally spoken faster and softer than its written/dictated form. Using these differences as a starting point, we show that it is possible to classify Indonesian speech segments as being spontaneous or dictated based on a small number of simple features including segment duration and average log energy levels.**

***Index Terms*—speech recognition,**

I. Introduction

The ability to accurately recognize spontaneous speech is currently the largest challenge to the practical use of speech recognition systems. This is seemingly in contrast to the recognition performance achievable on most state-of-the-art statistical-based systems on dictated speech tasks and limited domain spoken interactions. It has been proposed that spontaneous speech is less ordered both linguistically and acoustically, and differs spectraclly from dictated speech.

There are additional practical difficulties in the collection of spontaneous speech data.

It is important to understand that the distinction between spontaneous and dictated speech is not analogous to the distinction between conversational and formal speech, though some overlap may occur.

Acoustically, differences between spontaneous and dictated speech may occur at a number of different levels. Temporally, a specified number of (overlapping) audio samples are processed in a frame. A segment is comprised of a varying number of frames that physically describe a single phoneme, the smallest linguistic unit. Linguistically, phonemes are organized into words and sentences. However, for the purposes of acoustic analysis, the linguistic units are foregone and we instead consider utterances as collections of segments. A number of high level features are also considered acoustically important. These features are the specific speaker, their gender, race, and age.

Understanding exactly how spontaneous and dictated speech differ is an important step in improving recognition performance. In this paper we discuss how these differences manifest acoustically, specifically with regards to timing/temporal, spectral, and log energy features, at various units of consideration.

In machine learning-based document classification, an indexing procedure needs to be applied to the documents in order to obtain a compact representation of its contents [1]. Document are usually represented as a vector of term weights where a term is typically identified by word. Most of the time, term frequency-inverse document frequency (TF-IDF) weighting function is used to compute term weights. Before indexing, removal of function words (i.e., topic-neutral words such as articles, prepositions, conjunctions, etc.) is almost always performed. Stemming is widely used to perform such procedure [2]. A stemmer extracts the root word from a given word. In summary, standard procedure consists of case folding, tokenization, stop words elimination, stemming, and TF-IDF weighting.

The result of these procedures is used to train a classifier. The resulting classifier only covers words contained in the documents. Out of vocabulary (OOV) occurs when there are words that are not covered by the classifier. This condition can lead to performance decrease as shown in [3].

For illustration, consider an event of total solar eclipse that has occured in Indonesia on March 2016. Prior and some time after this event occured, there were a lot of news articles covering this event. Suppose we had a classifier that we had trained a year before this event occurs. Since total solar eclipse is a rare event, not many news articles would cover this event long before it happens. Because of that, our classifier did not learn this kind of atricle documents. For this illustration, let's say that correct label for articles about total solar eclipse is "natural phenomenon". Our classifier would might give label "tourism" for these atricles because they mentioned a lot of foreigner visited Indonesia on that time. This could happen because our classifier did not learn words such as "total", "solar", and "eclipse", which might be important to classify documents belong to "natural phenomenon".

To cover the OOV words, we need to learn those words. This can be done in two methods. The first method is batch learning.

In batch learning, we create a new classifier using previous documents we have used to train our old classifier and the new documents containing the OOV words. The new classifier will be used for future classifying task, and the old one can be discarded. The problem with batch learning is memory consumption and processing time for training will increase proportionally to the number of documents we use [4]. And also, availability of previous documents can not be guaranteed in the time of future training, making batch learning can not be performed. This is why we need the second method: incremental learning. In incremental learning, we do not use any previous documents, only the new ones [5]. That is, we can learn new documents as many times as we need without running into memory consumption, computational time, and previous documents availability limitations.

To our knowledge, existing incremental learning algorithms can only update its classifier parameters, they can not update its feature set. We can say that a vector of TF-IDF weights from indexing procedure is feature set, in the sense that it is what is used to train the classifier. Since the vector of weights is the words weights, to include the OOV words, we need to weight those words and add it into the vector. That being said, we need an algorithm that able to update its classifier feature set, not only its parameter.

For this reason, our solution is based on meta machine learning algorithm, an ensemble method. The idea is, when OOV occurs, we add a new classifier member that covers feature set including OOV words that are indexed from new documents. The new classifier will be trained by only using new documents, therefore this method still satisfies to incremental learning understanding we have explained previously.

In real world, documents can usually be classified to more than one label. For example, an article from www.voanews.com with the title "Ex-New Orleans Mayor Starts Long Jail Term for Corruption" can be classified as "criminal-law" and "politics" article. Classification of a document into one or more label is known as multilabel classification. The classifier which do such task is called multilabel classifier. Ensemble methods are usually designed for single-label classifier. To employ multilabel classifier with ensemble method, some adaptations need to be applied. In this paper, we use multilabel classifier with ensemble methods so that it can be used for real world document classification task.

Ensemble method with multilabel classifier has not been researched before for Indonesian news articles. This paper explains an ensemble approach for multilabel classification of Indonesian news articles in order to addres OOV. In Section II we discusses the related works. Our methods and adaptation strategy of ensemble and multilabel classification methods are explained in Section III. Our data is analyzed in Section IV. In section V, we present our experimental setup. The results are explained in section VI. The last section concludes and presents the further works.

II. Related Works

Wang, et al. [6] proposed an incremental ensemble method called Accuracy-Weighted Ensembles (AWE). It is a weighted majority ensemble algorithm with the weight is calculated based on error reduction analysis. Each classifier c_i is given a weight reversely proportional to the expected error MSE_i. The weight is then calculated as $w_i = \text{MSE}_i - \text{MSE}_r$ where MSE_r is the mean square error of a random classifier.

Another ensemble method for incremental learning is Streaming Ensemble Algorithm (SEA) [7]. It combines decision tree models using majority-voting. It does not weight each classifier and vote accordingly. It is based on investigation that weighting had little or no effect on ensemble accuracy.

Both ensemble methods are applied on concept drifting environment. We concluded that concept drift has the same meaning as what we called OOV condition in the sense of the underlying concept, which we considered as terms or words, does not reflected in the predictive features [7]. The concept can also change without warning. Zang, et al. [8] shows that AWE performs better than SEA.

For multilabel classification task, there are two main approach: problem transformation and algorithm adaptation [9]. In problem transormation approach, some commonly methods are Binary Relevance (BR), Label Powerset (LP), and Calibrated Label Ranking (CLR). Some examples of algoritm adaptation methods are Adaboost.MH, Adaboost.MR, and MLkNN.

In BR, multilabel dataset is transformed into n single-label dataset where n is the number of label [10]. CLR applies the comparisons among the pair of labels exist in dataset, then builds both pair and single-label models. LP method changes multilabel classification into a multiclass classification [11] by adding some new single-labels for each unique set of labels exists in multilabel dataset [10].

The first two methods of algorithm adaptation are the extension of AdaBoost algorithm that search for accurate classification rules by combining multiple weak hypothesis [12]. MLkNN is the adaptation of kNN algorithm which uses kNN separately for each label then determines the k closest instances to the tested one. Adaboost.MH and Adaboost.MR perform well in terms of one error, coverage, ranking loss and average precision, while MLkNN performs better in terms of hamming loss [13].

III. Ensemble Approach

AWE proposed by Wang, et al. [6] is used as ensemble method for our approach. Its weighting method is based on expected error of classifier. We compute the mean square error of classifier c_i by using formula

$$\text{MSE}_i = \frac{1}{|S|} \sum_{(x,y) \in S} (1 - f_i^y(x))^2$$

where S is the latest chunk of training data, (x, y) is a record in S with y is the true label, and $f_i^y(x)$ is the probability given by c_i that x is an instance of label y. And then, the

mean square error of each label according to its distribution is

$$\text{MSE}_r = \sum_c p(c)(1 - p(c))^2$$

Finally, the weight w_i of a classifier c_i is computed with formula

$$w_i = \text{MSE}_r - \text{MSE}_i$$

MSE_r does not contain useful information about the data. It is only used as a threshold to discard classifiers from ensemble. We eliminate classifiers that have negative weight, which means their error is equal or larger than MSE_r.

Adaboost.MH and Adaboost.MR is based on boosting ensemble method. Even though they provide what we need: ensemble-multilabel classifier, we can not employ one of these methods because they do not handle incremental learning. Since BR is the most widely used method [14], has low computational complexity [15], simple, and easy to use, we use this as our multilabel classifier. BR transforms multilabel classification problem by using l single-label classifier, where l is the number of label. All single-label classifier is trained using the same training data but with different label. When classifying documents, each single-label classifier classify the documents to label it handles. The classification result is multilabel consisting of each single label.

A. Training Step

Suppose we have chunks of training data $S = S_1, S_2, ..., S_n$. We perform indexing procedure that consists of case folding, tokenization, stop words elimination, stemming, and TF-IDF weighting. Every chunks will be processed the same way by all of indexing procedure steps, except TF-IDF weighting. It is because each chunk will have its own feature set. That being said, we only need one case folder, tokenizer, stop words eliminator, and stemmer, but different TF-IDF transformer for each chunk.

In training step, each chunk of documents will be transformed into TF-IDF weights resulting in a TF-IDF transformer that consists of vocabulary or word set and their document frequency, depends on the contents of documents in the chunk. After that, a classifier is trained using TF-IDF weights for each chunk. We, then, weight each classifier in the ensemble using MSE_r and MSE_h. The resulting product is an ensemble consists of classifiers that accompanied by its weight and TF-IDF transformer.

There are two condition that makes us remove a classifier in the ensemble. First, when there exists an ensemble member that has weight less or equal zero. If a classifier has error larger than MSE_r, it means it does not perform well in classifying current training data. We discard this classifier from ensemble because it is not good enough to be used in future classifying task, assuming the current training data represents what the next training data will be. Second condition occurs if we have a limit to the number of ensemble members. When we train a new classifier from new chunk of training data while the number of classifiers in the ensemble has already reached its limit, we remove a classifier from the ensemble and add the new classifier to it. Classifier weight is used to determine which classifier we remove. If there is k limit of classifiers, we choose the top k weighted classifiers, including the new one. Because of the weight is reversly proportional to the classifier expected error, we can expect the top k classifiers as the best classifiers.

In multilabel classification setup, the topology is not an ensemble of BR classifiers. But instead, it is a BR classifier of ensemble where the ensemble members are single-label classifiers. Fig. 1 and Fig. 2 illustrates ensemble of BR and BR of ensemble respectively. If we have 10 labels and the ensemble has 4 members, there will be one BR classifier consists of 10 ensembles, each ensemble classifies one label. And then, each ensemble will consist of 4 single-label classifier. That means, the total number of classifiers we create is $10 \times 4 = 40$ classifiers. We understand that the tradeoff of using this approach is there will be a lot classifiers.

Fig. 1. Ensemble of BR.

Fig. 2. BR of ensemble.

Note that what we remove from the ensemble is ensemble member. If the member is BR, we will remove the BR classifier with all single-label classifiers it contains. This is not right when the other single-label classifiers performs well. We need a topology that allows us to remove only one single-label classifier without removing the others. That is why we choose BR of ensemble topology where the member is single-label classifier. It is allows us to remove single-label classifier from ensemble without removing the other. With L ensembles, we can contain them in a single BR to have a multilabel classifier of L labels.

Each single-label classifier will be accompanied by one TF-IDF transformer. But, in different ensemble, there may be more than one classifiers uses the same TF-IDF transformer. For effiency, we should not use one-to-one relationship of TF-IDF transformer and classifier. Instead, we use one-to-many relationship, which means one TF-IDF transformer can be used by many classifiers. For the sake of simplicity, we do not consider this efficency aspect in the next explanation. We treat accompanionship of TF-IDF transformer and classifier as one-to-one relationship.

Note that each TF-IDF transformer covers its own word set. When OOV occurs, we can train a new TF-IDF transformer and a classifer using new training data that contains OOV words and add it to the ensemble. That being said, we can always cover new words using this strategy, hence OOV condition can be handled. Training procedure is summarized in Algorithm 1.

B. Classification Step

The first step is top-down procedure. The document we want to classify is "sent" to the system, BR of ensembles. The

Algorithm 1 Training procedure

procedure TRAIN
input:
S: new chunk
k: maximum number of classifiers
L: labels
B: Binary Relevance (BR) classifier of $|L|$ ensembles
algorithm:
fit TF-IDF transformer f^{new} on S
transform S into D^{new} by f^{new}
for $l \in L$ **do**
train classifier c_l^{new} on D^{new} using label l
compute weight w_l^{new} of c_l^{new}
for $c_l^i \in B_l$ (label l ensemble in BR) **do**
transform S into D_l^i by f_l^i
apply c_l^i on D_l^i using label l to derive MSE_i
compute weight w_l^i of c_l^i
end for
$B_l \leftarrow k$ of the top weighted classifiers in $B_l \cup \{c_l^{new}\}$
remove negative weighted classifiers in B_l
end for
end procedure

Algorithm 2 Classification procedure

procedure CLASSIFY
input:
s: a document
L: labels
B: Binary Relevance (BR) classifier of $|L|$ ensembles
output:
Y: multilabel result
algorithm:
for $l \in L$ **do**
$W_0 \leftarrow 0, W_1 \leftarrow 0$
for $c_l^i \in B_l$ (label l ensemble in BR) **do**
transform s into d_l^i by f_l^i
predict y by c_l^i on d_l^i
$W_y \leftarrow W_y + w_l^i$
end for
$Y_l \leftarrow \text{argmax}_y\, W_y$
end for
return Y
end procedure

BR forwards this document to all ensembles. Each ensembles forwards the document to every single-label classifier. TF-IDF transformer in each classifier performs indexing to this document and then the classifier performs classification.

The process is continued with bottom-up procedure. Each ensemble in the system votes the label class. This is done by choosing a class that has maximum support, total weight of classifiers that predict class y. The voting rule is formalized by

$$\hat{y} = \underset{y}{\text{argmax}} \sum_{c \in C \rightarrow y} w_c$$

where $\hat{y}$ is the ensemble result of label class, $C \rightarrow y$ is classifiers that predict y, and w_c is the weight of the classifier. And then, the result of each ensemble is combined to make one multilabel class of the document.

This procedure is summarized in Algorithm 2

IV. Data Analysis

Our data were taken from multilabel corpus constructed by Rahmawati [3] and extended with two new datasets. We consider one dataset as one chunk, that means there are 3 chunks. We also have a test data consists of 100 documents that are different from the previous ones, also taken from [3]. All chunks and test data are multilabel data with 10 labels. Data and label composition are shown in Table I and Table II. The number of words are counted after stop words elimination and stemming performed.

TABLE I
DATA COMPOSITION

	Chunk A	Chunk B	Chunk C	**Total**	Testing
Documents	690	1057	1479	3226	100
Cardinality	1.448	1.018	1.014	1.108	1.3
Words	5548	7788	8419	12286	1253

TABLE II
LABEL COMPOSITION

Label	Chunk A	Chunk B	Chunk C	**Total**	Testing
1	100	92	150	342	7
2	100	120	150	370	25
3	100	115	150	365	17
4	100	108	150	358	6
5	100	91	150	341	19
6	100	109	150	359	8
7	100	107	150	357	7
8	100	109	150	359	26
9	100	114	150	364	5
10	100	111	150	361	10
Total	1000	1076	1500	3576	130

V. Experiment

Our experiment compares the performance of our method against other methods. There are four methods in this experiment: batch, incremental, ensemble, and ensemble-incremental. Base classifier in every methods is Naive Bayes classifier. We choose Naive Bayes because it is fast and simple, and also produce the second best performance beside SVM in Rahmawati experiment [3]. We use the same metric used by Rahmawati to evaluate each method performance, which is sample average f-measure. The methods are evaluated using 10-fold cross validation and our test data.

In batch experiment, the first chunk is learned and the result is a BR classifier. This is the first iteration. After that, the data

from first chunk is appended with the second chunk to make a new classifier, discarding the old classifier. After the second iteration, we do the same process in third iteration with the third chunk. In every iteration we perform, we evaluate its performance.

In other methods, the chunks are trained incrementally. In incremental method, we train the BR of Naive Bayes classifier incrementally chunk by chunk. Note that Naive Bayes classifier can be trained incrementally by only updating the distribution $P(y)$ and $P(x_i|y)$. The next two methods use our method. In the first one, ensemble method, we only perform our method as is. In ensemble-incremental method, we furtherly employ incremental learning ability of Naive Bayes to existing classifiers in the ensemble.

We also perform chunk sorting in order to fit AWE weighting scheme analysis. Classifier weighting in AWE is estimated from its expected prediction error on the test examples, assuming class distribution of the most recent training data is closest to class distribution of test data [6]. To meet this assumption, we sort our chunks by evaluating three classifiers that are trained independently using each chunk.

We masure processing time and memory usage of each method to see whether incremental and ensemble methods use less resource. Implementation of ensemble methods that is used for this experiment is the efficient one, which has one-to-many relationship between TF-IDF transformer and classifier. OOV condition is also analyzed by calculating the number of words that are covered and not covered by the classifiers, its effect on class prediction, and a sample of document that has OOV words. Experiment results are explained in next chapter.

Natural language processing (NLP) tools we use for tokenization and stemming tasks is INA-NLP by Purwarianti [2]. We use stop words list taken from Tala [16]. For machine learning task, we use Python programming language and Scikit-learn [17].

VI. Result

The result of 10-fold cross validation is shown in Table III. Note that the result of first iteration in all methods are identical because they are actually the same classifier, trained with the same data. No re-learn procedure has happened in incremental, ensemble, and ensemble-incremental classifier. Incremental-trained classifier will be similar to batch-trained classifier. Also, ensemble and ensemble-incremental will consist of one member which is a classifier that similar to batch-trained one.

TABLE III
Cross validation result

	Iteration 1	Iteration 2	Iteration 3
Batch	0.785	0.768	0.787
Incremental	0.785	0.735	0.778
Ensemble	0.785	0.775	0.768
Ensemble-incremental	0.785	0.775	0.776

Ensemble and ensemble-incremental method has the same F1 score in first and second iteration. This happens because there are only two members in the ensemble, so the voting will be to choose one from two classifiers. This means, a classifier that has larger weight is the same one between ensemble and ensemble-incremental method.

The result against testing data is shown in Table IV. Here, like in cross validation, all methods has the same performance is first iteration. Ensemble-incremental is always outperform ensemble-only method in validation and testing. This concludes that existing classifiers in ensemble are also need to be re-trained with new training data in order to make them adapt with the most up-to-date condition.

TABLE IV
Testing result

	Iteration 1	Iteration 2	Iteration 3
Batch	0.778	0.798	0.823
Incremental	0.778	0.796	0.8
Ensemble	0.778	0.71	0.756
Ensemble-incremental	0.778	0.71	0.776

To see the effect of meeting AWE weighting scheme assumption, we use f-measure to estimate closeness of chunk class distribution to our test data. A chunk which has larger F1 score is assumed to indicate its class distribution is closer to test data. Classifier F1 score when trained with Chunk A is 0.778, 0.71 with Chunk B, and 0.728 with Chunk C. Therefore, we use Chunk B in first iteration, Chunk C in the second, and Chunk A in the last iteration.

Classifiers performance using test data after the chunks has been sorted is shown in Table V. We can see that both ensemble methods perform better than before sorted. Ensemble-incremental method even outperform batch method in the last iteration. This shows that the assumption of the most recent chunk as the closest to class distribution of test data affects performance.

TABLE V
Sorted chunk testing result

	Iteration 1	Iteration 2	Iteration 3
Batch	0.71	0.76	0.823
Incremental	0.71	0.745	0.811
Ensemble	0.71	0.728	0.791
Ensemble-incremental	0.71	0.728	0.834

Each methods computational time and memory usage is shown in Table VI. We can see that even though both ensembles take more time, their processing time is proportional to how much documents is used in the training. This also applied in incremental method, since they only use current chunk and do not use any previous chunk. Note that the last chunk used in third iteration contains the least number of documents. Incremental, ensemble, and ensemble-incremental methods took less time than previous iteration. Ensemble methods trained much longer because there are more classifiers and more process, such as weighting and TF-IDF transforming. In

contrast, batch method processing time is proportional to total number of documents its used, chunk size did not affect it. In memory usage we can also see batch method uses more memory in the last iteration than other methods. Incremental and ensemble methods memory usage of training data depends on chunk size. Ensemble methods require more memory than incremental method because they contain more classifiers.

TABLE VI
TRAINING TIME AND MEMORY USAGE

	Iteration 1	Iteration 2	Iteration 3
Training Time			
Batch	0.42	0.881	1.09
Incremental	0.42	0.47	0.291
Ensemble	0.702	1.523	1.241
Ensemble-incremental	0.702	1.59	1.356
Memory Usage			
Batch	20762	41953	52132
Incremental	20590	23224	14331
Ensemble	20604	29432	24568
Ensemble-incremental	20604	29683	25090

To investigate OOV occurence, we analyzed words contained in each chunk. The number of words that appear in chunk A and also appear in chunk B is 3882, and that makes the total of unique words of their union is 9454. It means, 1666 words are OOV words. That union and the chunk C has 5587 same words. It has 3867 OOV words from that union compared to words consisting in the chunk C.

OOV occurrence can be seen in prediction of a sample document in our test data with title "Partai Buruh Dinilai Sangat Mungkin Terbentuk" [Labor Party is Considered Very Likely to Formed]. The word "kecimpung" [involved in] appears in that document, but it is not covered by incremental method classifier. This OOV condition resulted in incorrect classification result. Ensemble method correctly classify the document to label "social and cultural", but incremental method does not. When we create a dummy document consisting only the word "kecimpung", we got the same result, ensemble method classifies it correctly but incremental method classifier does not. This is an example of OOV can affect prediction performance.

VII. CONCLUSION AND FURTHER WORKS

An ensemble approach to incrementally learn out of vocabulary words has been implemented in this experiment for multilabel classification of Indonesian news article. Our approach is a combination of Accuracy-Weighted Ensemble (AWE) for ensemble classifier and Binary Relevance (BR) for multilabel classifier. We provided explanation of the usage of both methods in order to employ them as one classifier system.

The results show that our ensemble methods could perform well when the last training data represent class distribution of test data the most. Our approach is incremental method, in the sense that it does not use any previous data, making its processing time and memory consumption does not proportionally increased by total number of all documents. OOV is also shown to have been handled using this approach.

Our method will be improved by making its processing time shorter. Parallel implementation can be employed to address this problem. There are some BR and AWE procedures that can be performed in parallel, since classifiers contained in them process data independently. Feature selection can also be performed. There are evidence that shows feature selection increases classifier performance for the same task [3]. Although this method still leaves some opportunity for improvements, we believe that this paper has covered the big picture of what our idea is.

REFERENCES

[1] F. Sebastiani, "Machine learning in automated text categorization," ACM computing surveys (CSUR) 34.1 (2002): 1-47.

[2] A. Purwarianti, "A non deterministic Indonesian stemmer," Electrical Engineering and Informatics (ICEEI), 2011 International Conference on. IEEE, 2011.

[3] D. Rahmawati and M.L. Khodra, "Automatic multilabel classification for Indonesian news articles," Advanced Informatics: Concepts, Theory and Applications (ICAICTA), 2015 2nd International Conference on. IEEE, 2015.

[4] R. Calandra, et al., "Learning deep belief networks from non-stationary streams," Artificial Neural Networks and Machine LearningICANN 2012, Springer Berlin Heidelberg, 2012, 379-386.

[5] M.D. Muhlbaier and R. Polikar, "An ensemble approach for incremental learning in nonstationary environments," Multiple classifier systems, Springer Berlin Heidelberg, 2007, 490-500.

[6] H. Wang, F. Wei, P. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," In Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 226-235, ACM, 2003.

[7] W. Street and Y.Kim, "A streaming ensemble algorithm (SEA) for large-scale classification," Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2001.

[8] W. Zang, et al., "Comparative study between incremental and ensemble learning on data streams: case study." Journal Of Big Data 1.1 (2014): 1-16.

[9] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Mining multi-label data," in Data mining and knowledge discovery handbook, Springer, 2010, pp. 667685.

[10] P. Prajapati, A. Thakkar, and A. Ganatra, "A Survey and Current Research Challenges in Multi-Label Classification Methods," Int. J. Soft Comput., vol. 2, 2012.

[11] K. Dembczynski, W. Waegeman, W. Cheng, and E. Hllermeier, "On label dependence in multi-label classification," in Workshop proceedings of learning from multi-label data, 2010, pp. 512.

[12] S.C. Dharmadhikari, M. Ingle, and P. Kulkarni, "A comparative analysis of supervised multi-label text classification methods," 2011.

[13] M.L. Zhang and Z.H. Zhou, "ML-KNN: A lazy learning approach to multi-label learning," Pattern Recognit., vol. 40, no. 7, pp. 20382048, 2007.

[14] A. Santos, A. Canuto, and A. Neto, "A comparative analysis of classification methods to multi-label tasks in different application domains," Int. J. Comput. Inform. Syst. Indust. Manag. Appl, vol. 3, pp. 218227, 2011.

[15] E.A. Cherman, M.C. Monard, and J. Metz, "Multi-label problem transformation methods: a case study," CLEI Electron. J., vol. 14, no. 1, p. 4, 2011.

[16] F.Z. Tala, "A study of stemming effects on information retrieval in Bahasa Indonesia," Institute for Logic, Language and Computation Universeit Van Amsterdam (2003).

[17] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," The Journal of Machine Learning Research 12 (2011): 2825-2830.