# INTRODUCTION TO NLP
## Project 1 Practical Tips

Prof. Larry Birnbaum

TA Victor Bursztyn
v-bursztyn@u.northwestern.edu

Northwestern | intelligent information laboratory

# Tip 1: Start with Expressions

**Task 1: Award Names.**



- The first advice is on where to start, and I've recommended as starting point to think about expressions that could indicate the presence of a particular piece of information.
- Take as an example the task of extracting the award names.

# Tip 1: Start with Expressions

Task 1: Award Names.

**Sam Mendes got Best Director!**

**Sam Mendes won Best Director!**



- If you bump into these two tweets here, both mentioning Sam Mendes as the winner of the Best Director award…
- …If you had to pick only one verb to use as a filter of relevant tweets, which one would you pick?

# Tip 1: Start with Expressions

Task 1: Award Names.

**Sam Mendes got Best Director!**

**Sam Mendes won Best Director!**



- And why would you pick “won” and not “got”?
- Because the verb win can be more specific and meaningful in this particular domain. “Got” could happen in tweets that have nothing to do with award names.

# Tip 1: Start with Expressions

Task 1: Award Names.

Sam Mendes got Best Director!

**Sam Mendes won Best Director!**

```
tweet.lower().split().index("won")
```



- Now, just to remember, with this original representation of the tweet, you can retrieve the position of the word “won”.
- Assuming that tweet is the variable storing the tweet, you can run: tweet.lower() to normalize to lowercase, .split() to split over the space characters, then look for the index of “won.”
- What should be returned by this line?

# Tip 1: Start with Expressions

Task 1: Award Names.

Sam Mendes got Best Director!

**Sam Mendes won Best Director!**
0 1 **2** 3 4

```
tweet.lower().split().index("won")
```



- Two, right? The word “won” is at the third position in the sentence, so it returns 2, from 0 to 2.
- And now that you have the position of the verb win, how could we parse the award name?

# Tip 1: Start with Expressions

Task 1: Award Names.

Sam Mendes got Best Director!




- We would need to scan the sentence forward, because the award name is the object of the verb win. So it follows the verb win here.

# Tip 1: Start with Expressions

Task 1: Award Names.

Sam Mendes got Best Director!

**Sam Mendes won Best Director! That's awesome!**



- And probably the tweet doesn't really stop there. You can have more stuff following that object. Here's just one example.
- Considering this, does it look easy to parse the award name as you scan the sentence forward?

# Tip 1: Start with Expressions

Task 1: Award Names.

Sam Mendes got Best Director!

**Sam Mendes won Best Director! That's awesome!**

**Best Screenplay**

**Best Motion Picture**

**Best Television Series - Drama**

**Best Original Song - Motion Picture**

**Best Original Score - Motion Picture**



- Here's a few award names. Would they be easy to parse or hard to parse this way?
- And why's that? You don't have a clear delimiter closing the award names, right? You can try to use the POS tagger and that should help a bit, but that's not super intuitive.

# Tip 1: Start with Expressions

Task 1: Award Names.

Sam Mendes got Best Director!

**Sam Mendes won Best Director! That's awesome!**

**:))) Best Director goes to Sam Mendes! That's awesome!**



- And what about with this other tweet here?
- Is it the same thing when you focus on the expression "goes to"? Why?

# Tip 1: Start with Expressions

Task 1: Award Names.

Sam Mendes got Best Director!

**Sam Mendes won Best Director! That's awesome!**

**:))) Best Director goes to Sam Mendes! That's awesome!**



- Yes, now the award name is the subject of the verb go. It means that, starting from the verb's position, you'd need to scan the sentence backward.

# Tip 1: Start with Expressions

Task 1: Award Names.

Sam Mendes got Best Director!

**Sam Mendes won Best Director! That's awesome!**

**:))) Best Director goes to Sam Mendes! That's awesome!**

**Best Screenplay**

**Best Motion Picture**

**Best Television Series - Drama**

**(…)**



- And what about now? If you had to parse the award names as the subject rather than the object, would it be equally hard?

# Tip 1: Start with Expressions

Task 1: Award Names.

Sam Mendes got Best Director!

**Sam Mendes won Best Director! That's awesome!**

**:))) Best Director goes to Sam Mendes! That's awesome!**

**Best** Screenplay

**Best** Motion Picture

**Best** Television Series - Drama

(…)



- If you scan the sentence back, you may be able to explore some pattern, some regularity, to know where to stop.
- This is just one way of exploring properties of language to extract the information for this project.

# Tip 2: ∑ (Weak Strategies)

$$candidate\_answers_1 \leftarrow extraction\_rule_1()$$

$$candidate\_answers_2 \leftarrow extraction\_rule_2()$$

$$candidate\_answers_3 \leftarrow extraction\_rule_3()$$

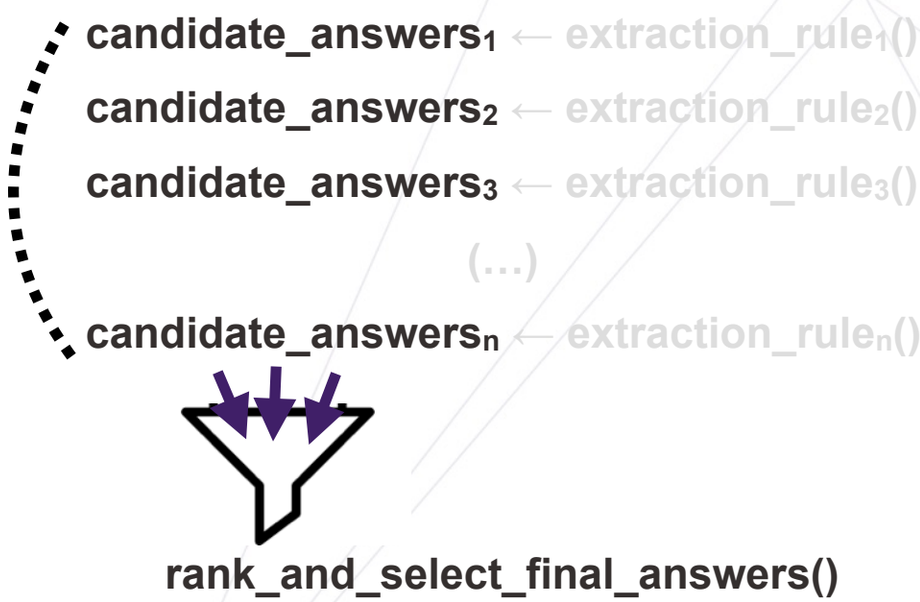
$$(\ldots)$$

$$candidate\_answers_n \leftarrow extraction\_rule_n()$$



- Which leads us to the second tip:
- The typical solution for this project is a combination of many strategies that can be individually weak or limited, but can be combined in a sensible way to produce the final set of answers.
- Here you can imagine different rules filtering subsets of tweets and extracting candidate answers.

# Tip 2: ∑ (Weak Strategies)






- Then part of your task is to figure out ways of merging these results, ranking them, and selecting the final set of answers.
- This is what Larry means by creating a voting system. And of course you can have votes with different weights if you can trust a particular rule more than you trust the others.

# Tip 2: ∑ (Weak Strategies)

**Concrete Example: Extracting Winners**

**$rule_1$ ← "X won Y"**

"I can't believe John Doe **won** Best Actor! So happy!"



- Let's see a concrete example for the task of extracting winners.
- Our first rule will be based on the pattern "X won Y". X, the subject here, is the winner, so we need to scan the sentence from right to left starting from "won."
- This rule will catch tweets like this one: "I can't believe John Doe won Best Actor! So happy!"

# Tip 2: ∑ (Weak Strategies)

**Concrete Example: Extracting Winners**

**$rule_1$ ← "X won Y"**

"I can't believe John Doe **won** Best Actor! So happy!"

**$rule_2$ ← "X goes to Y"**

"And Best Actor **goes to** John Doe! So well deserved!"



- Our second rule will be based on the pattern "X goes to Y." Y, the object here, refers to the winner, so we need to scan the sentence from left to right starting from "goes to."
- This rule will catch tweets like this one: "And Best Actor goes to John Doe! So well deserved!"

# Tip 2: ∑ (Weak Strategies)

**Concrete Example: Extracting Winners**

$\mathbf{r_1}$ ← “I can't believe John Doe **won** Best Actor! So happy!”



- As we apply this rule to this data we start generating candidate answers.

# Tip 2: ∑ (Weak Strategies)

**Concrete Example: Extracting Winners**

$\mathbf{r_1}$ ← “i can't believe john doe **won** best actor so happy”



- First let’s normalize this tweet by removing punctuation and transforming to lower case.

# Tip 2: ∑ (Weak Strategies)

**Concrete Example: Extracting Winners**

$\mathbf{r_1}$ ← “i can't believe john doe **won** best actor so happy”

**candidate_answers$_1$ ← []**



- And now we can start scanning this sentence from right to left, token by token, and parsing each n-gram as a candidate answer.

# Tip 2: ∑ (Weak Strategies)

**Concrete Example: Extracting Winners**

**$r_1$** ← “i can't believe john doe **won** best actor so happy”

**$ca_1$** ← **["doe"]**



- The first candidate answer is the unigram “doe.”

# Tip 2: ∑ (Weak Strategies)

**Concrete Example: Extracting Winners**

**$r_1$** ← “i can't believe john doe **won** best actor so happy”

**$ca_1$** ← ["doe", **"john doe"**]



- The second candidate answer is the bigram “john doe.”

# Tip 2: ∑ (Weak Strategies)

**Concrete Example: Extracting Winners**

**$r_1$** ← “i can't believe john doe **won** best actor so happy”

**$ca_1$** ← ["doe", "john doe", **"believe john doe"**]



- Then we have the trigram “believe john doe.”

# Tip 2: ∑ (Weak Strategies)

**Concrete Example: Extracting Winners**

$r_1$ ← “i can't believe john doe **won** best actor so happy”

$ca_1$ ← ["doe", "john doe", "believe john doe", "can't believe john doe", **"i can't believe john doe"**]



- And eventually we have a full set of candidates including clearly noisy ones.

# Tip 2: ∑ (Weak Strategies)

**Concrete Example: Extracting Winners**

$\mathbf{r_2}$ ← “and best actor **goes to** john doe so well deserved”

**candidate_answers$_2$ ← []**



- Now we do the same for rule number two, this time scanning the sentence from left to right because we want the object of goes to.

# Tip 2: ∑ (Weak Strategies)

**Concrete Example: Extracting Winners**

$r_2$ ← “and best actor **goes to** john doe so well deserved”

**$ca_2$ ← ["john", "john doe", "john doe so", "john doe so well", "john doe so well deserved"]**



- So we end up with all these additional candidate answers: “john”, “john doe”, “john doe so”, “john doe so well”, and finally “john doe so well deserved.”
- Again, we have a full set of candidates including noisy ones.

# Tip 2: ∑ (Weak Strategies)

**Concrete Example: Extracting Winners**

$ca_1 \leftarrow$ ["doe", "john doe", "believe john doe", "can't believe john doe", "i can't believe john doe"]

$ca_2 \leftarrow$ ["john", "john doe", "john doe so", "john doe so well", "john doe so well deserved"]



- And now that we have the two sets, what happens when we try to merge the two?

# Tip 2: ∑ (Weak Strategies)

**Concrete Example: Extracting Winners**

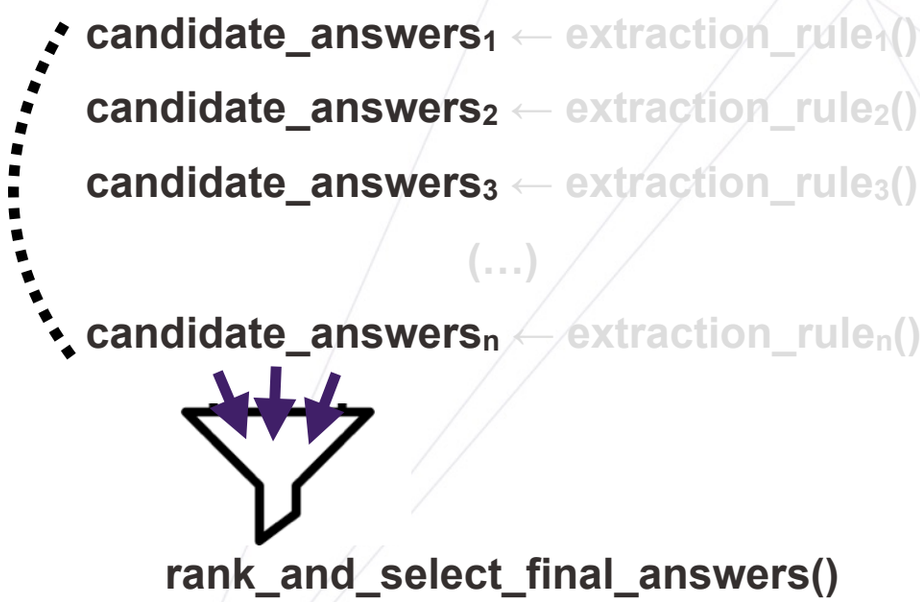
**$ca_1$** ← ["doe", **"john doe"**, "believe john doe", "can't believe john doe", "i can't believe john doe"]

**$ca_2$** ← ["john", **"john doe"**, "john doe so", "john doe so well", "john doe so well deserved"]



- The candidate answer “john doe” is the intersection of the two sets.
- Another way of putting it is that “john doe” received two “votes” from two independent extraction rules, allowing us to filter out noisy candidates by simply counting frequencies.

# Tip 2: ∑ (Weak Strategies)






- It doesn't mean that this process is going to be that clean or that easy. It won't be perfect, but it's generalizable.
- This is how you work with the requirement of **not** hardcoding parts of the answers you're trying to extract.

# Tip 3: Run the Autograder

**Open: gg2013answers.json & gg2015answers.json**

**Run: time python3 autograder.py 2013 (or 2015)**



- Tip #3 is to make sure you run the autograder on either 2013 or 2015 to test your solution and get some feedback.
- You have the answers in these files gg2013answers.json and gg2015answers.json, so you can at least open these files and check if a your strategy is generating the answers.

# Tip 4: Additional Goals

**Additional Goals: Sentiment Analysis (among others).**



- The last tip is: don’t forget about the additional goals. You can be creative here, but one thing that’s very feasible is to do sentiment analysis over the ceremony.

# Tip 4: Additional Goals

Additional Goals: Sentiment Analysis (among others).

**> pip install TextBlob**

**> pip install vaderSentiment**



- You can find many libraries that help you with that. These are two that are extremely simple to use: TextBlob and Vader. You have many others that are more sophisticated than these.

# Tip 4: Additional Goals

Additional Goals: Sentiment Analysis (among others).

> pip install TextBlob

> pip install vaderSentiment

```
> from textblob import TextBlob
> blob = TextBlob("I loved the Golden Globes!")
> blob.sentences[0].sentiment.polarity
> 0.5375
```



- But if you use, for instance, TextBlob, you can check the polarity of the phrase "I loved the Golden Globes!" with just three lines of code.
- The polarity can range from -1.0 to 1.0, meaning that something can be negative, positive or neutral depending on your interpretation of this range.
- Here the polarity is above .5 because of the word "loved."

# Tip 4: Additional Goals

Additional Goals: Sentiment Analysis (among others).

> pip install TextBlob

> pip install vaderSentiment

**> from textblob import TextBlob**
**> blob = TextBlob("I hated the Golden Globes!")**
**> blob.sentences[0].sentiment.polarity**
**> -0.2625**



- But if you change it to "hated", then you get -0.26, which can be interpreted as negative polarity.

# Tip 4: Additional Goals

Additional Goals: Sentiment Analysis (among others).

```
> from textblob import TextBlob
> blob = TextBlob("Not a fan of this Best Director")
> blob.sentences[0].sentiment.polarity
> _
```



- And what about this? You have the string "Not a fan of this Best Director." What do you think this polarity should be? Any guesses?

# Tip 4: Additional Goals

Additional Goals: Sentiment Analysis (among others).

```
> from textblob import TextBlob
> blob = TextBlob("Not a fan of this Best Director")
> blob.sentences[0].sentiment.polarity
> 1.0
```



- It thinks it's perfectly positive. Probably the word "Best" is nudging this model towards positive polarity, and even the word "fan" can be doing the same if it doesn't catch the negation.
- This is just to show that simple models are noisy and can't parse the nuances of sentiment as humans do.
- However, you can figure out ways of handling some of this noise if you want to use them. Maybe removing the word "Best" is a good idea here.

# Tip 4: Additional Goals

Additional Goals: Sentiment Analysis (among others).

```
> from textblob import TextBlob
> blob = TextBlob("Not a fan of this Best Director")
> blob.sentences[0].sentiment.polarity
> 1.0
```

**"I literally cried with the monologue! :)"**



- But as I mentioned, you can get creative when doing you additional goals.
- Can you assess the polarity of this tweet here?

# Tip 4: Additional Goals

Additional Goals: Sentiment Analysis (among others).

```
> from textblob import TextBlob
> blob = TextBlob("Not a fan of this Best Director")
> blob.sentences[0].sentiment.polarity
> 1.0
```

**"I literally cried with the monologue! :)"**
**"Ricky Gervais :D"**



- And what about this?

# Tip 4: Additional Goals

Additional Goals: Sentiment Analysis (among others).

```
> from textblob import TextBlob
> blob = TextBlob("Not a fan of this Best Director")
> blob.sentences[0].sentiment.polarity
> 1.0
```

**"I literally cried with the monologue! :)"**
**"Ricky Gervais :D"**
**"The ceremony is about to start ❤️"**



- And this tweet? Is it positive or negative?

# Tip 4: Additional Goals

Additional Goals: Sentiment Analysis (among others).

```
> from textblob import TextBlob
> blob = TextBlob("Not a fan of this Best Director")
> blob.sentences[0].sentiment.polarity
> 1.0
```

**"I literally cried with the monologue! :)"**
**"Ricky Gervais :D"**
**"The ceremony is about to start ❤️"**
**"Oh man 😄 #goldenglobes"**



- Is this one positive or negative?
- You can create custom dictionaries to measure things that you might think are interesting in this dataset.
- In the context of an additional goal, there's value in anything that can act as a cheap proxy for measuring sentiment over time.

# Tip 5: all() and any()

```python
fashion_dict = ['dress', 'skirt', 'tuxedo']

tweet = "House ambassador Margot Robbie attended the 77th Golden Globes awards ceremony\
         in an embroidered metallic bustier top with a long skirt in satin from the\
         Fall-Winter 2019/20 Haute Couture collection."

[(kw in tweet) for kw in fashion_dict]
```

```
[False, True, False]
```

```python
fashion_dict = ['dress', 'skirt', 'tuxedo']

any([kw in "Margot Robbie (...) a long skirt (...)" for kw in fashion_dict])
```

```
True
```



- When working with these custom-made dictionaries in Python (dictionaries not as in the data structure, but as in a custom list of tokens), the operators all() and any() are your friends.

# Thank you! Questions?

v-bursztyn@u.northwestern.edu

Northwestern | intelligent information laboratory

- And that’s it!