Information Extraction for Articles on Mass Shooting

Angel Yu, Yonglin Wu, Brian Sun

Massachusetts Institute of Technology
77 Massachusetts Avenue, Cambridge, MA 02139, USA
E–mail: {angelyu, ylwu, sunset}@mit.edu

Abstract

This project is interested in the problem of information extraction across multiple documents that are related to a specific topic. In particular, we build a system that uses a search engine to find relevant documents and extrapolates information from those documents. The topic of focus is events about mass shootings. Given an article, our task is to conclude facts about an event, specifically, the name of the shooter, the number of people killed and wounded as well as the location of the shooting. In order to predict facts from a single article, we use a Maximum-Entropy Markov Model (MEMM) to tag individual words in documents, which are news articles from the web. Using these tags, we use an averaging method over words with certain tags to predict event information. To improve accuracy, we then introduce prediction averaging over multiple articles by majority voting as well as a method of query finding to acquire relevant articles. Our code for this project is available publicly on github.[1]

1 Introduction

Automatic information extraction on news article is a broad field and has many interesting applications. In general, news articles are well structured and present information in a straightforward way, which makes it easier for a machine to extract information than a random article. The particular subset of newspaper article we are interested in are articles about mass shooting happening in United States. Our goal is to build a system that can automatically extract four entities: shooter name, number of people killed, number of people injured, and name of the city of the shooting given the title and text of the news article. Those are the four entities most of articles on mass shooting have, and they are what people care about the most. Currently there are databases online on mass shootings in U.S, which are generated manually. This system can automate the process of building the database and make the process more scalable.

The dataset we used came from `http://www.shootingtracker.com/wiki/Main_Page`, which lists about 800 shootings between 2013 and 2015 in U.S. For each shooting, it listed a few news article URLs on that event, along with the name of shooter, number of people killed, number of people injured and the location of the shooting, which we use as ground truth. We use a third party news scraping package to extract text and title from given URLs. We are able to extract 800 news articles, from which we used 500 articles to form our training set, 150 for our dev set and 150 for our test set.

[1] `https://github.mit.edu/angelyu/Information-Extraction`

2 Related Work

Information extraction on events has been explored in other contexts. Siddharth Patwardhan of IBM [1] has explored event extraction at the sentence level. More specifically, features were associated with entire sentences. The model is able to detect events occuring in sentences as well as the nature of the event (terrorist attack, disease outbreak, etc). There has also been work on join event extraction [2] where multiple triggers and arguments are extracted simultaneous to capture dependencies. This is used in conjunction with global features.

3 Information Extraction

The information extraction system we designed is made up of a few components. The entire pipeline of our system is illustrated in Figure 1. Given an article we would like extract information from, we have a predictor that extracts information from the article text. This component is described in detail in Section 3.1. However, information on articles are often imperfect and in our case, not all the information is given in one news article. Hence, it is beneficial to find articles describing the same event and combine information extracted from all the articles. We will use a search engine such as Google and Bing to aid us in finding articles describing the same event. Similar to everyday searching, the query is often key to finding good relevant results from a search engine. Section 3.2 will describe in detail how we find relevant articles. After finding the relevant articles, we will run our predictor for each of them and combine the results of the predictor into one combined result. Section 3.3 will specify how we combine them.

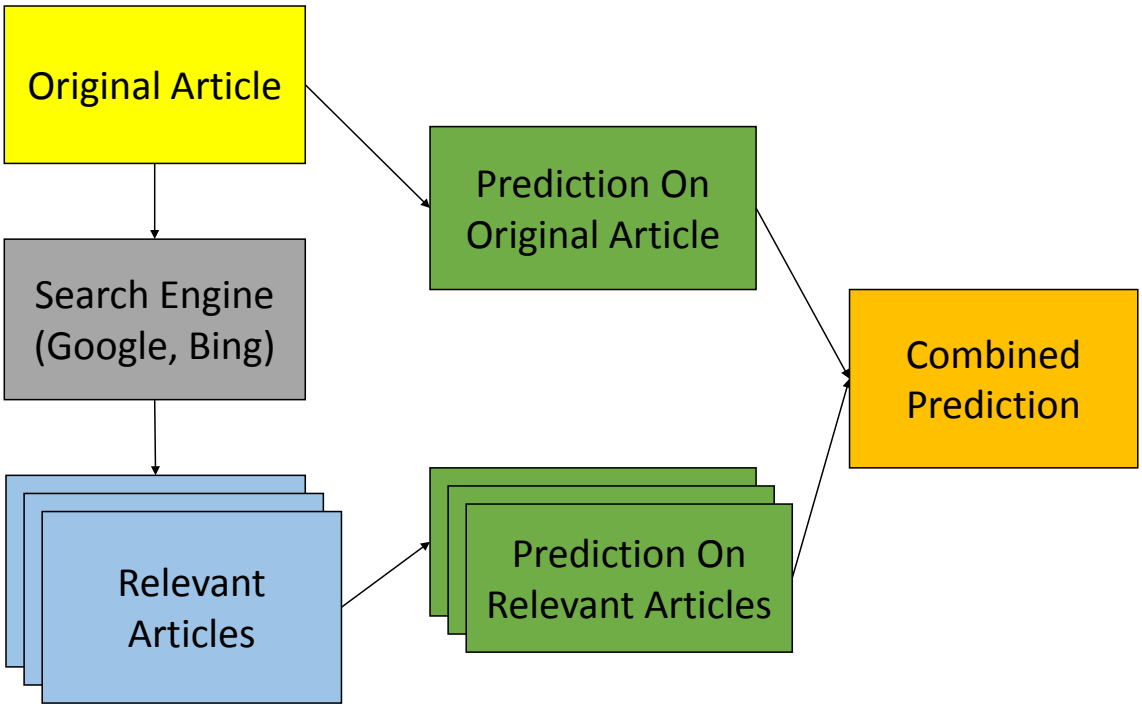

Figure 1: Overview of our Information Extraction system which makes use of a search engine to find relevant articles and aggregates predictions.

3.1 Single Article Extraction

3.1.1 Algorithm

Given an article, we first split the article into a sequence a words and we try to predict tags for each words. The tags that we will predict are: `shooterName`, `numKilled`, `numWounded`, `city` and `TAG`. Each of these tags other than `TAG` signal that the word associated with it is an entity of that tag. The tag `TAG` signals that the word associated with it is not an entity of interest. We will tag the dataset by tagging words that occur in the ground truth with the corresponding tag in order to create a supervised learning problem. So, we have now reduced the problem to a Named Entity Recognition (NER) problem.

We use a Maximum Entropy Markov Model (MEMM) to predict the tags. MEMM is an algorithm used to tag a sequence of observations $X = \{x_1, x_2, \ldots, x_n\}$ with $Y = \{y_1, y_2, \ldots, y_n\}$. In our case, x_i are words of a sentence and $y_i \in \{\texttt{shooterName}, \texttt{numKilled}, \texttt{numWounded}, \texttt{city}, \texttt{TAG}\}$. We would like to find:

$$Y^* = \text{argmax}_Y(P(Y|X))$$

To do this we will calculate the probability:

$$P(Y|X) = \prod_i P(y_i|y_1, \ldots, y_{i-1}, X)$$

And we model each term in the above product to be:

$$P(y_i|y_1, \ldots, y_{i-1}, X) \propto \exp(\theta \cdot \phi(y_1, \ldots, y_{i-1}, X))$$

where θ is a vector of weights to be trained and $\phi(y_1, \ldots, y_{i-1}, X)$ is a feature vector. The weights θ is trained using a Maximum-Entropy (MaxEnt) classifier that maximizes the likelihood of the training data. Selecting useful features in this model determines the success of the overall prediction and hence feature engineering is extremely important. We will discuss our choice of features in detail in Section 3.1.2. To predict a sequence of tags, we will just greedily predict the best tag for each observation.

Once we have obtained the tags for all the words in the article, we will aggregate all the entities for each tag. For tags `numKilled` and `numWounded`, we simply take the most frequent entity tagged. For `shooterName` and `city`, we look at entities that appear together and merge those as a single token and output the most frequent token. Because, we noticed that we do not always give a prediction for the tag `city`, we additionally had a database of the top 3000 US cities and we tried find them in article if we did not have any entities for the tag `city`. In the case of multiple cities being found in the article, we again take the most frequently occurred one.

3.1.2 Features

Generating good features is crucial to a good learning result, and it requires a combination of intuition and experiment. As a baseline, we use one hot vector of the current word as our feature. We collect the set of all words that appeared above a certain number of times, and if current word matches one of the word in the set, we set a specific position in the vector to be 1. This threshold can be learned in model selection phase.

This model should be able to learn the difference between a number tag and a name tag, but it would be hard to differentiate between killed number and wounded number, as the only thing it is possible to learn is whether current word is a number. This can be solved by adding context to the

feature, specifically, adding features on previous m words and following n words. Intuitively, just by looking at whether the following word is "killed" or "wounded", we can differentiate between `numKilled` and `numWounded` tag for current word. This is confirmed later in our training phase.

Simply using a one hot vector of the current word and nearby words can suffer from overfitting, as a word or a phase that never appeared in the training set will have vector containing all zeros as feature, and that is often the case with name of shooter. To deal with this, we include some custom created general features, such as whether a word contains digit, starts with capital letter, is a common male name, etc. In addition to the custom features that only depends on the word itself, we add two features that capture the position and co-occurrence of the words in a sentence. One such feature is whether the word is the 1^{st}, 2^{nd}, ... 10^{th} word in the paragraph. In general, if a word is a number and it appears in the first word in the paragraph, it has a much higher chance of being a `numKilled` or `numWounded` than if it appears in the middle of a paragraph. Adding this feature is an attempt to give our model the chance to learn that. The other feature is the appearance of other words in the same sentence as the word we are predicting. This feature comes from our assumption that some key words, such as "killed", "police", etc, will increase the possibility of other words having a like `numKilled` or `numWounded`.

We do a few things to prevent overfitting. We prune our word vocab so that it only includes words that have appeared in a threshold of K articles.K is a model parameter we will tune. We also limit the number of possible previous M and next words N in our model to 5 to prevent our model from growing to large. The specific parameters we use have a big impact on our feature vector length; we tune them carefully on dev set instead of picking randomly. There are a total of 4520 words in our word set when we don't prune and all, and 2475 words when we set K to 3. Our models have a maximum feature vector length of 54262, and a minimum length of 2475. However, our feature vector is very sparse as it consists of several long one-hot vectors. Our training set has 147098 words, so we are in danger of over-fitting if we further increase our feature vector length.

3.2 Finding Relevant Articles

3.2.1 Constructing Queries

Our first task in leveraging predictions on multiple articles is to use a suitable search query to find the articles. We choose to hand generate a large set of potential queries. For each of these queries we test their performance in their ability to retrieve a predetermined set of most relevant articles when run on a search engine. The queries are mainly permutations of article attributes. In large part, we use the core event attributes: "shooter name", "killed number", "wounded number", and "location (city)". Some examples taken from the set of tested queries are:

- "Shooting in [location]"
- "Shooting in [location] on [date]"
- "Shooting in [location] on [date] by [shooter name], [killed num] killed"
- "1 officer killed, 2 wounded near Louisiana casino"
- "Charenton, LA 1 officer killed, 2 wounded near Louisiana casino"
- "1/27/2013 1 officer killed, 2 wounded near Louisiana casino"

The first three queries shown are examples of templates where we construct arbitrary sentences that include event attributes. In this case, we have location, date, as well as names and casualty numbers. We also test queries as shown in the following three where we use title as well as title and some other attribute appended. With these query templates, we have to find the one that performed the best. Our approach is to use these query templates on a every event in a set of events. Whatever query performs the best over these events we select that query as the best query and subsequently use that template for the system. We choose to look at events from 2013 with an event set of about 364 events. Our metric for performance is to look at how many relevant articles a query returns from a search engine. We use two methods as a measure for relevance. The first is to look at how many articles in the search result intersect with the given set of relevant articles, or the original articles, for an event. The second method is to look at the percentage of search results that belong in the given set of relevant articles. For each method, we sum up the scores for each query template. The template which gives the best score will be the chosen template for the system.

3.2.2 Ranking Articles

With the query template chosen and fixed, the remaining task is to choose the best articles that are returned by the search engine. To do this, we look at the cosine similarity of the tf-idf values of the articles. Articles that have a similarity score of over a certain threshold are included in our selected relevant subset of articles. We introduce another metric which is article date. We make sure to only include articles that are within a month of the event date so as to not incorporate other events with the same location. The reason is that often times multiple shooting events occur within the same location over a long period of time. Selecting a short time period of validity allows us to filter out in part some of these overlaps.

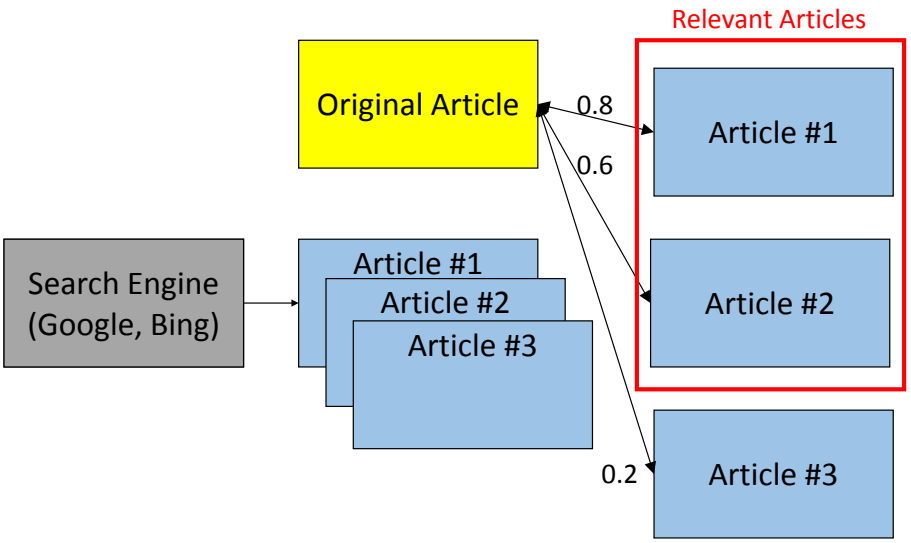

Figure 2: Overview of how relevant articles and retrieved and filtered.

3.3 Combining Predictions

Once we have all the relevant articles, we will run our single article predictor described in Section 3.1 on all of the articles we found. We then use a majority voting system to combine the predictions we made with more weight on the original article.

4 Results & Analysis

4.1 Evaluation Metrics

We evaluate our predictions by comparing to the ground truth given in the data set. For the tag `shooterName`, we use precision, recall and f1 score as our metric. For the tags `numKilled`, `numWounded` and `city`, we use accuracy as our metric. This is because there are sometimes multiple shooters but the number of people killed and wounded along with the city of the shooting has only one correct entity. We train our model on the training set, tune our parameters using the dev set and finally evaluate our model using the test set. When tuning parameters in our single article predictor described in Section 3.1, we instead use precision, recall and f1 on the entity level.

4.2 Single Article Extraction

Single article extraction is a two-step process. We first predict the tags for each word, and aggregate all entities for each tag to give prediction for a single article. The assumption we make is that better accuracy on tag prediction for each word will lead to better accuracy of entity accuracy for the entire article, and the experiment result confirms that.

Features	`shooterName`	`numKilled`	`numWounded`	`city`
One-hot vector only (baseline)	0.03	0.26	0.59	0.42
Above + prev M + next N	0.24	0.59	0.67	0.44
Above + custom features	0.27	0.56	0.67	0.62
Above + tag of previous M words	0.35	0.52	0.66	0.63

Table 1: F1 score of word level tag accuracy

Table 1 shows the result of word level F1 score for each tag each time when we add more features to our model. Our baseline is the model where we use only one-hot vector of words. As shown in the table, each time we add more features to our model, there are significant improvements in at least one type of tag. In our final model that included all features listed in the table, there are four different parameters we tuned: regularization parameter of our MaxEnt classifier (C), number of previous words we include (M), number of next words we include (N), and the threshold frequency (K) of words in our word dictionary for one-hot vector. The best parameters turn out to be $C = 10$, $M = 5$, $N = 4$ and $K = 3$, which means previous 5 words and next 4 words are predicative of tag of current word, and if a word appear in at least 3 articles, we should consider it common enough to use as a feature.

Out of the four tags we are trying to predict, `numWounded` is the easiest one to predict, for which the baseline (0.59) is almost as good as final model (0.66). However, `numKilled`, which is also a number, proves to be much harder to predict and requires more feature to get right. After looking at the data set, we think the difference comes from the nature of shooting events. A shooting event always has `numWounded`, but not necessary the `numKilled`. (We ignored the case where the number of people killed is 0, which we are not able to learn using our current model) Hense more `numWounded` appear

in the article more often than `numKilled`. Since the training size is bigger, it is easier to obtain a good model.

For tag `city`, we see the biggest improvement comes from adding custom general features to the model. By inspecting the features with most weight in our MaxEnt classifier, we can see the classifier tries to learn the city names that appear in multiple articles, such as "Dallas", "Detroit","New". Although this works to a certain extent, it fails to predict smaller cities that appear less often in our training set. Adding custom general features helps the classifier to better generalize. Specifically , two features, "Does the word start with capital letter" and "Is the word the name of a city" (from our pre-populated database of 3000 most populous US cities) are the most predicative of the "city" tag among the custom features.

Tag `shooter name` is the hardest to predict for several reasons. Many of the articles in our training set do not have a shooter name, which give us a smaller sample size to train comparing to other entities. Also as opposed to predicting an single number, or a single city, it predicts a list of shooter names, which by itself is a harder prediction problem. In addition to that, inherently there are more ambiguity to shooter names. The names appear in an article can be name of victims, name of police officers, or even name of the reporter, which raises significant challenges for our predictor.

4.3 Finding Relevant Articles

Our results show that article title as the search query template performances best against our test set. We ran our query templates against 364 events from 2013 from a given corpus. We also have two metrics of performance, number of original articles found in the search results and percentage of relevant search queries with respect to the original articles. With respect to the first metric, we see a margin of approximately 9% for title alone when compared against all other query templates. This means that on average the title query is able to produce 9% more articles in the original article set when compared to other search query templates. With respect to the second metric, we see a margin of approximately 11%, again with title compared against all other query templates. Thus, for the purposes of this system, we choose to use article title alone as the query template.

To filter the search results and extract a subset of articles, we need to select a relevancy score threshold. In practice, we choose 0.42 as the threshold. This gives us a large enough subset of articles to use during prediction while also being sufficiently high enough in relevancy for good prediction performance.

4.4 Combining Predictions

	Baseline			Single Article Predictor			Combined with Relevant Articles		
Tag	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
`shooterName`	0.17	0.04	0.06	0.57	0.31	0.40	0.59	0.45	0.51
	Accuracy			Accuracy			Accuracy		
`numKilled`	0.30			0.74			0.81		
`numWounded`	0.54			0.70			0.75		
`city`	0.37			0.68			0.71		

Table 2: Article level extraction result

Table 2 shows the side-by-side comparison of the accuracy on article level for our baseline, single article predictor and our final predictor that combines result from query. For example, 0.71 in last row of last column means that given a random article in our test set, there is a 71 percent chance

that our best predictor will predict the city name correctly. We can see from the table that by combining results from relevant articles, we improve prediction accuracy across all tags, especially with `shooter name`, which is hardest to predict.

5 Further Work

There are still many areas to explore in future work. Our experiments show that shooter name is difficult to predict due to ambiguity. Names in the article could refer to many individuals such as a victim, a police officer, or witnesses. It was more difficult to differentiate the identity of an agent in this context. Another problem was that it was often the case that shooter name was not mentioned. Thus, it may be useful in the future to acquire a larger set of training data to improve results for shooter name.

Another possibility is to explore the performance among different types of documents. In this paper, for example, we did not differentiate a news article from a blog post. We extracted information automatically from a search result URL. It may be useful to explore how different document types perform under different prediction models.

6 Conclusion

In this paper, we implement a system that uses a Maximum Entropy Markov Model (MEMM) to tag the words in an article about mass shootings. Given these tags, we extract the attributes of the event that the article described. In particular, the attributes are shooter name, number of people killed, number of people wounded, and location. We extend this by running prediction over a larger set of articles. We find a search query template that returns the most relevant articles and then use tf-idf cosine similarity to find a subset of articles with the highest relevancy scores. Running prediction on all relevant articles, we use majority voting to find a consensus prediction among them. This method of leveraging the results from multiple articles leads to significant prediction improvements in practice.

Acknowledgments. The authors would like to thank Professor Regina Barzilay and Professor Tommi Jaakkola for offering 6.864, through which the authors were exposed to the fascinating subject of natural language processing as well as for offering valuable feedback throughout the project, which we incorporated into many of our decisions.

References

[1] Patwardhan, Siddharth. (2010). “Widening the Field of View of Information Extraction Through Sentential Event Recognition”. (Ph.D). The University of Utah.

[2] Li, Qi, Heng Ji, and Liang Huang. “Joint Event Extraction via Structured Prediction with Global Features.” ACL (1). 2013.

[3] Berger, A.L. and Pietra, V.J.D. and Pietra, S.A.D. “A maximum entropy approach to natural language processing”. Computational Linguistics (MIT Press). 1996.

[4] McCallum, Andrew; Freitag, Dayne; Pereira, Fernando. “Maximum Entropy Markov Models for Information Extraction and Segmentation.” Proc. ICML 2000