a) How to run the code

It uses Python 3 and requires some packages (looking at extra sources for details)
The data stream can either from file or other api stream.
Run the main.py (right now the data comes from the training_tweets I used for training maxEnt model)
To change it with twitter stream, just replace the “file_stream” with “twitter_stream” to get data from twitter api.
We can change the threshold and period before running the program.

b) My approach to solve the problem

Information collection

First of all, I gather information from social media. Take Twitter here as an example, by using its Streaming API, I can get real time data. Given that, I filtered out all tweets those don’t have keyword “Japanese”. Consider we only get English tweets and most of them are not generated in Japan, those tweets will always mention “Japanese” if a famous Japanese passed away. If we want to, in the future, we can analysis those. I didn’t use keywords such as “died, passed away, RIP” directly to filter the tweet since there are a lot more forms to talk about death with or without those expressions.

Tweets classification

Now we have tweets related to Japanese. I want to classify them into 2 categories, tweets about death notification and ones not. I used historic tweets and trained a classifier.

I have chosen MaxEnt to do this classification task. Given more time, I would try linear SVM as well to compare the accuracy.

For training data, I used twitter api to extract tweets about five people who recently died (these tweets most likely will be related to their death) and label them as “related”: Seijun Suzuki, Jiro Taniguchi, Masaya Nakamura, Alan Cdolmes and Brenda Buttner.
Twitter REST API only provides data from the previous 7 days, so I couldn’t get tweets before they died (which would not be about their deaths). Therefore, I just extracted random tweets labeled as “notRelated”. The training tweets can be found under maxEnt_model/training_tweets.txt. I split some of the training data to be the test data in order to test the accuracy (maxEnt_model/test.txt). After using clean_data.py to convert the data into feature vector forms, I used mallet to train the model. From the maxEnt_model/me.stdout we can see the accuracy is really high:
train accuracy mean = 0.9947950553025374

test accuracy mean = 0.9430051813471503
But in reality, the model is not good enough because:
1) The training data is too small. I only included tweets about five people's deaths which indicates that their names has big weights as features. To make the model better, we should include more people who already passed away and get their death-related tweets. On the other hand, due to the restriction of Twitter REST, I couldn't access tweets more than a week ago, if I could, it will also improve the accuracy by using tweets about those people before their deaths.
2) Not all the tweets are death-related. I assume that all tweets published right after people died would be about their deaths, But there can obviously be exceptions. In reality, if it is possible, we may need to manually check each tweet to make sure it is death related.
3) Among the features I used in the model, there are a lot of function words such as "a, the, of…" that are not strong features for classification (Same thing with Japanese functioning characters such as に、は、を). The accuracy can be improved if I remove those word as features.

Name extraction

Now that I have tweets about death, Japanese syllables are used to check if a name is Japanese or not. Since Japanese names can only consist of certain syllables such as "a, ka, ga...", I can check if a name only consist of these combination, if yes, it should be a Japanese name (in Roman letter form).

Output result

The program keeps track of how many tweets mention the death of a person. It will be considered as genuine famous person dead if it go above the threshold within a certain period. And the result will be written to a file. Also, I keep a passed list to store people who already are reported dead to prevent the same person to be reported more than once.

Other questions

How would you define and quantify 'famous'?

To me, 'famous' means popular. Whenever popularity is mentioned, first thing comes to my mind is social media. One person is 'famous' if he / she has influence and people mention him or her a lot. In this project, if a person's death is mentioned for many times(threshold) within a period, I define him as 'famous'. The project could be extended to, say, when you get the name of of a dead person, you adopt some algorithms like Pagerank to see how many times the name is mentioned in other places, like some Japan's local media. There is another way too. Usually the age or DOB will be mentioned in the notification, so that we can use Wikipedia API to see if a page is created for that person, by this way, it eliminates people of the same name. In the

project, currently a person is ‘famous’ if his / her death has been published at least 10 times in a day period(values selected to demonstrate code with sample data).

What are the (online and publicly accessible) sources the tool is using?

python-twitter: https://github.com/bear/python-twitter (A wrapper of Twitter’s API for Python. It supports Twitter Streaming API)
Install: `pip install python-twitter`
Nameparser: https://pypi.python.org/pypi/nameparser (A python package to extract name entity from a sentence)
Install: `pip install nameparser-0.5.1-py2.py3-none-any.whl`
Mallet: http://mallet.cs.umass.edu/ (a Java-based package for statistical natural language processing)

For Japanese text:
Kuromoji morphological analyzer: http://www.atilika.org/ (a Java-based package for Japanese tokenization, POS tagging, word segmentation)

How does your tool identify its target strings?

Use “Japanese” as keyword to filter tweet stream.

Use MaxEnt to classify the tweet is death-related or not.

Use nameParser to extract the name from the tweet.

Japanese Text:
After finishing the English pipeline, I started to look into good Japanese tokenizers that can analyze Japanese text directly since people who are famous in Japan are not necessary famous in other countries, or local news will not report them in English. Ideally, a japanese counterpart is even more useful since you definitely get more tweets in Japanese, so the result will be more reliable. But the idea is similar to the English pipeline.

I have made a JapaneseNameExtractor class with Kuromoji morphological analyzer java package and call it from Python by executing a command and processing the output. You can test the name extractor with a japanese sentence and it will return a list of Japanese names extracted from the sentence. For example, given a Japanese sentence: “あの歌手の名前は宇多田光です”(That singer’s name is Utada Hikaru), [‘宇多田光’] (Utada Hikaru) will be returned.

Using the JapaneseNameExtractor, it can extract names from Japanese text. In the next step, I can integrate this into the DeathDetector with a model in Japanese and detect deaths with

Japanese tweets (Just change the language setting in api to Japanese). Regarding to the model in Japanese, maxEnt should be a good choice too since the order of Japanese words in a sentence is not very important.

How do you measure your tool's performance?

With a the training tweets as a test, I can extract all three Japanese people who passed away. (The default setting of main.py)

To test the real time stream, we can use the twitter stream and let it run for some time and check if the names in the list are real deaths or not. I ran it myself it gives a lot of false positive due to the inaccuracy of model, so there's definitely room for improvement here.

If we have a similar tool that's based on other social media, we could compare results of them. Also, we could manually watch the news at the same time to see if there is any missing death or misclassified death to calculate the recall and precision.

How do you present your findings?

Due to time limit, for now the program only writes the names to an output file.
There are some other potential improvements as well. One is storing data on an online database, say, Amazon DynamoDB. This not only prevents data loss from disk crashes, also, one single running program can be flooded if there are lots of tweets, an online database can coordinate many running program for better throughput.
The other thing is, besides writing to a file, we can also shoot user email to make sure timeliness.