ABSTRACT
We built a real-time news classifier which will collect news from some U.S. news websites and classify the collected news by our machine learning news topic classification. We crawled news from the websites of New York Times, Fox News and NBC News and use the articles crawled from the New York Times to train our topic classifier. We also build a web interface from which users can access the classified news articles at real-time. In the report, we will introduce the architecture of our system and the machine learning methods that we use. We also provide our testing evaluation and analysis for our system.

Keywords
Python, Tornado Web Server, HTTP, HTML, Real-time, Distributed System, Machine Learning, Naive Bayes Classifier, Support Vector Machines

1. INTRODUCTION
Documents classification is one of the critical problems in natural language processing and information retrieval fields. News articles provide great examples for such task as the semantic content of news articles is accurate, concise and coherent. Besides, most news groups publish their news articles on their websites freely so that we can obtain these news reports by using a web crawler.

Automatic news classifier is a hot topic in the media industry. People require more and more conciseness and timeliness for getting news and that can be solved perfectly by machine intelligence. On-line crawlers can rapidly get sources and classifier will automatically tag those news and aggregate them to users. Also, for social science researchers, such textual analysis and modeling will help them better understand the news industry, and thus uncover patterns and biases behind each news. Based on above background information, we choose to implement a real-time news classifier. We support features such as distributed documents storage and retrieval, automatic classification and real-time news updates.

In this paper, we first introduce our high-level system architecture, then dive into details regarding our machine learning modules and our testing results based on the data set that we have crawled by ourselves. At last, we discuss some limitations and possible future improvements about our project.

2. ARCHITECTURE
Our system consists of news crawlers, a news topic classifier and a web interface, as shown on figure 1.

2.1 News Crawler
We crawled news from the websites of the New York Times [5], NBC News [3] and Fox News [1]. Each crawler will be launched every 10 minutes, 20 minutes, 30 minutes respectively according to news articles update frequency on their websites. Up the to project due day, we crawled about 2000 articles from the New York Times, 500 from NBC News and 200 from Fox News. The crawled news articles are encoded in JSON format with the news source, URL, published time and body text attributes, and are stored on disk as plain text.

We use the New York Times Newswire API [6] to fetch the
metadata for each document we crawled from the New York Times. The metadata from this API contains a “section” attribute for each document so that we can use this to label the crawled New York Times articles as our classifier training set. As shown in Table 1, The New York Times originally classify their articles into 20 kinds of category, but we re-organize them into seven topics as our machine learning labels to have a better generality and have higher classification accuracy.

<table>
<thead>
<tr>
<th>Class</th>
<th>NYT Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>us</td>
<td>U.S.</td>
</tr>
<tr>
<td>world</td>
<td>World</td>
</tr>
<tr>
<td>opinion</td>
<td>Opinion</td>
</tr>
<tr>
<td>business</td>
<td>Job Market, Your Money, Business Day, Real estate</td>
</tr>
<tr>
<td>tech</td>
<td>Science, Technology</td>
</tr>
<tr>
<td>entertain</td>
<td>Food, Travel, Magazine, Watching, Style, Movies</td>
</tr>
<tr>
<td>sport</td>
<td>Well, Sports, Automobiles</td>
</tr>
</tbody>
</table>

2.2 Topic Classifier

The topic classifier will be launched every one minute to examine whether there are new incoming documents crawled by crawlers. If there are new documents, the classifier will use our pre-trained model to classify the articles into one of the seven classes (Table 1) and include them into our section lists and document corpus. Our section list and document corpus are persisted by using the Python Pickle package. Once the persistence is done, the topic classifier will also inform our Section Server and Document Server to use the latest persisted data so that the latest articles are retrieved and the news classification can be served at real-time.

Details of our pre-trained classification model will be illustrated in the next section.

2.3 Web Application

The web application allows users to access the real-time classified news articles. Our web application consists of three parts: the front-end server, the section servers, and the document servers. All the communications in between the web application components are conducted by the HTTP protocol [2] by using the Python Tornado Web Server package [7].

2.3.1 Front-end Server

Front-end server receives users’ HTTP requests and renders the result HTMLs to users. When the front-end server receives a section HTTP request from the user, it will ask the section servers for the document list corresponding to that given section. The section list is sorted by the time when an article is published, and then the article information such as title, HTTP, news source and text snippet are further retrieved from the documents servers. The front-end server in our system is centralized.

2.3.2 Section Server

Section Server is serving for retrieving the document list corresponding to a given section string. Our section server is distributed: documents are stored among the section servers according to the hash value of their URL string so that we can add more section servers to our system if a section is over-loading.

2.3.3 Document Server

Document Server is serving for retrieving the document detail information such as document title, URL, news source and text snippet according to a given document id. Our document server is distributed: documents are stored among the document servers according to the hash value of their URL string so that we can add more document servers to our system if a document is over-loading.

3. PRE-TRAINED CLASSIFIER

3.1 Data set

Since the widely used the New York Times annotated corpus[9] is currently not publicly accessible, we decide to use news data crawled by ourselves from the New York Times website. The New York Times provides the public a Newswire API [6] from which we can fetch metadata from the documents that we crawled from the New York Times and label them with section tag. Up to the date the project is done we have crawled about 2000 articles from the New York Times as our training set. Though it is enormously small for a machine learning task, we still get a fairly reasonable result at the end, especially with categories such as sport, entertainment and technology. And after all the classifier gives satisfactory result in our real time news application. An example of our running application with sport topic is shown in figure 2.

The data set distribution is presented in Table 2.

![Real time sports news](image)

3.2 Feature selection

We choose unigram and bigram bag-of-words as features. For each class, we choose the top 100 words with highest
Table 2: Crawled NYT data set

<table>
<thead>
<tr>
<th></th>
<th>us</th>
<th>world</th>
<th>opinion</th>
<th>business</th>
<th>tech</th>
<th>entertain</th>
<th>sport</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>254</td>
<td>306</td>
<td>287</td>
<td>276</td>
<td>153</td>
<td>418</td>
<td>273</td>
</tr>
</tbody>
</table>

Figure 3: Top 30 tf-idf words in class Sport

tf-idf values and combine all these words into out feature set. The whole feature set size we are using this time is 512. Examples of words with highest tf-idf values in sport and world is shown in figure 3 and figure 4. We can find that we successfully capture important keywords that can help differentiate multiple classes such as league and north korea.

We only use word-level features this time and we ignore other important information such as part of speech, chunk and sentence dependency. We will try to add more features as our continuously run crawlers get us more labeled data.

3.3 Learning Model

In document classification field, naive Bayes and support vector machine models are often used as baselines for other more complex methods. However, despite their relatively simple theory, they provide robust performance in document classification tasks[8]. For this project, we choose the multinomial naive Bayes model and linear kernel svm model as a comparison. Since our features are bag-of-words that we selected from each class, we make the assumption that out data are multi-nomially distributed.

In this section, we first examine the theoretical parts of these two models, then we give our testing results and error rate details regarding each class. Finally, we summarize three limitations of our classification model.

3.3.1 Multi-nomial Naive Bayes

Multi-nomial naive Bayes model is trained with parameterized vectors $\theta_y = (\theta_{y1}, ..., \theta_{yn})$ for each class $y$, where $n$ is the number of feature and $\theta_{yi}$ is the probability $P(x_i|y)$ of feature $i$ appearing in a sample belonging to class $y$. The parameters are estimated by a smoothed version of maximum likelihood:

$$\tilde{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

where $N_{yi} = \sum_{x \in T} x_i$ is the number of times feature $i$ appears in a sample of class $y$ in the training set $T$, and $N_y = \sum_{i=1}^{T} N_{yi}$ is the total count of all features in class $y$. $\alpha$ is smoothing priors.

3.3.2 Linear SVM

For SVM with linear kernel, we simply get the prediction through a binary linear classifier:

$$y^{(k)} = \text{sign}(w^T x^k + b)$$

And during training, $w, b$ are obtained by minimizing

$$w^T w + C \sum_i \max(0, 1 - y^{(i)} (w^T \tilde{f}^{(i)} + b))^2$$

Since we have more than two classes, we choose the strategy known as one versus all. For each classifier, the class is fitted against all the other classes. It selects the class with the highest aggregate classification confidence by summing over the pair-wise classification confidence scores computed by the underlying binary classifier.

For multi-nomial naive Bayes and linear SVM implementation, we choose the python library scikit-learn[4], where we feed our encoded training data and save the prediction model. All the training work has been finished offline and we implement the pre-trained prediction model in our system.

4. EVALUATION AND ANALYSIS

We test our model on the separated testing data set. We first test the error rate performed by these two methods. Then, we compare the error rate in each class that we classify.
Finally, we investigate limitations of our classification model and propose our ideas which we will improve in the future.

The overall error rates tested on multi-nomial naive Bayes and linear SVM are shown in table 3 and table 4.

<table>
<thead>
<tr>
<th>Table 3: Overall training error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-nomial naive Bayes</td>
</tr>
<tr>
<td>0.1281</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4: Overall testing error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-nomial naive Bayes</td>
</tr>
<tr>
<td>0.1947</td>
</tr>
</tbody>
</table>

We can find these two gives very similar performances in our data set. Our model considerably underperformed comparing with state-of-art document classification results trained on much larger data set. However, with merely around 2000 data, it produces fairly reasonable results. The classifier performs fairly well in our real-time application.

We then dive deep into our classification model and find its performance in each class. Table 5 is the results of multinomial naive Bayes model separated by each class.

From analysis of error rate in each class, we can find that our classifier performs well in classes such as tech, entertain and sport. These three classes all have obviously discernible word features and thus have good results. The highest error rate occurs in opinion class, which is hard to classify only on word level. News articles in opinion class can have several topics which make them easily matched with other classes. Example of highlighted features we are using from class opinion is shown is Figure 4. We can find that top scored words such as women, tax and trump can be easily overlapped with other class feature words, say class U.S.. Therefore, for improvement in class like opinion, we need to examine more features beyond word level, which definitely requires much more data for us to investigate.

Apart from error analysis with learning model and class features, we also find that our model has following limitations.

- Since our data are crawled from the most updated new york times news feed, features of our model are consist of many prevailing words that are only active recently. For example, we can find that the words trump and macron have relatively high tf-idf scores in world class. Such words might not be that important in news occurred several years ago. So for a robust news topic classifier, we should capture more general words that are important in a specific category. To solve this problem, we will keep getting news feed and try to get some outdated news as well. A large and distributed diversely data set is imperative for classifying news.

- We ignore features beyond word level. This can explain our bad result in class opinion as mentioned above. Intuitively, to improve such performance, we might include features that can capture more complex semantic meanings such as part of speech tags, chunk tags and sentence information like sentence dependency tree. Due to limited experiment time for this project, we only encode our texts with bag-of-words rather than traditional natural language processing pipeline. We will investigate more features and try to add such pipeline into our encoding process in the future.

- Though the two models we choose are widely used in documents classification and perform fine in our system, they still lack the ability to learn more complex feature such as long-term dependency in a document. For example, if we want to improve our accuracy in opinion class, we might need to investigate long-term dependency in the content to differ between an opinion article and a regular news article. RNN models such as LSTM are used to learn long-term dependency in a document. We might further investigate other models in the future.

<table>
<thead>
<tr>
<th>Table 5: Testing error rate by class</th>
</tr>
</thead>
<tbody>
<tr>
<td>us</td>
</tr>
<tr>
<td>0.2588</td>
</tr>
</tbody>
</table>

5. FUTURE WORKS

Though our application gives fairly satisfactory result in this time, we still have more improvements to do in the future.

Firstly, due to the limited time for our crawlers to produce labeled data this time, we definitely need much more samples to make our classification model more accurate and robust.

Secondly, we should deal with issue how a file can be shared efficiently crossing different machines. Currently our Classifier, Section Server and Document Server access the same
Python persisted serialization files on the same disk since currently our Classifier, Section Server and Document Server are launched on the same machine. We should think about how we can share the serialization files once they are really distributed among different machines.

Thirdly, we will import more searching and ranking features in our application. This time we only support search by single topic and all retrieved news are ranked by their updating time. We will keep adding more searching options, such as personalized filters, multiple keywords with topics, etc.

In addition, we still need to think of a good way to “score” our retrieved news, in other words, to deliver more “valuable” news feed to users. In this time, we just rank our news by their published time and that is apparently naive in a real-time application. We also have tried to use the confidence score, which is the log probability computed by the learning model, to rank the news and it turns out making no sense. Ideally we should have a mathematical ranking model that considers some traits to decide which news are more valuable to users. We will investigate more in these areas since it is the value of news that really matters.

6. ACKNOWLEDGEMENT
We would like to thank Professor Doherty for his brilliant class. Most of our project ideas and infrastructures are inspired by class assignments which are challenging as well as fun. We also want to thank all the colleagues for sharing each others’ project ideas.

References