Automatic Knowledge Extraction for News Articles

Mengtian Jin[1], Yilong Li[2], and Linying Yang[1]

[1]Institute for Computational and Mathematical Engineering, Stanford University
[2]Department of Computer Science, Stanford University
{mtjin, yilong, lilyy}@stanford.edu

***Abstract*—In this article we used machine learning algorithms to extract knowledge like named entity and topic labels from Internet news articles. We found that in named entity extraction, neural network based spaCy performed better on person and places while SVM outperforms on regions. For topic labelling, the convolutional neural network with pretrained model and chained classifier outperform in the large multi-label news dataset.** [1]

I. Motivation and Tasks

A structured knowledge database of news articles is very important in a variety of areas like news agencies, media websites and some knowledge-based decision making systems. It can be used to recommend customized news articles for users, analyze development of a business by synthesizing similar articles. In this project, we aim at building an interface that will use NLP to automatically relate articles and knowledge by topic. We aim to have a system that integrate news into a predefined knowledge base, based on a database of tagged news articles, and to identify key entities, topics and even high level relationships between articles automatically is important.

Before integrating news articles into a predefined knowledge base, there are some basic tasks for the news dataset, and these are what we will do in this project:

A. General Topic Labelling

For this dataset and this task, there are basically two types of topics we need to get for each new article: the "topic" of the news, which is the broader category of the news like sports, technologies, and world news; and the "theme" of the news, which is the specific topic an article is about, like hurricane, gun control, Nobel prize, etc. One thing we should note is that there can be multiple tags for a single article, for example, the article *US judge orders temporary pause on deportations of reunited migrant families* can be related to tags like "Immigration" "Law & Justice" and "World News".

B. Named Entity Extraction and Linking

For news categorization we also need to **extract the key named entities** in the article and divide them into different categories like *Places*, *Regions*, *Persons* and *Groups*. There are usually a lot of named entities included in this article but what we need is only the key entities for article tagging. Also for one word or phrase, it can refer to different named entities based on its contexts. So it will be also important to **link the named entity** we extracted to the semantic unit itself.

II. Related Works

A. Named Entity Extraction

Handcraft rules and machine learning algorithms can classify techniques of entity extraction. Typical rule-based systems like SystemT [1] and GATE [2] make use of rules throughout the entire flow. Due to many special cases, handcraft rules are not sufficient and time-consuming. Supervised learning approaches such as HMM, Decision Trees, SVM and ME are current dominant techniques. However, the crucial dependence on the availability of large, representative and high-quality labeled training datasets makes building large-scale NER systems hard. Thus, researches on applications of semi-supervised learning and unsupervised learning without the prerequisite of an annotated corpus are more popular these days and we will try different techniques in this part.

B. Topic Labelling

For the topic labelling, this is basically a multi-labeled classification problem, where multiple labels may be assigned to a single instance. For multi-label classification problems, classification labels may be correlated and we have to take the correlation between labels into consideration.

Per [3], there are several state-of-the-art multi-label classification algorithms, which can be divided into two major classes — transformation methods and adaptation methods. Transformation methods transfer the multi-labeled classification problems to simpler problems, like binary classification, label ranking or multi-classification [3]. Chained classifiers [4] transfer the problem into a continuous chain of binary classifiers, where for each classifier its input is the original input data plus all the previous calculated labels $(x^{(i)}, l_1, \ldots, l_j)$, and it outputs whether the input belongs to the the next-label $l_{j+1} \in \{0, 1\}$. Since the order of labels will definitely affect the classification performance, we use ensemble methods to vote for the best possible label sets. Other adaptation-based methods include Rank-SVM [5], decision-tree based ML-DT [6] and modified k-nearest neighbor algorithm ML-KNN [7].

Improvements in deep learning also enabled us to use deep neural networks for text classification. Kim [8] first uses convolutional neural network to categorize sentences by sentiments, which will be described in detail later. Lai et al. [9] propose to use recurrent neural networks (RNN) to classify texts. Zhang et al. [10] use character-level CNN to solve text classification problem without involving words.

III. Dataset and Features

A. Choice of dataset

In this project, we use the database provided by a news platform led by journalist Jens Erik Gould, which is a CSV file containing the title, contents, and human-labelled information like topics, regions, events, persons, groups, places and themes for all news articles collected from the news agency during a long time period. It contains about 1200 news articles published in 2017 and most of the news are international / political news.

This dataset is perfect for named entity recognition tasks since it contains a lot of hand-labelled named entities of different categories (persons, groups, locations, etc.). For labelling, the problem is that both the training and testing set are too small and too unbalanced. Large topics like War and Conflict, Trump,

[1]Code available at https://github.com/gnoliyil/229proj

and Law & Justice can contain over 200 articles, while topics like Religion, France, and Sports may have only a few (less than 20) articles. This imbalance severely affected the accuracy for our baseline classifiers, since most of them have a very strong tendency to return 0 after training.

In order to solve this problem we introduced news articles collected in Wikinews [11] from 2004 to 2018, where there are totally 21000 articles are used for our training and testing set. Each article has different tags (for example, "Soccer", "Sports" and "England" may appear together at a single news article).

So totally we have about 22000 articles for the categorization task, and we divide them into 60% of training set, 20% of validation set and 20% of testing set.

B. Preprocessing

For text data, we need to preprocess it before feeding them into the vectorizer and classifier. We did following preprocessing: (1) Remove all the unnecessary punctuations; (2) Clean up all the stopping words from the stopping word list; (3) Create a dictionary for indexing and convert string of tokens into a list of word indices. We also tried methods like word stemming but we didn't see any performance increase, partly because the vectorizer captured the non-stemmed words when building the model.

C. Vectorization

1) TF-IDF: TF-IDF is a numerical statistic to measure the importance of words in documents. It increases proportionally to the "term frequency" i.e. the times a word occurs in a document, and it is also proportional to the inverse document frequency, which is larger if the word appears in fewer documents:

$$\text{TF-IDF}(d, w) = \frac{\sum_{i=1}^{|d|}[d_i = w]}{|d|} \log \frac{N}{|\{d' : w \in d'\}|}$$

For each word we can calculate the TF-IDF value, so we can get a V-dimensional vector for each article, where V is the total number of words in the corpus.

2) Word2vec: One of the problem in TF-IDF is that it doesn't reflect the relationship between words, and the dimensionality is too high for some tasks. Word2vec [12] was introduced to learn vector representations of words, named "word embeddings". It projects words from the one-hot space to the embedded space, where similar words are close to each other in the embedded space.

The algorithm trains a "skip-gram" model which receives a word as input and tries to output all words inside its context. The model network has a weight matrix (for hidden level), which is our final objective, and several output layers. In training process, we minimize the loss function, the conditional probability of the output:

$$E = -\log \prod_{i=1}^{C} P(w_{O,i}|w_I)$$

D. GloVe

Similarly, GloVe [13] is also an unsupervised learning algorithm to obtain vector representations for words. GloVe constructs a global co-occurrence matrix to measure how often two words are connected within a context. Then it calculates word vectors by optimizing the following loss:

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - \log P_{ij})^2$$

GloVe doesn't differ so much in performance compared with word2vec, while it is faster to train to get better performance sooner.

Entity Name	Description
PERSON	People,including fictional.
NORP	Nationalities or religious or political groups.
FAC	Buildings, airports, highways, bridges,etc.
ORG	Companies, agencies,instituitions,etc.
GPE	Contries, cities,states.
LOC	Non-GPE locations, mountain ranges, bodies of water.
PRODUCT	Objects, vehicles, foods, etc.
EVENT	Named hurricanes, battles, wars, sports events,etc.
WORK OF ART	Titles of books, songs, etc.

TABLE I. A selection of entities provided by OntoNotes 5

IV. Methods

A. Named Entity Extraction

Our group first experimented on entity extraction. However, we encountered problems at the very beginning for the lack of data to train our own models. We only have about 250 articles and no samples of reliable entities.

Given the fact that NER field has been thriving for more than twenty-five years, although most of the work has concentrated on limited domains and textual features, good techniques are available in terms of news articles. In this part, we tried to implemented two popular techniques in NER, while evaluating the outcome based on the database we have in this task, trying to pick up the more suitable one for future use.

1) spaCy: spaCy [14] is a relatively new package for Industrial strength NLP in Python. Based on **Neural Network** algorithms, it provides a one-stop-shop for tasks commonly used in any NLP project, including Tokenizations, POS tagging and Named Entity Recognition.

Named entities can be successfully accessed through the *Doc* objects *.ents* method. By applying this method through the whole data set, we successfully extracted entities of each articles for further classification.

Given bunches of built-in models, we picked up the one traind on OntoNotes 5 corpus, providing entities such as *PERSON*, *GPE (Countries, cities, states)*, *ORG (Organizations)* and *EVENT*, which performed well in our task. Although the built-in entity types are complete enough, we may still want to add our own entities based on specific data set, which can also be realized through the `add_label` method on *spaCy*. We can achieved this goal if there are more data in the future.

Although it is quite powerful, we still need to figure out the fundamental theories of *spaCy* on specific projects. It is also important to evaluate the performances of it when processing our project. Thankfully, *spaCy* offers evaluation metrics(function `Score()`, providing details for model selection.

2) Support Vector Machines: Another model we implemented on NER is Support Vector Machines (SVMs). In the field of natural language processing, SVMs are applied to text categorization, and are reported to have achieved high accuracy without falling into over-fitting even though with a large number of words taken as the features.

We first applied the method TF-IDF to vectorize articles. Then we used this to construct the feature vector for the words. The feature vector used in this SVM model comes from the current word interested, the words preceding it, and the words following it. The number of words to take preceding or following is considered as a parameter to be adjusted during the training, and this is called window size. Based on the work of Li et al[15], we used the optimal window size 2.

Due to the fact that our data set is unlabelled, we performed the SVM training on CoNLL 2003 shared task (NER) data. This training data set contains four NER categories: Location, Person, Organization and names of miscellaneous entities. The fitted SVM model is then applied on our news data set and classifies the

words. The kernel we used when training the SVM model is quadratic Kernel.

B. Topic Labelling

1) Baseline: Binary-classification methods: For the topic labelling part, we use single-labeled (i.e. binary) classifier as our baseline to evaluate all the other methods.

Here we use three methods:

1) k-nearest neighbors (k-NN). In k-NN method, for each input we find k neighbors in the feature space which has has the least geometric distance to it, and let them vote for the class of the input. For multi-label classification problem, for each label, we let k neighbors vote if the input belongs to this label.
2) Naive-Bayes classifier. Naive Bayes classifier calculate the posterior probability $P(c|x)$ using Bayes' Rule:

$$P(c|x) \propto P(x|c)P(c)$$

where we apply the Bayes' assumption that $P(x|c) = \prod P(x_i|c)$, which means each element (word) is independent from its neighbors in the article.
To estimate the parameters for features' distribution, we need to assume distributions, i.e. the event model of the classifier. For discrete features, e.g. multivariate Bernoulli distribution, we can use Bernoulli Naive Bayes, where $P(x|c)$ is defined as

$$P(x|c) = \prod_{i=1}^{n} p(x_i|c)^{x_i}(1 - p(x_i|c))^{1-x_i};$$

For continuous data, we can assume the data follows Gaussian distribution and have

$$P(x|c) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-(x-\mu)^2/\sigma^2).$$

3) Support Vector Machine. In linear space for binary classification problems, the objective of SVM is to find a decision boundary which maximizes the geometric margin by solving the optimization problem

$$\min_{w,b} s.t. \frac{1}{2}\|w\|^2 \text{ s.t. } y^{(i)}(w^T x^{(i)} + b) \geq 1, i = 1, \ldots, m$$

By applying kernel tricks using kernel functions, it is easy to extend the data to a very high-dimensional space. For most of our experiments, we tried the basic linear SVM as well as SVM with polynomial kernels and radial-basis function ($\phi(r) = \exp(-(\epsilon r)^2)$).

2) Classifier Chains: This algorithm [4] finds the labels by transforming the multi-label problem to a chain of binary classification problems, each of which related to the previous one. It generates a chain of binary classifier $C_1, \ldots, C_{|l|}$ to calculate $P(l_1|x), P(l_2|x, l_1), \ldots, P(l_{|l|}|x, l_1, \ldots, l_{|l|-1})$. We can use any type of binary classifier as the base classifier to build the chain. Since the sequence of l affect the result greatly, in practice we usually use ensembles of classifier chains, creating several random sequences of l, building classifier based on them and take the average.

3) ML-kNN: This method was first proposed by Zhang et al. in 2007 [7]. It is derived from the traditional k-NN algorithm to gain information about its nearest neighbors. For each point x, assume it has a neighbor set $N(x)$, for each class l, the algorithm counts the numbers of neighbors belonging to this class: $C_x(l) = \sum_{a \in N(x)} 1[a \in l]$. Then it applies the maximum a posteriori (MAP) rule to calculate probability of each class and determine the label set. Let H_1^l and H_0^l be events that the data point is included in class l or not, and E_j^l be the event that test data has exactly j neighbors in class l. Then the objective is to determine if we have

$$P(H_1^l|E^l_{C_x(l)}) > P(H_0^l|E^l_{C_x(l)})$$

According to Bayes rule we know that

$$P(H_1^l|E^l_{C_x(l)}) \propto P(E^l_{C_x(l)}|H_1^l)P(H_1^l),$$

and both values on the right hand side can be calculated by counting frequencies in the train set. Thus it is possible to derive all the labels.

4) Convolutional Neural Network: In 2014, Kim et al. [8] proposed a method to classify sentence sentiment using convolutional neural network. It uses a embedding layer to convert all words into vector representation, and use convolutional and pooling layers to obtain features of the same dimension for different input data. Different Conv-Maxpool layers capture information within different local context. Here we adapt his method to multi-labelled classification problem and have the following neural network as shown in Figure 1.

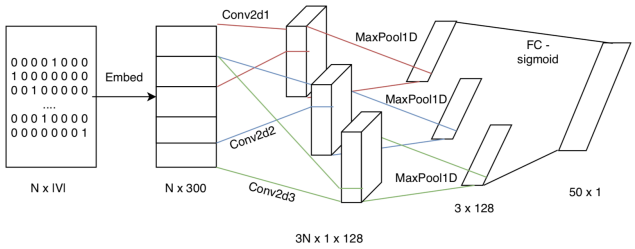


Fig. 1: Convolutional Neural Network Architecture

The output is the probability for each class. In the optimizer we calculate the Binary Cross Entropy between the ground truth and the predicted result as the loss function:

$$\ell = \frac{1}{d} \sum_{n=1}^{d} [y_n \log x_n + (1 - y_n) \log(1 - x_n)]$$

V. Evaluation Methods

A. Named Entity Extraction

As mentioned before, due to the limitation of the dataset, it is unlikely for us to train our own model. However, we still need to evaluate models so that we can pick up the most suitable ones. One purpose of NER in the task is to identify not only **accurate**, but also **valuable** entities. In our dataset, each article have 2-3 entities as tags extracted mannually:**Persons / Groups, Regions, Events and Places**. The model picked should make sure that those tags are extracted and classified properly, which should be taken into account when evaluating models.

Thus, we designed one way to evaluate models' perfomance: every time we extracted entities using a specific model, we compared the outcome with those existing tags in the dataset, presented as the **similarity** rate, calculated as:

$$\text{similarity}(i_j) = \frac{\#(\text{data}_i \cap \text{model}_j)}{\#(\text{data}_i)}$$

where $\text{data}_i \in \{\text{existed tags of data}\}$, $\text{model}_j \in \{\text{entities of model}\}$. We can also sum up similaties in each tags, as a measurement of performances of entities belong to a specific tag being sucessfully extracted.

For model training, we use a standard evaluation method. The **accuracy score** is misleading in NER. To fix this problem, we will look at **precision**, **recall rate** and **F1 score** in addition. They are computed using tp (true positives, refering to number of labels of a class that are predicted correctly), fp(false positives, refering to number of predictions of a class that are wrongly predicted), and

fn(false negatives, refering to number of predictions that predict a class but are not labeled as belonging to the class). Then we have equations:

$$\text{Pre} = \frac{TP}{TP+FP}, \text{Rec} = \frac{TP}{TP+FN}, \text{F}_1 = 2 \cdot \frac{\text{Pre} \cdot \text{Rec}}{\text{Pre}+\text{Rec}}.$$

We will evaluate different NER methods by these numbers.

B. Topic Labelling

For multi-label classification problem, we have a lot of metrics which can be divided into two basic types [3] : **example-based metrics**, focusing on tagging accuracy of each article, and **label-based metrics**, focusing on the label-wise statistics for each tag.

For example-based metrics, we can use metrics like example-based accuracy, precision, recall and F-value, as well as hamming loss and ranking based metrics to evaluate our algorithms. For label-based metrics, we usually calculate the true/false positive/negative values to evaluate the binary classification metrics for each class and then do some averaging over classes.

Here we choose the following metrics to evaluate our classifier: For examples, we calculate the hamming loss [3]

$$\text{Hamming}(h) = \frac{1}{m} \sum_{i=1}^{m} |h(x^{(i)}) \Delta y^{(i)}|$$

where $A \Delta B$ means the symmetric difference between sets A and B. This measures the fraction of misclassified instance-label pairs, i.e. a relevant label is missed or an irrelevant is predicted. And we also evaluate the F-1 score over all examples:

$$\text{F}_e(h) = \frac{2\text{Pre}_e(h)\text{Rec}_e(h)}{\text{Pre}_e(h) + \text{Rec}_e(h)}$$

where $\text{Pre}_e = \frac{1}{m} \sum \frac{|h(x^{(i)}) \cap y^{(i)}|}{|h(x^{(i)})|}, \text{Rec}_e = \frac{1}{m} \sum \frac{|h(x^{(i)}) \cap y^{(i)}|}{|y(i)|}$.

For labels, we calculate the weighted average of F-1 score in all classes, i.e calculate

$$TP_j = |\{x|y_j = h(x)_j = 1\}|; FP_j = |\{x|y_j = 0, h(x)_j = 1\}|;$$
$$TN_j = |\{x|y_j = h(x)_j = 0\}|; FN_j = |\{x|y_j = 1, h(x)_j = 0\}|$$

and thus

$$\text{F}_l(h) = \sum_{i=1}^{l} \frac{w_i}{\sum_{j=1}^{l} w_j} \cdot \frac{2TP_i}{2TP_i + FP_i + FN_i}$$

where w_i is the weight for label i.

VI. Experiments and Discussion

A. Named Entity Extraction

According to [16], when evaluating the named entity extraction systems we usually use the precision, recall and F measures as the metrics to evaluate the named-entity systems. We can give more weights to precision if we need to remove all unneeded entities or give more weights to get more entities collected.

Based on the evaluation proposed above, we compared the two NER models.

We can see that spaCy did better in the extraction of entities on type Person and Places, and SVM did better in the extraction of entities on type Regions. spaCy performed poorly on the extraction of event, and SVM did not extract the event because it was not in the given entity categories. Also notice that in the SVM model, one word could be classified into multiple entities. A possible reason for the performance of SVM was that the feature vector was a long sparse vector, which did not contain a large set of information about the word. Therefore, the result of the SVM model was not as ideal as we would expect.

(a) Evaluation of spaCy and SVM NER

		Persons	Regions	Events	Places
spaCy	PERSON	0.480	0.111	0	0.051
	NORP	0.013	0	0	0
	FAC	0.006	0	0	0.007
	ORG	0.315	0	0	0.044
	GPE	0.022	0.333	0	0.859
	LOC	0.003	0.333	0	0.015
	PRODUCT	0.010	0	0	0
	EVENT	0.003	0	0.021	0
	WORK OF ART	0.003	0	0	0
	Others	0	0	0	0
SVM	PERSON	0.239	0.444	0.007	0.311
	ORGANIZATION	0.124	0.111	0	0.104
	LOCATION	0.019	0.667	0	0.363
	MISC	0	0	0	0.037

(b) Sum up similarities of two NER methods

	Persons	Regions	Events	Places
spaCy	0.857	0.777	0.024	0.977
SVM	0.382	1	0.007	0.815

TABLE II. Evaluation of two NER Methods

Table II presented the percentage of named entities that were captured by the models. Words belonging to Persons, Regions and Places were mostly captured as named entities in the spaCy model. Words belonging to Regions and Places were mostly captured as named entities in the SVM model. Although the percentage in the SVM model was not exact because there were chances that one word was classified into multiple entities, the number could still give us a view of named entities being captured.

In terms of model efficiency, we found that spaCy took less time to train, but SVM model took about an hour to train the same amount of data.

B. Topic Labelling

Using the Wikinews dataset, we do topic labelling on the top 50 labels by frequency. The labels include large categories (Crime and law, Internet, Human Rights, etc.), locations (North America, United States, California, etc.), detailed entity (Republican, Iain Macdonald, etc.), and they have a lot of overlapping with each other — the distribution heatmap original data pairs (see Figure 2) is usually less than 0.4, and we can see there are some vertical stripes with high frequency which represents *North America, Asia, USA, Germany*, etc.

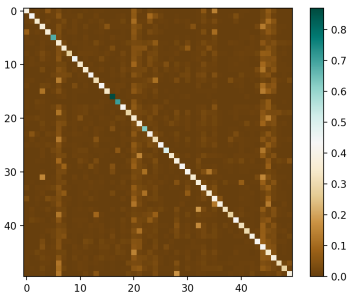


Fig. 2: Distribution heatmap of the ground truth y

We tried different vectorization methods mentioned in and used the methods mentioned in Section IV.B, including baseline methods (kNN, Naive Bayes and SVM), multi-label methods (ML-kNN and Classifier Chain), and convolutional neural networks. We used some implementations in `scikit-learn` [17] like k-NN, Naive Bayes and SVM.

The results of non-CNN methods is shown below.

For the CNN classifier, we implemented the network architecture using PyTorch 1.0[18]. For embedding, we tried three

	KNN	Bayes	SVM	Chain	ML-kNN
TF-IDF	2.0560	3.0286	1.6786	**1.6732**	1.9309
	0.7269	0.6867	0.7894	**0.7883**	0.7656
	0.6204	0.5745	0.7185	**0.7203**	0.6829
Word2vec	2.3790	8.2244	**2.0224**	2.0688	2.1373
	0.6891	0.4838	**0.7429**	0.7337	0.7181
	0.6281	0.5585	0.6854	0.6308	**0.6938**
GloVe	2.6835	9.6642	**2.6065**	1.8893	2.7054
	0.6443	0.4199	0.6378	**0.7643**	0.6463
	0.4756	0.5622	**0.6899**	0.6818	0.6481

TABLE III. Non-CNN Classifier Metrics on test set, red = Hamming, blue = F-1_e, black = F-1_l

methods: using only random uniform values, using word2vec pretrained outputs and using GloVe outputs. we do dropout with probability 0.5 after embedding. In the network, we define 4 convolutional layers, each of which has size 3, 4, 5, and 6, and generates output of size 128. Rectified linear unit (ReLU) is used as activator in all conv layers. We use Adam optimizer with learning rate 2e-4 to train the network.

The results of CNN-method is shown in Table IV.

	Word2Vec	GloVe	No Pretrain
Static	0.718	**0.726**	0.648
Nonstatic	0.717	/	/

TABLE IV. Label-based weighted F-1 values of CNN method

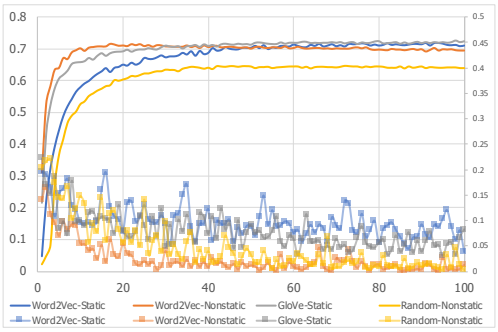
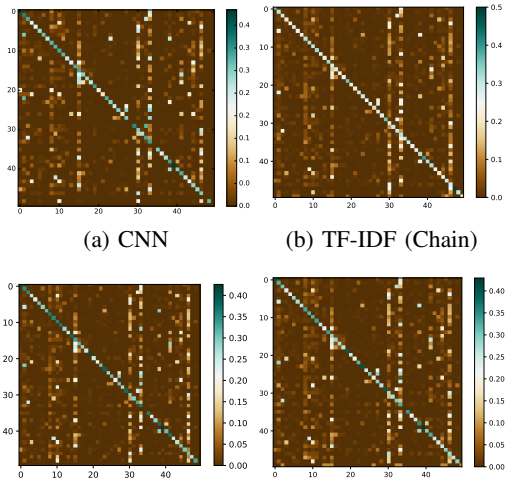


Fig. 3: Training Process. x axis: epoch, left: F value, right: training loss

Our experiments have shown that:

(1) For non-CNN methods, simple vectorization (i.e. using tf-idf weights directly) works better, since it contains the most complete information and the corpus provided is large enough to cover most possible words. While it has the highest precision and recall, the problem is that the tf-idf space has too many dimensions, which make some methods (like ML-ARAM we also tried) fail to running on the dataset. And one another reason that word2vec and GloVe doesn't work well in non-CNN method is that is contains few temporal information — we see that the performance goes up after using convolution layers. Another thing is that both training word2vec and GloVe requires a very large amount of input data so that we have to use pretrained models. Difference between training set and our dataset in word2Vec and Glove also affect the performance.

(2) Compared with other multi-label classification methods, chaining classifier creates a chain of conditional possibility classifier and uses ensemble method to build the best classifier, which is easy to implement, and has better performance in practice due to combination with CNN classifiers. ML-kNN runs fast (can calculate all classification results in one pass, runs faster compared with all other methods) but the performance is not so outstanding compared with other methods.



(a) CNN (b) TF-IDF (Chain)

(c) Word2vec (SVM) (d) GloVe (SVM)

Fig. 4: Comparison of Heatmaps

(3) Simple CNN with little hyperparameter tuning achieves good result on our dataset, and pre-trained model helped a lot on embedding. The pre-trained weight parameters help the model avoid overfitting so that it achieves higher performance. We find that if we start from the random state or start from a pretrained vector but the weight can be modified, it will be easier to get overfitting so that the overall performance is affected.

(4) The accuracy of models are highly different by type of classes. For large and generic classes (like *Disasters and accidents, Politics and conflicts, Sports*), or specific things (*Football (soccer), Weather*), the F-value is very high, compared with that of locations (*South America, Germany, France, London, California and World*), which can be less than 0.3, greatly affecting the total F value. That is partly because data with location labels are usually scattered in the text vector space — this can be of multiple topics, and thus the classification method won't work. Things we can do include that we increase the weight of named entities we found, or we use different methods for named entity labelling.

(5) If we plot all the heatmaps for four best methods, as shown in Figure 4, we can find that performance for different algorithms has some difference in amount of noise and wrong labels. Here we can see a bright strip at label 10, 15, 30, 33, 44, 46, which corresponds to *Canada, Iraq, Transport, Economy and Business, Politics and conflicts, Crime and law*. The first four of them are dim in Figure 2, so it means that might be some wrong classifications.

VII. Conclusion and Future Work

In this article we tried to use natural language processing techniques and machine learning algorithms to extract useful information from network news articles.

We find that in terms of multi-label tagging / labelling, using chaining classifier, or using CNN with pretrained word embedding can have some good results, while for some classes there are still false positive problems. For named-entity extraction, neural-network-based spaCy outperforms in person and places recognitions while for regions SVM works better.

For future projects, we can try to do better feature engineering to see how to weigh important words, and solve location tagging problems. For NER part, as the dataset getting more and more mature, we will try to build our own NER model.

VIII. Contribution

All members contributed equally to this project.

Named entity recognition part is done by Linying and Mengtian. Wikinews data processing, text vectorization and topic labelling / classification are done by Yilong.

References

[1] L. Chiticariu, R. Krishnamurthy, Y. Li, S. Raghavan, F. R. Reiss, and S. Vaithyanathan, "Systemt: an algebraic approach to declarative information extraction," in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2010, pp. 128–137.

[2] H. Cunningham, V. Tablan, A. Roberts, and K. Bontcheva, "Getting more out of biomedical documents with gate's full lifecycle open source text analytics," *PLoS computational biology*, vol. 9, no. 2, p. e1002854, 2013.

[3] M.-L. Zhang and Z.-H. Zhou, "A review on multi-label learning algorithms," *IEEE transactions on knowledge and data engineering*, vol. 26, no. 8, pp. 1819–1837, 2014.

[4] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," *Machine Learning*, vol. 85, no. 3, p. 333, Jun 2011. [Online]. Available: https://doi.org/10.1007/s10994-011-5256-5

[5] A. Elisseeff and J. Weston, "A kernel method for multi-labelled classification," in *Advances in neural information processing systems*, 2002, pp. 681–687.

[6] A. Clare and R. D. King, "Knowledge discovery in multi-label phenotype data," in *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 2001, pp. 42–53.

[7] M.-L. Zhang and Z.-H. Zhou, "Ml-knn: A lazy learning approach to multi-label learning," *Pattern recognition*, vol. 40, no. 7, pp. 2038–2048, 2007.

[8] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, 2014, pp. 1746–1751. [Online]. Available: http://aclweb.org/anthology/D/D14/D14-1181.pdf

[9] S. Lai, L. Xu, K. Liu, and J. Zhao, "Recurrent convolutional neural networks for text classification." in *AAAI*, vol. 333, 2015, pp. 2267–2273.

[10] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in neural information processing systems*, 2015, pp. 649–657.

[11] "Wikinews." [Online]. Available: https://www.wikinews.org/

[12] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[13] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation." in *EMNLP*, vol. 14, 2014, pp. 1532–1543.

[14] M. Honnibal and M. Johnson, "An improved non-monotonic transition system for dependency parsing," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, September 2015, pp. 1373–1378. [Online]. Available: https://aclweb.org/anthology/D/D15/D15-1162

[15] Y. Li, K. Bontcheva, and H. Cunningham, "Svm based learning system for information extraction," in *Deterministic and Statistical Methods in Machine Learning*. Springer, 2005, pp. 319–339.

[16] M. Marrero, S. Snchez-cuadrado, J. M. Lara, and G. Andreadakis, "Evaluation of named entity extraction systems," in *Advances in Computational Linguistics, Research in Computing Science*, 2009, pp. 41–47.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[18] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.