

CS 337: Tips + Tricks for Project 1

Simon Benigeri

simon.benigeri@northwestern.edu

Credit to past TAs Chris Coleman and Victor Bursztyn – some examples/ideas taken from their versions of this deck!

Outline

1. Framing the tasks
 - a. How-to: searching strings → candidate answers
 - b. How-to: syntactic parsing and entity recognition
2. Pre-processing the dataset
 - a. How-to's: cleaning text in individual tweets
 - b. How-to: language detection
3. Tweet-specific features
 - a. Hashtags and usernames
 - b. Re-tweets and quote tweets
 - c. Tweet timestamps

Framing: evaluation setting

Core challenge of Project 1: how can we build a robust information extraction system that can generalize to unseen, real-world data?

- You'll build-test-iterate using tweets from the 2013 Golden Globes
- But, you'll be **graded on a hidden test set of tweets from other award shows**
 - Could be other years of Golden Globes, other entertainment-related award shows
 - Structure and general domain of data won't change, though

Task components: faithfully reconstruct information about these categories:

host(s); award names;

presenters for each award; nominees for each award; and winner(s) of each award

Framing: challenges across project tasks

- **Dataset noise**

- Typos, internet slang and syntax, tweets in different languages, diverse formats for quote tweets/retweets, ...

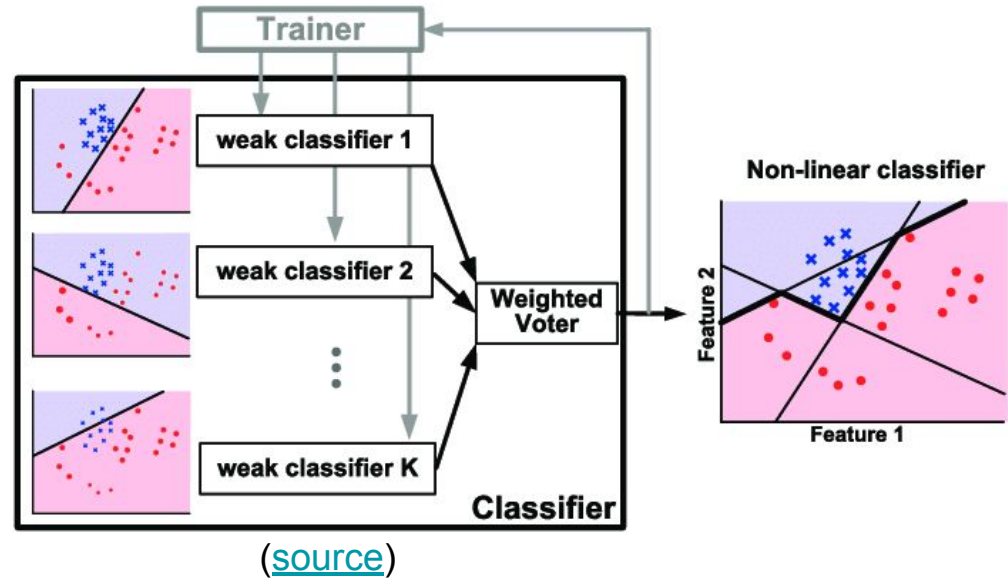
- **Task category complexities** – e.g., for award names:

- A single “real” award is often referred to with various shortened forms
- Less interest in some awards results in disproportionately fewer tweets
- Popularity of “unofficial awards” (e.g. best dressed) makes ^ problematic

- **Evaluation setting:** useful features in training data have no guarantee of being useful in evaluation data

Framing: ensembling weak rules → strong parser

1. Curate a diverse set of weak parsing rules: each one should extract *decent* predictions related to one of the award categories.
2. Using the sets of weak predictions, merge related answers and return a re-scored set of candidate answers.



Example: weak rules for award names

- One approach – searching for specific words/expressions:
 - Award names: might include the words “best” or “award”
 - Use the “script” of an award show to brainstorm:
 - Pre-winner: “nominees for X”, “is nominated for X”, “I bet X goes to”, ...
 - Reporting winner: “Y wins X”, “X goes to Y”, “Y takes home X”, ...
 - Post-winner: “Y celebrates X win”, “X should have gone to”, ...
 - ...

Example: matches for “[entity] wins [award]”

*RT @HuffingtonPost: Anne Hathaway wins best supporting actress #GoldenGlobes
<http://t.co/fXsNg1h>*

- How do we know where [entity] and [award] begin and end?

Example: matches for “[entity] wins [award]”

*RT @HuffingtonPost: Anne Hathaway wins best supporting actress #GoldenGlobes
<http://t.co/fXsNg1h>*

- How do we know where [entity] and [award] begin and end?
- One option: collect all possible candidate answer phrases
 - best
 - best supporting
 - best supporting actress
 - best supporting actress #GoldenGlobes
 - best supporting actress #GoldenGlobes <http://t.co/fXsNg1h>

How-to: default methods for string search

```
tweet = "RT @HuffingtonPost: Anne Hathaway wins best supporting actress  
#GoldenGlobes http://t.co/fXsNg1h"
```

Splitting on " wins " via python built-ins

```
win_split = tweet.split(" wins ")  
>> [  
    'RT @HuffingtonPost: Anne Hathaway',  
    'best supporting actress #GoldenGlobes http://t.co/fXsNg1h'  
]
```

Splitting on " "

```
left_side_words = win_split[0].split()  
>> ['RT', '@HuffingtonPost:', 'Anne', 'Hathaway']
```

How-to: getting candidate answer strings

```
tweet = "RT @HuffingtonPost: Anne Hathaway wins best supporting actress  
#GoldenGlobes http://t.co/fXsNg1h"
```

Right side of rule: collect candidate answers

```
if " wins " in tweet: # assuming only one instance of " wins " exists...  
    win_split = tweet.split(" wins ")  
    words, candidate_answers = win_split[1].split(), []  
    for ix in range(len(words)):  
        candidates.append(" ".join(words[:ix + 1]))  
  
>> ['best', 'best supporting', 'best supporting actress',  
    'best supporting actress #GoldenGlobes',  
    'best supporting actress #GoldenGlobes http://t.co/fXsNg1h']
```

How-to: getting candidate answer strings

```
tweet = "RT @HuffingtonPost: Anne Hathaway wins best supporting actress  
#GoldenGlobes http://t.co/fXsNg1h"
```

Left side of rule: collect candidate answers

```
if " wins " in tweet: # assuming only one instance of " wins " exists...  
    win_split = tweet.split(" wins ")  
    words, candidate_answers = win_split[0].split(), []  
    for ix in range(len(words)):  
        candidates.append(" ".join(words[ix:]))  
    candidates = candidates[::-1] # reverse order of results  
  
>> ['Hathaway', 'Anne Hathaway', '@HuffingtonPost: Anne Hathaway',  
    'RT @HuffingtonPost: Anne Hathaway']
```

How-to: regex methods for string search

```
tweet = "RT @HuffingtonPost: Anne Hathaway WINS best supporting actress  
#GoldenGlobes http://t.co/fXsNg1h"
```

Flexible splitting using regex:

```
re.findall(r"(.+) (wins|Wins|WINS) (.+)", tweet)
```

```
>> [('RT @HuffingtonPost: Anne Hathaway',  
      'WINS',  
      'best supporting actress #GoldenGlobes http://t.co/fXsNg1h']
```

How-to: regex methods for string search

```
tweet = "RT @HuffingtonPost: Anne Hathaway receives best supporting actress  
#GoldenGlobes http://t.co/fXsNg1h"
```

Flexible splitting using regex:

```
re.findall(r"(.+) (wins|Wins|WINS|receives|received|gets) (.+)", tweet)
```

```
>> [('RT @HuffingtonPost: Anne Hathaway',  
    'receives',  
    'best supporting actress #GoldenGlobes http://t.co/fXsNg1h']
```

Example: matches for “[entity] wins [award]”

*RT @HuffingtonPost: Anne Hathaway wins best supporting actress #GoldenGlobes
http://t.co/fXsNg1h*

- How do we know where [entity] and [award] begin and end?
- Returning to our possible candidate answers...
 - best
 - best supporting
 - best supporting actress
 - best supporting actress #GoldenGlobes
 - best supporting actress #GoldenGlobes http://t.co/fXsNg1h

Example: matches for “[entity] wins [award]”

*RT @HuffingtonPost: Anne Hathaway wins best supporting actress #GoldenGlobes
<http://t.co/fXsNg1h>*

- How do we know where [entity] and [award] begin and end?
- Simple filters can improve quality of candidates
 - Award names aren't just single-word superlatives/adjectives
 - best supporting
 - best supporting actress
 - Award names aren't mixtures of text and hashtags
 - Award names don't include links

Example: matches for “[entity] wins [award]”

*RT @HuffingtonPost: Anne Hathaway wins best supporting actress #GoldenGlobes
<http://t.co/fXsNg1h>*

- How do we know where [entity] and [award] begin and end?
- Useful to explore the diversity of “good” matches:
 - *Les Miserables wins Best Motion Picture, Comedy or Musical.*
 - *#LesMiserables wins for Best Comedy/Musical*
 - *Les Miserables wins Best Comedy or Musical*
 - *Les Mis wins comedy/musical award.*
 - *Les miserablés wins best motion picture!!*
- ... 4 distinct [entity] spans; 5 distinct [award] spans ...

Example: matches for “[entity] wins [award]”

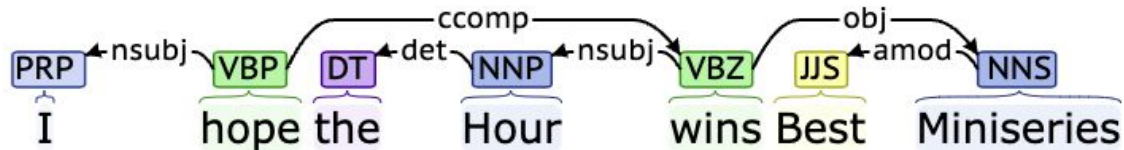
RT @HuffingtonPost: Anne Hathaway wins best supporting actress #GoldenGlobes
<http://t.co/fXsNg1h>

- How do we know where [entity] and [award] begin and end?
- Useful to explore the diversity of “bad”/“tricky” matches:
 - I hope the Hour wins Best Miniseries
 - Maggie Smith, the Dowager Countess, wins Best Supporting Actress
 - Adele's Skyfall wins best original song
- ... Motivates a broader (but quick) aside on syntactic parsing

How-to: motivating syntax ([useful reference](#))

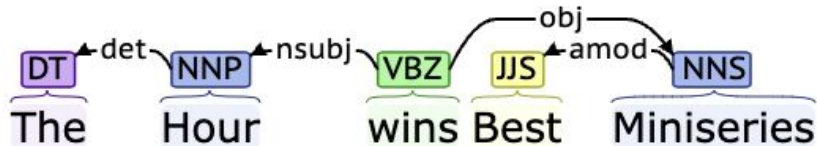
“I hope the Hour wins Best Miniseries”

- **root** of syntax tree is “**hope**”; “**wins**” is a **clausal complement** of “**hope**”



“The Hour wins Best Miniseries”

- New **root** of syntax tree is “**wins**”



How-to: syntactic parsing – dependency trees

Parsing dependency trees ([spacy link](#)):

```
import spacy

spacy_model = spacy.load("en_core_web_sm")
spacy_output = spacy_model(input_sentence)
for token in spacy_output:
    print(
        token.text,
        token.dep_,
        token.head.text,
        token.head.pos_,
        [child for child in token.children]
    )
```

I hope the Hour wins Best Miniseries

Dependency parse:

- I | nsubj | hope | VERB | []
- hope | **ROOT** | hope | VERB | [I, **wins**]
- the | det | Hour | PROPN | []
- Hour | nsubj | wins | VERB | [the]
- wins | **ccomp** | hope | VERB | [Hour, Miniseries]
- Best | compound | Miniseries | PROPN | []
- Miniseries | dobj | wins | VERB | [Best]

How-to: syntactic parsing – noun chunks

Noun chunks:

```
import spacy

spacy_model = spacy.load("en_core_web_sm")
spacy_output = spacy_model(input_sentence)
for chunk in spacy_output.noun_chunks:
    print([chunk.text, chunk.root.head.text])
```

Results on the right:

- Use caution: Model fails with @/# changes and less grammatical syntax
- Extra motivation to do good tweet-aware preprocessing of data

Input sentence	Child noun chunks of "won"
Ben Affleck won best director!	"Ben Affleck", "best director"
I just heard that Ben Affleck won best director and he was beyond grateful!	"Ben Affleck", "best director"
@BenAffleck won #BestDirector!	"BestDirector"
YAY Ben Affleck won best director omg!	"YAY Ben Affleck", "best director omg"

How-to: entity recognition

Entity recognition with [SpaCy](#):

```
import spacy
```

```
spacy_model = spacy.load("en_core_web_sm")
spacy_output = spacy_model(input_sentence)
for entity in spacy_output.ents:
    print([entity.text, entity.label_])
```

Results on the right:

- Same issues with @/#s
- Caution: the entity label isn't always correct!
Lots of noise here

Input sentence	Entity output
Ben Affleck won best director!	[Ben Affleck, PERSON]
I just heard that Ben Affleck won best director and he was beyond grateful!	[Ben Affleck, PERSON]
@BenAffleck won #BestDirector!	[#, CARDINAL]
YAY Ben Affleck won best director omg!	[YAY Ben Affleck, PERSON]

Framing: choosing the best candidate answers

```
["Les Miserables", "Les Miserables", "#LesMiserables", "Les Mis", "Les miserablés", ... ]  
["Best Motion Picture, Comedy or Musical", "Best Comedy/Musical", "Best Comedy or Musical"]
```

- **Merging answers:**
 - “Best Comedy/Musical” == “Best Comedy or Musical”
 - “best motion picture - comedy or musical” != “best motion picture - drama”
- **Ranking final answers** – should we prefer...
 - “Best Motion Picture, Comedy or Musical”?
 - “Best Comedy/Musical”?
 - ...

Framing: choosing the best candidate answers

```
["Les Miserables", "Les Miserables", "#LesMiserables", "Les Mis", "Les miserablés", ... ]  
["Best Motion Picture, Comedy or Musical", "Best Comedy/Musical", "Best Comedy or Musical"]
```

- **Merging answers:**

- Edit distance – “Miserablés” vs “Miserables”
- Co-occurring context – “Anne Hathaway” often mentioned w/ “Les Miserables”
- Component words – ignore order/punctuation → [“best”, “comedy”, “musical”]
- External knowledge – querying imdb for “Les Mis” vs. “Les Miserables”
- Tweet timestamps – (in later slides)
- ...

Framing: choosing the best candidate answers

`["Les Miserables", "Les Miserables", "#LesMiserables", "Les Mis", "Les miserablés", ...]`
`["Best Motion Picture, Comedy or Musical", "Best Comedy/Musical", "Best Comedy or Musical"]`

- **Ranking final answers:** can use frequency of answer candidates, **but beware!**
 - Example case: failure modes when choosing final list of award categories
 - Award name categories are often referred to by shortened names
 - Unofficial awards like “Best Dressed” are extremely common
 - Some awards are tweeted about less, e.g. best foreign film
- **Contextual metrics for confidence are extremely useful!** Examples:
 - Some parsing patterns might be more accurate – upweight those!
 - What are signs that a tweet is reputable? An account?

Pre-processing the dataset

Pre-processing: misc. text processing tips (1)

- The data has some artifacts that were introduced when the tweets were initially scraped from Twitter
- Example:

"Kerry Washington in Miu Miu **&**
Prada!!!! OH.....EM.....GEE!!!!!!!!!! **<**3
#GoldenGlobes #redcarpet"

- I universally recommend doing this!
- We can use the [ftfy library](#) to fix these:

```
pip install ftfy
```

Code:

```
from ftfy import fix_text  
  
fixed_text = fix_text(original_text)
```

Output:

"Kerry Washington in Miu Miu **&** Prada!!!!
OH.....EM.....GEE!!!!!!!!!! **<**3 #GoldenGlobes
#redcarpet"

Pre-processing: misc. text processing tips (2)

- **Info:** diversity of individual characters in tweets can prevent our parsing rules from working as intended (i.e. string matching, parsing syntax)
- **Examples:** emojis, non-latin alphabet letters, rare unicode symbols, etc.
- **Heavy-handed catch-all:** [unicode](#) tries to replace non-ascii characters with their ascii counterparts, otherwise it removes the characters:

```
pip install unicode
```

Input:

“Tina Fey & Amy Poehler 🤗 #goldenglobes”

Code:

```
import unicode  
  
new_text = unicode(original_text)
```

Output:

“Tina Fey & Amy Poehler #goldenglobes”

Pre-processing: misc. text processing tips (3)

- URLs are common in the data – it's your choice whether to ignore them, remove them, or integrate them into your work!
 - Can detect with regular expressions

Pre-processing: misc. text processing tips (4)

- Extra whitespace can also mess with string matching / regex patterns
- Tabs and newline characters are uncommon and can usually be replaced by a single space without affecting tweet meaning / syntax
- I'd recommend this one too, but it's up to you!

Code to substitute all whitespace:

```
fixed = " ".join(original_text.split())
```

Code to keep tabs/newline characters:

```
import re  
  
fixed = re.sub(' +', ' ', original_text)
```

Pre-processing: misc. text processing tips (5)

Most of our tweets are in English, but a sizable fraction of the data isn't

- Detecting and removing non-English can be helpful; can often misclassify, though
- **Rule of thumb:** use confidence of language detectors to choose when to discard/keep an example
- There are a few libraries that do language detection – I've had the some success with [langdetect](#):

pip install langdetect

Input:

'Qué tal? Qué impresiones tienen de la alfombra roja de los #GoldenGlobes ???'

Code:

```
from langdetect import detect, detect_langs

# detect returns the most probable language

detect(tweet) -> 'es'

# detect_lang returns the most probable languages
and probabilities

detect_langs(tweet) -> [es:0.9999931512174858]
```

Tweet-specific features

Framing: capitalizing on tweet-specific properties

- Hashtags and account names are information-rich
- Handling/leveraging quote-tweets and re-tweets is powerful
 - If something is re-tweeted a lot, it's probably important to the project tasks
 - Unique parsing rules and types of additional analysis can be performed with quote tweets
- Syntax of tweets can throw off 3rd party tools you might want to use. Examples:
 - Syntactic parsers (e.g. if you want to use part-of-speech tagging) – these often use statistical models that are probably trained on standard text
 - Language detection models – our dataset has a lot of non-English text

... **TL;DR:** even if we don't build features around these, useful for cleaning our data

Hashtags + usernames: motivation

- #/@'s are easy to extract from text
 - Limited to alphanumeric characters + underscore → easily found with regular expressions
 - No spaces allowed → certainty that we have the full hashtag or username
- #/@'s are often easy to convert into natural language
 - Most of the high-frequency #/@'s are CamelCase or snake_case (more on that later)
 - For lower-case multi-word ones, we can use context to resolve them
 - Example – counts of hashtags that match any casing of “SelmaMovie”:

```
{'#SelmaMovie': 4424, '#selmamovie': 259, '#SELMAmovie': 4, '#SELMAMOVIE': 2}
```

Hashtags + usernames: motivation

- @'s are meaningful! 1-to-1 link to a person/organization/entity/other
 - Famous people tend to have usernames similar to their real name
 - Outliers exist: Amy Poehler == @smrtgrls
 - But, despite 1-to-1 link, people still make typos with usernames too!
- #'s are meaningful! Hashtag popularity implies trending/important things
 - Useful task-related information, e.g. #BestPicture
 - Could even use hashtag co-occurrence to build a weighted graph!
 - Contrast with usernames: no guaranteed 1-to-1 relations; ability to link hashtags is useful
 - Example: #OITNB & #OrangelsTheNewBlack

Hashtags + usernames: syntactic roles

The way that #/@'s are used in tweets typically fits into a limited set of cases

1. Using #/@'s within the context of natural language

- Example: “I wanna see #AmyPoehler win one of these days for #ParksandRec”
- To convert to “standard” text (code for this on the next slide):
 - For each hashtag/username, capitalization indicates where to insert spaces
 - Substitute these back into the tweet

Hashtags + usernames: parsing example

This code works for references that are either CamelCase or snake_case

For other capitalization patterns: check if we've previously seen other hashtags with the same text + different casing

Code:

```
import re
from inflection import humanize, underscore

usernames = re.findall(r"@(\w+)", tweet_text)
hashtags = re.findall(r"#(\w+)", tweet_text)

for part_list in [usernames, hashtags]:
    processed_parts = []
    for part in part_list:
        snake_name = part[:] if "_" in part else underscore(part)
        processed_parts.append(humanize(snake_name))
```

Example input strings → output strings:

#BestDirector → Best director

#best_director → Best director

#bestdirector → Bestdirector

Hashtags + usernames: syntactic roles

2. Denoting the original tweet author in the context of retweets/quote tweets

- "Aaand now I'm mad. RT @goldenglobes: Best Motion Picture - Comedy or Musical - Les Miserables - #GoldenGlobes"

To make analysis easier, use the "RT @..." to split the full string:

- The retweeted text → *Best Motion Picture ...*
 - Suggestion: store the retweeted text and the original username – easier to track how often the original tweet was RT'd
- Any newly added text → *Aaand now I'm mad.*

Hashtags + usernames: syntactic roles

3. Username-specific: tweeting at / tagging another account

"@joshuahorowitz RT @goldenglobes: From the pressroom: Jodie Foster says that she's not retiring from acting. ... <http://t.co/a48rr4e2>"

"@EmWatson you're so right is doesn't get better than this :) #GoldenGlobes"

Hashtags + usernames: syntactic roles

4. Inserted outside of any natural language context (and not fitting into cases 2/3)

"Django wins it's first award! #BestSupportingActor #GoldenGlobes"

... Beware of edge cases! This one is tricky to partition:

"#DjangoUnchained and Quentin Tarantino reeling in the #GoldenGlobes #Deserving #BestFilm"

(suggestion: syntax parsing!)

One application: improving language detection

Language detection accuracy drops in the presence of hashtags and usernames

- In this example, the hashtags aren't integral to the natural language's syntax

Example on the right: **misclassified as Danish!**

Note: This example uses the language detection tool [PyCLD3](#). Langdetect did not misclassify this example but we still think that removing hashtags should improve language detection.

Input:

“All these hot men in tuxes... I love it. Ben affleck, Hot! #eredcarpet #GoldenGlobes”

Outputs:

```
LanguagePrediction(language='da',  
probability=0.476..., is_reliable=False,  
proportion=1.0)
```


One application: improving language detection

Dropping the hashtags fixes the output!

New Input:

“All these hot men in tuxes... I love it. Ben affleck, Hot!”

New Outputs:

```
LanguagePrediction( language='en',  
probability=0.905..., is_reliable=True,  
proportion=1.0 )
```

Input:

“All these hot men in tuxes... I love it. Ben affleck, Hot! #eredcarpet #GoldenGlobes”

Outputs:

```
LanguagePrediction(language='da',  
probability=0.476..., is_reliable=False,  
proportion=1.0)
```

Re-tweets + quote tweets: motivation

RT/QTs are fairly straightforward to detect and parse – cases:

1. Simple retweet: tweet starts with “RT @”
 - a. “RT @goldenglobes: Best Director - Ben Affleck (@BenAffleck) - Argo - #GoldenGlobes”
2. Simple quote tweet: includes “ RT @” (note the space before “RT”!)
 - a. “Congratz beautiful! RT @TVGuide: Best actress for comedy/musical goes to Jennifer Lawrence #GoldenGlobes”
3. Can retweet/quote tweet existing retweets/quote tweets
 - a. “RT @AmazedByRobsten: RT @goldenglobes: From the pressroom: Jodie Foster says that she's not retiring from acting. She's looking forward to directing more.”
 - b. “RT @MsAmberPRiley: Truth RT @bernardx: you could hear a pin drop. Who else can command that kind of respect and admiration. Very few.#GoldenGlobes”
4. Other QT formats exist, though – e.g. [added text] “@username ...”

Re-tweets + quote tweets: motivation

RTs are meaningful!

- a. Expression of agreement/interest/etc.
- b. Can suggest importance of the **content** of the original tweet
 - i. Tracking the number of retweets for each original tweet could be very useful
 - ii. More retweets → information in original tweet is more likely to be reliable
- c. Can suggest importance of the **account** of the original tweet
 - i. Most retweeted accounts in GG2013 include the official Golden Globes account, news organizations, etc.
 - ii. Lots of different ways to use this in your analysis!

Re-tweets + quote tweets: motivation

QTs are meaningful!

- a. QTs can be used to measure importance, like RTs
- b. Structurally, QTs guarantee that the text of the quote tweet is directly related to the content of the original tweet
 - i. If some tweet is quote-tweeted a reasonable amount, the added texts in QTs can be a cheap way to get overall reactions to something – e.g. were there any nominees who really got snubbed for an award?
 - ii. The average length of added text in QTs is much shorter than standard tweets – additional tasks like sentiment analysis are especially good fits

Re-tweets + quote tweets: motivation

QTs are meaningful!

- a. QTs can be used to measure importance, like RTs
- b. Structurally, QTs guarantee that the text of the quote tweet is directly related to the content of the original tweet
- c. QT-specific parsing rules can be interesting and useful!
 - i. Linking awards + other project tasks:
 - "Boo Brave. It's all about #WreckItRalph RT @CNNshowbiz: Best animated film is awarded to \"Brave\" #GoldenGlobes"
 - ^ likely that "#WreckItRalph" is a nominee for "Best animated film"
 - ii. Sentiment analysis related to specific awards/events/etc.:
 - "Yes!! I am SO happy for Anne! RT @goldenglobes Best Supporting Actress in a Motion Picture - Anne Hathaway - Les Miserables - #GoldenGlobes"
 - iii. ... and more!

Tweet timestamp: motivation

- Our data also includes “timestamp_ms” for each tweet == when the tweet was posted
- Consider a stripped-down script of the progression of award shows:
 - People arrive – often a red carpet-type thing
 - The show starts – hosts introduce themselves
 - Awards are presented sequentially
 - The show ends

Tweet timestamp: motivation

- These script events afford specific types of things we can learn/extract:
 - People arrive – often a red carpet-type thing
 - Can learn what people are wearing (who is best dressed?), who arrives or takes pictures together, etc.
 - Can learn about award names, nominees, hosts (likely released before the show)
 - *Cannot* learn about award winners! Could ignore all tweets from this part of this script for winner-related parsing rules
 - The show starts – hosts introduce themselves
 - Awards are presented sequentially
 - The show ends

Tweet timestamp: motivation

- These script events afford specific types of things we can learn/extract:
 - People arrive – often a red carpet-type thing
 - The show starts – hosts introduce themselves
 - My intuition is that the number of host-related tweets dramatically increases here
 - Still no award winners to parse, yet!
 - Awards are presented sequentially
 - The show ends

Tweet timestamp: motivation

- These script events afford specific types of things we can learn/extract:
 - People arrive – often a red carpet-type thing
 - The show starts – hosts introduce themselves
 - Awards are presented sequentially
 - Intuition: tweets related to each award name/nominees/winners/presenters spike when each award's phase starts
 - Trivially: we can only know the winner of an award once we reach its individual presentation → can turn on winner-related parsing rules
 - The show ends

Tweet timestamp: motivation

- All to say – extremely useful to estimate the timing of these events:
 - Start of actual award show
 - The start and end times for each award
 - During this range (and immediately after): more likely that tweets are about the award
 - Example use case: merging shortened forms / typos / etc. related to award names
 - Concretely: suppose that you extract “best movie”, “best film”, “best picture” from tweets in a specific award’s window
 - Then, we can be more confident that those expressions refer to the same award!
 - If less confident about some candidate parses related to names of awards, do those potential answers resolve any “gaps” in your estimated schedule for all awards?

Tweet timestamp: processing raw timestamps

- Each example has a POSIX timestamp (time since Jan 1 1970) in milliseconds
- We can use python's built-in [datetime](#) library to easily parse it into a human-readable format

Code:

```
import datetime

raw_time = example["timestamp_ms"]

dt = datetime.datetime.fromtimestamp(raw_time/1000.0)
```