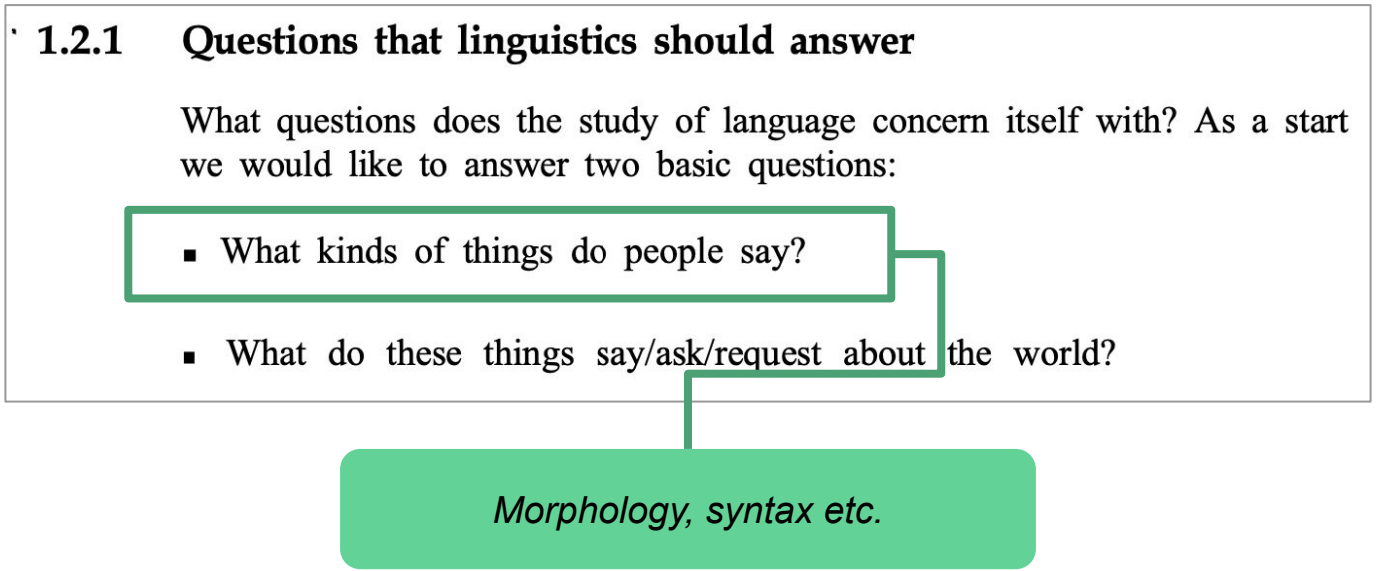
Good ol’ NLP

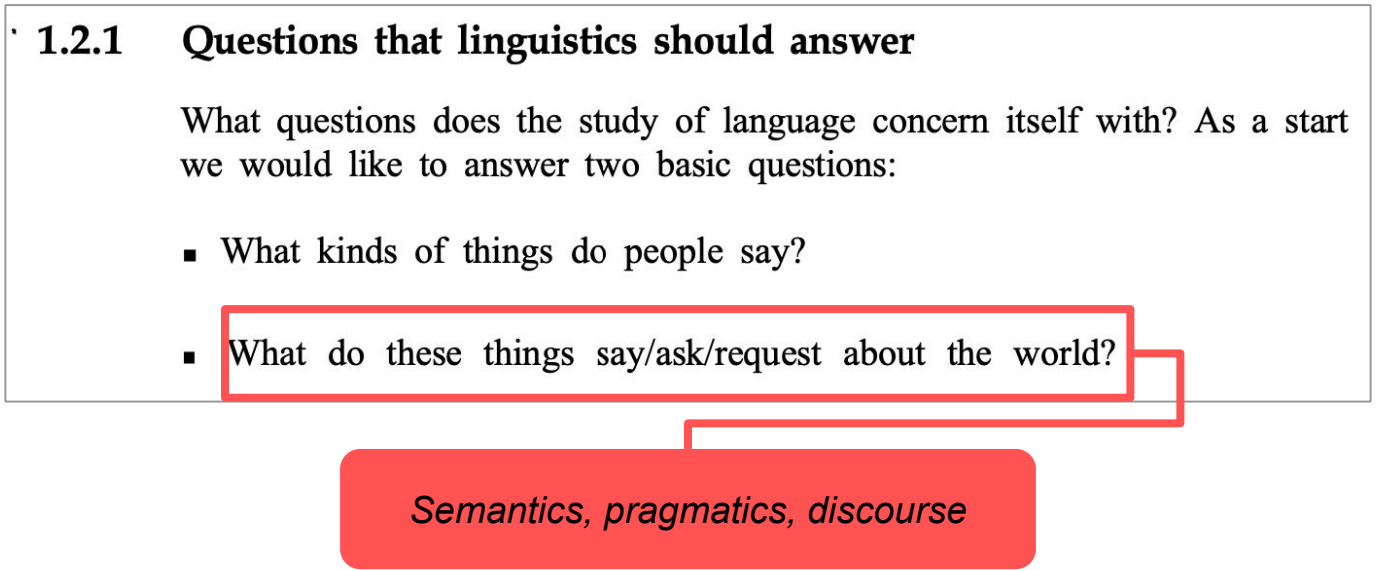
What is language?

- Language is an incredibly complex object, and a quintessential human prerogative (pending some birds).



What is language?

- Language is an incredibly complex object, and a quintessential human prerogative (pending some birds).



What is language?

- Sound-stuff: phonetics and so on
  - Note: speech-to-text is pretty advanced, and we mostly deal with written language
- Morphology: word structure
  - *bellissimo* = “bell” (root) + “issim” (superlative) + “o” (male, singular)
- Syntax: how words are combined together
  - Colorless green ideas sleep furiously (and Broca’s area!)
  - Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo
- Lexical semantics: the meaning of words
  - Man : King = Woman : Queen
- Compositional Semantics: meaning of sentences (truth / entailment)
  - Every man is mortal, Socrates is a man, Socrates is ....
  - The meaning of a sentence is how the world would look like, if the sentence was true
- Pragmatics: language in context
  - “Can you speak English?” vs “Can you pass me the salt?”
  - - How is your new CS Ph.D.? - He is always on time for meetings and has a very pleasant voice.
- Discourse: language in turns
  - - I can’t meet you today. - What if we do in two days?

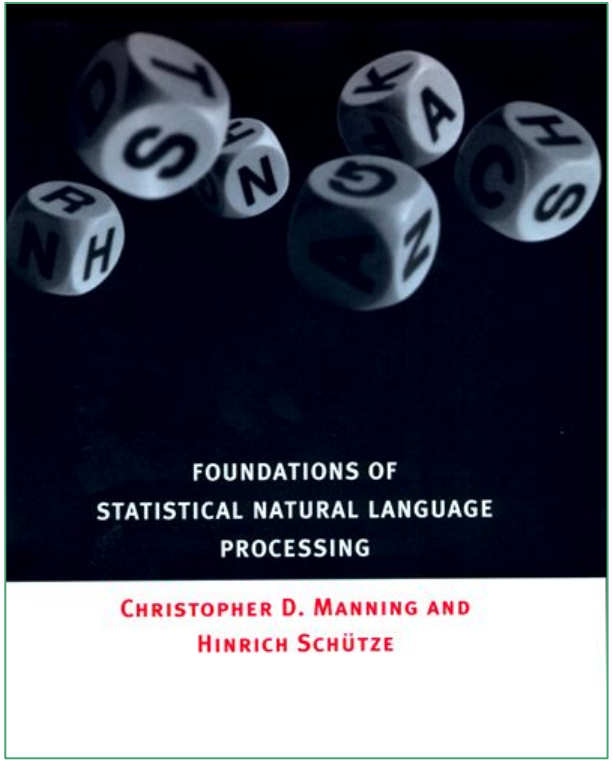
What is language at FRE 7773?

- Sound-stuff: phonetics and so on
  - Note: speech-to-text is pretty advanced, and we mostly deal with written language
- Morphology: word structure
  - *bellissimo* = “bell” (root) + “issim” (superlative) + “o” (male, singular)
- Syntax: how words are combined together
  - Colorless green ideas sleep furiously (and Broca’s area!)
  - Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo
- Lexical semantics: the meaning of words
  - Man : King = Woman : Queen
- Compositional Semantics: meaning of sentences (truth / entailment)
  - Every man is mortal, Socrates is a man, Socrates is ....
  - The meaning of a sentence is how the world would look like, if the sentence was true
- Pragmatic: language in context
  - “Can you speak English?” vs “Can you pass me the salt?”
  - - How is your new CS Ph.D.? - He is always on time for meetings and has a very pleasant voice.
- Discourse: language in turns
  - - I can’t meet you today. - What if we do in two days?

Language as a statistical phenomenon

- To answer “What kind of things people say?” we take a *statistical approach,* that is, we try and identify *common patterns* that occur in language use (i.e. we’ll do a lot of counting and probabilities).
    - On Chomsky and the Two Cultures of Statistical Learning

- “While practical utility is something different from the validity of a theory, the usefulness of statistical models of language tends to confirm that there is something right about the basic approach.”



Language modelling

Given a language L with terms t_1, t_2, ... t_n, and a set of sentences S from t_1, t_2, ... t_n, a language model (**LM**) is a function *f* assigning a probability to each sentence in S. We require *f* to be a probability distribution, i.e.:

1. The sum of all probabilities for all sentences sums up to 1;
2. Each sentence gets assigned a probability, that is for each sentence *s*, f(*s*) >= 0.

Given a LM, we can:

- given a sentence, calculate its probability under the LM: P(“I ate an apple”) > P(“I ate an embassy”)
- given *n* tokens starting a sentence, predict what is likely to come next: P(“apple” | “I ate an”) > P(“embassy” | “I ate an”) -> **if I concatenate prediction after prediction, what do I get?**

Language modelling

- **Terms**: { Jacopo, NYU, NLP, NYC, lives, teaches, is, Italy, and ... }
- **Sample sentences S**: { Jacopo teaches at NYU; Jacopo is from Italy; Jacopo is from NYC; Jacopo teaches NLP in NYC; Jacopo teaches NLP in Italy }
- **Task**: *learn* a LM for S.
  - “Learn” implies that our LM should reflect the statistical patterns of our sample, instead of, for example, simply assigning arbitrary probabilities to sentences;
  - the fact that a sentence is true or false for humans is irrelevant (i.e. I’m not from NYC): LMs capture statistical patterns of “plausibility”, not truthfulness.

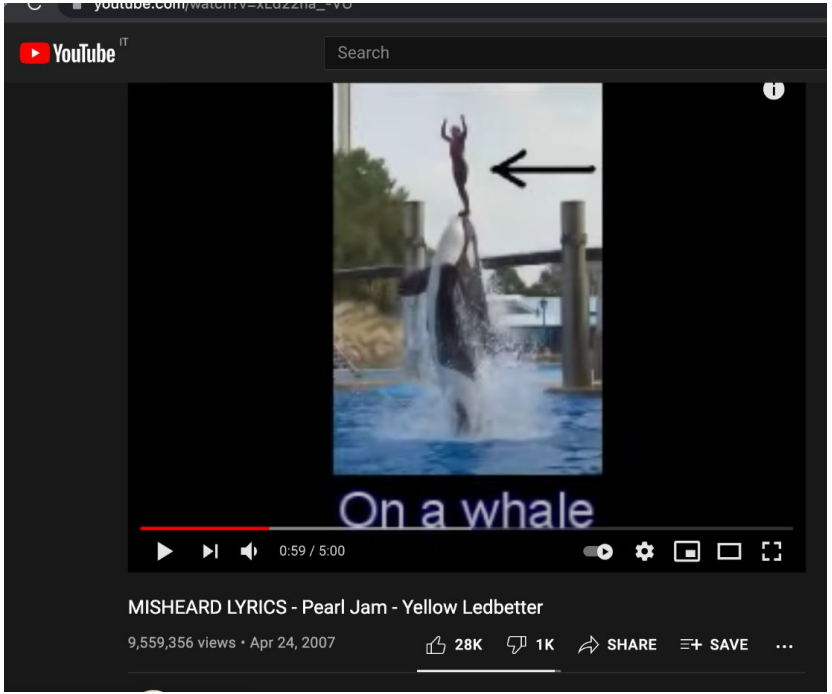
Language modelling

- **Terms**: { Jacopo, NYU, NLP, NYC, lives, teaches, is, Italy, and ... }
- **Sample sentences S**: { Jacopo teaches at NYU; Jacopo is from Italy; Jacopo is from NYC; Jacopo teaches NLP in NYC; Jacopo teaches NLP in Italy }
- **Task**: *learn* a LM for S.
  - “Learn” implies that our LM should reflect the statistical patterns of our sample, instead of, for example, simply assigning arbitrary probabilities to sentences;
  - the fact that a sentence is true or false for humans is irrelevant (i.e. I’m not from NYC): LMs capture statistical patterns of “plausibility”, not truthfulness.

A trivial LM: *f(s)* = empirical frequency of s!

Why should *you* care?

- Speech recognition: what you hear is a “sound” + your linguistic expectations (check this on misheard lyrics!)
- LMs are in your everyday life. e.g. Smart Compose for Gmail!
- LMs are a (plausible) way to treat language as a statistical phenomenon: their shortcomings are interesting.
- LMs are **very central** to contemporary NLP.



Markovian language models

- **Goal**: estimate the probability of a sequence of terms, taken from our vocabulary, that is $P(t_1, t_2, \dots t_n)$
  - Since for *n* there are $|\text{Vocabulary}|^n$ sequences, we want a compact model!
- **First step**: re-write the joint distribution with the chain rule:

  $$P(t_1, t_2, \dots t_n) = P(t_1) \prod_{i=2}^{n} P(t_i \mid t_1, \dots t_{i-1})$$
- **Second step**: Markov assumption, the probability of a term *only depends on the previous term.*

  $$P(t_1, t_2, \dots t_n) = P(t_1) \prod_{i=2}^{n} P(t_i \mid t_{i-1})$$
- **Bonus step**: second degree Markov, third, etc.

A bigram language model

- Let's augment our vocabulary t_1, t_2, ... t_n with two special tokens, * and |.
  - * is to be used as a special "start sentence" sign
  - | is to be used as a special "stop sentence" sign
- **Goal #1**: since the probability of any sentence for our LM is the product of the probabilities of each bigram...
- **Goal #2**: estimate the probability of each bigram, that is, each pair of terms.
  - P("Jacopo teaches NLP") = P("* Jacopo") x P("Jacopo teaches") x P("teaches NLP") x P("NLP |")
  - We would like to estimate then P(Jacopo | *), P(teaches | Jacopo), etc.
- **Estimation**: "maximum likelihood estimation"
  - Given a bigram <u, w>, P(w | u) = Count(u, w) / Count(u)
  - Example: P(teaches | Jacopo) = Count(Jacopo teaches) / Count(Jacopo)
- **Let's check the notebook now to see how that looks in practice**.

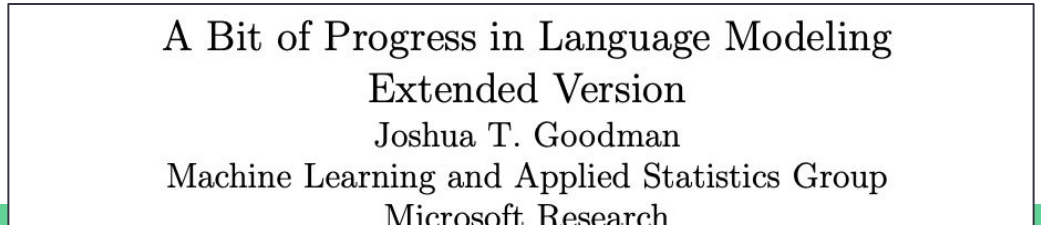
A trigram language model

- One more time, with feelings!
- Trigram LMs are exactly the same as bigram LMs, but now we employ a second-order Markov condition
    - i.e. the probability of “States”, in “Biden is the President of the United States”, *depends only on “United” and “the”.*
- **Estimation**: “maximum likelihood estimation”
    - Given a trigram <u, y, w>, P(w | u, y) = Count(u, y, w) / Count(u, y)
    - Example: P(in | Jacopo teaches) = Count(Jacopo teaches in) / Count(Jacopo teaches)
- **Let’s check the notebook now to see how that looks in practice**.

How good is a LM?

- Qualitative evaluation:
  - If we are fluent in the underlying language/vocabulary, we can “unit test” our LM:
    - quality checks depends heavily on the tester knowledge / assumptions.
    - It doesn’t scale, BUT qualitative tests are very useful in practice (e.g. corner cases, biases).

How good is a LM?

- Qualitative evaluation:
  - If we are fluent in the underlying language/vocabulary, we can “unit test” our LM:
    - quality checks depends heavily on the tester knowledge / assumptions.
    - It doesn’t scale, BUT qualitative tests are very useful in practice (e.g. corner cases, biases).
- Quantitative evaluation:
  - Intrinsic, with **perplexity**:
    - Intuition: given a standard train/test split, a good LM would evaluate as highly probable the unseen sentences in the test set.
    - You first compute the log probability of test under LM, and normalize by the number of words: call it LP; then perplexity = 2^{-LP}; i.e. the *smaller perplexity is*, the better the LM.
    - **Q**: **what happens if *any* of the sentence in the test set gets P=0 under the LM**?
  - 
A Bit of Progress in Language Modeling
Extended Version
Joshua T. Goodman
Machine Learning and Applied Statistics Group
Microsoft Research

How good is a LM?

- Qualitative evaluation:
  - If we are fluent in the underlying language/vocabulary, we can “unit test” our LM:
    - quality checks depends heavily on the tester knowledge / assumptions.
    - It doesn’t scale, BUT qualitative tests are very useful in practice (e.g. corner cases, biases).
- Quantitative evaluation:
  - Intrinsic, with **perplexity**:
    - Intuition: given a standard train/test split, a good LM would evaluate as highly probable the unseen sentences in the test set.
    - You first compute the log probability of test under LM, and normalize by the number of words: call it LP; then perplexity = 2^{-LP}; i.e. the *smaller perplexity is*, the better the LM.
    - **Q**: **what happens if *any* of the sentence in the test set gets P=0 under the LM**?
  - Downstream tasks: we will discuss LMs as building blocks for other tasks in future lecture.

Dealing with long range dependencies

- The Markov assumption seems very plausible for certain contexts:

  In “Biden is the President of the United States”, the high probability of P(States | the, United) does a good job in narrowing down candidates.

- ...but certainly not in others:

  “I like the book ...”: “I am reading now”, “sitting on my desk”, “that I bought yesterday”... completion are much more open ended.

We can build higher-order LMs (four-gram models etc.), but data sparsity would make the models marginally better after a while. Truth is, we will revisit this with neural networks.

Dealing with rare events

- Remember *perplexity* goes to infinity when any P(sentence) is 0, which happens anytime the test set has unseen n-grams.
- The solution is called "*smoothing*", and involves providing estimates for unseen n-grams.

METHOD #1: ADD ONE (LAPLACE LAW)

- Given <u, w>, P(w | u) = Count(u, w) **+ 1** / Count(u) **+ |V|** (where |V| is the vocabulary size)
- When Count(u, w) = 0, the LM will still assign to it a positive probability

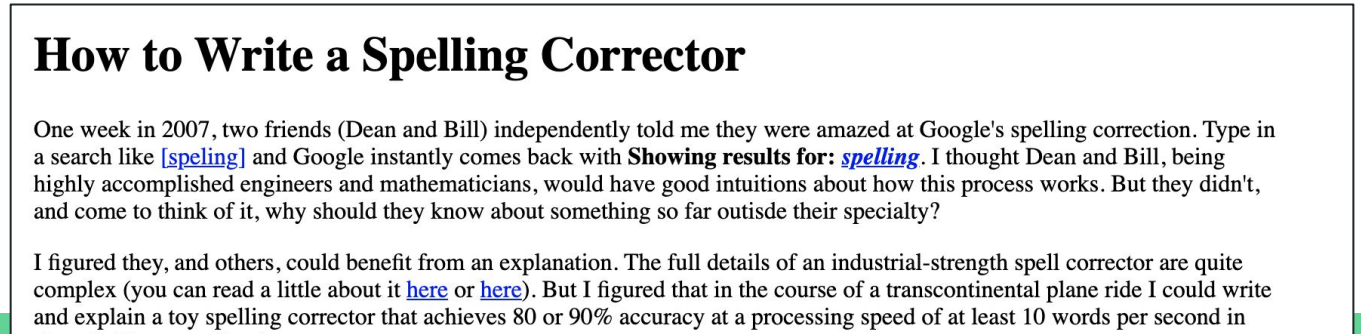
Dealing with rare events

METHOD #2: LINEAR INTERPOLATION

- Given <u, y, w>, we smooth **P(w | u, y)** by: λ_1 Trigram + λ_2 Bigram + λ_3 Unigram, where $\lambda_1 + \lambda_2 + \lambda_3 = 1$, where:
  - Trigram = Count(u, y, w) / Count(u, y)
  - Bigram = Count(y, w) / Count(y)
  - Unigram = Count(w) / |V|
- The intuition is that we use lower-level probabilities (as data is sparser in higher order) to compensate for our estimates when data is missing.
- **How do we pick the lambdas?** We use a *validation set* and pick the lambdas that maximizes the log probability over the dataset.

Application: typo-correction

- How to Write a Spelling Corrector (the “old” way)
- We model spell checking as a noisy channel:
  - Imagine a sender S sending a message M to a receiver R, *but M may be corrupted in the process.*
  - Example: S sends “apple” to R, but R receives “apkle” - R needs to be able to reliably recover “apple”
  - **Goal for R**: rank possible messages from S according to P(message | text I received)
  - Example: P(apple | apkle) > P(car | apkle) (**why?**)



How to Write a Spelling Corrector

One week in 2007, two friends (Dean and Bill) independently told me they were amazed at Google's spelling correction. Type in a search like [speling] and Google instantly comes back with **Showing results for:** ***spelling***. I thought Dean and Bill, being highly accomplished engineers and mathematicians, would have good intuitions about how this process works. But they didn't, and come to think of it, why should they know about something so far outisde their specialty?

I figured they, and others, could benefit from an explanation. The full details of an industrial-strength spell corrector are quite complex (you can read a little about it here or here). But I figured that in the course of a transcontinental plane ride I could write and explain a toy spelling corrector that achieves 80 or 90% accuracy at a processing speed of at least 10 words per second in

A noisy channel model

- Consider a vocabulary of t_1, t_2, ... t_n terms. *S* picks one term to send (the message, M = some *t*), R tries to decode what she receives (the actual string A).
- For all *t*, R needs to compute **P(*t* | A)**, that is, through Bayes: **P(t) X P(A | t),** and then pick the term with the highest probability.

- We need to estimate two terms then:
  - P(t): a language model (unigram, in fact), that is, the prior probability of “apple” vs “car” in the general language;
  - P(A | t): an error model, that is, the probability that, given that S really wanted to say “apple”, “apkle” resulted instead.

i.e. the “right” correction is a trade-off between popularity and possible mistakes

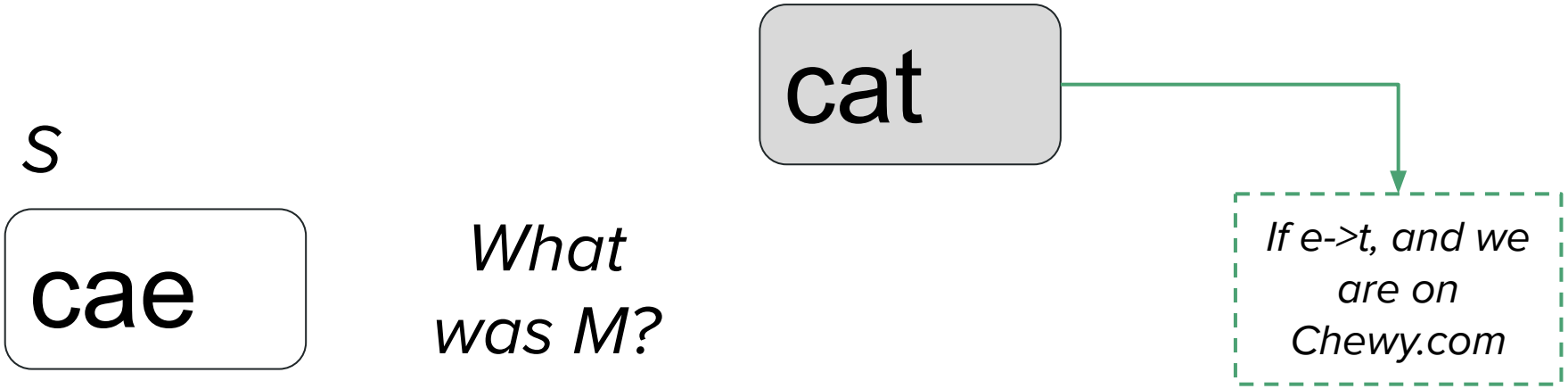
A noisy channel model

S

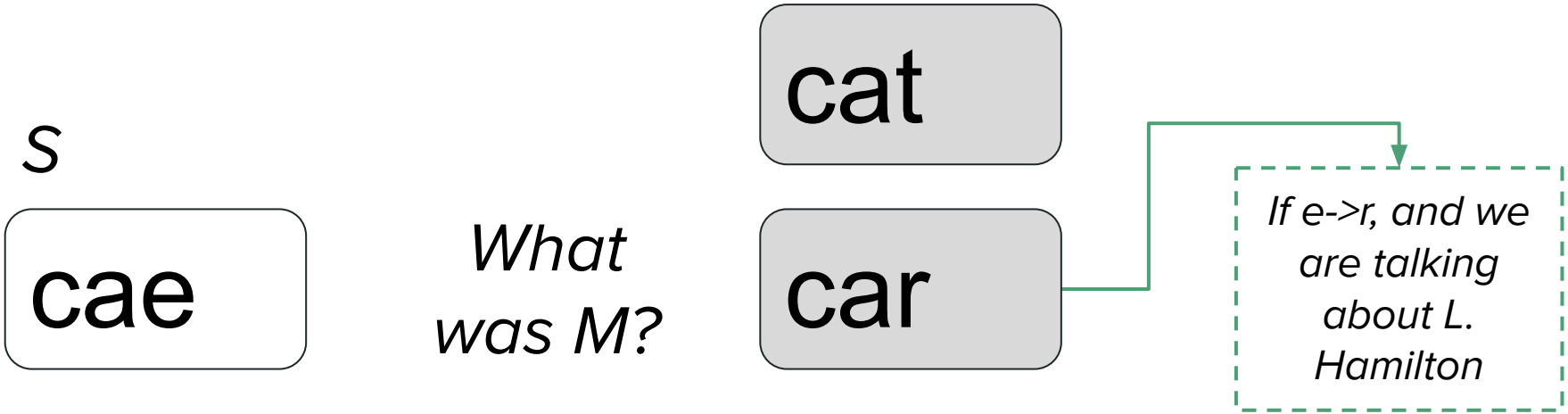
cae

What was M?

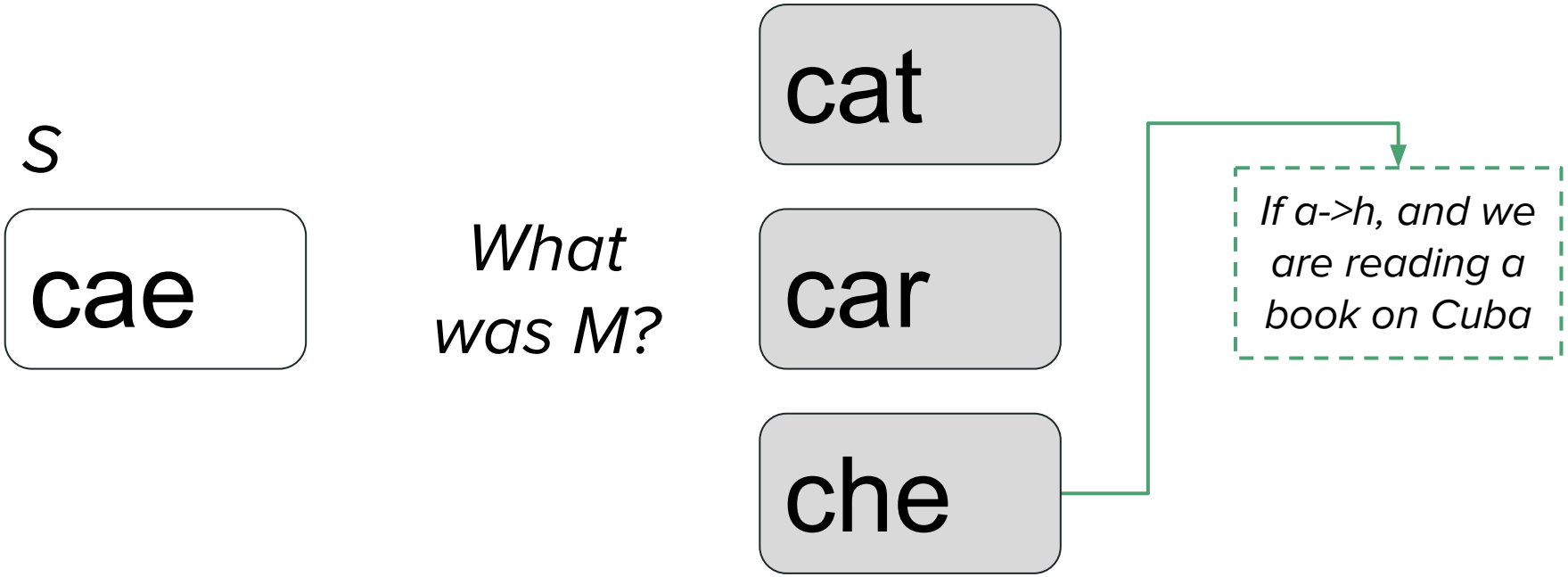
A noisy channel model



A noisy channel model



A noisy channel model



A noisy channel model

- Estimate the LM: *unigram* LM, that is, for each term *t* calculate P(t) as **Count(t) / # of tokens**
- Estimate the error model: “error model that says all known words of edit distance 1 are infinitely more probable than known words of edit distance 2, and infinitely less probable than a known word of edit distance 0”
  - Example: P(apkle | apple) = P(spple | apple) = P(appl | apple)
  - Example: P(apkle | apple) > P(apkles | apple)
- **Let’s check the notebook now to see how that looks in practice**.

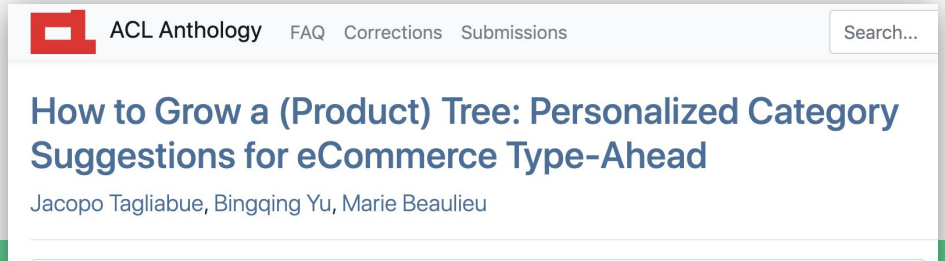
A noisy channel model - Homework hints

How can we go and improve upon Norvig's model?

- We can improve the language model: "cae" is more likely to be "car" than "cat" in the sentence "I drive a fast **cae**". What happens if we consider the context?
- We can improve the error model: given the QWERTY layout, some errors are more likely than others - can we incorporate this intuition in the spelling corrector?

Bonus: **advanced** noisy channel model

- Can we explicitly condition the language model depending on context?
- Type-ahead is a good example: for all completions Cs and query Q, we compute **P(c** | **Q)**, i.e. **argmax P(c) X P(Q** | **c).**
- **P(c)** becomes **P(c** | **context)**, so that the probability of a completion change based on the history of the user, her geolocation etc.



From words to vectors

- **Q**: We are used to feed “scikit models” with numbers for, say, regression, but how do we feed them *words*?
- **A**: We feed words by converting them to numbers!

Option #1: count vectorizer

Option #2: TF IDF vectorizer

One-hot encoding for words

Test corpus:

- “I live in NYC”
- “I teach in NYC”
- “I leave and teach in NYC”

Total of 6 words: [I, live, in, NYC, teach, and]

One-hot encoding for “I”:

1	0	0	0	0	0

One-hot encoding for “NYC”:

0	0	1	0	0	0

One-hot encoding for words

Test corpus:

- “I live in NYC”
- “I teach in NYC”
- “I leave and teach in NYC”

Total of 6 words: [I, live, in, NYC, teach, and]

One-hot encoding for “I”:

1	0	0	0	0	0

One-hot encoding for “NYC”:

0	0	1	0	0	0

What happens with a bigger corpus?

Count vectorizer

Test corpus:

- “I live in NYC”
- “I teach in NYC”
- “I leave and teach in NYC”

Total of 6 words: [I, live, in, NYC, teach, and]

Vector for “I teach in NYC”:

1	**0**	**1**	**1**	**1**	**0**

Q: what is the vector for “in NYC live I”?

Count vectorizer

- Visually, documents map to a point in the vocabulary space: in this example, when V=2, we can draw four Shakespeare plays and see that similar plays point to the same region of the space.
- How do we quantify “similar” then?

	As You Like It	Twelfth Night
battle	1	0
good	114	80
fool	36	58
wit	20	15

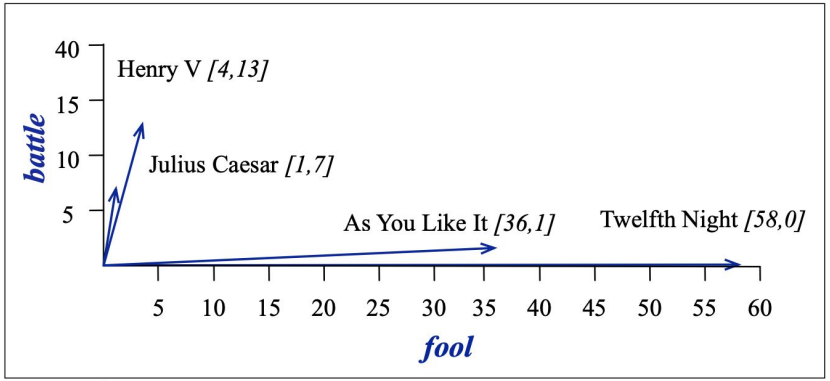


Figure 6.4 A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words *battle* and *fool*. The comedies have high values for the *fool* dimension and low values for the *battle* dimension.

Count vectorizer

- Visually, documents map to a point in the vocabulary space: in this example, when V=2, we can draw four Shakespeare plays and see that similar plays point to the same region of the space.
- How do we quantify “similar” then? -> **COSINE SIMILARITY!**

$$\text{cosine}(\mathbf{v},\mathbf{w}) = \frac{\mathbf{v}\cdot\mathbf{w}}{|\mathbf{v}||\mathbf{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

Dot Product

From counts to weights

- In the Count vectorizer, all words get the same importance
- Intuitively, some words however are more important than others: the presence of the word “growth” or “liability” in a financial article is more salient than generic words like “and” or “ company”.
- TF-IDF (“term frequency–inverse document frequency”) is an effective weighting scheme used in IR, text classification etc.

  tf-idf (term, document, corpus) = frequency (term, document) * idf (term, corpus)

  [typically idf = log (# documents / # documents with term)]

Q: how do we get a high tf-idf (term, document, corpus)?

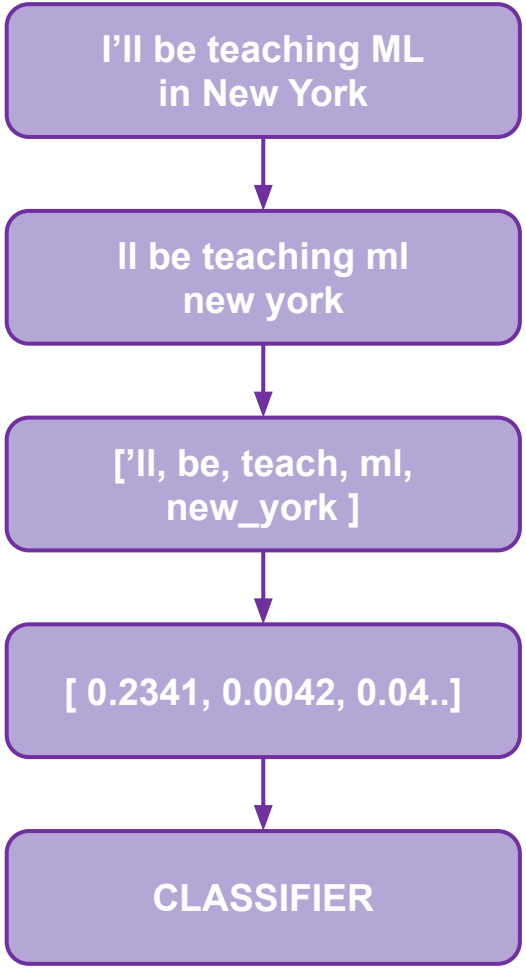
See the notebook on text classification for hand-on experience and tips on vectorization!

Application: text classification

- Text classification is one of the oldest tasks in NLP, and very relevant to Finance: for example, we may want to classify information based on a topic (“is this article about politics?”), or we may want to classify the general sentiment of a financial announcement (“is this tweet by Bank of America positive or negative?”)
- The general flow is similar to the “scikit patterns” you have seen earlier in the course, but with some caveats, required by the peculiar nature of text data.
- Make sure to check the notebook on text classification for some hand-on experience!

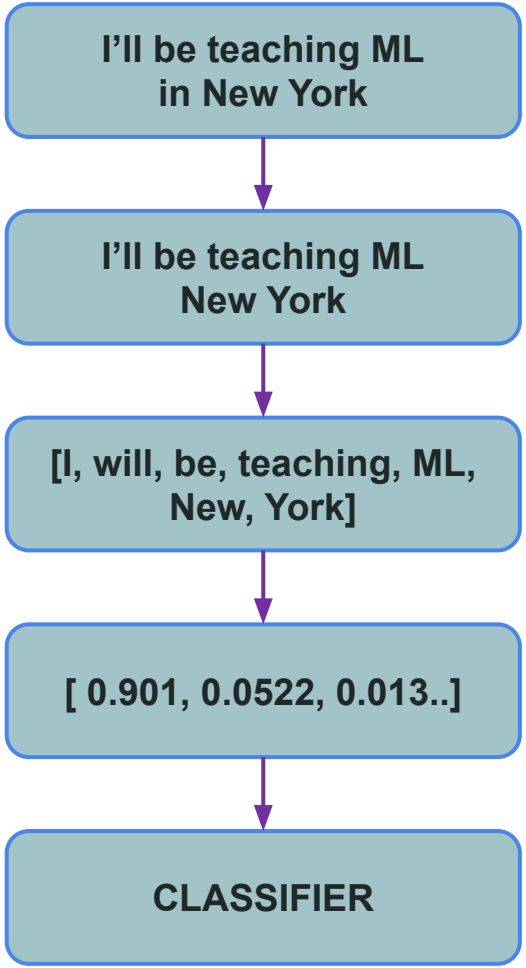
Text classification flow

- Pre-processing: casing (?), punctuation (?), stop words (?).
- Tokenization: split text in tokens { stemming (?), lemmatization (?), phrases (?) }.
- Vectorization: apply a vectorization technique (count or tf-idf) [Vocabulary size (?)]
- Modelling (training and testing): train a classifier model over text vectors - this is equivalent to your previous experience with scikit-like APIs.



Text classification flow

- Pre-processing: casing (?), punctuation (?), stop words (?).
- Tokenization: split text in tokens { stemming (?), lemmatization (?), phrases (?) }.
- Vectorization: apply a vectorization technique (count or tf-idf) [Vocabulary size (?)]
- Modelling (training and testing): train a classifier model over text vectors - this is equivalent to your previous experience with scikit-like APIs.



Additional materials

THEORY

- Aside from the classic book from Manning & Schutze, I love Michael Collins old notes on “foundational” NLP; the classic Russell & Norvig also contains some NLP stuff, and Bender’s book is a survey of linguistics concepts for NLP.
- For a more recent treatment, check out the excellent Jurafsky & Martin.

PRACTICE

- Good NLP libraries in Python: NLTK is where everybody starts; check out Spacy for a modern (and more opinionated) approach.