Sequential Labelling with Active Learning to Extract Information about Disasters

Mustafa Erkan Başar
Master's Thesis
Department of Artificial Intelligence,
Radboud University Nijmegen, the Netherlands
ebasar@science.ru.nl
27/08/2017

Abstract

Learning from past incidents has a great importance for disaster managers. Estimation of the outcomes beforehand can improve preparations for the next incidents. To make this a less labour-intensive task, we aim to automate extracting information from past events. We focus on extracting critical information about flooding events from newspaper articles as our use case. We treat this information extraction task as a sequential labelling task and create an ensemble of two supervised machine learning algorithms, namely Conditional Random Fields and Structured Support Vector Machines, to achieve our goal. However, supervised learning requires manually annotated training data, which is very expensive and time-consuming to obtain. To reduce the need for manual annotation, Active Learning, a human-in-the-loop method, is explored. We obtain improvement on f1-score up to 25% and observe that Active Learning drastically reduces the effort required by annotation.

1 Introduction

In emergency situations, even seconds can make a difference between life and death. The decisions may save lives or result in a loss of valuable time. Disaster managers have to respond quickly to the situations during disasters. In this sense, they need to be aware of the possible dangers and threats even before they actually see the effect of the disaster.

One of the biggest problems in disaster management is the distribution of emergency equipment. Disaster management organisations such as Red Cross store their equipment in regional warehouses. During a disaster, if the equipment stored in a regional warehouse is not enough for that region, they have to activate another regional warehouse. However, bringing equipment from another region increases the response time. Moreover, a warehouse is often equipped based on its capacity. The disaster managers should be able to say that a warehouse is lacking some of the items inside because of the limitations of its space. Thus, they may consider building another one. Realising that the capacity of a warehouse is not enough should be based on precise information. By analysing the past operations, the risk profiles of the areas can be created. Eventually, disaster managers can use the risk profile to capacitate the warehouses based on the actual needs, not on its space.

Relief distribution is another problem that disaster managers encounter. If the disaster managers can identify the needs, they can distribute the goods efficiently. However, the number of affected people is often obscured in the early stages of a disaster. By looking at the risk profiles, the disaster managers can estimate beforehand the quantities of the items they should include in certain regions. Thus, estimating the number of people who have been under risk can help them to respond needs effectively.

Having the information of past operations can improve the preparedness and the anticipation of quantities. The disasters are one of the main interests of news agencies. Moreover, reporters often collect and include detailed information such as casualties and economical damage while reporting the incidents. Therefore an archive of news articles about disasters is also an archive of critical information about disaster events. Hence disaster managers can have access to a history of events by extracting information from digitalised newspaper archives. With using the extracted information, they can prepare themselves better for future disasters. However, collecting such amounts of data is long, expensive and hard to maintain when it is done manually by people. We aim to automate the data collection and interpretation to quickly supply the needed information. We aim for this automated extraction to be as accurate as possible, knowing that our methods will probably not be as precise as human annotation.

The task of automatically extracting information from unstructured text is called Information Extraction (IE) and is studied as part of the Natural Language Processing (NLP) area. Although many techniques have been introduced in recent years to resolve information extraction problems, in this study, we treat it as a sequential labelling problem. In sequential labelling tasks the goal can be described as labelling the sequences of relevant information by a single categorical class. In other words, the goal is to find and classify the relevant word sequences in the text. Furthermore, we apply machine learning techniques to accomplish this goal.

A classic machine learning framework for labelling sequential data is linear-chain Conditional Random Fields (CRF) (Lafferty et al., 2001). CRF prevents the label bias problem that is encountered in its predecessor algorithm, Maximum Entropy Markov Models, which occurs when there is an uncertainty in the previous tag of the sequence (Peng and McCallum, 2006). The strength of the CRF is also coming from the ability of dealing the arbitrary, overlapping features of the input (Culotta and McCallum, 2004). CRFs have been applied to many information extraction tasks and are accepted as one of the state-of-the-art approaches. Moreover, it is currently used in many leading information extraction tools including the one in Stanford CoreNLP (Finkel et al., 2005).

Another state-of-art machine learning approach is Structured Support Vector Machines (Structured SVM or SSVM) (Altun et al., 2003). Like Conditional Random Fields, Structured SVM is also applied to sequential labelling successfully. Massachusetts Institute of Technology (MIT) prefers to use Structured SVM in their information extraction tool named MITIE (King, 2009). Many comparisons have been made between CRF and SSVM algorithms. On the one hand, some studies claimed that SSVM outperforms the rest (Nguyen and Guo, 2007). On the other hand, some studies reported that there is not a significant difference (Keerthi and Sundararajan, 2007).

CRF and SSVM are supervised machine learning approaches, meaning that we teach the machine how to predict by using pre-annotated data. The gold standard of the pre-annotated data can be created by human annotators. However, manual annotation is time-consuming and expensive. Active Learning is one of the human-in-the-loop concepts that has been proposed to reduce the time required for manual annotation. In principle, an initially trained classifier automatically labels an unseen data and human annotators are asked to correct only some of them. Thus, the annotation process gets faster.

There are several strategies in Active Learning to decide on what part of the data will be presented to the human annotators (Lewis and Gale, 1994). In Passive Learning strategy, every prediction is presented to the human annotator to be checked. In another strategy called Random Sampling, certain portions of the predictions randomly selected and submitted to the human annotator. Because the selections are made randomly, this strategy focuses on exploration rather than exploitation. Here in this study, we use the Uncertainty Sampling strategy where the labels about which the machine-learning classifiers are the least confident are returned to be checked. Because of that, the Uncertainty Sampling helps to sharpen the learning curve by exploring the decision boundary further (Cawley, 2011). In other words, the system redirects the hardest and most confusing decisions to the human annotators and learns from the answers returned.

In conclusion, besides using supervised machine learning approaches to perform information extraction, we focus on building an Active Learning system to overcome the time issue with manual annotation in supervised learning. Time is one of the major problems with manual annotation and we intend to find out if the Active Learning approach accelerates the annotation process. Our hypothesis is that it is possible to obtain more training data in a shorter time by using the Active Learning techniques. To test this hypothesis, we define our baseline as the time it takes for initial manual annotation without involving any automated process. At the end of our study we compare our baseline to the time it takes for the annotations with the Active Learning system involved.

With the usage of Active Learning, the quality of the annotations emerges as another question. Thus, we aim to find out if involving Active Learning improves the quality of the annotations. We hypothesise that the quality will be improved. We test this hypothesis by comparing the Active Learning assisted system to our baseline human-in-the-loop system.

2 Information Extraction

2.1 Data Retrieval

Flood-related news articles are collected from The Guardian API[1] and the Global Database of Events, Language, and Tone (GDELT) Project[2]. The Guardian API is the easiest to use and has the most organised data. The content is a clean plain text. In contrast, GDELT does not provide anything but the URLs of the online news articles. Thus, we scrape the content of the articles from those URLs by using the Beautiful Soup library (Richardson, 2017) in Python. The output of the scraping method is cleaned from leftover HTML tags by using regular expressions.

2.2 Manual Annotation

Before starting to annotate the data, we decide on the class types. To be able to define an event, it is required to know at least the name the event goes by (henceforth referred to as the identifier), the location, and the time of the event. However, our intention is extracting further information about events besides detecting them. Therefore, we include other critical information such as the casualties, damages, responses etc.

Aside from the classes directly connected to the event, we add a helper class to detect the exact locations. In our data, we encounter long described locations. For instance, the location of the event may be mentioned as 'villages on the slopes of the Mayon volcano'. Moreover, we see that the information of the exact location is spread out in the sentence. Training the classifier to detect the location as a whole is a bottleneck. To overcome this issue, the idea is to annotate the supplementary parts of the location as additional information. We call this extra class the supplementary of the location. Thus, in our example, the 'Mayon volcano' should be annotated as the specific location. The 'villages' and the 'slopes' should be annotated as the supplementary. The supplementary and the specific location classes complete each other. They work together to provide the details of a location.

The final version of our class list is the following.

1. Identifier of the event,
2. Specific location of the event,
3. Supplementaries of the locations,
4. Time of the event,
5. Number of casualties,
6. Number of people injured,
7. Number of people missing,
8. Number of people evacuated,
9. Damage to properties,
10. Damage to economy,
11. Damage to business continuity,
12. Responses to event in the form of donations,
13. Responses to event in the form of supplies.

We use an online annotation interface called Format for Linguistic Annotation Tool (FLAT) to manually annotate the documents. FLAT employs a special XML format called Format for Linguistic Annotation (FoLiA; van Gompel and Reynaert, 2013). FoLiA is developed specifically for linguistic annotation tasks. Thus, we convert the articles into FoLiA XML format.

[1]`http://open-platform.theguardian.com/`. Accessed August 26, 2017.

[2]`http://www.gdeltproject.org/`. Accessed August 26, 2017.

Initially, 59 news articles are annotated. 44 of them are used to train the initial classifier. The rest of them are stored to be used as test data and never included in any training process. The data is manually annotated by the author.

2.3 Beginning-Inside-Outside Formatting

Sequential labelling has different interpretations; 'raw labelling' and 'joint segmentation and labelling' (Daume, 2006). In the raw labelling approach, each token receives a single tag. Part-of-Speech tagging is one of the most common examples of raw labelling. The joint segmentation and labelling approach is typically used for tasks where multiple tokens may form a single entity that receives a single tag. Therefore, the system should be able to detect the entities with multiple tokens before the classification. Joint segmentation and labelling is commonly used to solve Named Entity Recognition problems. On the one hand, joint segmentation and labelling is the most effective approach because it allows us to use features related to sequences. On the other hand, joint segmentation and labelling is more complicated to implement than raw labelling because it requires an additional segmentation process.

Beginning-Inside-Outside (BIO) formatting allows us to convert a joint segmentation and labelling problem into a raw labelling problem. The tag of the first token of an entity becomes the beginning tag (notated as 'B.') of the class of that entity. If there is more than one token in the entity, the other tokens get the inside tag (notated as 'I.') of the same class. For instance, if the entity is *'26 March 2015'*, *'26'* is labeled as beginning, *'March'* and *'2015'* are labelled as inside tag of the corresponding class.

Besides changing the formatting inside the entities, BIO formatting suggests an additional class for uninformative tokens that we are not interested in. This class is commonly called Outside and is used as a generic negative class label that indicates the corresponding token does not belong to any entity.

BIO doubles the number of classes by producing two versions (the beginning and inside) of each tag. Hence, when the BIO formatting applied to a set of n classes, the number of informative classes becomes $2n$. Finally, when the outside class is included, the total number of classes that the classifier has to take into account becomes $(2n + 1)$.

2.4 Threshold on Sample Sizes

Initially, we define and manually annotate 13 classes as it is explained in Section 2.2. However, some of the classes are not mentioned in the articles at a sufficient rate. For instance, the damage on the business continuity class is used only twice during the manual annotation. Moreover, during the experiments, we observe that the classes with insufficient sample sizes confuse the classifier. Thus, they decrease the overall accuracy. Therefore, we set a threshold on the sample size up to 20 samples. The informative classes without a sufficient number of samples are replaced by the class of the uninformative tokens, the Outside class. Hence, those classes are ignored from the dataset as they are never used during the manual annotation.

The list of the classes that are not ignored and their initial sample sizes are listed in Table 2. The experiments are performed with using the classes in the same table. In our case, the damage to the economy, the damage to the business continuity, the number of people missing and the number of people injured classes are ignored. The classes that are ignored in this step are never included back into the dataset in any step of this study.

2.5 Feature Extraction

Feature extraction is one of the most important processes in machine learning because the features have a great impact on the classification accuracy. Here we use features that express the characteristics of the tokens in the data.

We use a sliding window with a length of 5 tokens to extract the token level features. The token in the middle of the window is the token that will be annotated. Thus the first two tokens in the window are the left side of that corresponding token, and the last two tokens are the right side of the same token. More formally, if we are extracting features of a token at position i, the sliding window will contain token_{i-2}, token_{i-1}, token_{i}, token_{i+1}, and token_{i+2}.

Take the sentence 'Typhoon Nona paralyses central Philippines.' as an example. If we extract the context features of the tokens in the sentence starting from the third token, context_{i} would be ‘paralyses’, $\text{context}_{\text{i-1}}$ would be ‘Nona’ and $\text{context}_{\text{i+1}}$ would be ‘central’ etc. After the features of the third token are extracted, the system would proceed to extract the features of the fourth token in the sentence. Thus, this time $\text{context}_{\text{i}}$ would be ‘central’, $\text{context}_{\text{i-1}}$ would be ‘paralyses’ and $\text{context}_{\text{i+1}}$ would be ‘Philippines’. As a difference, for the fourth token, $\text{token}_{\text{i+2}}$ would not exist because the sentence ends after the fifth token. In such a case the corresponding context feature, $\text{context}_{\text{i+2}}$, would be left empty.

Besides getting the features of individual tokens in the sliding window independently, we also include n-grams of these tokens. N-grams are combinations of the information derived from the surrounding tokens. N-grams make the classifier consider the combinations as a single feature. We use bi-grams and tri-grams as demonstrated in Table 1.

If we take the sentence ‘Typhoon Nona paralyses central Philippines.’ again and extract the bi-grams of the third token, $\text{context}_{\text{i-1}}$ + $\text{context}_{\text{i}}$ combination would be ‘Nona+paralyses’ and $\text{context}_{\text{i}}$ + $\text{context}_{\text{i+1}}$ combination would be ‘paralyses+central’. In the end, including the ones in mentioned in the previous paragraph, the list of context features for the third token would be ‘Nona’, ‘paralyses’, ‘central’, ‘Nona+paralyses’, ‘paralyses+central’ etc.

Bi-grams	$\text{token}_{\text{i-1}}$ + token_{i} token_{i} + $\text{token}_{\text{i+1}}$
Tri-grams	$\text{token}_{\text{i-2}}$ + $\text{token}_{\text{i-1}}$ + token_{i} $\text{token}_{\text{i-1}}$ + token_{i} + $\text{token}_{\text{i+1}}$ token_{i} + $\text{token}_{\text{i+1}}$ + $\text{token}_{\text{i+2}}$

Table 1: Representations of n-grams that we have used.

Features

Hereunder we describe the features that are used in this study.

- **Context**

  We use the context of the tokens as features. Sliding window and n-gram logics are applied while adding the context of the tokens to the feature vectors.

- **Orthographic Types**

  The orthographic types of the tokens are extracted during the creation of FoLiA XML files by the built-in tokenizer in FoLiA library. The major orthographic types can be listed as word, punctuation, and number. We only included the orthographic types of $\text{token}_{\text{i-2}}$, $\text{token}_{\text{i-1}}$, token_{i},$\text{token}_{\text{i+1}}$, and $\text{token}_{\text{i+2}}$ individually because in most of the cases, the orthographic type is word. Thus we find forming bi-grams and n-grams of it useless.

- **Part-of-Speeches**

  We use the pre-trained part-of-speech tagger in the Natural Language Toolkit (NLTK) to obtain the part-of-speech tags (Bird et al., 2009). The tagger is trained on the Penn Treebank corpus consisting of news articles from Reuters newswire. Moreover, the Penn Treebank part-of-speech tagset contains 36 different classes[3]. We include the part-of-speech tags of the tokens by applying the sliding window and n-grams methods as described above.

- **Time Expressions**

  By adding the time expressions as a feature, we aim to improve especially the detection of time of the event class. We use a multilingual cross-domain temporal tagging tool called HeidelTime to find out whether a token is a time expression or not (Strötgen and Gertz, 2013). HeidelTime accepts an optional argument to define the type of the data source. We run the tool with specifying the type of the data source as ’NEWS’ to obtain results as accurate as possible.

- **Named Entities**

  We use the 7-class Stanford Named Entity Recognizer (NER) model (Finkel et al., 2005) to decide on the named entities. The model is trained on MUC 6 and 7 datasets. The implementation of the classifier is done by applying it in NLTK. According to the official results, the performance of the tagger is around 90% when it is used on news articles

[3] `https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html`. Accessed August 2017.

Classes	Notation	Sample Size
Identifier of the event	event.flood	292
Specific location of the event	loc.focus	192
Supplementaries of the locations	loc.supp	101
Time of the event	time	96
Number of casualties	dmg.ppl.kill	93
Number of people evacuated	dmg.ppl.evac	44
Damage to properties	dmg.prop	70
Responses to the event in the form of donations	resp.donat	22
Responses to the event in the form of supplies	resp.supply	91
Outside	O	39612

Table 2: The number of samples for each class that has a sample size above the threshold and their notations.

(Stanford NLP Group, 2003). We include the named entities of the tokens by applying the sliding window and n-grams methods as described in Section 2.5.

The 7 classes that can be detected by Stanford NER are LOCATION, ORGANIZATION, DATE, TIME, MONEY, PERSON, PERCENT. Some of these classes are helpful to detect specific information. In the news articles, specific locations of the events contain location names (e.g. 'Manila', 'the Philippines'). Thus, detecting the location names in a text is crucial to find locations of events. The MONEY class is effective on detection of information such as donations or economic damage because the entities of such classes consist of words that express money (e.g. '$90 million'). DATE and TIME classes are helpful for detecting the time of the events. Although the knowledge of the time expressions is also added as a separate feature (Section 2.5), the NER classes are kept to obtain the possible n-gram combinations with other NER classes. For instance, tri-gram NER class combination of 'Philippines 13 August' sequence would be 'LOCATION, DATE, DATE'. Thus, a tri-gram combination provides a unique information. The rest of the NER classes, PERSON and PERCENT, are helpful to distinguish the Outside class.

- **Class Occurrence Probabilities of Sentences**

The idea behind this feature is about mimicking one of the most basic human behaviours observed during manual annotations. Sometimes the true meaning of a token may be hidden outside of the boundaries of a sliding window. Human beings may also not be able to decide on the class by only looking at the close surrounding elements. In such cases, we tend to read the whole sentence to disambiguate the meaning of the information. Thus, this feature can be considered as an attempt to help the machine see the bigger picture.

We apply the idea by adding the probability of classes being used in the corresponding sentence as prior information. In other words, we calculate how likely a class might be used to label any of the tokens in a given sentence. Then, we add this probability into the feature vector of each token in that sentence. Thus, the classifier takes this sentence level information into account while deciding on the class of a single token. The classes that have high probabilities get higher chances to be chosen by the classifier. Likewise, the classes that are less likely to be used in that sentence become much less considerable. Eventually, this technique may help the classifier to focus on a smaller subset of classes per sentence.

In order to calculate the chances of occurrences, we train another classifier that returns the probabilities of the classes for sentences. We apply Linear Support Vector Classification by using Scikit-learn Python Library (Pedregosa et al., 2011) to train this additional sentence classifier. The sentence classifier is trained on the same data that used to train the information extraction classifier. Thus to use the class occurrence probabilities feature in the training of the information extraction classifier, we collect the probability predictions of the sentence classifier by using 10-fold cross-validation. We use tf-idf scores of the words as the only feature of the sentence classifier. Tf-idf score roughly gives an idea about the important and decisive words in the corpus.

For instance, assume that a sentence contains information about the identifier, the location and the time of an event as given in Figure 1. First, the sentence classifier runs on the sentence and calculates a single occurrence probability for each class. In the ideal world, we expect that the system returns high occurrence probabilities for these three classes and lower probabilities for the rest. As shown in the example, 75% probability is predicted for the occurrence of tokens related to the identifier of the event class and 98% probability for the occurrence of location related tokens. These probabilities are added to the feature vector of each token in the sentence. Therefore, for each token, the identifier-class feature is '75', the location-class feature is '98', and the time-class feature is '87' etc.

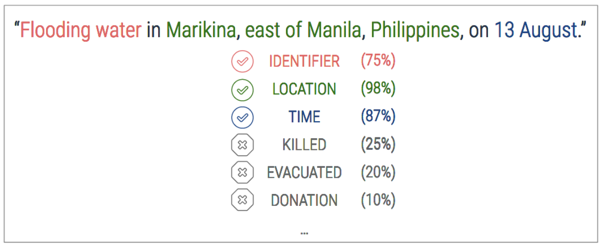


Figure 1: An example to the obtained class probabilities for the given sentence. Given probabilities are assumptions and are here only to serve as examples.

The class occurrence probabilities are also included in the feature selection process that is explained in Section 2.6. The fact that none of the occurrence probabilities is eliminated by the feature selection algorithm shows that there is a statistical correlation between the occurrence probabilities and the labels.

- **Cyclone Names**

  Tropical cyclones are named to make them more memorable (World Meteorological Organization, 2017). Moreover, tropical cyclones often cause floods, so that the cyclone names are often used in the news articles referring to flooding disasters. Since they indicate a disastrous event, these names are included in the entities labelled with the identifier of the event class. Thus, knowing that a token is a cyclone name is a clue to detect the events in an article. Therefore we use the information of a token being a cyclone name as a token based feature. Cyclone names are collected from Wikipedia[4].

- **Other Features**

  Besides the features mentioned, we include a few more such as a boolean value indicates whether the first letter of the token_i is capitalised or not. We also include the lemma of the token_i that is obtained by using the built-in lemmatizer in Natural Language Toolkit.

2.6 Feature Selection

Feature selection is an essential boosting method for the classification tasks. It is the process of detecting which features are the most relevant or useful and which features are irrelevant, redundant or useless. In other words, feature selection is designated to return the feature set that helps to obtain the best results by the classifier. It reduces overfitting, increases accuracy and shortens the training time because it reduces redundancy and removes misleading features.

We can divide feature selection methods into three categories: filter, wrapper and embedded (Guyon et al., 2008). The filter methods work to find the correlation between each feature and the labels. A score per feature is created by using the statistical measurements. Features are kept or eliminated based on their statistical scores. Thus, a feature is evaluated independently from the others. Examples of filter methods are ANOVA,

[4]`https://en.wikipedia.org/wiki/List_of_historic_tropical_cyclone_names`. Accessed August 26, 2017.

LDA, χ^2 (chi-square) test. In the wrapper methods, the models are created using different feature combinations. These models are evaluated and the scores are compared to each other. Thus, the feature combination with the highest score is chosen to be used in the classifier generation. Forward Selection and Recursive Feature Elimination algorithms are examples of wrapper methods. The embedded methods are applied during the time the model is being created. Common applications of the embedded methods are the regularization methods such as LASSO and Elastic Net. In the regularization methods, constraints are added into the optimization of the classifier to penalize the biased model (Brownlee, 2014).

Although embedded methods are accepted as the most effective approaches, in this study, we focus on filter methods due to time and computational limitations. Unlike the wrapper and the embedded methods, filter methods are not specific to a classifier. It allows us to eliminate unnecessary features by running once and for all algorithms in the pipeline. Filter methods are less time taking yet still being effective.

We implement the feature selection to choose the highest scoring 70% of the features by using Scikit-learn Python Library. For the statistical measurements, χ^2 test is applied. The χ^2 test first assumes that the features are independent of the labels and measures how far the features are away from this assumption. In other words, the test focuses on detecting the features that are irrelevant for classification.

Feature vectors of the tokens with the Outside class are not included in the feature selection measurements because they can be misleading. A reason is that the Outside class covers almost 90% of the data. Moreover, the feature vectors of the Outside class are extremely diverse even in the class itself. Consequently, we leave them to find out the features distinctive to the informative classes.

2.7 Classifier Generation

We employ Conditional Random Fields by using the Python binding of CRFSuite library (Okazaki, 2007). To apply the Structured Support Vector Machines approach we use the PyStruct Python library (Müller and Behnke, 2014). Hyper-parameter optimisation is applied to both of the classification algorithms by using Randomize Search method (Bergstra and Bengio, 2012), especially to optimize the regularization parameters known as 'C' parameters.

2.8 Ensemble of Classifiers

We observe that some of the labels missed by CRF classier are correctly found by SSVM classifier, and vice versa. Thus we decide to ensemble the two classifiers to make them correct each other. We ensemble them by combining the probabilities of the two classifiers with Equation 1, which is inspired by Bayes Theorem (Stwrt and Ord, 1994). However, PyStruct library only returns the predicted label and does not produce label confidences. Thus, we calculate the probability of the class as being correctly predicted by the SSVM algorithm in general. In contrast, CRFSuite library is able to produce a confidence per class for each sample.

We only combine the probability of the SSVM prediction with the CRF confidence of the same class. To calculate the probabilities of the rest of the classes, we applied the chance rate as the probability returned by the SSVM algorithm. Finally, as usual, we choose the class with the maximum final confidence. In sum, the equation works to tweak the CRF confidences by taking the SSVM predictions into account.

For instance, assume that the sample token is 'August', SSVM predicts that the class is going to be the time of the event, the SSVM's true-positive rate for the time class is 0.50, and CRF returns 0.80 probability for the time of the event class. Thus, the final calculated confidence for the time class is the multiplication of 0.50 with 0.80 divided by the marginal likelihood. For the rest of the classes, the equation is the same except the SSVM probability is set to random chance.

2.9 Classifier Evaluation

Beginning-Inside-Outside formatting allows us to predict a single tag for a single token. Thus, it saves us the hassle of segmenting the text beforehand in a separate process. However, in the end, we are after the complete entities that may be formed by more than one token.

Furthermore, knowing the existence of a single inside tag alone without any beginning tag does not provide the actual complete information we want to find. Likewise, missing an in-

$$P(class_{Ensemble}) = \frac{P(class_{SSVM})P(class_{CRF})}{P(class_{SSVM})P(class_{CRF})+(1-P(class_{SSVM}))(1-P(class_{CRF}))} \quad (1)$$

side tag that should normally come after a beginning tag is considered as an information loss, even if the beginning tag alone is correct. Therefore, we should perform the evaluation on the whole chunks. Thus, the predicted entities are penalized if they do not fully match with the corresponding manually labelled chunk.

To apply such an evaluation method, we use the evaluation script used in shared tasks of The Conference on Computational Natural Language Learning in 2000 (Sang, 1998). Thus, we calculate the precision, recall and f1-score per class alongside with the average scores.

2.10 Cross-validation Over Articles

In the real life version of our system, a classifier generated by using our training set would run on unseen articles one by one to detect and return the event related critical information. Therefore, to be able to test how our system would perform in a situation as close as possible to real life, we apply k-fold cross-validation (Geisser, 1993) on our training set on the article level.

To create the folds of the cross-validation, we split the articles into groups of equal numbers, namely four articles per group. Then, we leave the corresponding group of articles out as our test set. Next, we generate a classifier with training it on the rest of the articles. Finally, we predict the tags in the articles left out and save the predictions aside. After this method is applied to each fold, we obtain the predictions for the whole dataset. Hence, we can evaluate the performance of our system by comparing these predictions with the manual annotations.

2.11 Random Permutation Test

Besides the evaluation scores, we also perform a random permutation test on the classifiers to see how reliable they are. To apply the test, we rearrange the labels in the training data by randomly shuffling the label list. Thus the labels match with the wrong feature vectors. Then, we train the classifiers on the shuffled training data. Finally, we evaluate the classifiers to see how they perform with the new set. Hence the aim of the test is to see if a classifier actually learns from the given features or only coincides the correct labels (Ojala and Garriga, 2010).

2.12 Classification Results

When we compare the SSVM classifier, CRF classifier and the ensemble of them, we observe that the ensemble system performs slightly better than CRF classifier on the average f1-score as shown in Figure 2. However, based on McNemar's test (McNemar, 1947), the improvement is not significant. There are more downsides of the ensemble such as high computational requirements and time consumption. Since the ensemble system is built on training more than one classifier, it demands more computational power than training a single classifier. Furthermore, PyStruct library consumes 60 GB of memory for training. Meanwhile, the CRFsuite library requires only 2 GB of memory for using the same feature set. Consequently, training both of the classifiers at each Active Learning step would slow down the overall system drastically.

Thus, despite the slight but not significant improvement provided by the ensemble system, we decide to use only CRF algorithm in Active Learning. However, the main reason that we choose CRF classifier over SSVM classifier is that CRF classifier significantly outperforms the SSVM classifier. The significance of the comparison is indicated by McNemar's statistical test ($p < 0.05$).

Next, we test the effect of the class occurrence probabilities feature that explained in Section 2.5. Although the sentence classifier by itself does not perform well with 47% f1-score, using the class occurrence probabilities as a feature significantly improves the information extraction classifiers. To see the effect of the feature, we train two separate information extraction classifiers. One of them is generated by using the class occurrence probabilities as features, and the other one is generated in the same way except only without using the class occurrence probabilities. We observe that the classifier with the class occurrence prob-

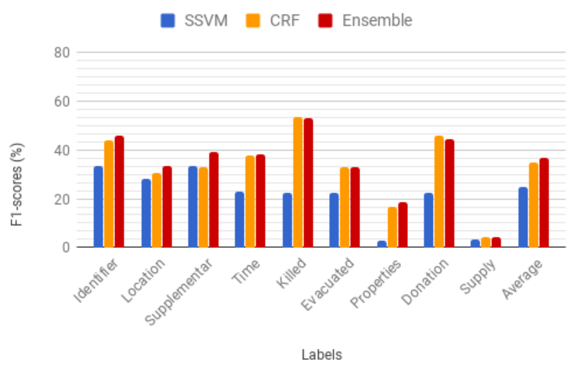


Figure 2: F1-score comparisons of the SSVM and CRF classifiers with their ensemble.

abilities obtains higher evaluation scores. Using the feature increases the both precision and recall scores by 3 to 5 percent as shown in Figure 3. McNemar's statistical test proves that the improvement obtained by using the feature is significant ($p < 0.05$).

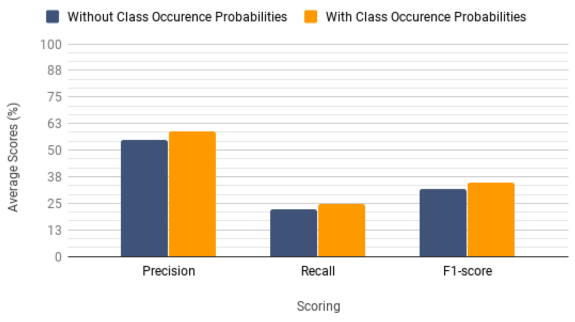


Figure 3: Comparison of precision, recall, and f1-scores of the classifiers with and without the class occurrence probabilities.

Finally, we apply the random permutation test to CRF classifier. We observe that the classifier can not correctly find more than a single label when the label list is permuted. F1-score of the classifier varies from 0.0% to 0.25%. This result means that our feature set is meaningful and directly affects the classifier performances. Thus, the classifier actually learns from the features and does not predict the correct labels by chance. With the p-value being 0.01 after 100 iterations, we have a clear indication that the results are significant ($p < 0.05$).

3 Active Learning

3.1 Methods

Active Learning is an iterative process. To initiate the system, first, we train the classifier with the methods explained throughout Section 2 on the initial data which consists of 44 news articles. By using the classifier we automatically annotate 10 more articles at each iteration. The automatically annotated articles are uploaded back to the FLAT online annotation tool to be corrected by the human annotator. However, we display only the annotations that have confidences below 0.80. In other words, the human annotators see only the uncertain labels. Next, the uncertain labels are checked, corrected, and submitted back by the human annotators. A step of the Active Learning system is completed when these new manual annotations are added into the initially annotated data. The second step begins with training a new classifier on the data enriched in the previous step. The steps of a single iteration are demonstrated in Figure 4.

Because of the imbalance between the sample sizes, if we train the classifier including the complete set of Outside class, the classifier becomes discouraged to predict informative classes. Therefore, the system ends up with only a few predictions out of 10 articles to display to human annotator. Consequently, the human annotator can not encounter enough data that is worth adding to the training data for the next step. Likewise, because of the same reason, the classifier predicts the entities closer to the decision boundary as Outside class at a high rate. Therefore, the decision boundary can not be explored at all and the goals of the system cannot be met. To overcome this issue, we undersample Outside class exemplars in the training data during the Active Learning steps. This method encourages the classifiers to predict more of the informative classes. Therefore, the human annotator is provided with more options to consider and there is a higher chance to enrich the data.

However, the Outside class contains information as well. A complete removal of the class would mean valuable information loss. Furthermore, CRF algorithm makes a prediction for a word by using the conditional dependence on the tag of the previous word meaning that the order in

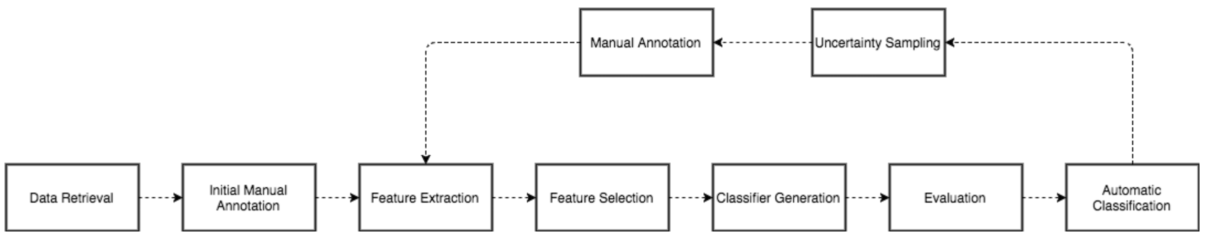


Figure 4: The data flow in the system.

a sentence should not be disrupted. Therefore, we consider sentences as a whole to be added to the training data. The classifiers are trained only on the sentences with at least one manual annotation. The sentences without any manual annotation are excluded from the training data including the initial data as well.

Furthermore, only the manual annotations or the manually confirmed automatic annotations in a sentence are accepted. Any automatic annotation without manual confirmation would be replaced by Outside class in the training data. However, including informative entities into the uninformative class set can create contradictions in the training data. For instance, assume that a sentence contains an entity that should be labelled with an informative class and the human annotator leaves it as unannotated. The entity would be considered as labelled with Outside class meaning that the classifier would learn not to label an entity that should be labelled, during the training in the next iteration. Because of the iterative nature of Active Learning, the number of contradictions can expand incrementally and reach to a point to cause confusions in the predictions after a few iterations. Therefore, we require the human annotator to manually label any information in a sentence once any manual annotation is applied to the sentence.

Finally, in a real case, the system would be iterated until it is observed that enrichment of the training data does not affect the performance of the classifier anymore, or sufficient performance scores are obtained. In this study, we iterated the system 10 times at maximum.

3.2 Experiments

We evaluate the classifier performances at each step of the Active Learning to see whether there is an improvement on the performance. To properly test the performances, we use a separate test set that consists of 15 news articles. The test set is never included in any training process.

We start with making use of the classifiers produced during Active Learning, which are trained only on the sentences with at least one manual annotation. In the first test, the classifier is run on the unfiltered test data. Thus the predictions are made on any sentence in the test data including the ones without any manual annotation. As the second test, we filtered the sentences in the test data as well. In other words, the classifiers predict labels only on the sentences with at least one manual annotation. For the third test, we train the classifiers on unfiltered training data and run them on unfiltered test data. So that any sentence is included in both training and the test processes.

Last but not least, we created a baseline human-in-the-loop system that assigns labels randomly with random confidences. However, if we assign the labels purely random, almost half of the each sentence in the test data is labelled by informative classes. This can force the human annotator to read each sentence in the data. Thus, we want to discourage the usage of the informative classes in order to save from time. To achieve that, we use the counts of the classes in the training data as the weights of labels. Since the Outside class is always the majority class in the data, using the weights helps to reduce the number of informative classes that randomly picked. Although the number of random predictions is reduced, the human annotator still has to read almost each sentence in the dataset. Thus, to make the creation of the baseline system more feasible, we decide that human annotator should check only if the predicted entities instead of the whole sentences.

Any test or situation applied to the Active Learning system is also applied to the baseline system, and vice versa. By comparing the results of these two systems, we aim to find out first whether Active Learning speeds up the annotation process or not, and second how Active Learning affects the quality of the labels.

3.3 Results

For the Active Learning and the baseline system, we observe similar graphs in Figure 5. For both of the systems, the worst scores are obtained when the classifier is trained and tested on unfiltered sentences. The best scores are obtained by the classifier both trained and tested on the filtered sentences.

If we look at the average f1-scores of the Active learning system in Figure 5, it can be easily seen that there is a clear correlation between the first and the second tests. The same correlation occurs in the results of the baseline system in their own shapes. Thus, when the classifier is trained on only the sentences with manual annotations, the addition of the same data causes the same changes regardless of the type of human-in-the-loop system or a number of unannotated sentences in the test data.

In general, for any case that we test the learning systems, we observe that the recall scores of the systems increase during the iterations while the precision scores decrease. In the end, f1-scores are not affected by this exchange between the scores, and an overall increase can be observed on the both of systems under almost any configuration that we tested.

When we compare the performances of the Active Learning and the baseline systems, we observe that they present similar behaviours under similar circumstances (Appendix A, Figure 6, Figure 7). The only exception is the situation we train and test the classifiers on unfiltered data, which produces the most unpredictable results (Appendix A, Figure 8). In any case, Active Learning system significantly outperforms the baseline system based on McNemar's test ($p < 0.05$).

Another comparison between the systems can be made on the rate of the improvements. While the Active Learning system improves the performances from 10% to 25% in 10 steps, the improvement in the baseline system is observed from 6% to 17%.

In a sense, both human-in-the-loop systems reduce the effort and the annotation time in comparison to pure manual annotation. However, the baseline system returns 12805 entities to be checked by the human annotator while Active Learning system returns 2524 entities in total. Thus, the baseline system requires 5 times more effort than the Active Learning system. Likewise, we observe that the Active Learning system speeds up the whole process by reducing the time cost to 3 minutes per article. Conversely, the baseline system takes around 20 minutes while pure annotation can take 60 minutes to annotate a single article.

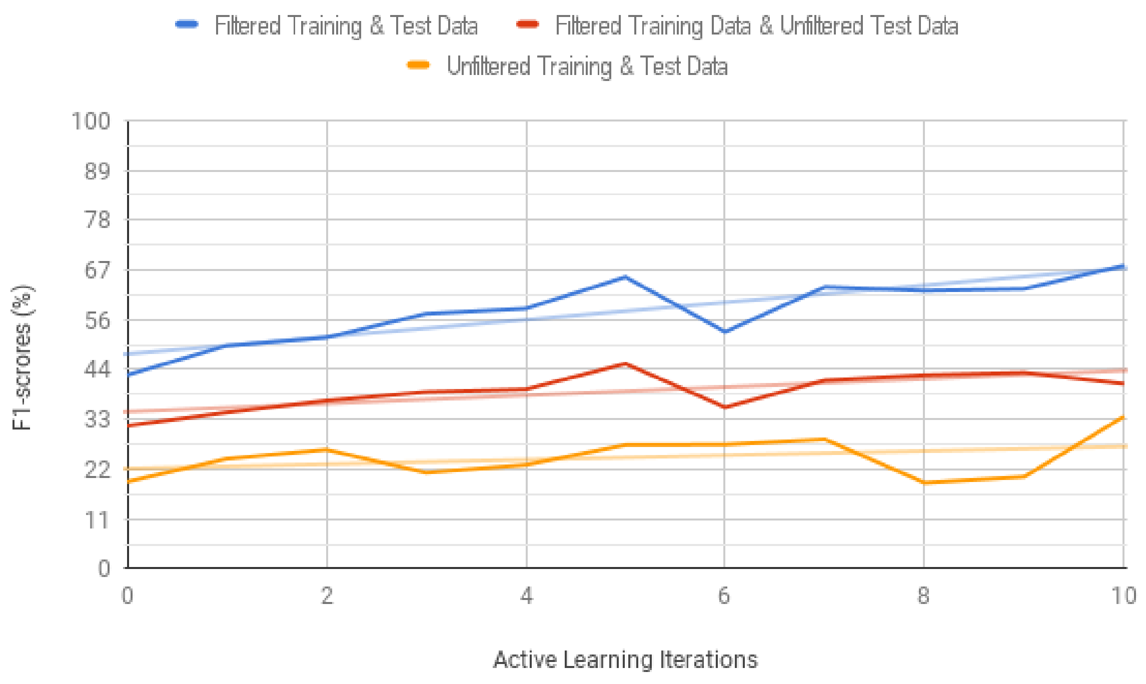
4 Conclusions

We have shown that adding data that is annotated by a human in the loop improves the performance of a sequential labelling based information extraction classification and Active Learning is a rapid and useful approach to do it. Furthermore, we introduced the idea of using occurrence probabilities of the classes as a token level feature and discussed its contribution to the classification performances.

Active Learning only shows the points that human annotators need to focus on, and thus saves time. Moreover, it learns to deal with hard decisions by presenting the most uncertain annotations to the humans. Conversely, as our baseline human-in-the-loop system represents, randomly annotating the data does not provide the same usefulness. Active Learning returns less entity to be checked by the human annotator and yet improves the system better than the baseline.

As we observed in our experiments, using the complete data to train an information extraction classifier can cause contradictions that result in high ambiguity because the majority class is always the uninformative class. Consequently, low-performance scores are observed. Thus, undersampling the uninformative class helps us to get better chances of detecting the critical information. However, different sampling methods might be explored in a further research to overcome the issues with imbalanced data better.

Although the best scores are obtained by the classifier both trained and tested on the filtered sentences, the test does not represent the real life situation because the system is not capable of finding which sentences would be manually an-



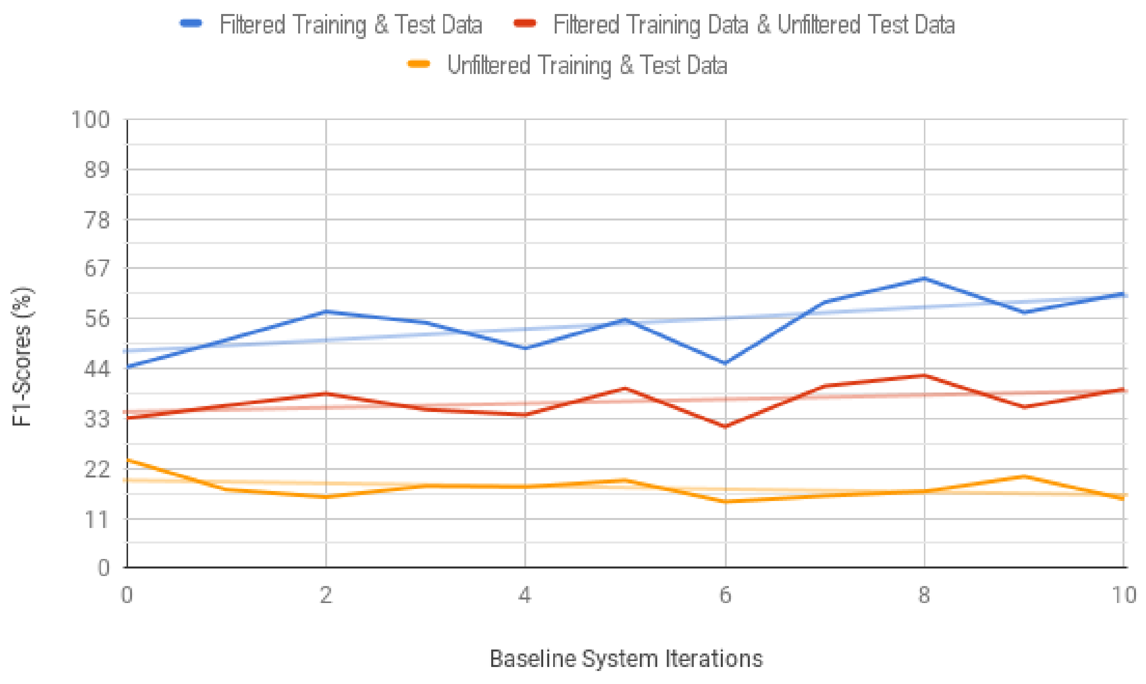


Figure 5: The progress throughout the learning steps of Active Learning and baseline human-in-the-loop system, respectively. ‘Filtered Training & Test Data’ means that the classifier both trained and tested only on the sentences with at least one manual annotation. ‘Filtered Training Data & Unfiltered Test Data’ indicates that the classifier trained only on the sentences with at least one manual annotation and tested on the data including the sentences without a manual annotation. ‘Unfiltered Training & Test Data’ means that the classifier both trained and tested including the sentences without a manual annotation. Straight lines with low opacity demonstrate the overall trend.

notated in the test data. It is only the case because we use a manually pre-annotated data for the tests. However, when the same classifier is used to extract information from unfiltered sentences, it results in a high recall and a low precision score. This is also an unwanted result because incorrect information can slip through. Therefore, in future research, we may want to study an approach to detect the sentences that are worth labelling. Thus, we can artificially filter the sentences before we run the classifier to predict.

Besides the fact that an improvement is possible with Active Learning, further improvements can be applied to the classification in further studies. For instance, by putting more work on the sentence classification, the contribution of the class occurrence probabilities can be increased. Likewise, more sentence based features can be added alongside with paragraph and article based features. Thus, the classifier can be made more aware of the structure of a news article.

References

Yasemin Altun, Ioannis Tsochantaridis, Thomas Hofmann, et al. 2003. Hidden markov support vector machines. In *ICML*, volume 3, pages 3–10.

James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.".

Jason Brownlee. 2014. An introduction to feature selection. `https://machinelearningmastery.com/an-introduction-to-feature-selection/`. Accessed August 26, 2017.

Gavin C Cawley. 2011. Baseline methods for active learning. In *Active Learning and Experimental Design@ AISTATS*, pages 47–57.

Aron Culotta and Andrew McCallum. 2004. Confidence estimation for information extraction. In *Proceedings of HLT-NAACL 2004: Short Papers*, pages 109–112. Association for Computational Linguistics.

Hal Daume. 2006. Getting started in: Sequence labeling. `https://nlpers.blogspot.nl/2006/11/getting-started-in-sequence-labeling.html`. Accessed April 4, 2017.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 363–370. Association for Computational Linguistics.

Seymour Geisser. 1993. *Predictive inference*, volume 55. CRC press.

Isabelle Guyon, Steve Gunn, Masoud Nikravesh, and Lofti A Zadeh. 2008. *Feature extraction: foundations and applications*, volume 207. Springer.

S. Sathiya Keerthi and Sellamanickam Sundararajan. 2007. Crf versus svm-struct for sequence labeling. In *Technical Report*. Technical report, Yahoo Research.

Davis E King. 2009. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10(Jul):1755–1758.

John Lafferty, Andrew McCallum, Fernando Pereira, et al. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the eighteenth international conference on machine learning, ICML*, volume 1, pages 282–289.

David D Lewis and William A Gale. 1994. A sequential algorithm for training text classifiers. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–12. Springer-Verlag New York, Inc.

Quinn McNemar. 1947. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157.

Andreas C. Müller and Sven Behnke. 2014. pystruct - learning structured prediction in python. *Journal of Machine Learning Research*, 15:2055–2060.

Nam Nguyen and Yunsong Guo. 2007. Comparisons of sequence labeling algorithms and extensions. In *Proceedings of the 24th international conference on Machine learning*, pages 681–688. ACM.

Markus Ojala and Gemma C Garriga. 2010. Permutation tests for studying classifier performance. *Journal of Machine Learning Research*, 11(Jun):1833–1863.

Naoaki Okazaki. 2007. Crfsuite: a fast implementation of conditional random fields (crfs).

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Fuchun Peng and Andrew McCallum. 2006. Information extraction from research papers using conditional random fields. *Information processing & management*, 42(4):963–979.

Leonard Richardson. 2017. Beautiful soup: We called him tortoise because he taught us. `https://www.crummy.com/software/BeautifulSoup/`. Accessed August 26, 2017.

Erik Tjong Kim Sang. 1998. Conlleval source code: version 2004-01-26. `http://www.cnts.ua.ac.be/conll2000/chunking/conlleval.txt`. Accessed May 20, 2017.

Stanford NLP Group. 2003. Stanford nlp named entity recognition results. `https://nlp.stanford.edu/projects/project-ner.shtml`. Accessed April 4, 2017.

Jannik Strötgen and Michael Gertz. 2013. Multilingual and cross-domain temporal tagging. *Language Resources and Evaluation*, 47(2):269–298.

A Stwrt and K Ord. 1994. Kendall's advanced theory of statistics.

Maarten van Gompel and Martin Reynaert. 2013. Folia: A practical xml format for linguistic annotation-a descriptive and comparative study. *Computational Linguistics in the Netherlands Journal*, 3:63–81.

World Meteorological Organization. 2017. Tropical cyclone naming. `https://public.wmo.int/en/About-us/FAQs/faqs-tropical-cyclones/tropical-cyclone-naming`. Accessed August 26, 2017.

A Supplemental Materials

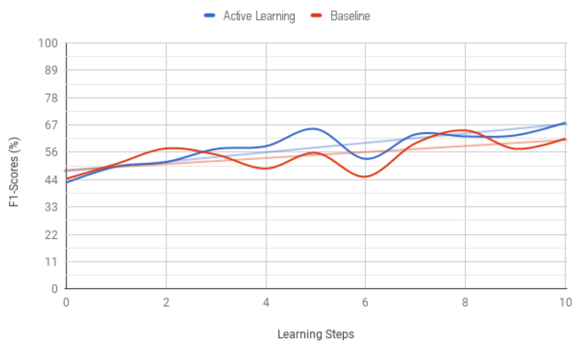


Figure 6: Comparison of Active Learning and baseline human-in-the-loop systems both trained and tested only on the sentences with at least one manual annotation.

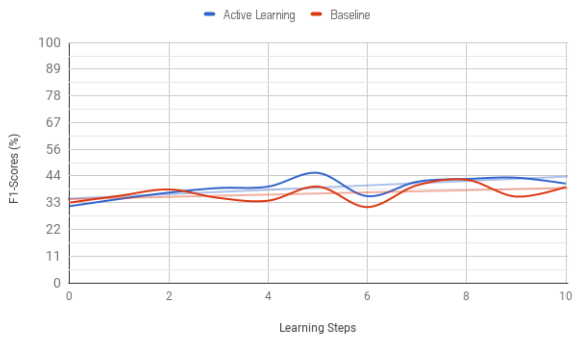


Figure 7: Comparison of Active Learning and baseline human-in-the-loop systems trained only on the sentences with at least one manual annotation and tested on the data including the sentences without any manual annotation.

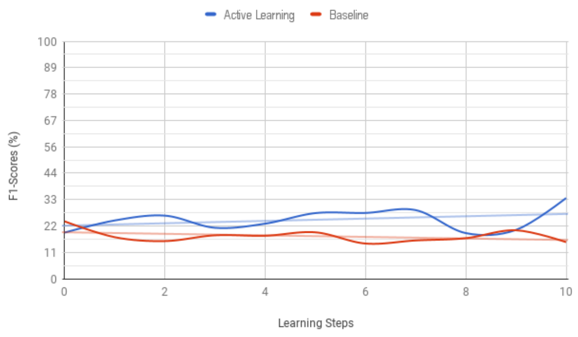


Figure 8: Comparison of Active Learning and baseline human-in-the-loop systems both trained and tested including the sentences without any manual annotation.