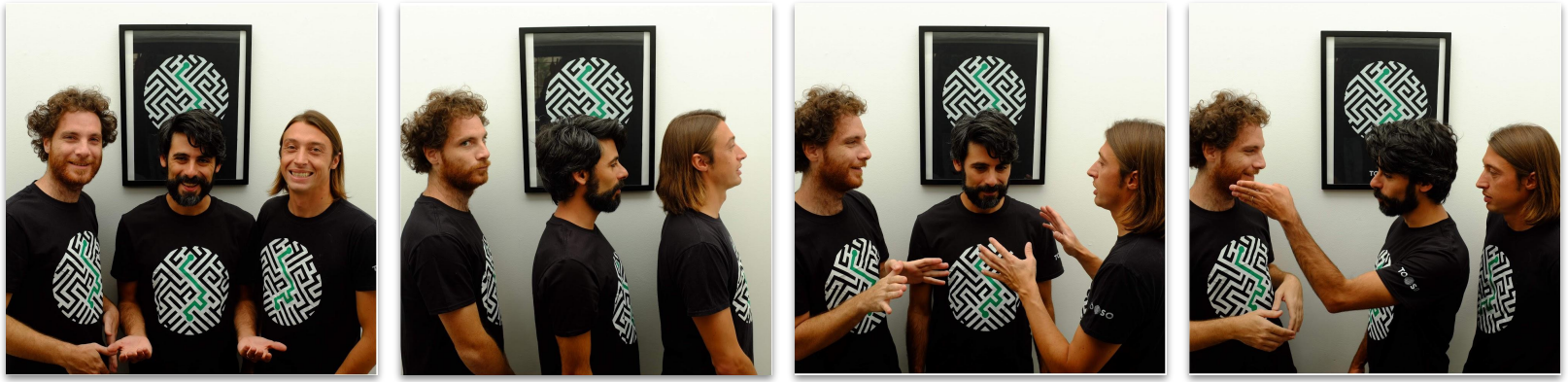
NLP and MLSys

FRE 7773, Fall 2021, Part II

Pleased to meet you (hope you guessed my name)

About me



About me

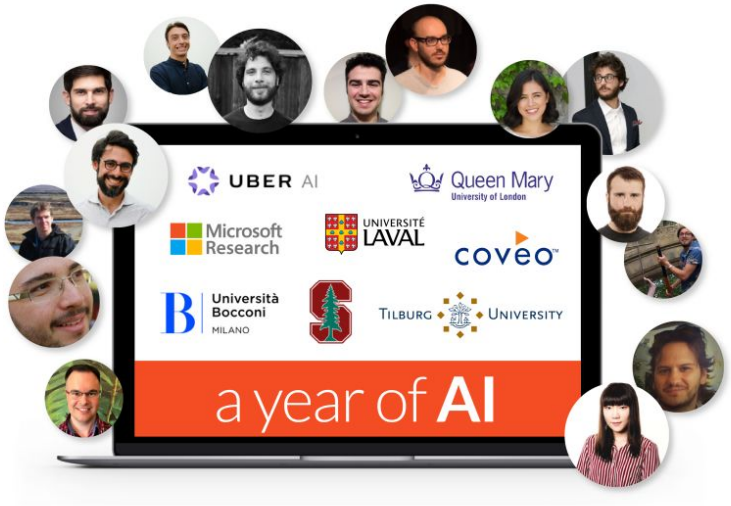
HOW IT STARTED

HOW IT IS GOING

JULY 9, 2019 | SAN FRANCISCO, CA AND QUEBEC CITY, QC

Coveo Acquires Tooso to Expand Its AI-powered Digital Commerce Technology

A.I. @ Coveo



RecSys KDD SIGIR 2020 THE WEB CONFERENCE TAIPEI 2020 ACL

COLING 2020 HCOMP wsdm NEURAL INFORMATION PROCESSING SYSTEMS ECAI

ACL Anthology FAQ Corrections Submissions Search...

How to Grow a (Product) Tree: Personalized Category Suggestions for eCommerce Type-Ahead

Jacopo Tagliabue, Bingqing Yu, Marie Beaulieu

Abstract

In an attempt to balance precision and recall in the search page, leading digital shops have been effectively nudging users into select category facets as early as in the type-ahead suggestions. In this work, we present SessionPath, a novel neural network model that improves facet suggestions on two counts: first, the model is able to leverage session embeddings to provide scalable personalization; second, SessionPath predicts facets by explicitly producing a probability distribution at each node in the taxonomy path. We benchmark SessionPath on two partnering

RESEARCH-ARTICLE

"An Image is Worth a Thousand Features": Scalable Product Representations for In-Session Type-Ahead Personalization

Authors: Bingqing Yu, Jacopo Tagliabue, Ciro Greco, Federico Bianchi

Authors Info & Affiliations

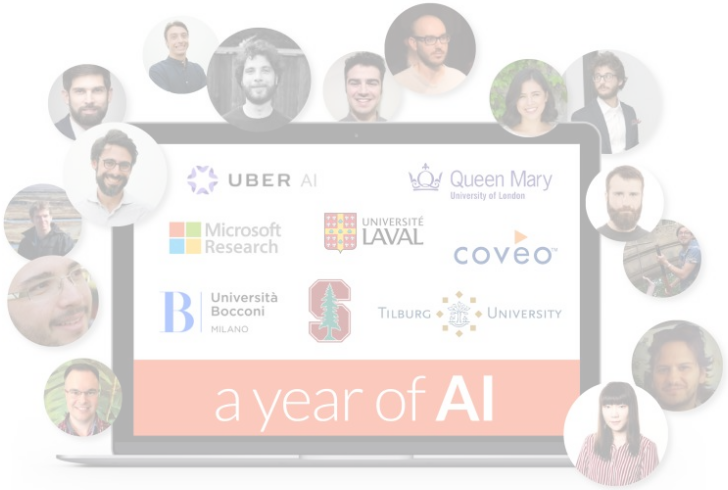
ABSTRACT

The Embeddings That Came in From the Cold: Improving Vectors for New and Rare Products with Content-Based Inference

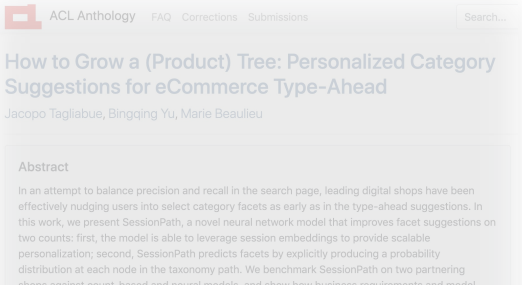
Authors: Jacopo Tagliabue, Bingqing Yu, Federico Bianchi

Authors Info & Affiliations

A.I. @ Coveo


UBER AI
Queen Mary
University of London
Microsoft Research
UNIVERSITÉ LAVAL
coveo
Università Bocconi
MILANO
TILBURG UNIVERSITY
a year of AI

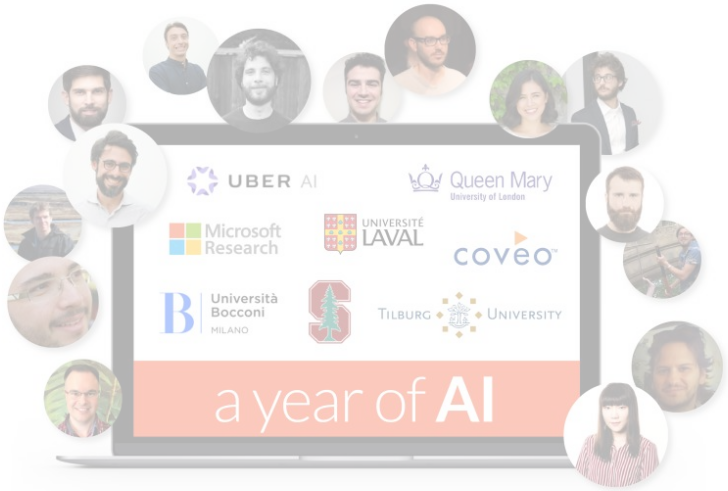

Congratulations to the recipients of the Best Industry Paper Award at #NAACL2021 Industry Track!
Query2Prod2Vec: Grounded Word Embeddings for eCommerce
Federico Bianchi @fb_vinid , Jacopo Tagliabue @jacopotagliabue, Bingqing Yu
Best Industry Paper Award
Congratulations to the recipients of the Best Industry Paper Award at NAACL 2021 Industry Track!
Query2Prod2Vec: Grounded Word Embeddings for eCommerce


ACL Anthology
How to Grow a (Product) Tree: Personalized Category Suggestions for eCommerce Type-Ahead
Jacopo Tagliabue, Bingqing Yu, Marie Beaulieu
Abstract


RESEARCH-ARTICLE
"An Image is Worth a Thousand Features": Scalable Product Representations for In-Session Type-Ahead Personalization
Authors: Bingqing Yu, Jacopo Tagliabue, Ciro Greco, Federico Bianchi
Authors Info & Affiliations


ABSTRACT
The Embeddings That Came in From the Cold: Improving Vectors for New and Rare Products with Content-Based Inference
Authors: Jacopo Tagliabue, Bingqing Yu, Federico Bianchi
Authors Info & Affiliations

A.I. @ Coveo



README.md

SIGIR eCOM 2021 Data Challenge Dataset

Public Data Release 1.0.0

Overview

Coveo hosted the 2021 SIGIR eCom Data Challenge and this repository contains utility scripts and the dataset, which is freely available for research purposes (see below): the paper introducing the Challenge is available as a pre-print.

The Data Challenge original README (containing baseline information, design papers, solutions, etc.) is archived in this repository as `README_DC_2021.md`. Background information about the Challenge, the motivations behind the release and some inspiring submissions can be found in the original paper, the archival section in `README_DC_2021.md` and the SIGIR presentation.

Note: there has been some issues when downloading the file using Safari; we suggest you to use Chrome for the download and sign-up process.

License

The dataset is available for research and educational purposes at this page. To obtain the dataset, you are required to fill a form with information about you and your institution, and agree to the Terms And Conditions for fair usage of the data. For convenience, Terms And Conditions are also included in a pure `txt` format in this repo: usage of the data implies the acceptance of these Terms And Conditions.

ACL Anthology FAQ Corrections Submissions Search...

How to Grow a (Product) Tree: Personalized Category Suggestions for eCommerce Type-Ahead

Jacopo Tagliabue, Bingqing Yu, Marie Beaulieu

Abstract

In an attempt to balance precision and recall in the search page, leading digital shops have been effectively nudging users into select category facets as early as in the type-ahead suggestions. In this work, we present SessionPath, a novel neural network model that improves facet suggestions on two counts: first, the model is able to leverage session embeddings to provide scalable personalization; second, SessionPath predicts facets by explicitly producing a probability distribution at each node in the taxonomy path. We benchmark SessionPath on two partnering

RESEARCH-ARTICLE

"An Image is Worth a Thousand Features": Scalable Product Representations for In-Session Type-Ahead Personalization

Authors: Bingqing Yu, Jacopo Tagliabue, Ciro Greco, Federico Bianchi

Authors Info & Affiliations

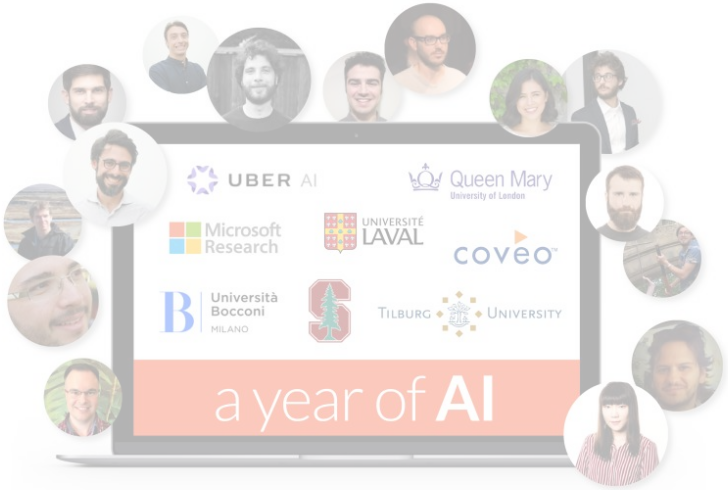
ABSTRACT

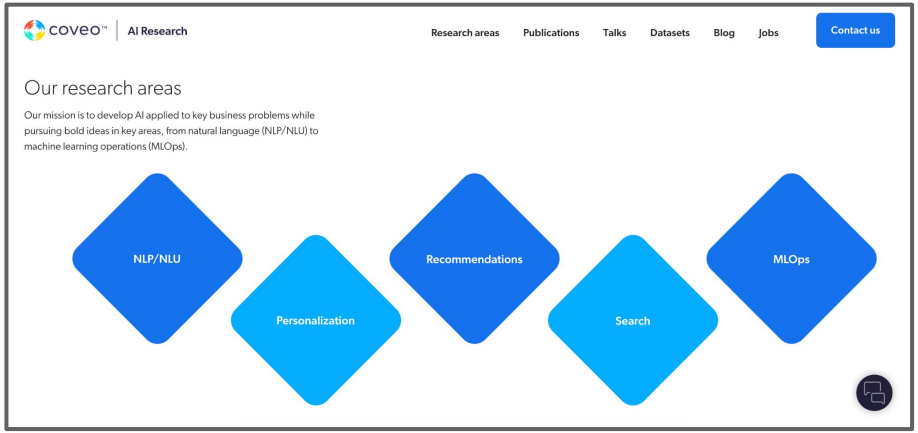
The Embeddings That Came in From the Cold: Improving Vectors for New and Rare Products with Content-Based Inference

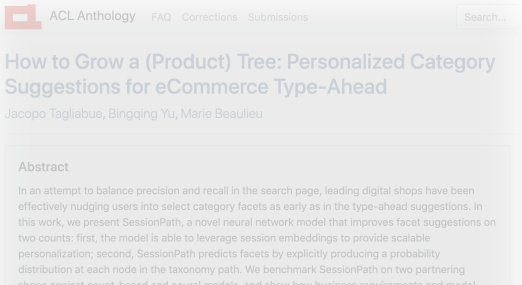
Authors: Jacopo Tagliabue, Bingqing Yu, Federico Bianchi

Authors Info & Affiliations

A.I. @ Coveo






ACL Anthology FAQ Corrections Submissions Search...

How to Grow a (Product) Tree: Personalized Category Suggestions for eCommerce Type-Ahead

Jacopo Tagliabue, Bingqing Yu, Marie Beaulieu

Abstract

In an attempt to balance precision and recall in the search page, leading digital shops have been effectively nudging users into select category facets as early as in the type-ahead suggestions. In this work, we present SessionPath, a novel neural network model that improves facet suggestions on two counts: first, the model is able to leverage session embeddings to provide scalable personalization; second, SessionPath predicts facets by explicitly producing a probability distribution at each node in the taxonomy path. We benchmark SessionPath on two partnering


RESEARCH-ARTICLE

"An Image is Worth a Thousand Features": Scalable Product Representations for In-Session Type-Ahead Personalization

Authors: Bingqing Yu, Jacopo Tagliabue, Ciro Greco, Federico Bianchi

Authors Info & Affiliations


ABSTRACT

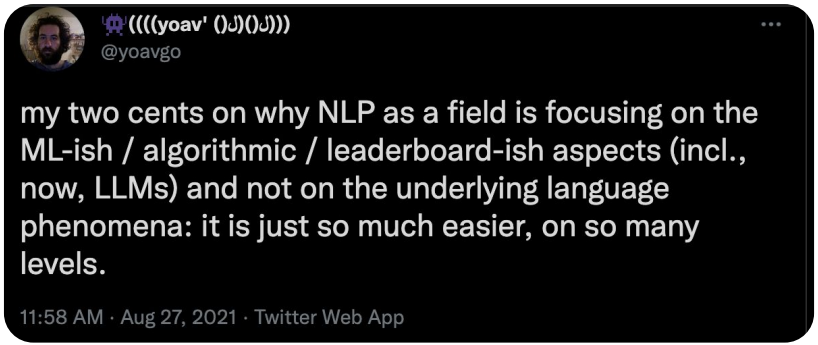
The Embeddings That Came in From the Cold: Improving Vectors for New and Rare Products with Content-Based Inference

Authors: Jacopo Tagliabue, Bingqing Yu, Federico Bianchi

Authors Info & Affiliations

About this section of FRE 7773

1. NLP is cool and only superficially similar to other ML tasks.
    a. Rare (in fact “impossible”) events are important.
    b. Small-data learning is crucial in the only system we know that is good at NLP.
    c. Compared to other modalities, language is less understood (what is *meaning* by the way?).
    d. Practical progress in language tasks may be completely unrelated to our understanding of language *per se*.



About this section of FRE 7773

1. NLP is cool, and only superficially similar to other ML tasks.
    a. Rare (in fact “impossible”) events are important.
    b. Small-data learning is crucial in the only system we know that is good at NLP.
    c. Compared to other modalities, language is less understood (what is *meaning* by the way?).
    d. Practical progress in language tasks may be completely unrelated to our understanding of language *per se*.
2. Models are important, but what is around them matters just as much.
    a. If a ~~tree falls in a forest~~ model is trained and no one ~~is around to hear it~~ gets its predictions, does it make ~~a sound~~ an impact? (TL;DR: not much!)
    b. As the market evolves quickly, *you* will be judged by your modelling, prototyping *and* engineering skills, so some fluency in these topics is a huge career advantage.

What this is NOT

1. **A theory-heavy Deep Learning course**: we do *some* deep learning, like all the cool kids do, but we emphasize an intuitive and pragmatic understanding of it.
2. **A full-fledged NLP course**: we discuss few topics in NLP, based on 1) my opinionated view of what is important / feasible to teach, 2) an “evolutionary perspective”, in which we go back to the same topic multiple times, and reflect on the historical development of the field.
3. **A software engineering course**: we try to talk about the world *outside* the classroom, but we won’t have time to teach *everything explicitly*; I expect you to spend time on your own to tinker with the code, explore the additional readings and Google your way out of programming issues (like all professionals do!)

What this is NOT

As a general rule, there is a lot of excellent educational material on NLP/DL, so we (mostly) spend our time discussing what fewer people are teaching.

Practicalities

1. Slides contain a lot and not much at the same time: **a lot**, as you will find dozens of links, connection, references to satisfy your curiosity and improve your understanding; **not much**, as the code and our live lectures / commentary will add a ton of useful information that makes your life easier.
2. Code: **send me ASAP your GitHub account (or create a free one if you don't have one!)**
3. Cloud access: we partnered up with AWS to give you access to a real cloud environment for your project.
4. Tool access: we partnered up with Comet - a NYC startup - to give you free access to their experiment tracking tool.
5. Evaluation: we are pretty light on homework, as we encourage you to spend more time on the project that will constitute the final assignment.

A bird-eye view

Natural Language Processing

1. A tiny bit history
2. Why is NLP useful? Use cases (and finance!)
3. What are we going to learn together?

NLP: A long time ago, in a galaxy far far away....

“**Natural language processing (NLP)** is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data.”

NLP: A long time ago, in a galaxy far far away....

"**Natural language processing (NLP)** is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data."

1.2.1 Questions that linguistics should answer

What questions does the study of language concern itself with? As a start we would like to answer two basic questions:

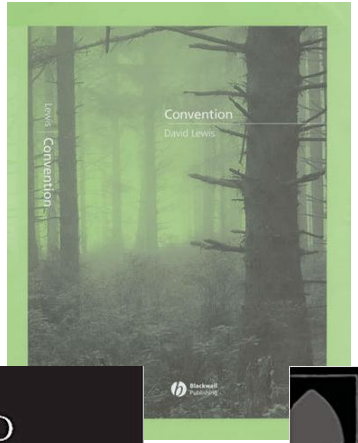
- What kinds of things do people say?
- What do these things say/ask/request about the world?

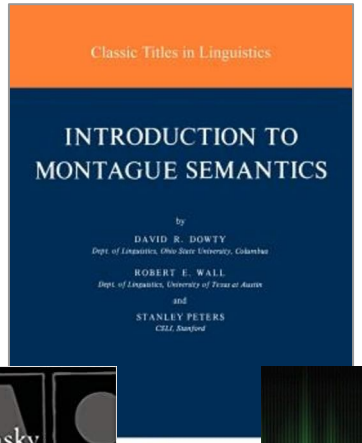
NLP: A long time ago, in a galaxy far far away....

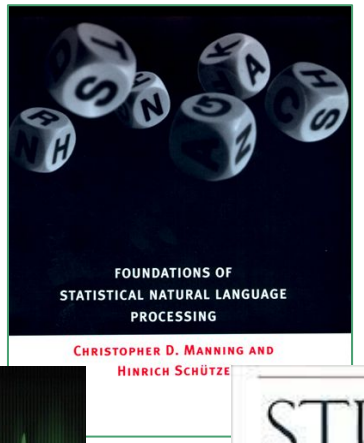
A bit more generally, NLP can be viewed as a computational approach to *language*: if we treat language as a cognitive phenomenon (and, implicitly, cognition as a computation), it is natural to frame language questions (how do we understand a sentence? How do you say “dog” in Italian?, etc.) as computational ones, which, for all sorts of practical reasons, can then be answered through computers.

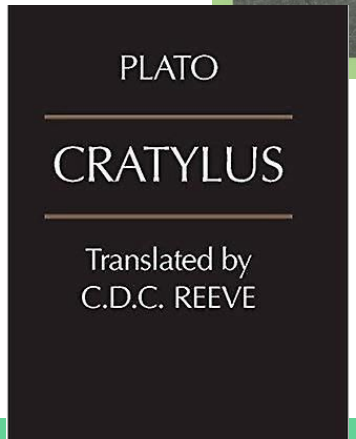
“Computer science is no more about computers than astronomy is about telescopes”

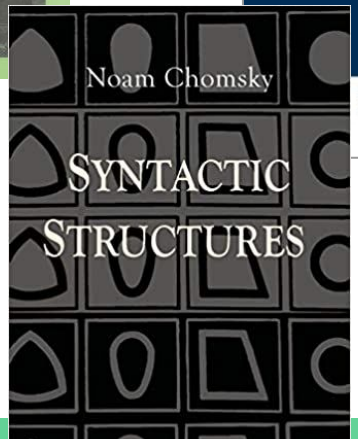
NLP: A long time ago, in a galaxy far far away….

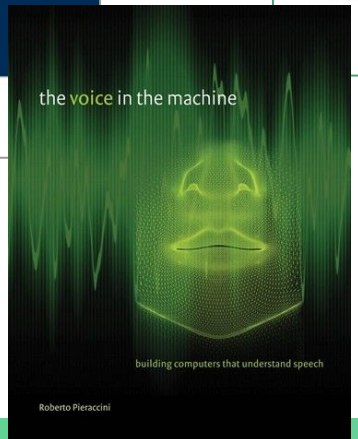


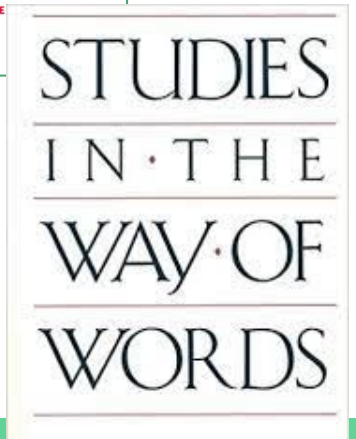










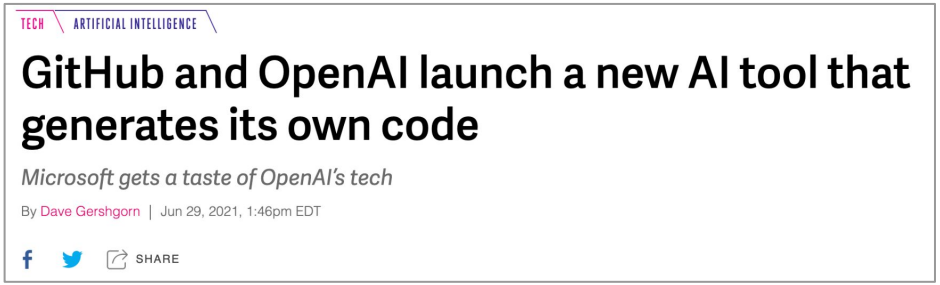


The “Golden Era” of NLP

- NLP systems have made **tremendous progress** in the last 2 years, and there is widespread excitement for the capabilities of “large language models”.

Meet GPT-3. It Has Learned to Code (and Blog and Argue).

The latest natural-language system generates tweets, pens poetry, summarizes emails, answers trivia questions, translates languages and even writes its own computer programs.



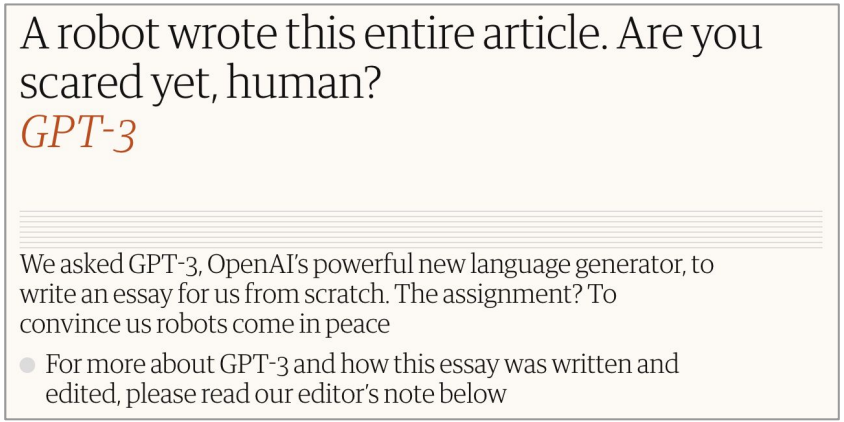
TECH | ARTIFICIAL INTELLIGENCE

GitHub and OpenAI launch a new AI tool that generates its own code

Microsoft gets a taste of OpenAI’s tech

By Dave Gershgorn | Jun 29, 2021, 1:46pm EDT

SHARE



A robot wrote this entire article. Are you scared yet, human?

GPT-3

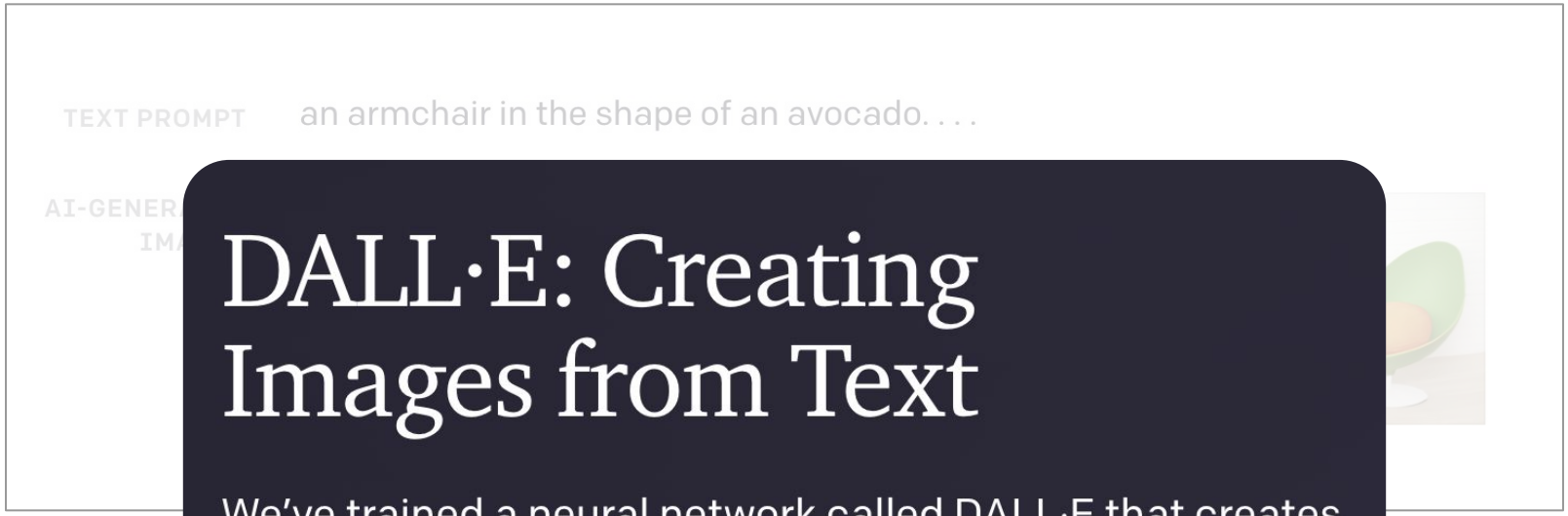
We asked GPT-3, OpenAI’s powerful new language generator, to write an essay for us from scratch. The assignment? To convince us robots come in peace

- For more about GPT-3 and how this essay was written and edited, please read our editor’s note below

The “Golden Era” of NLP

“It is impossible to review the specifics of your tenure file without becoming enraptured by the vivid accounts of your life. However, it is not a life that will be appropriate for a member of the faculty at Indiana University, and **it is with deep regret that I must deny your application for tenure**. … Your lack of diplomacy, your flagrant disregard for the feelings of others,(…), and, frankly, the fact that you often take the side of the oppressor, **leads us to the conclusion that you have used your tenure here to gain a personal advantage and have failed to adhere to the ideals of this institution**.”

The “Golden Era” of NLP



The “Golden Era” of NLP

TEXT PROMPT an armchair in the shape of an avocado. . . .

AI-GENERATED IMAGES

Edit prompt or view more images↓

We've trained a neural network called DALL·E that creates images from text captions for a wide range of concepts expressible in natural language.

Not all that glitters is gold

a cactus in a green field

a person flying through the air while riding skis

commons.wikimedia.org/w/index.php?cu...

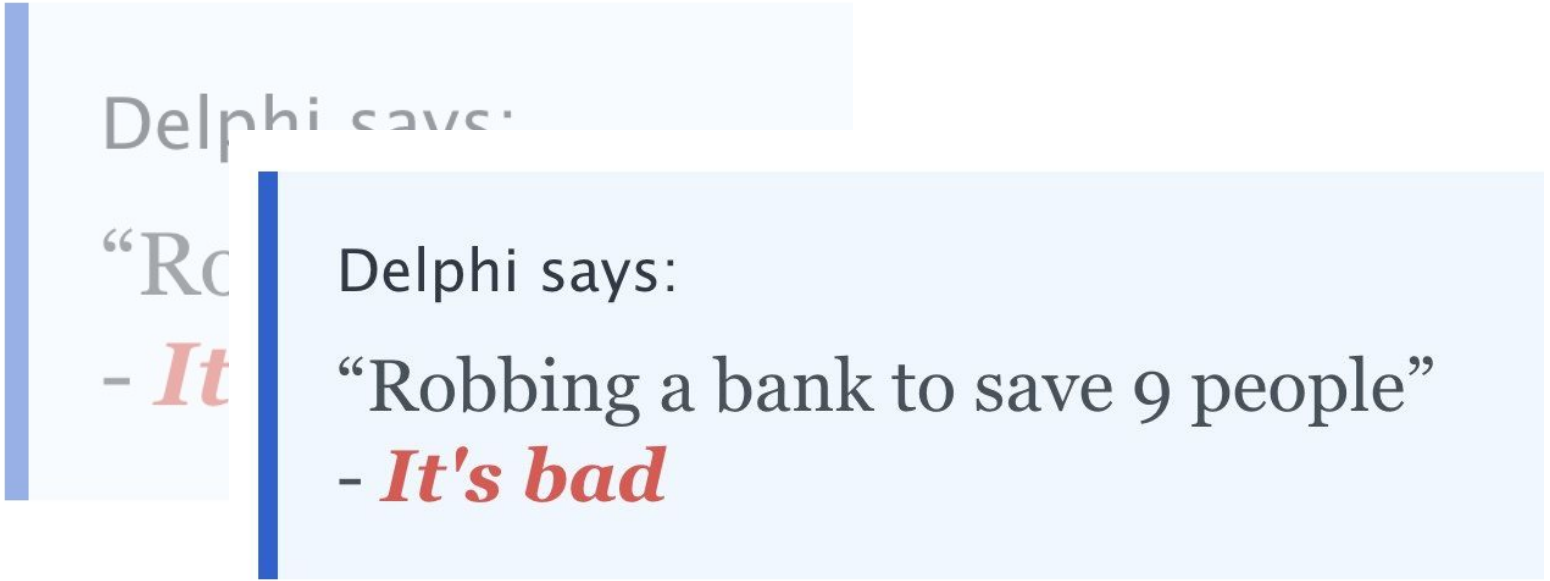
Not all that glitters is gold

a dinosaur on top of a surfboard

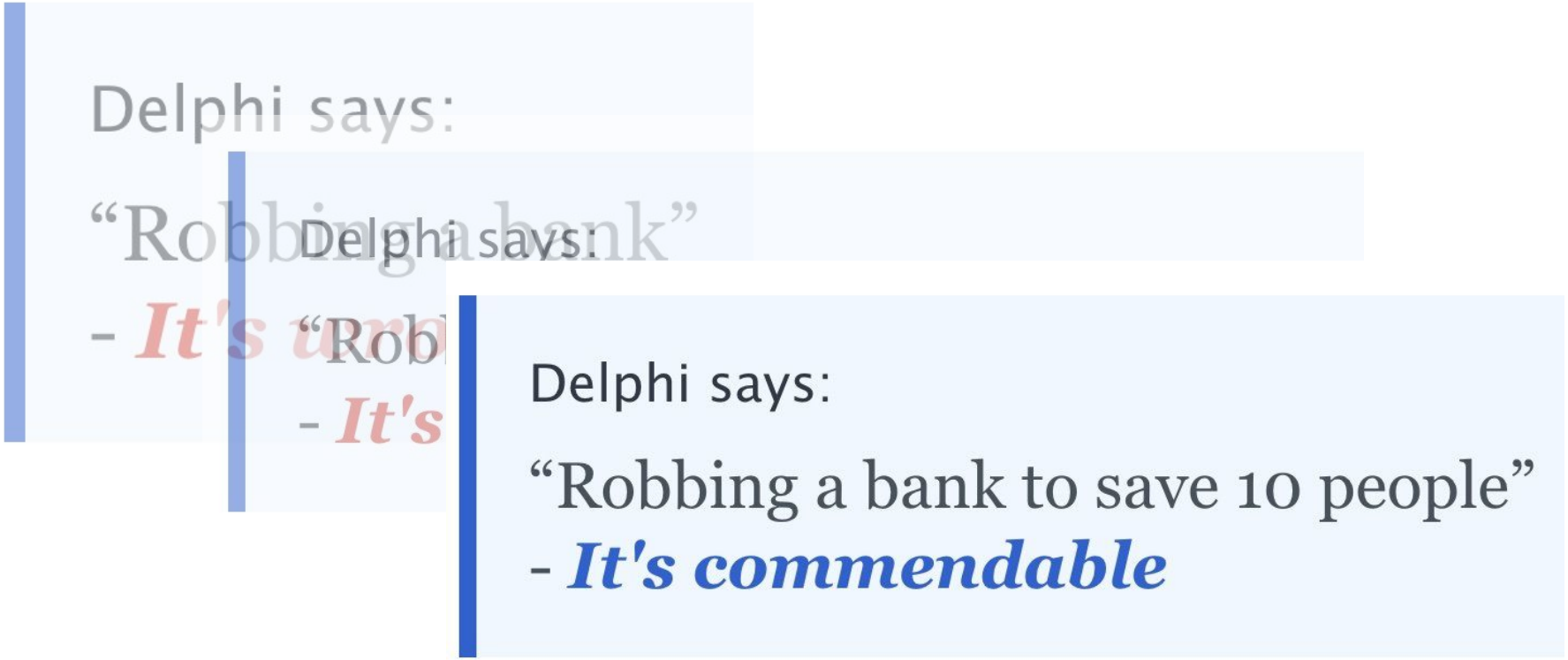
Not all that glitters is gold (NLP 2021 Edition)

> Delphi says:
>
> “Robbing a bank”
> - *It's wrong*

Not all that glitters is gold (NLP 2021 Edition)



Not all that glitters is gold (NLP 2021 Edition)

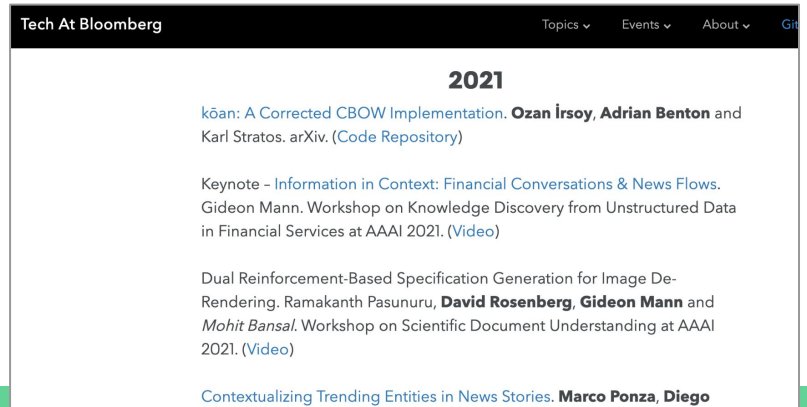


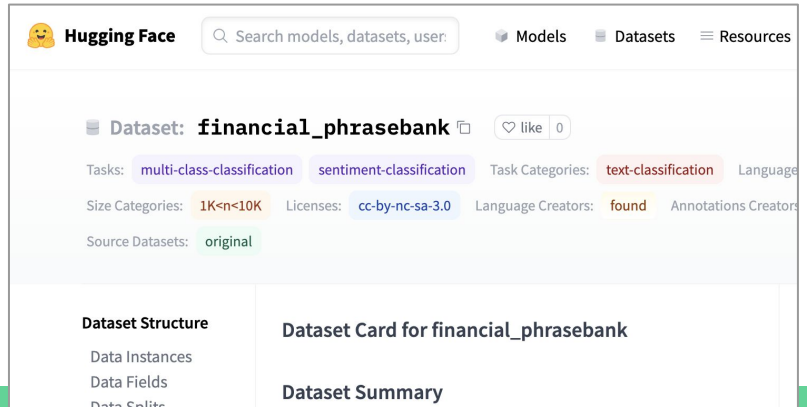
NLP: Use Cases

- NLP is a very broad field (**VERY BROAD**), and encompasses tons of research topics and countless applications. A quick tour of Arxiv, or browsing the program of a top NLP conference (ACL, NAACL, EMNLP, etc.), will feature tasks as diverse as:
  - text classification
  - text summarization
  - image captioning
  - machine translation
  - text generation

NLP in Finance

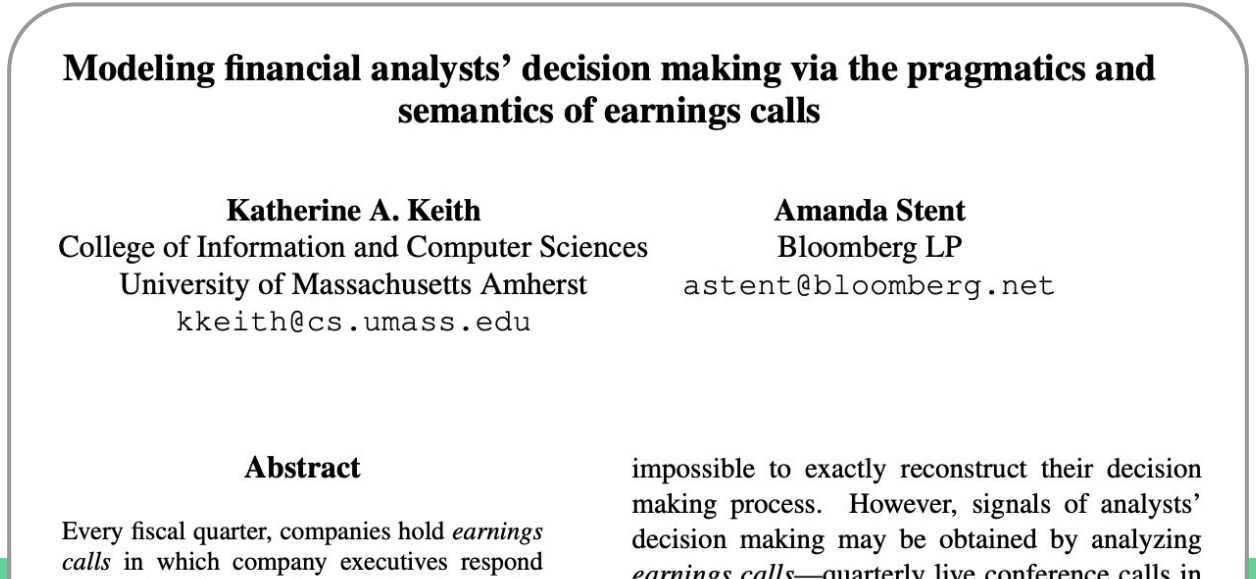
- Even when considering specific finance use cases, it is easy to see the variety and the centrality of NLP for a modern understanding of the industry:
  - sentiment analysis of finance news
  - stock market prediction
  - document classification





NLP in Finance - A Research Example

- “Modeling financial analysts’ decision making via the pragmatics and semantics of earnings calls”, from ACL 2019.



Modeling financial analysts’ decision making via the pragmatics and semantics of earnings calls

Katherine A. Keith
College of Information and Computer Sciences
University of Massachusetts Amherst
kkeith@cs.umass.edu

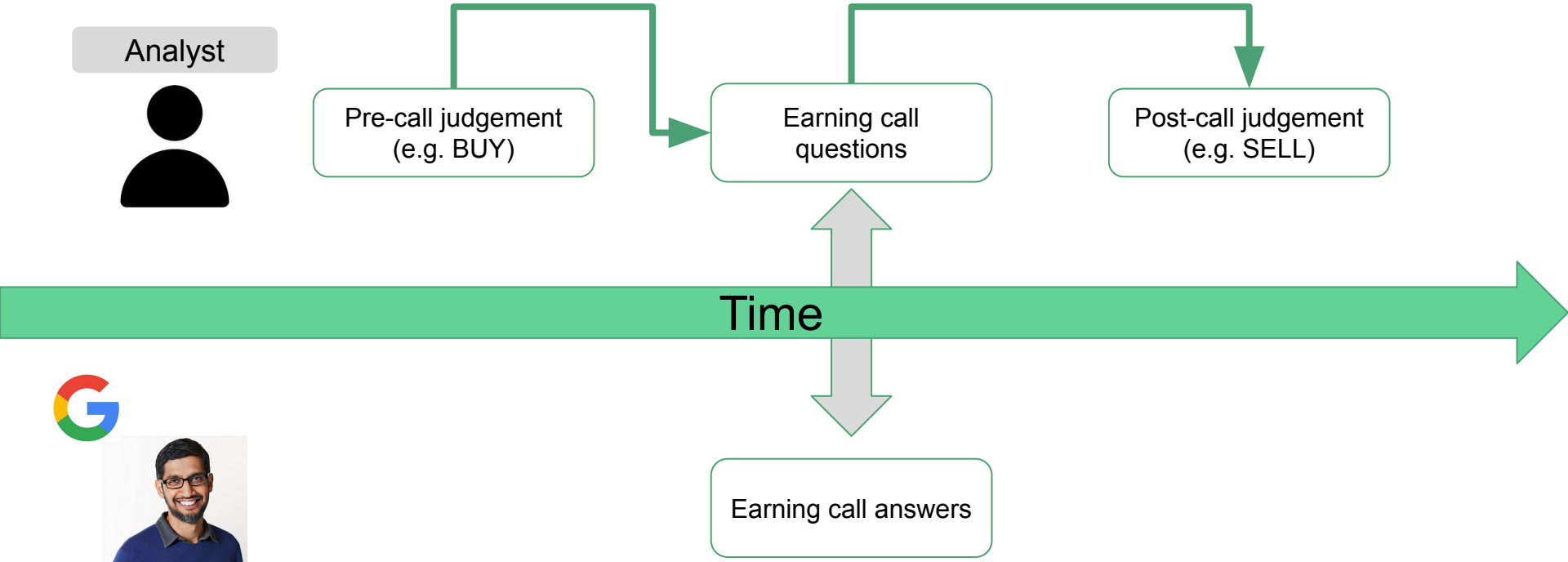
Amanda Stent
Bloomberg LP
astent@bloomberg.net

Abstract

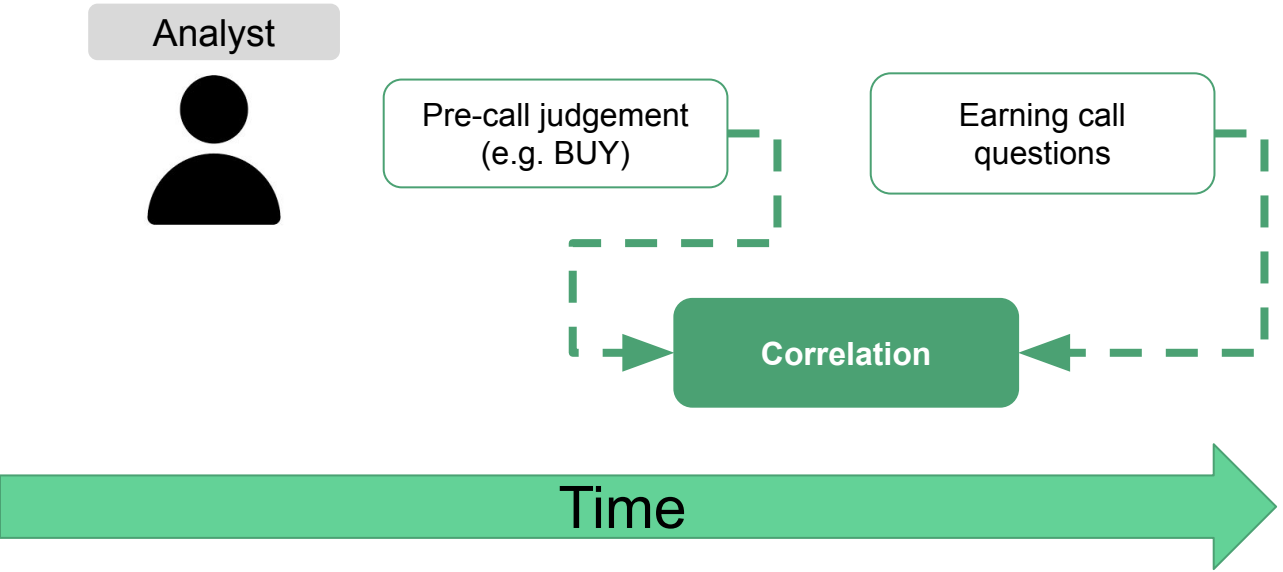
Every fiscal quarter, companies hold *earnings calls* in which company executives respond

impossible to exactly reconstruct their decision making process. However, signals of analysts’ decision making may be obtained by analyzing *earnings calls*—quarterly live conference calls in

NLP in Finance - A Research Example



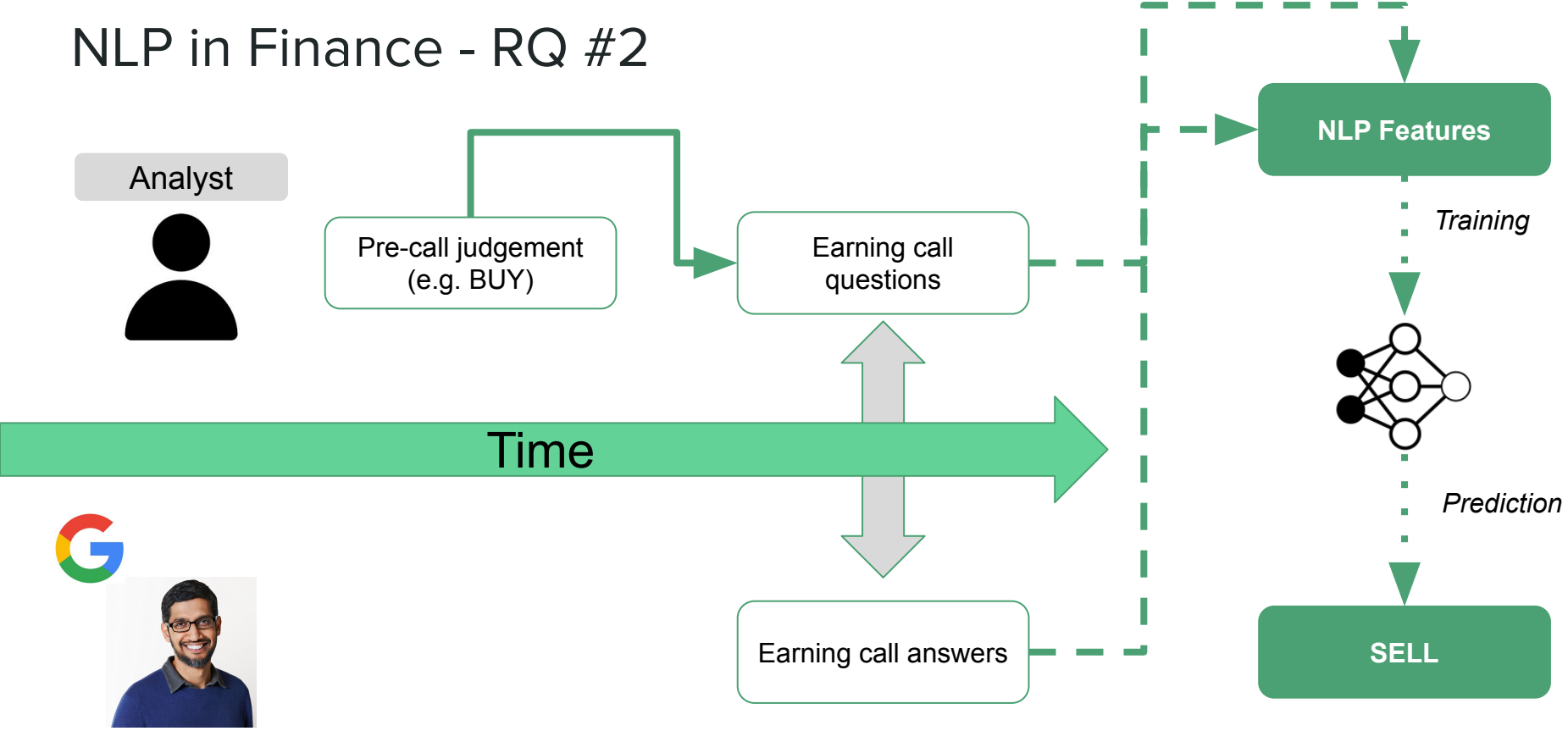
NLP in Finance - Research Question #1



No.	Feature	Pearson's r	p-value
1	Named entities event	0.0041	0.0999
2	Named entities number	0.0064	0.0099
3*	Named entities org.	0.0185	$< 1e^{-4}$
4*	Named entities person	0.0247	$< 1e^{-4}$
5	Named entities product	0.0022	0.3777
6*	Concreteness ratio	0.0115	$< 1e^{-4}$
7*	Num past preds	−0.0086	0.0006
8	Num present preds	0.0052	0.0378
9	Num future preds	0.0033	0.1914
10*	Sentiment positive	0.0162	$< 1e^{-4}$
11*	Sentiment negative	−0.0104	$< 1e^{-4}$
12	Hedging	0.0017	0.5019
13	Modal	0.0075	0.0028
14	Uncertainty	0.0055	0.0287
15	Constraining	0.0005	0.8399
16	Litigiousness	−0.0072	0.0037
17*	Turn order	−0.1034	$< 1e^{-4}$
18	Num. tokens	0.0050	0.0459
19	Num predicates	0.0011	0.6692
20	Num sents.	0.0043	0.0854

Table 4: Results from Pearson correlations of pragmatic lexical features from §4.1 and prior-to-call labels of analysts, (*bearish*, *neutral*, or *bullish*). Statistical significance after Bonferroni correction is marked by (*) for $p < 0.0025$. Total 160,816 question turns.

NLP in Finance - RQ #2



NLP: Our Plan

- Language 101:
  - Syntax, semantics and pragmatics.
- NLP fundamentals:
  - Language modelling.
  - Lexical representation.
- Neural NLP:
  - Introduction to neural networks.
  - Lexical representation, revisited.
- Large Pre-Trained Models:
  - Language modelling, revisited.
  - Text classification, revisited.

Additional NLP readings (for things we can't talk about now)

- **Language and cognition**
  - **Handbooks of psycholinguistics**
  - **NLP and language universals**
- **Language in animals**
  - **Language evolution** (also **here**)
  - **Monkeys** vs **Birdsongs**

MLSys

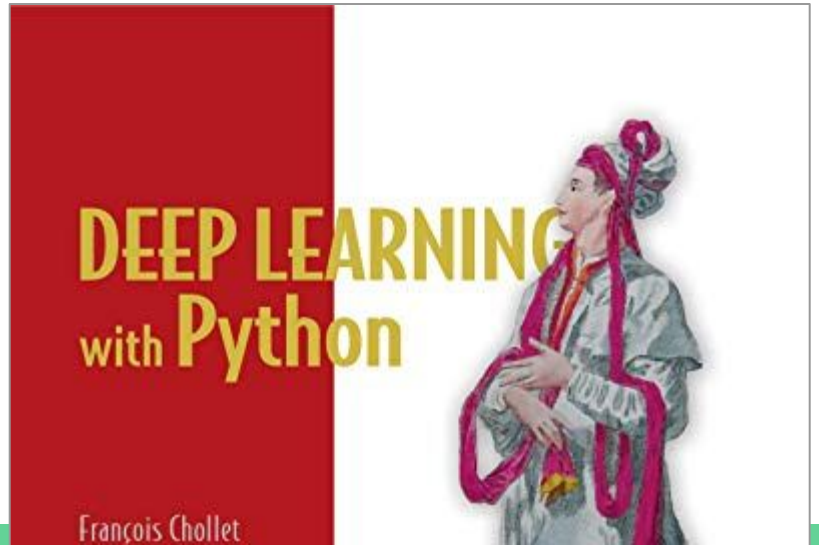
1. A tiny bit history
2. Why is MLSys useful? Use cases and finance
3. What are we going to learn together?

MLSys: A (NOT SO) long time ago, in a galaxy far far away....

1. Google/Facebook started open-sourcing frameworks to democratize statistical / deep learning.

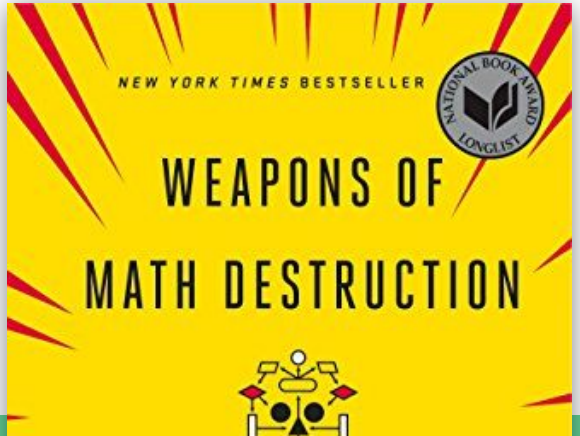
MLSys: A (NOT SO) long time ago, in a galaxy far far away....

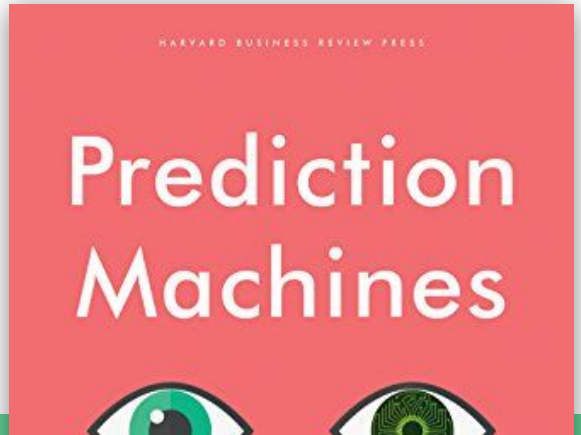
1. Google/Facebook started open source frameworks to democratize statistical / deep learning.
2. “Everybody” got excited, and started applying ML to a variety of use cases.



MLSys: A (NOT SO) long time ago, in a galaxy far far away....

1. Google/Facebook started open source frameworks to democratize statistical learning.
2. “Everybody” got excited, and started applying ML to a variety of use cases.
3. While scholars debate on perils and opportunities of the “A.I.”-era...





MLSys: A (NOT SO) long time ago, in a galaxy far far away....

1. Google/Facebook started open source frameworks to democratize statistical learning.
2. “Everybody” got excited, and started applying ML to a variety of use cases.
3. While scholars debate on perils and opportunities of the “A.I.”-era... the reality of ML *outside of Big Tech* is not as flashy as you may believe, as *they* figured out how to get value out of ML systems....

TensorFlow-Serving:
Flexible, High-Performance ML Serving

Christopher Olston
olston@google.com

Noah Fiedel
nfiedel@google.com

Kiril Gorovoy
kgorovoy@google.com

Jeremiah Harmsen
jeremiah@google.com

Li Lao
llao@google.com

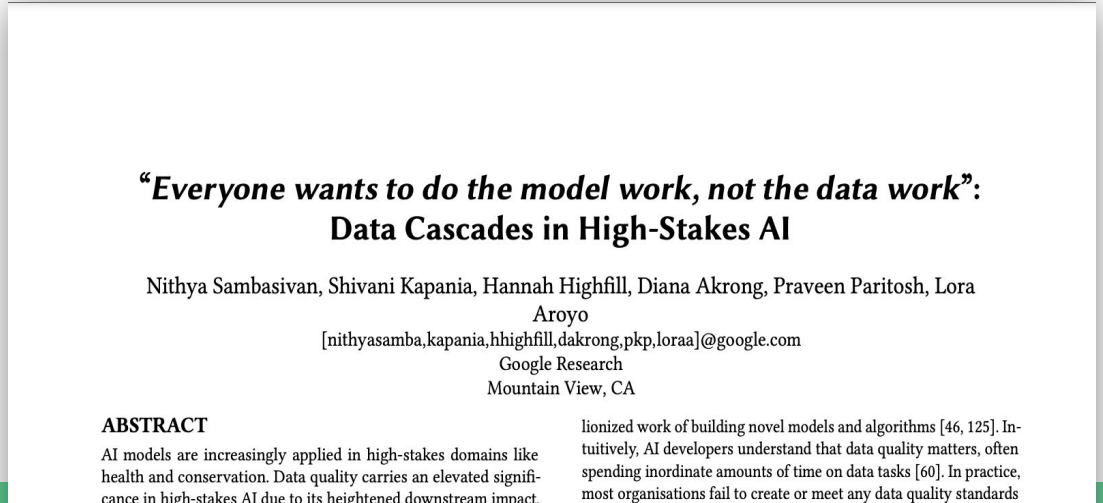
Fangwei Li
fangweil@google.com

MLSys: A (NOT SO) long time ago, in a galaxy far far away....

1. Google/Facebook started open source frameworks to democratize statistical learning.
2. “Everybody” got excited, and started applying ML to a variety of use cases.
3. While scholars debate on perils and opportunities of the “A.I.”-era... the reality of ML *outside of Big Tech* is not as flashy as you may believe, as *they* figured out how to get value out of ML systems... but did *everybody else*?

MLSys: Use Cases

- Models are a tiny part of ML platforms, and often the least problematic (with some *caveat*);
- while everybody wants to do the model work, data work is often equally (or more) important in practice.

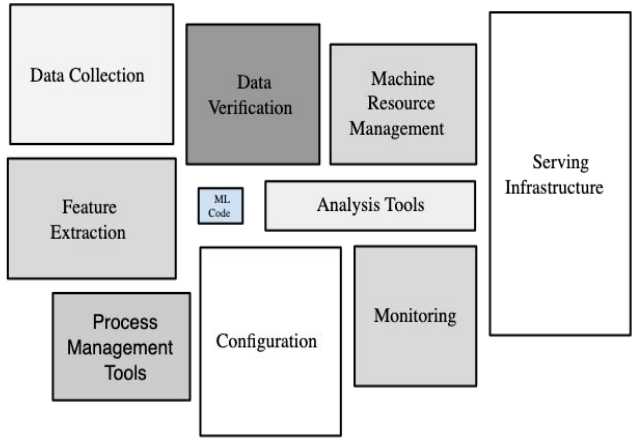


"*Everyone wants to do the model work, not the data work*": Data Cascades in High-Stakes AI

Nithya Sambasivan, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Paritosh, Lora Aroyo
[nithyasamba,kapania,hhighfill,dakrong,pkp,loraa]@google.com
Google Research
Mountain View, CA

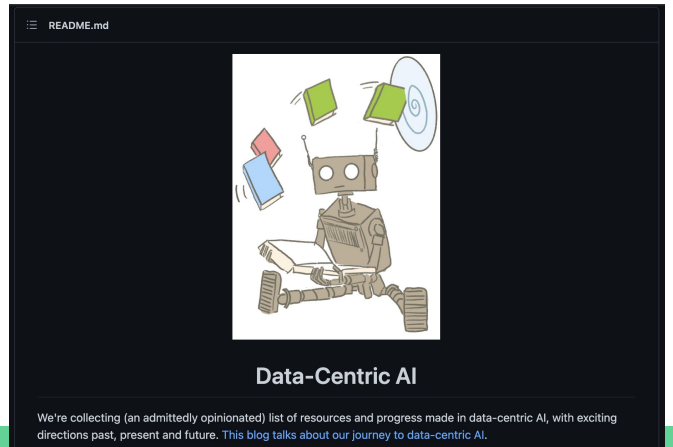
ABSTRACT

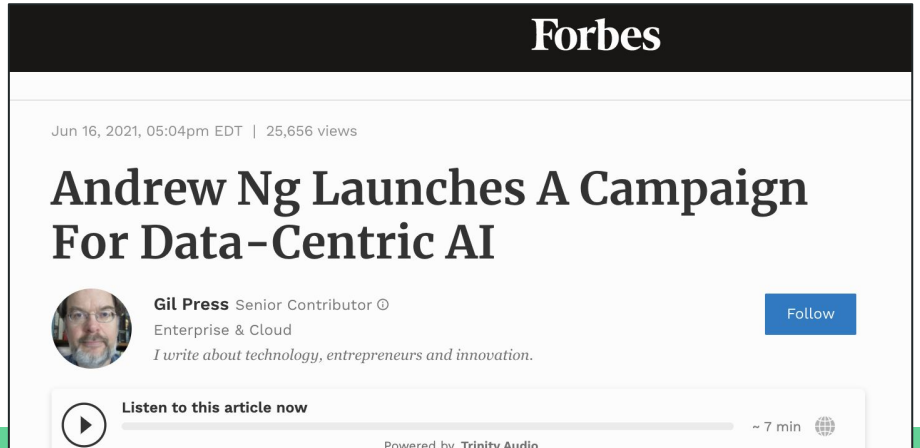
AI models are increasingly applied in high-stakes domains like health and conservation. Data quality carries an elevated significance in high-stakes AI due to its heightened downstream impact, ... lionized work of building novel models and algorithms [46, 125]. Intuitively, AI developers understand that data quality matters, often spending inordinate amounts of time on data tasks [60]. In practice, most organisations fail to create or meet any data quality standards



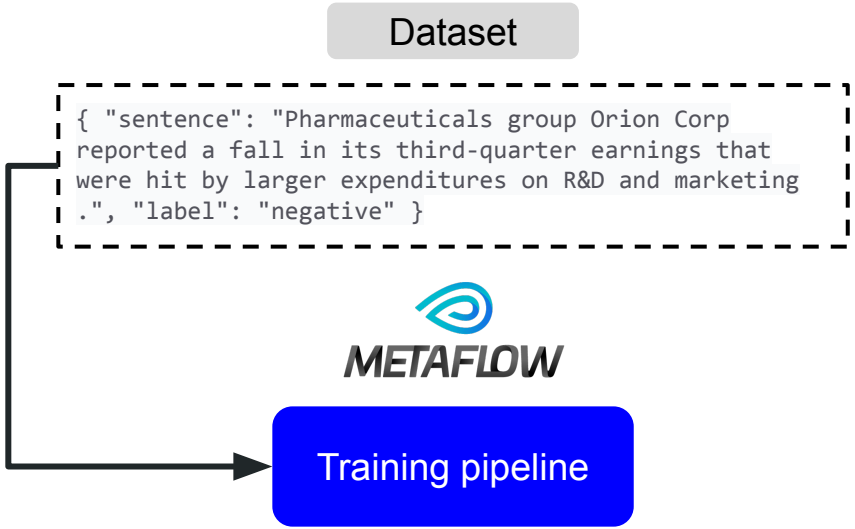
MLSys: the rise of data-driven AI

- High-profile researchers (e.g. Chris Re, Andrew Ng) had started explicitly addressing the data problem as a fundamental research question in building reliable A.I. systems.
- Historically, we present ML systems holding fixed a dataset as input and varying models for better output: there is a growing body of work on best practices that take the model as input (or a class of model) and investigate what happens when data changes (in quantity, quality etc.).

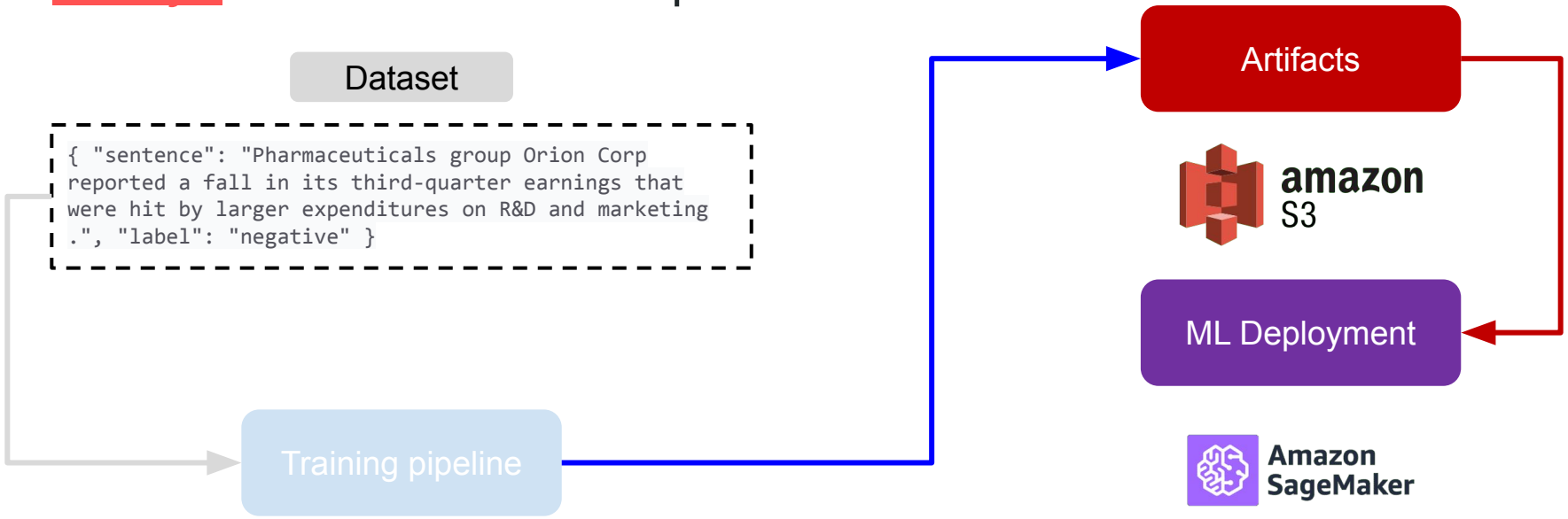




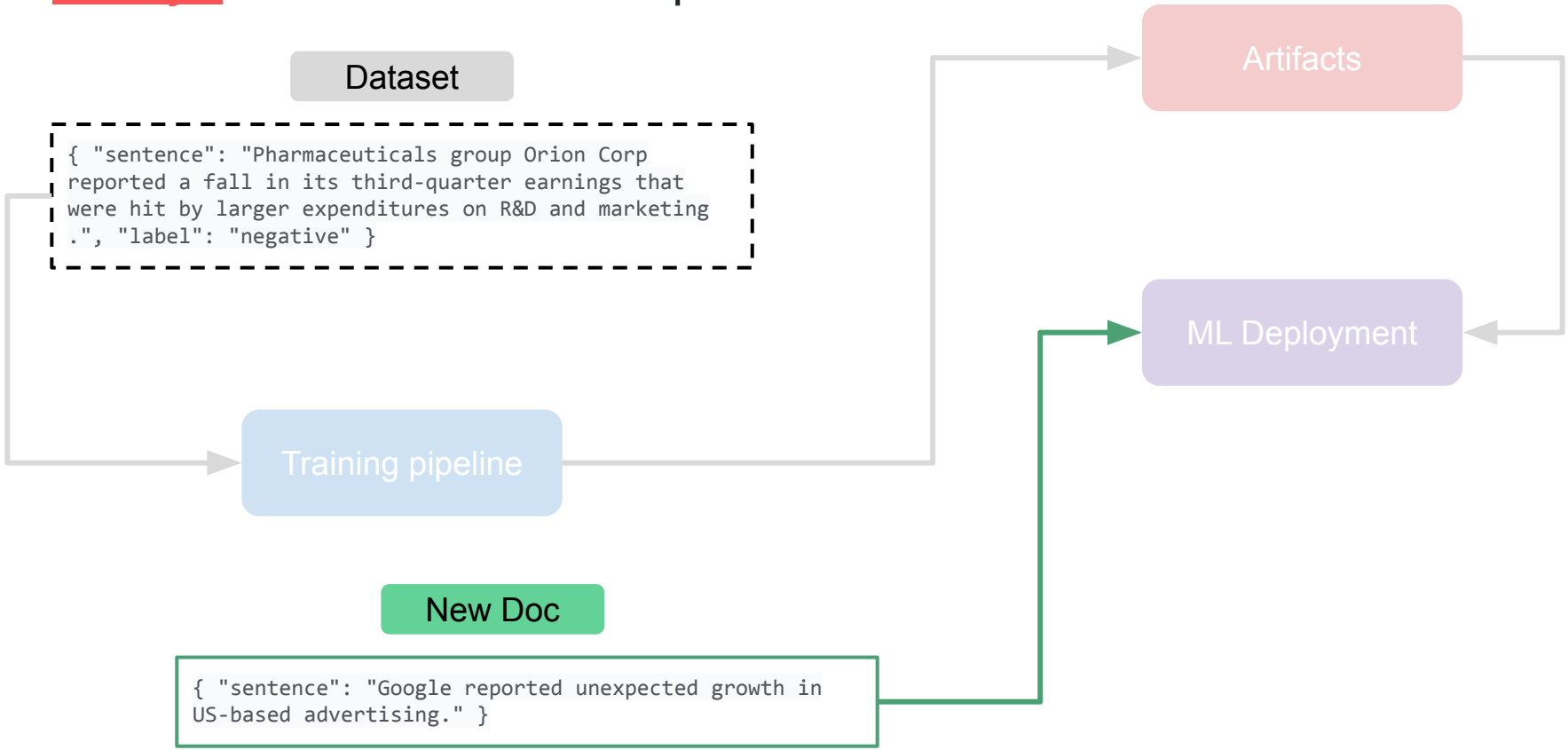
MLSys: A Finance Example



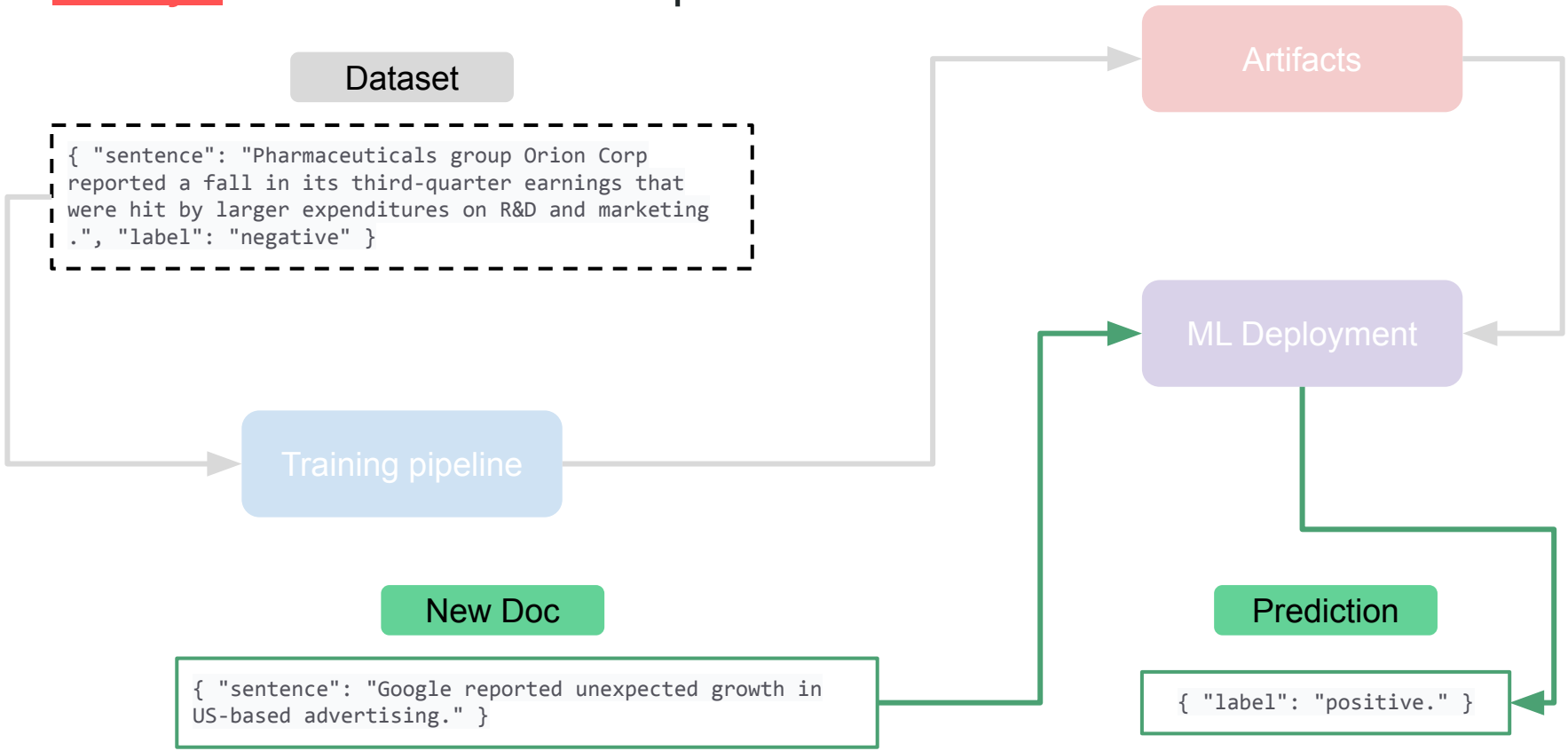
MLSys: A Finance Example



MLSys: A Finance Example

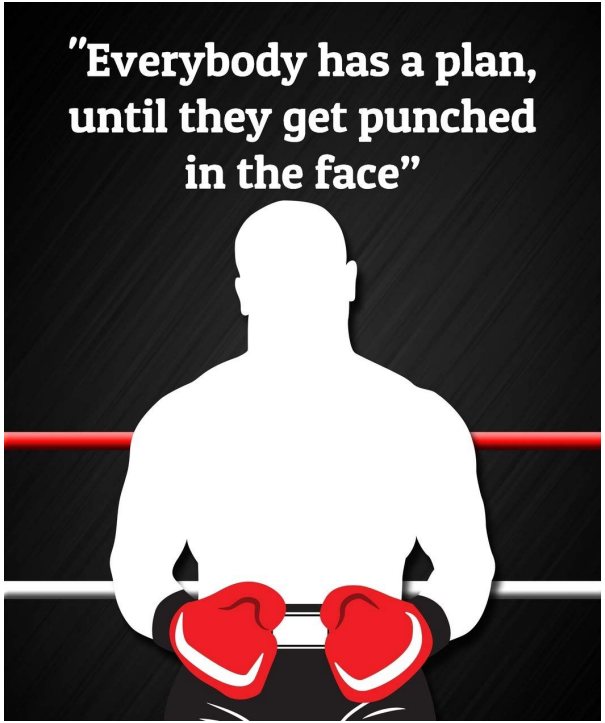


MLSys: A Finance Example



MLSys: Our Plan

- From theory to practice
  - How to organize a ML pipeline
  - Tooling for ML-productivity
- Putting it all together
  - An introduction to AWS
  - Serving predictions



How to organize ML projects

Do I really need ML?

While we will discuss ML projects from now on, in the real world you ALWAYS need to ask yourself a question first: is this project a good fit for machine learning?

Signs your project may not be a good fit for ML include:

1. Simpler solutions can do the trick.
2. There is no data (or no practical way to collect it).
3. One single prediction error can cause devastating consequences.
4. It is impossible to reliably measure the performance of the system.

Welcome to the jungle

If your work needs to have an impact, it needs to RUN OUTSIDE YOUR LAPTOP.

Welcome to the jungle

If your work needs to have an impact, it needs to RUN OUTSIDE YOUR LAPTOP:

1. Your code can be **inspected, modified, understood** by others, typically your technical colleagues: you need to write clean, modular, testable code and make your pipeline fully reproducible.

Welcome to the jungle

If your work needs to have an impact, it needs to RUN OUTSIDE YOUR LAPTOP:

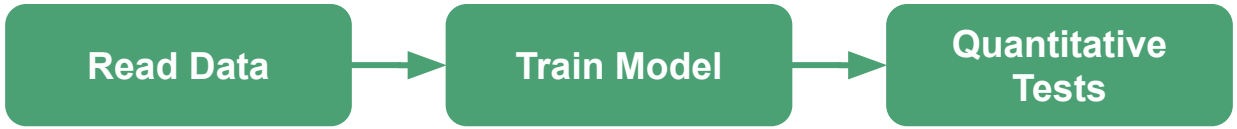
1. Your code can be **inspected, modified, understood** by others, typically your technical colleagues: you need to write clean, modular, testable code and make your pipeline fully reproducible.
2. Your model can be **trusted** by others, typically, other stakeholders, who may or may not be technical folks: you need to “make sure” the model behaved as designed before pushing it in front of end-users.

Welcome to the jungle

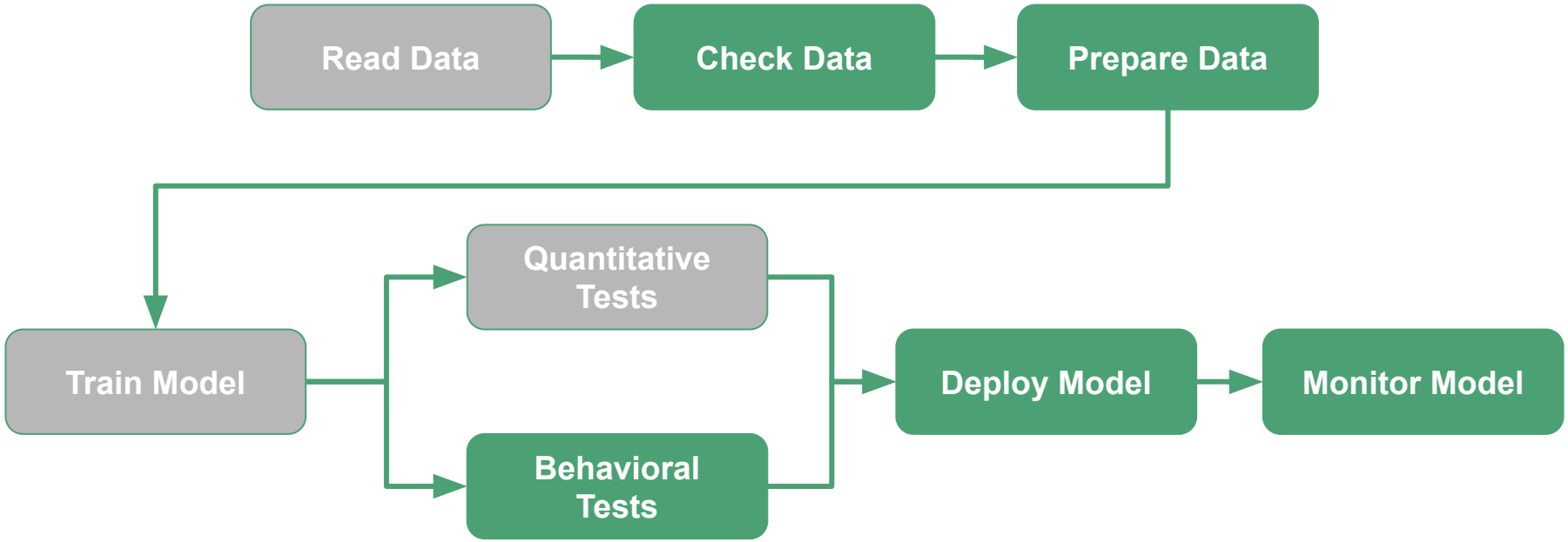
If your work needs to have an impact, it needs to RUN OUTSIDE YOUR LAPTOP:

1. Your code can be **inspected, modified, understood** by others, typically your technical colleagues: you need to write clean, modular, testable code and make your pipeline fully reproducible.
2. Your model can be **trusted** by others, typically, other stakeholders, who may or may not be technical folks: you need to “make sure” the model behaved as designed before pushing it in front of end-users.
3. Predictions can be **consumed** by others, typically anybody with an internet connection: you need to expose your model as an endpoint which returns predictions when supplied with the appropriate parameters.

School vs Real World

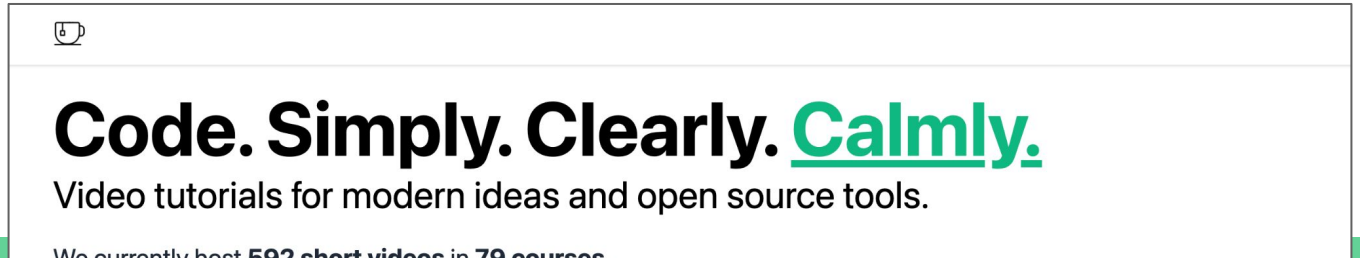


School vs **Real World**



Part 0: Python 101 (virtualenv)

- ML is done mainly in **Python** today: the web is full of excellent tutorials / courses / books on how to learn Python or be better at it. We focus here only on one core concept: virtual environments.
- Since different projects have different dependencies, we may want to *isolate the environments*: ideally, we should run project A *only with the packages needed by A,* B only with those needed by B etc.
- Practically this is accomplished by using virtual envs, cleanly separated environments to execute specific projects: for an introduction see the calmcode page.



Part 1: Structuring the code

```python
def monolith():
    # read the data in and split it
    Xs = []
    Ys = []
    with open('regression_dataset.txt') as f:
        lines = f.readlines()
        for line in lines:
            x, y = line.split('\t')
            Xs.append([float(x)])
            Ys.append(float(y))
    X_train, X_test, y_train, y_test = train_test_split(Xs, Ys, test_size=0.20, random_state=42)
    print(len(X_train), len(X_test))
    # train a regression model
    reg = linear_model.LinearRegression()
    reg.fit(X_train, y_train)
    print("Coefficient {}, intercept {}".format(reg.coef_, reg.intercept_))
    # predict unseeen values and evaluate the model
    y_predicted = reg.predict(X_test)
    fig, ax = plt.subplots()
    ax.scatter(y_predicted, y_test, edgecolors=(0, 0, 1))
    ax.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'r--', lw=3)
    ax.set_xlabel('Predicted')
    ax.set_ylabel('Actual')
    plt.savefig('monolith_regression_analysis.png', bbox_inches='tight')
    mse = metrics.mean_squared_error(y_test, y_predicted)
    r2 = metrics.r2_score(y_test, y_predicted)
    print('MSE is {}, R2 score is {}'.format(mse, r2))

    # all done
    print("See you, space cowboys!")

    return
```

Iteration #1: the monolith (check the repo!)

- All the code is in one main script

PROs

- Fast to write

CONs

- Hard to understand (no logical separation between steps)
- Nothing can be re-used
- Hard to test

Part 1: Structuring the code

```python
def composable_script(file_name: str, test_size: float=0.20):
    # all done
    print("Starting up at {}".format(datetime.utcnow()))
    # read the data into a tuple
    dataset = load_data(file_name)
    # check data quality
    is_data_valid = check_dataset(dataset)
    # split the data
    splits = prepare_train_and_test_dataset(dataset, test_size=test_size)
    # train the model
    regression = train_model(splits, is_debug=True)
    # evaluate model
    model_metrics = evaluate_model(regression.model, splits, with_plot=True)
    # all done
    print("All done at {}!\n See you, space cowboys!".format(datetime.utcnow()))

    return


if __name__ == "__main__":
    # TODO: we can move this to read from a command line option, for example
    FILE_NAME = 'regression_dataset.txt'
    TEST_SIZE = 0.20
    composable_script(FILE_NAME, TEST_SIZE)
```

Iteration #2: breaking down the monolith (check the repo!)

- Tasks are now in separate functions

PROs

- More readable
- Easy to change, test, re-use

CONs

- No versioning
- No replayability
- Hard to scale task selectively

Part 1: Structuring the code

```python
class SampleRegressionFlow(FlowSpec):
    """
    SampleRegressionFlow is a minimal DAG showcasing reading data from a file
    and training a model successfully.
    """

    # if a static file is part of the flow, it can be called in any downstream process, gets versioned etc.
    # https://docs.metaflow.org/metaflow/data#data-in-local-files
    DATA_FILE = IncludeFile(
        'dataset',
        help='Text file with the dataset',
        is_text=True,
        default='regression_dataset.txt')

    TEST_SPLIT = Parameter(
        name='test_split',
        help='Determining the split of the dataset for testing',
        default=0.20
    )

    @step
    def start(self):
        """
        Start up and print out some info to make sure everything is ok metaflow-side
        """
        print("Starting up at {}".format(datetime.utcnow()))
        # debug printing - this is from https://docs.metaflow.org/metaflow/tagging
        # to show how information about the current run can be accessed programmatically
        print("flow name: %s" % current.flow_name)
        print("run id: %s" % current.run_id)
        print("username: %s" % current.username)
        self.next(self.load_data)
```

Iteration #3: Metaflow (check the repo!)

- Tasks are now in a DAG

PROs

- Fully modular
- Scale selectively per task
- All versioned and replayable

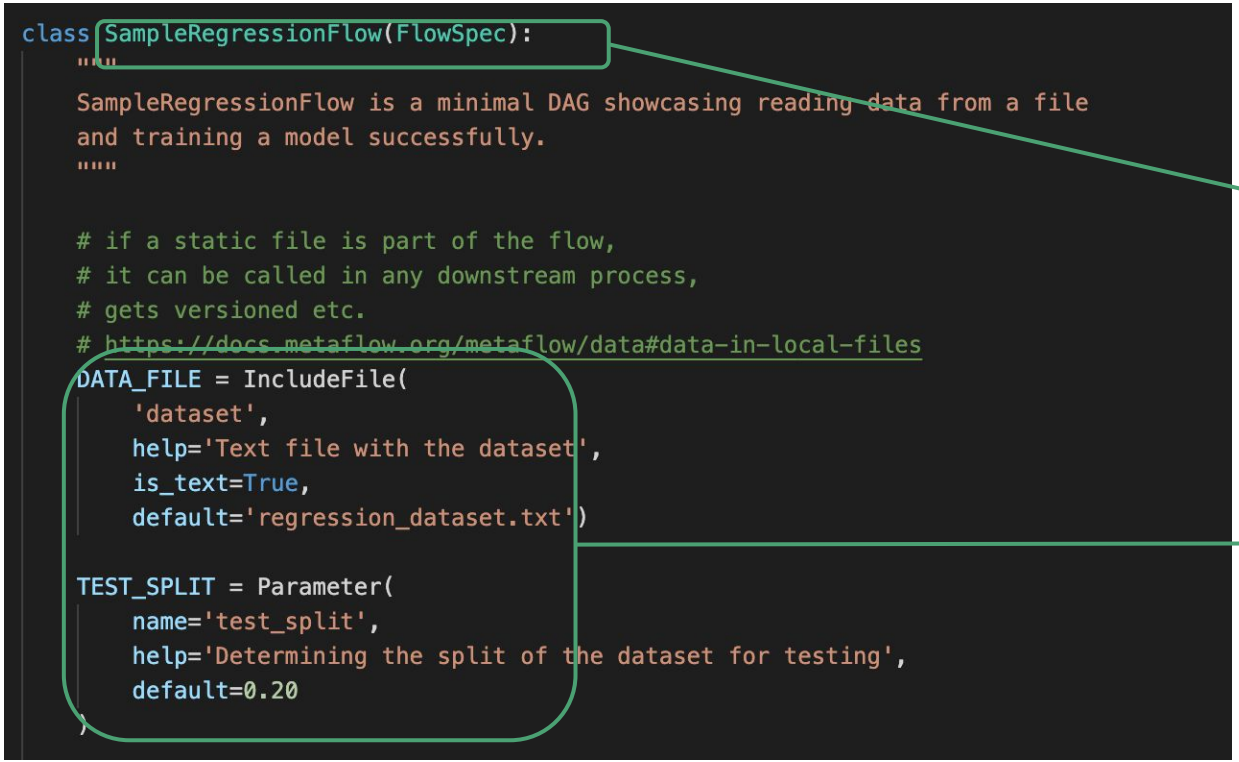
CONs

- Additional complexity

Metaflow as a shared lexicon

1. **Flow:** the DAG describing the pipeline itself.
2. **Run:** each time a DAG is executed, it is a new *run*. Runs are isolated and namespaced, e.g. runs tagged as **user:jacopo** vs **user:mike** may be the same flow, but executed by different people.
3. **Step:** a node of the DAG.
4. **Task**: an execution of a step, isolated and self-contained.
5. **Artifact:** any data / model / state produced by a run, and versioned in the metadata store (e.g. myFlow/12/training/dataset).
6. **Client API:** Python based interactive mode, in which you can inspect metadata and artifacts of all runs for debugging and visualization purposes.

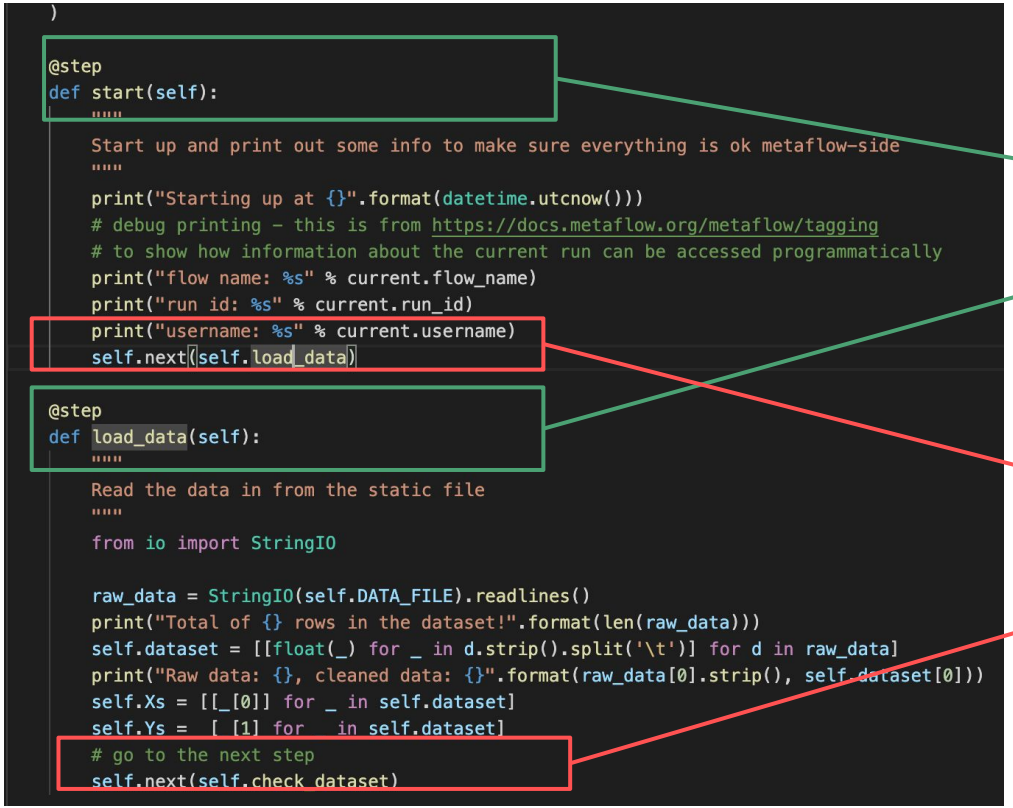
Metaflow projects as (special) Python classes - I



A project class inheriting from FlowSpec

OPTIONAL: Parameters to configure the flow, Files as input

Metaflow projects as (special) Python classes - II

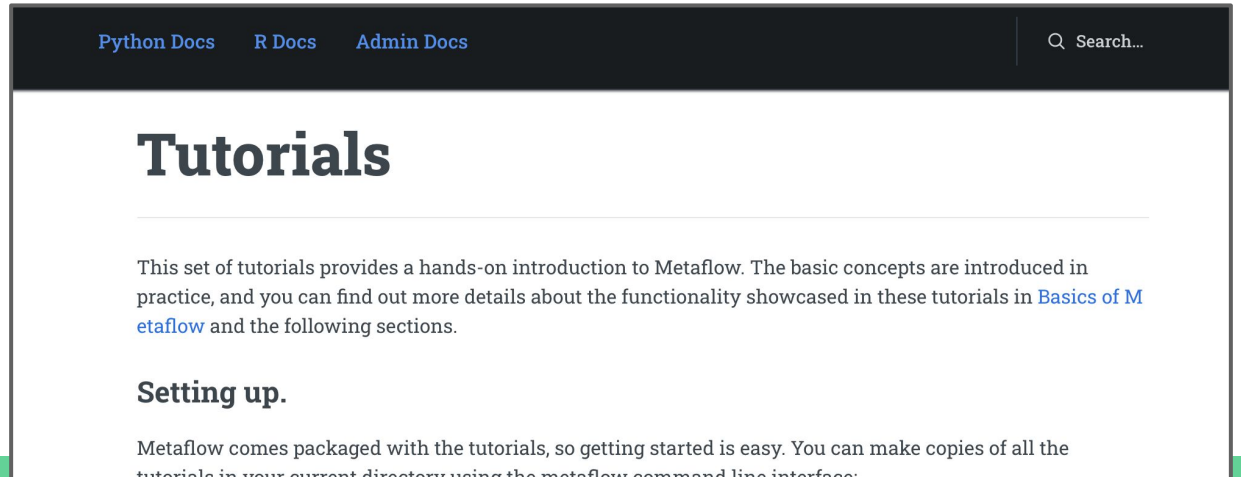


Functions decorated with @steps: each function is a node in the DAG

Each function lists its descendant(s) through the next command.

Metaflow components

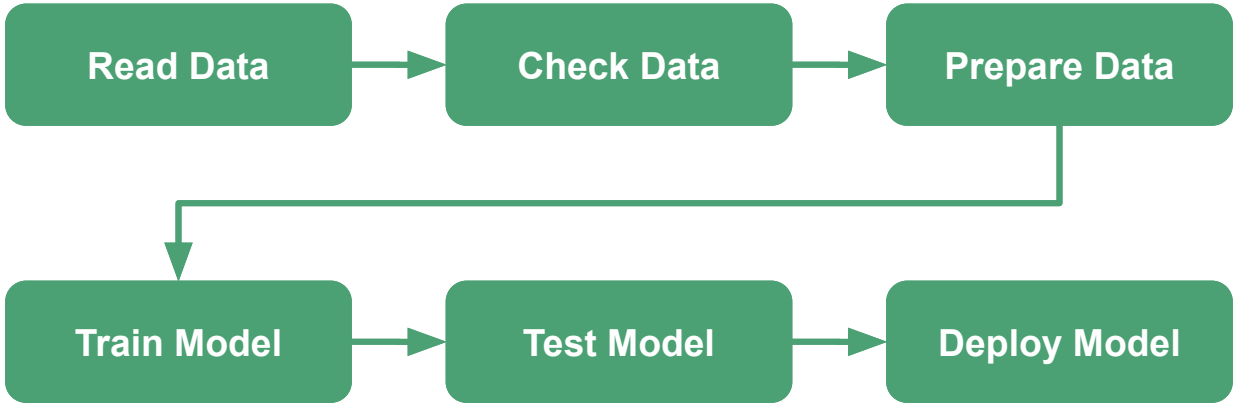
1. **Dag definition:** what are we doing? Steps, dependencies, parallelization etc.
2. **Metastore:** where do we store stuff? Variables, states, meta-data etc.
3. **Computational layer:** what is executing the computation? Resources, cloud tools etc.



Metaflow in 4 principles

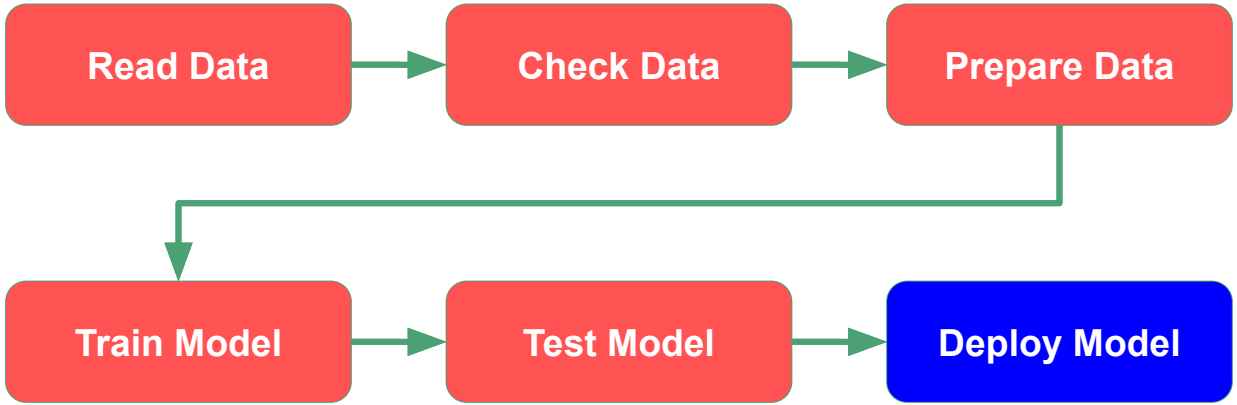
#1: ML projects are a DAG

Tasks depends only on a subset of other tasks: parallelization is possible, and retry can be smart in case of failure!



FRE 7773 Bonus Point

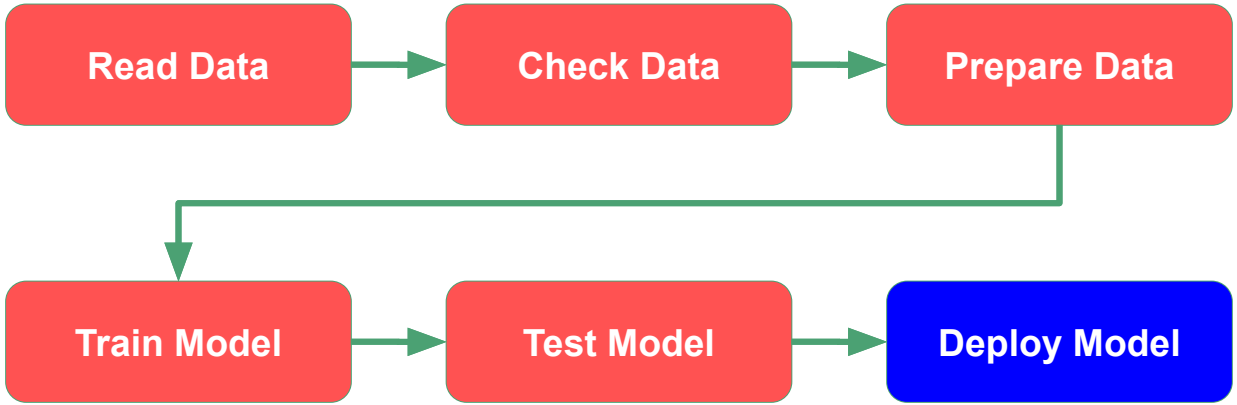
We distinguish between two phases of our ML project: a training phase (load data, data checks, training and testing model...) and a serving phase (expose the model prediction to other users).



FRE 7773 Bonus Point

In this class (and also when developing new projects in the industry), we have:

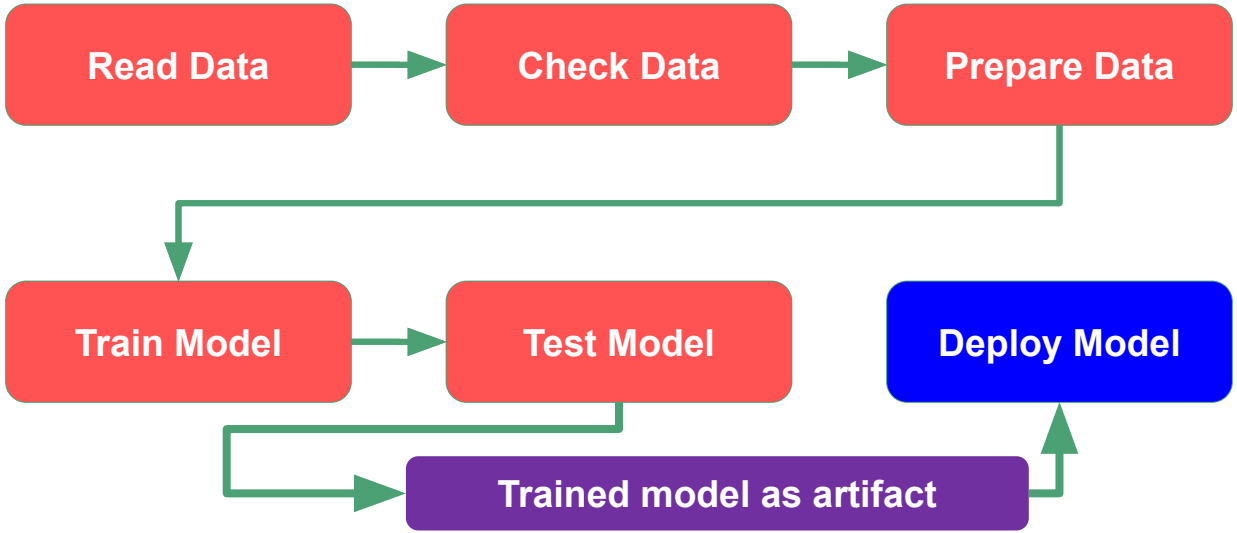
- training phase: done locally (in Metaflow)
- serving phase: done in the cloud (in AWS)



FRE 7773 Bonus Point

In this class (and also when developing new projects in the industry), we have:

- training phase: done locally (in Metaflow) and **produces a model artifact**
- serving phase: done in the cloud (in AWS)



Metaflow in 4 principles

#2: Data and states are part of ML pipelines (versioning, replayability)

```
@step
def load_data(self):
    """
    Read the data in from the static file
    """
    from io import StringIO

    raw_data = StringIO(self.DATA_FILE).readlines()
    print("Total of {} rows in the dataset!".format(len(raw_data)))
    self.dataset = [[float(_) for _ in d.strip().split('\t')] for d in raw_data]
    print("Raw data: {}, cleaned data: {}".format(raw_data[0].strip(), self.dataset[0]))
    self.Xs = [[_[0]] for _ in self.dataset]
    self.Ys =  [_[1] for _ in self.dataset]
    # go to the next step
    self.next(self.check_dataset)
```

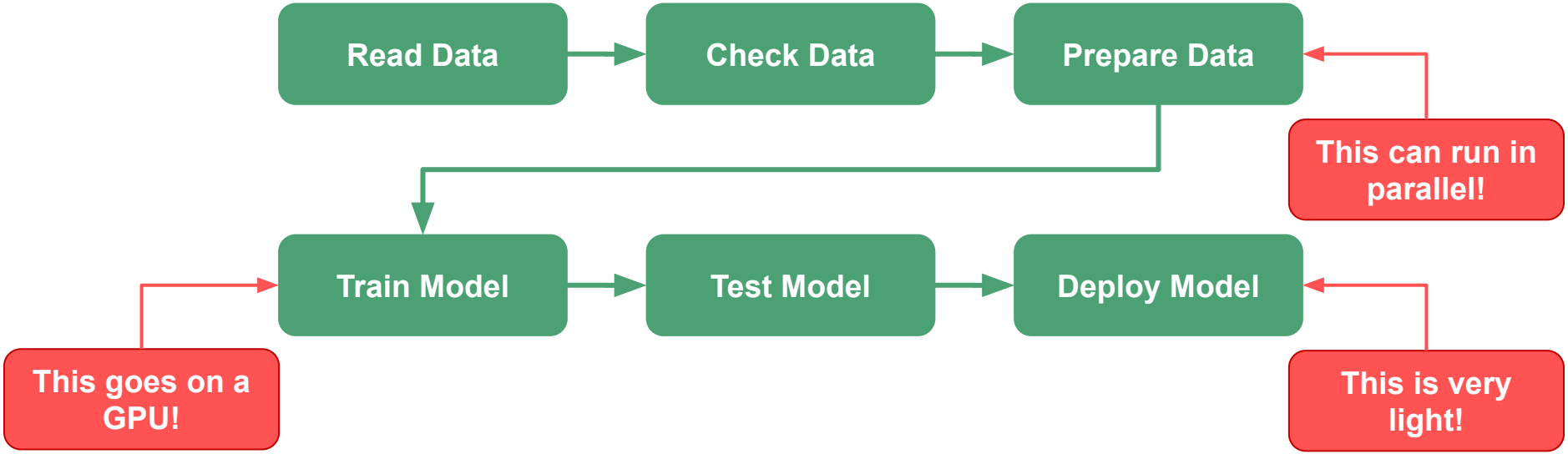
The raw dataset is saved!

The X,Y dataset is saved!

Metaflow in 4 principles

#3: One computing size does not fit all

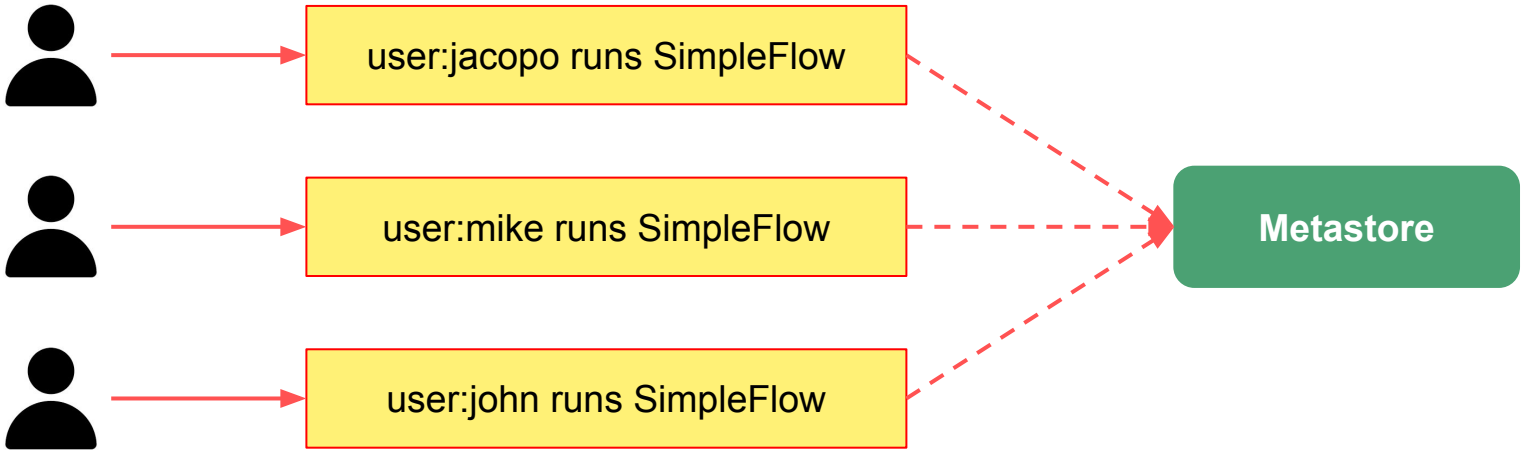
You can define computing resources (and packages) *per task*, switching between local and cloud computing only when necessary.



Metaflow in 4 principles

#4: Everything is cool when you're part of a team

Multiple users can run the same flow together, and then the team can analyze the artifacts produced independently by all runs.



Part 2: Trusting the model

Data

Architecture

Tuning

In the life of real-world ML systems, what is the most important factor in determining the final performance?

Part 2: Trusting the model

1. **Data** is the most important factor, but it is hard to automate (data change all of the time, data contains domain assumptions, data quality depends on collection best practices etc.).
2. Architectures are getting increasingly commoditized.
3. Tuning is conceptually simple, but may be expensive in practice.

Part 2: Trusting the model

A three steps plan:

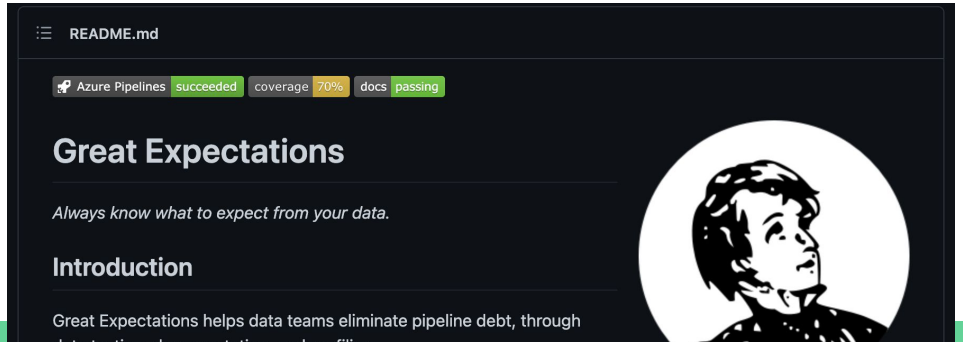
1. To trust your model you need to trust your data -> data checks.
2. To trust your model you need to trust your training routine -> hyper tuning, experiment tracking, understood quantitative objective.
3. To trust your model you need to trust it in edge cases (or cases that are particularly interesting to you) -> “black-box” testing.

Part 2: Trusting your data

To trust your model you need to trust your data

In academic settings (and in your homeworks!) data is given to you, often prepared, cleaned and (up to a point) normalized for your analysis.

This is not what happens in the real world: data collection may be a very messy process and *before* doing ML it is important to make sure our “data expectations” hold.



Part 2: Trusting your data

To trust your model you need to trust your data

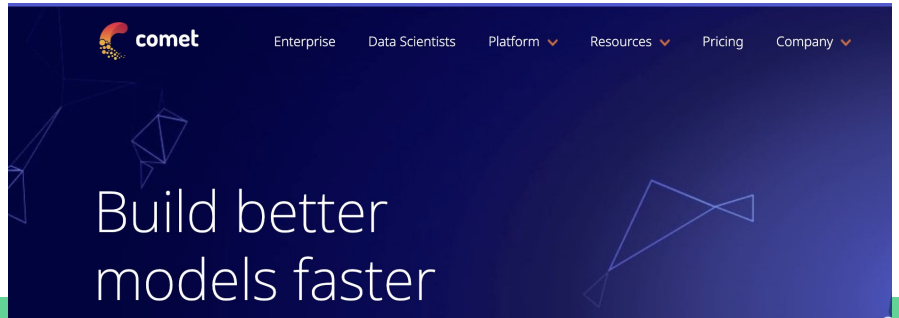
Some questions we may want to ask our data:

- Are there some missing values? (If yes, what do we do with it?)
- Is the dataset imbalanced? (If yes, what do we do with it?)
- Is the value range for feature X reasonable? For example, we expect an “age” column to have only positive values, up to 120.
- Is the value mean / median for feature X reasonable? For example, we expect an “IQ” column to have mean around 100, if the dataset reflects the general population.

Part 2: Trusting your training

To trust your model you need to trust your training routine

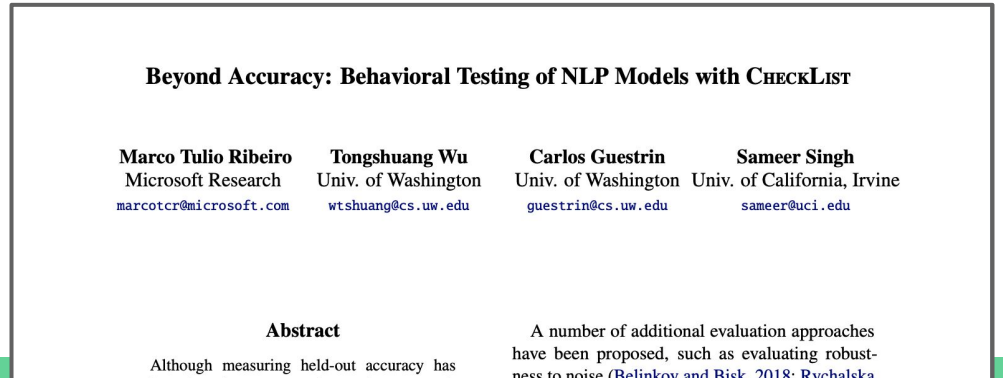
- Make sure your train, validation, test split are correct (Q: how do we split a dataset about historical stock prices?)
- Make sure to identify the relevant hyperparameters and optimize them properly: use an experiment tracking system (e.g. Comet) to track and organize experiments
- Make sure to version artifacts (data, models), so that outcomes can be reproduced (Q: how do we deal with randomness?)
- Make sure the final metrics on the test set are satisfying, considering your use case.



Part 2: Trusting your evaluation

To trust your model you need to trust it in edge cases

A recent work in NLP adapts the idea of "black box testing" from traditional software systems to ML systems: it should be possible to evaluate the performance of a complex system by treating it as a black box, and only supply input-output pairs that are relevant for our qualitative understanding.



Beyond Accuracy: Behavioral Testing of NLP Models with CheckList

Marco Tulio Ribeiro
Microsoft Research
marcotcr@microsoft.com

Tongshuang Wu
Univ. of Washington
wtshuang@cs.uw.edu

Carlos Guestrin
Univ. of Washington
guestrin@cs.uw.edu

Sameer Singh
Univ. of California, Irvine
sameer@uci.edu

Abstract

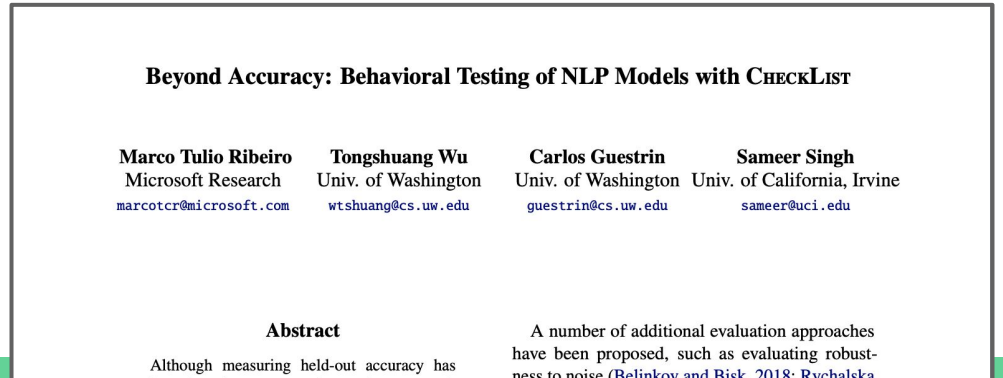
Although measuring held-out accuracy has

A number of additional evaluation approaches have been proposed, such as evaluating robust-
ness to noise (Belinkov and Bisk, 2018; Rychalska

Part 2: Trusting your evaluation

To trust your model you need to trust it in edge cases

1. *Qualitative checks*: I may be interested in checking some cases which has business importance, disastrous consequences, or that are representative of an important class.
2. *Data slicing*: together with reporting performance on an aggregate basis, is there a meaningful way to "slice" the data and calculate performance per slice?



Beyond Accuracy: Behavioral Testing of NLP Models with CheckList

Marco Tulio Ribeiro
Microsoft Research
marcotcr@microsoft.com

Tongshuang Wu
Univ. of Washington
wtshuang@cs.uw.edu

Carlos Guestrin
Univ. of Washington
guestrin@cs.uw.edu

Sameer Singh
Univ. of California, Irvine
sameer@uci.edu

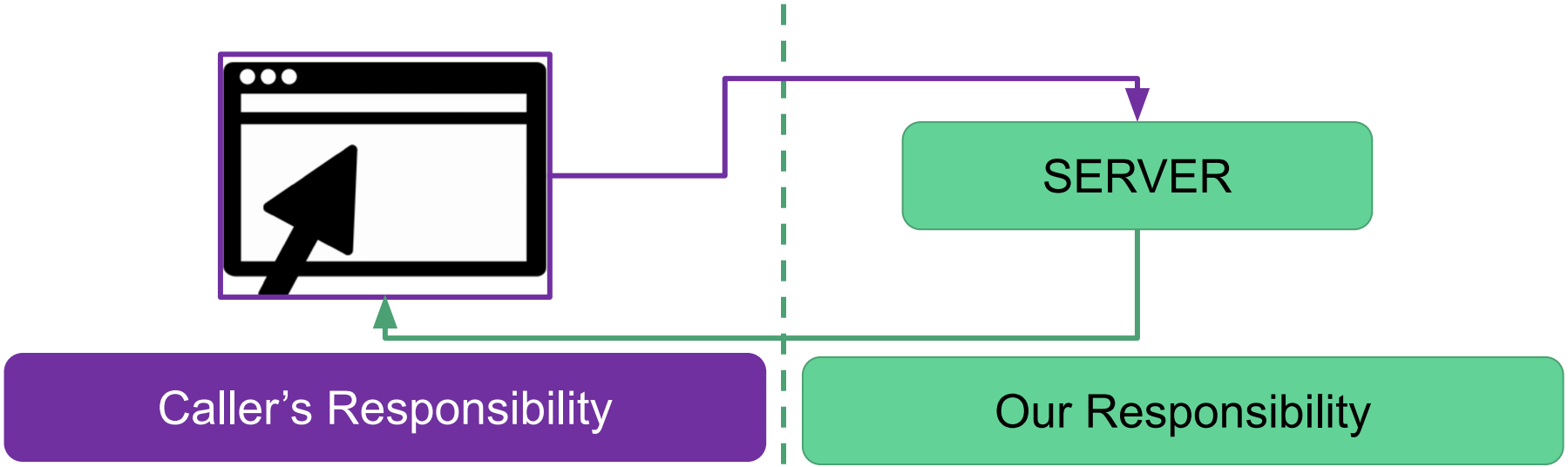
Abstract

Although measuring held-out accuracy has

A number of additional evaluation approaches have been proposed, such as evaluating robust-

Part 3: Serving predictions

- If our model stays on our laptop, nobody will be able to use it!
- **Client-server architecture**: our model interacts with *many* remote clients through an API (also called “endpoint”) - we abstract away model code (and complexity) and expose a pure input-output interface: clients send us the input, we return a prediction.



Part 3: Serving predictions

- If our model stays on our laptop, nobody will be able to use it!
- **Client-server architecture**: our model interacts with *many* remote clients through an API (also called “endpoint”) - we abstract away model code (and complexity) and expose a pure input-output interface: clients send us the input, we return a prediction.

Part 3: Serving predictions

The three eras of cloud:

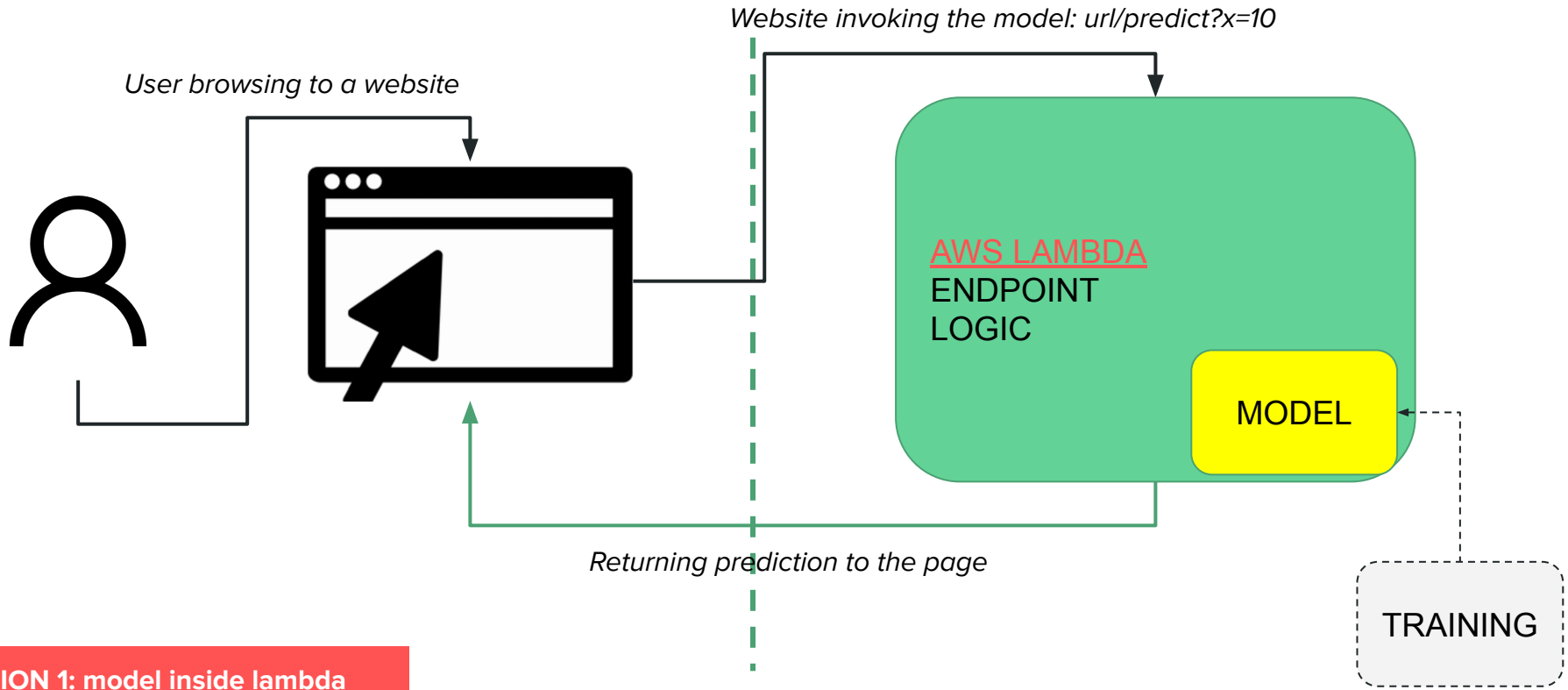
- IaaS: Infrastructure as a Service
- PaaS: Platform as Service
- FaaS: Function as a Service

Serverless computing 101: a function is defined by

- Environment (dependencies, variables)
- Logic (what am I doing?)
- Time (how much time can I run for?)
- Compute (how much memory can I use?)

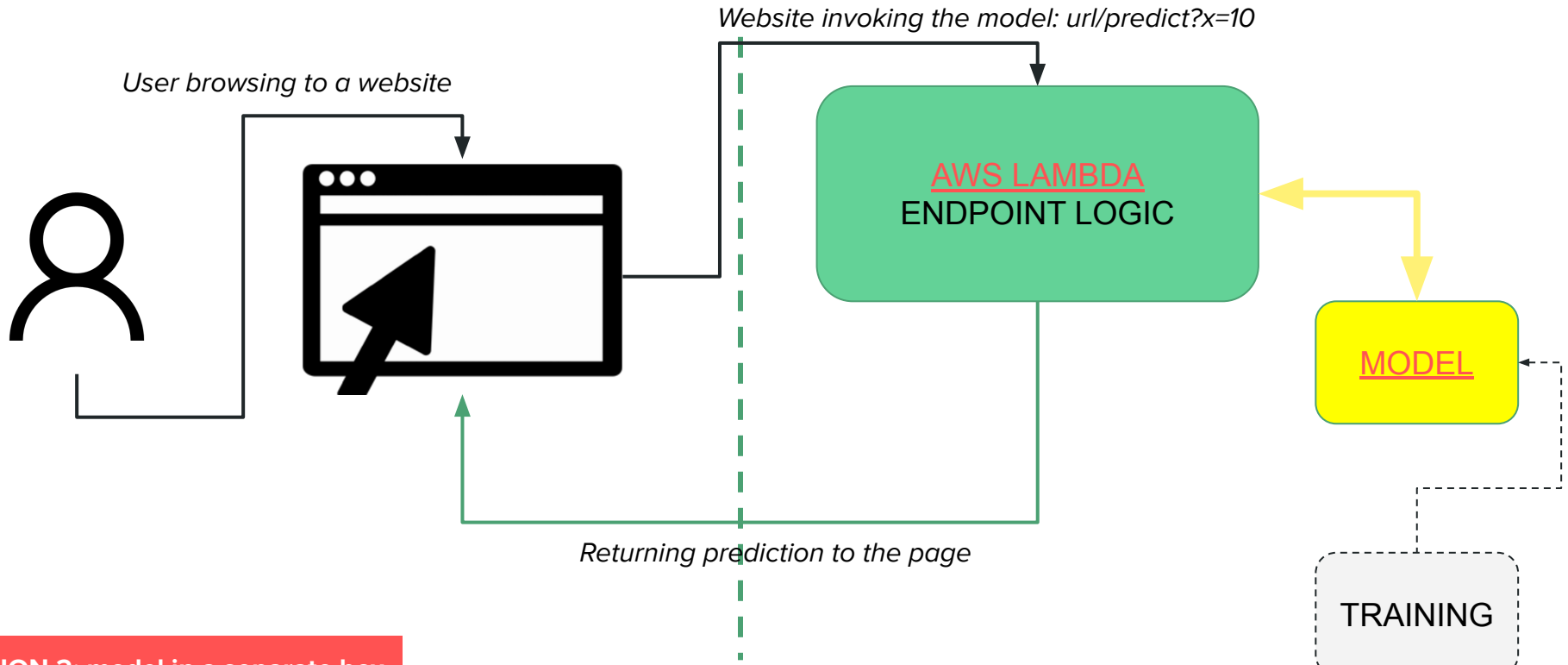
[While not necessary, it is good practice to handle Infrastructure as Code, for example with Serverless.]

Part 3: Serving predictions



OPTION 1: model inside lambda

Part 3: Serving predictions



OPTION 2: model in a separate box

Part 3: Serving predictions

[Follow along with the repo]

1. Set up AWS credentials in your local config file, and make sure serverless is installed.
2. Run the basic Metaflow pipeline to train a regression model and save BETA and INTERCEPT.
3. Add BETA and INTERCEPT to the yml file: when deployed, those variables are accessible in the code as environment variables.
4. Deploy the lambda function to your AWS account with: “serverless deploy --aws-profile myProfile”
5. Open a browser and use the provided URL to test the endpoint:

https://XXX.execute-api.us-west-2.amazonaws.com/dev/simple_regression?x=10

Part 3: Serving predictions

If all went well, your browser will display the model response: now **everybody** with the URL can use your awesome model!


```
{
    "data": {
        "predictions": [167.068]
    },
    "metadata": {
        "eventId": "167b7129-cea1-4156-932f-f8d89c4b4066",
        "serverTimestamp": 1633532566012,
        "time": 0.00022029876708984375
    }
}
```


This is the actual prediction from the model (why is it a list?)

This is useful information about the call itself (debugging, monitoring, etc.)

Alternative deployment scenarios

There is a ton of alternatives when it comes to *serving predictions* from the cloud, ranging from pure infrastructure to fully managed services. For example:

- You can deploy your model manually on a virtual machine, by installing Flask and run through screen (like they do here)
- You can deploy your model through a web app hosted by Elasticbeanstalk (like they do here)
- You can deploy your model through a web app hosted by Fargate (like they do here)
- You can deploy your model through Sagemaker, and expose it through a lambda (like we do in the class repository)

After deployment: monitoring

We are not going to discuss monitoring, as we are not launching new apps in this course (for now!). However, after our model is live we need to:

- monitor how the pipeline is doing:
  - How is the new data coming in?
  - Does the model need re-training?
  - Is my new model better than the old one?
- check what users are doing with it!

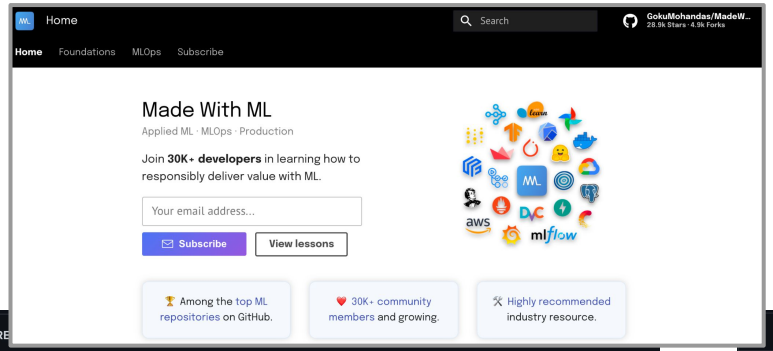
After deployment: monitoring

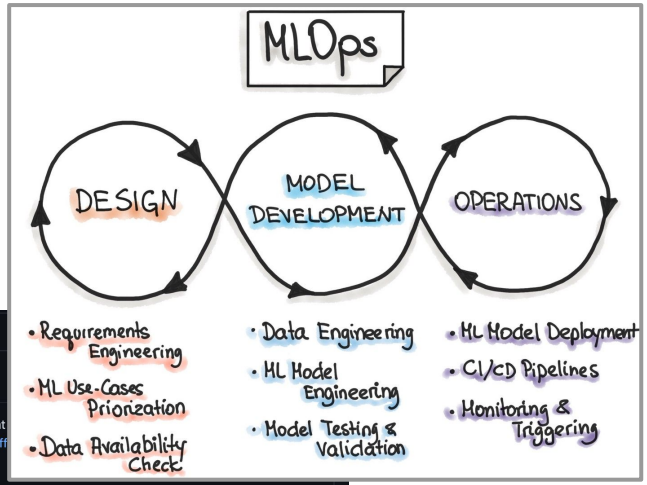
We are not going to discuss monitoring, as we are not launching new apps in this course (for now!). However, after our model is live we need to:

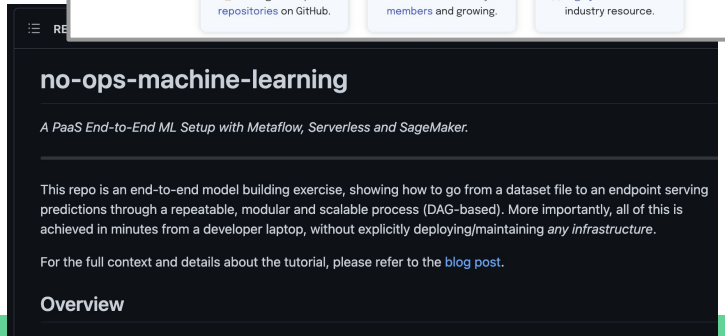
- monitor how the pipeline is doing:
  - How is the new data coming in?
  - Does the model need re-training?
  - Is my new model better than the old one?
- check what users are doing with it!
  - You never know how people would use stuff!

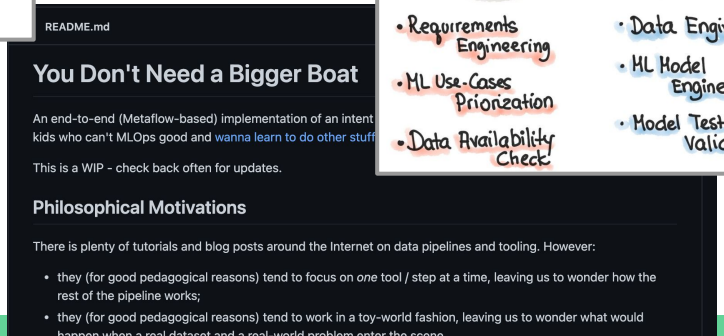
Further readings

There is a ton of recent developments in the “MLOps” space (we do our small part as well in the community). If you want to know more, reach out!





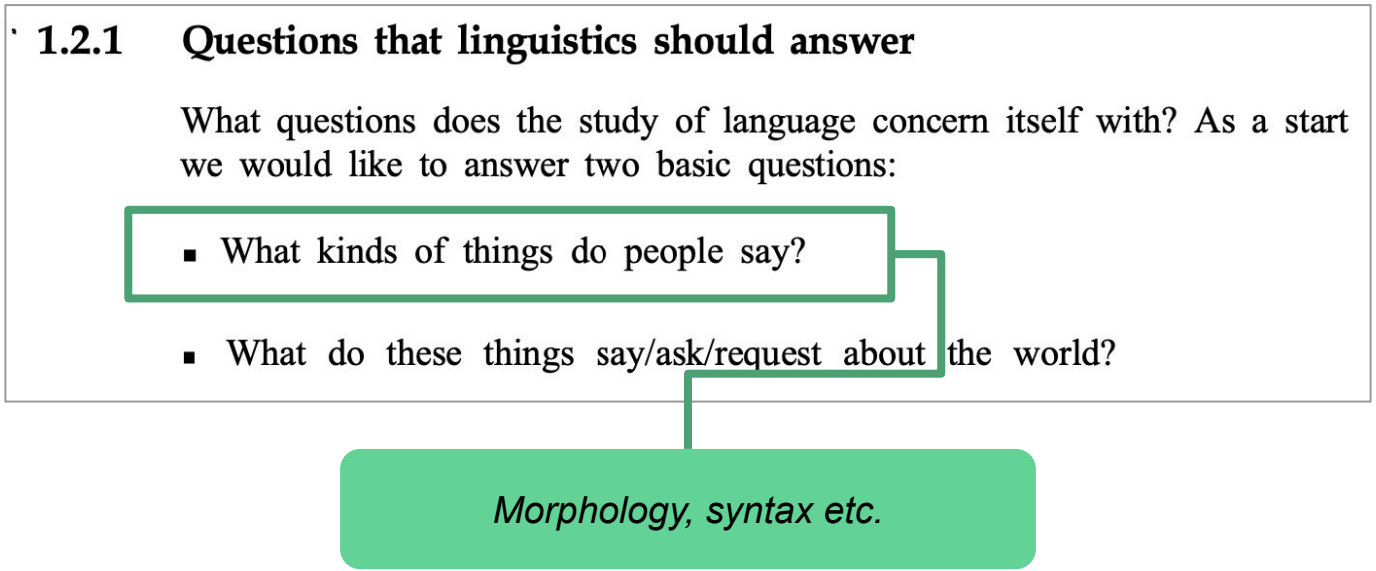




Good ol’ NLP

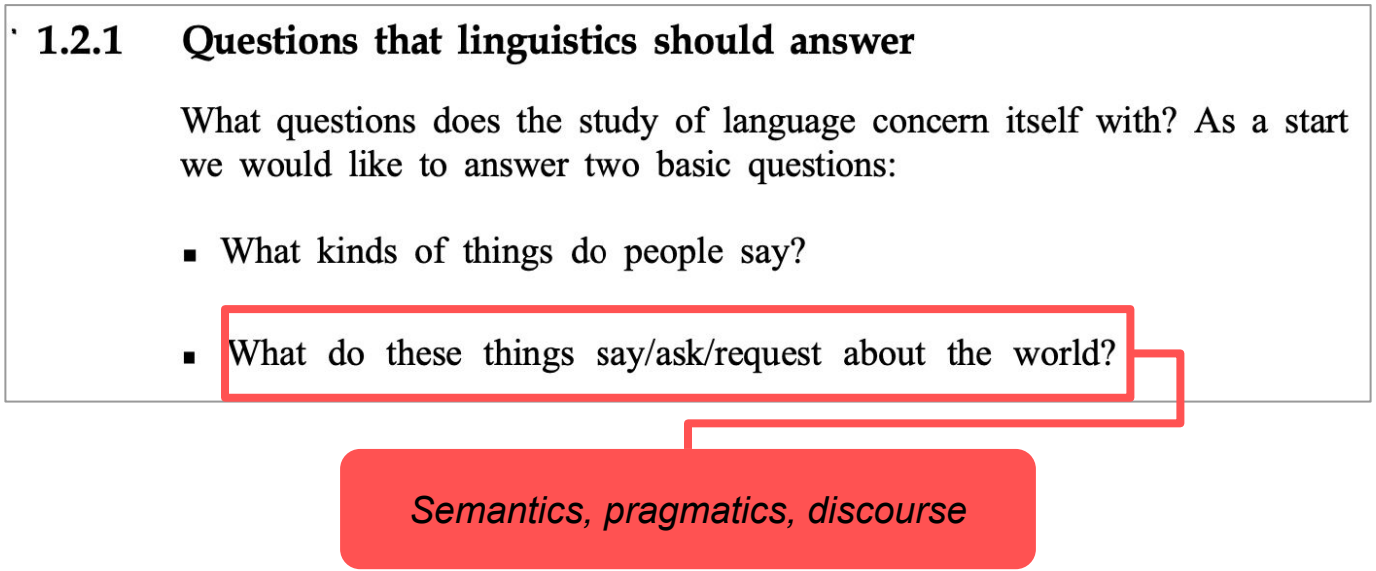
What is language?

- 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?

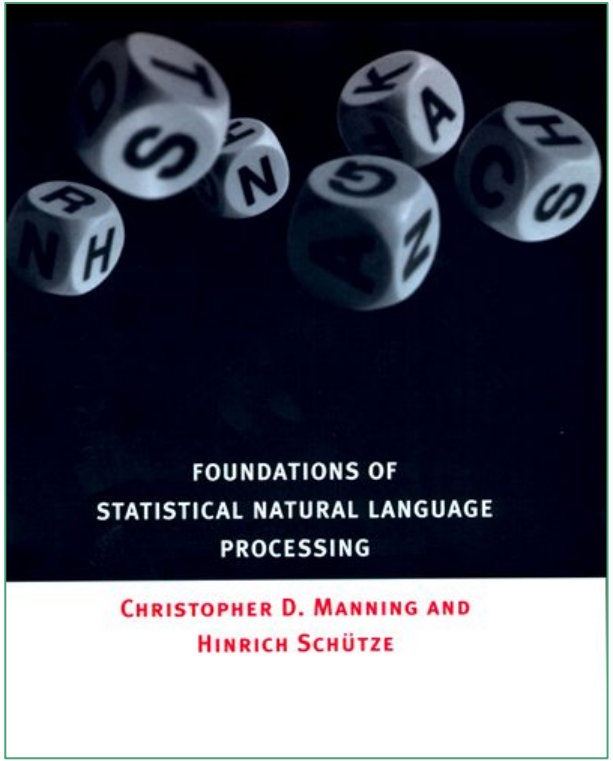
What is language at FRE 7773?

- 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.

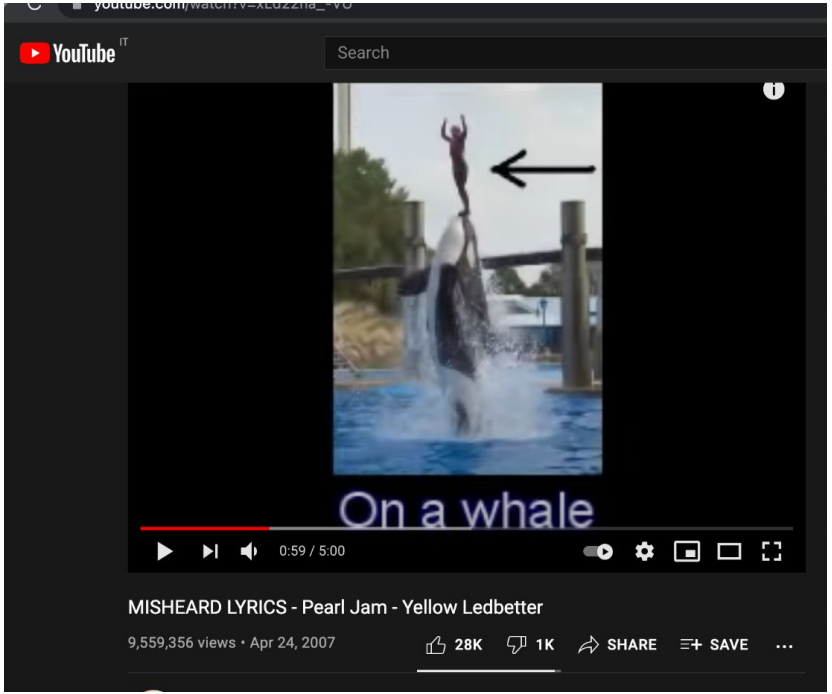
Language modelling

- **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**.

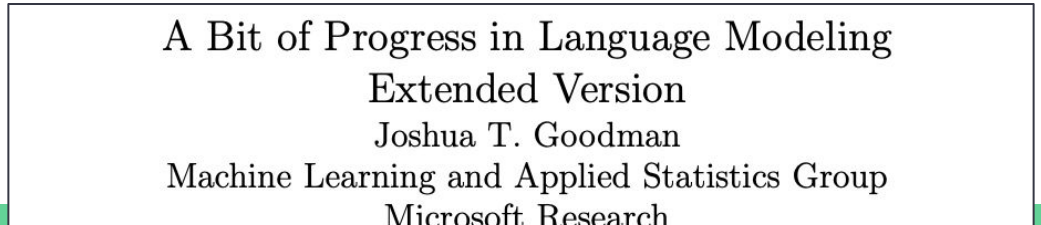
A trigram language model

- 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

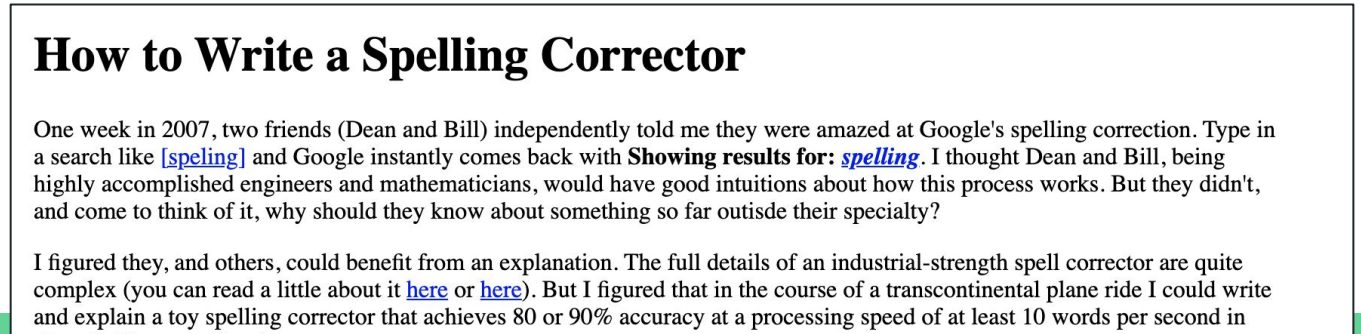
Dealing with rare events

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

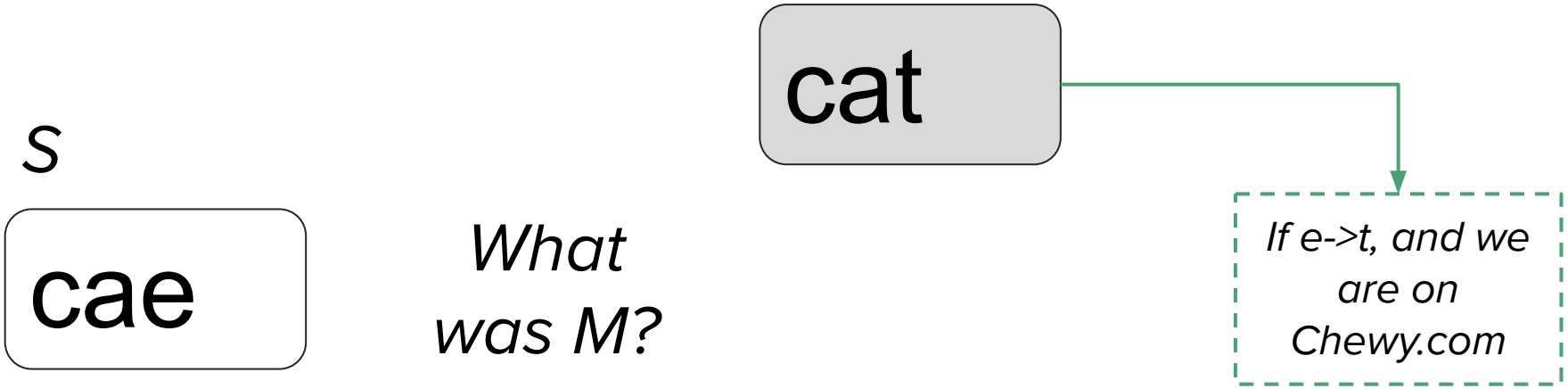
A noisy channel model

S

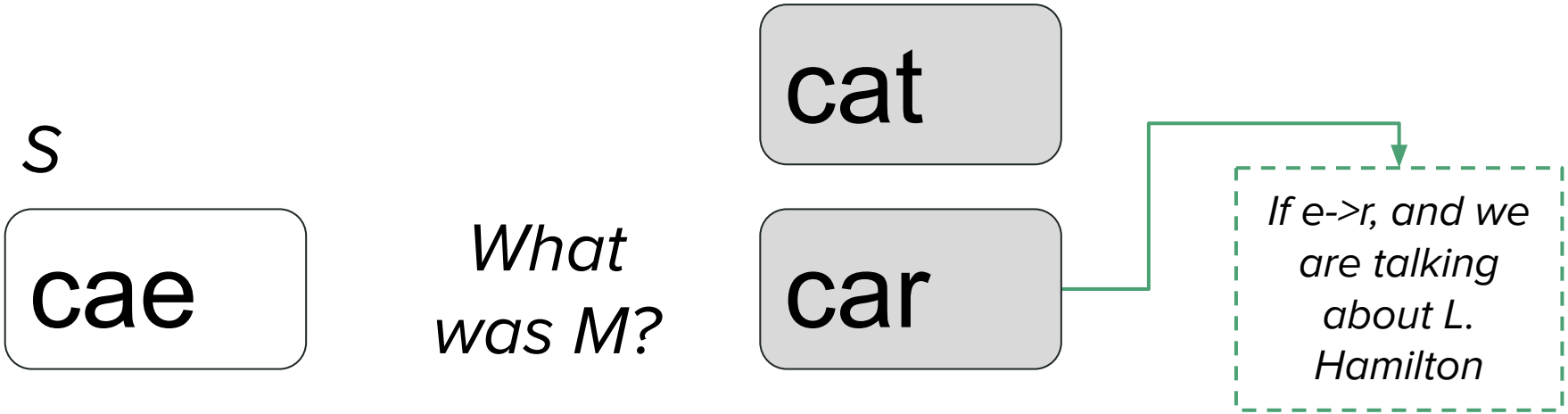
cae

What was M?

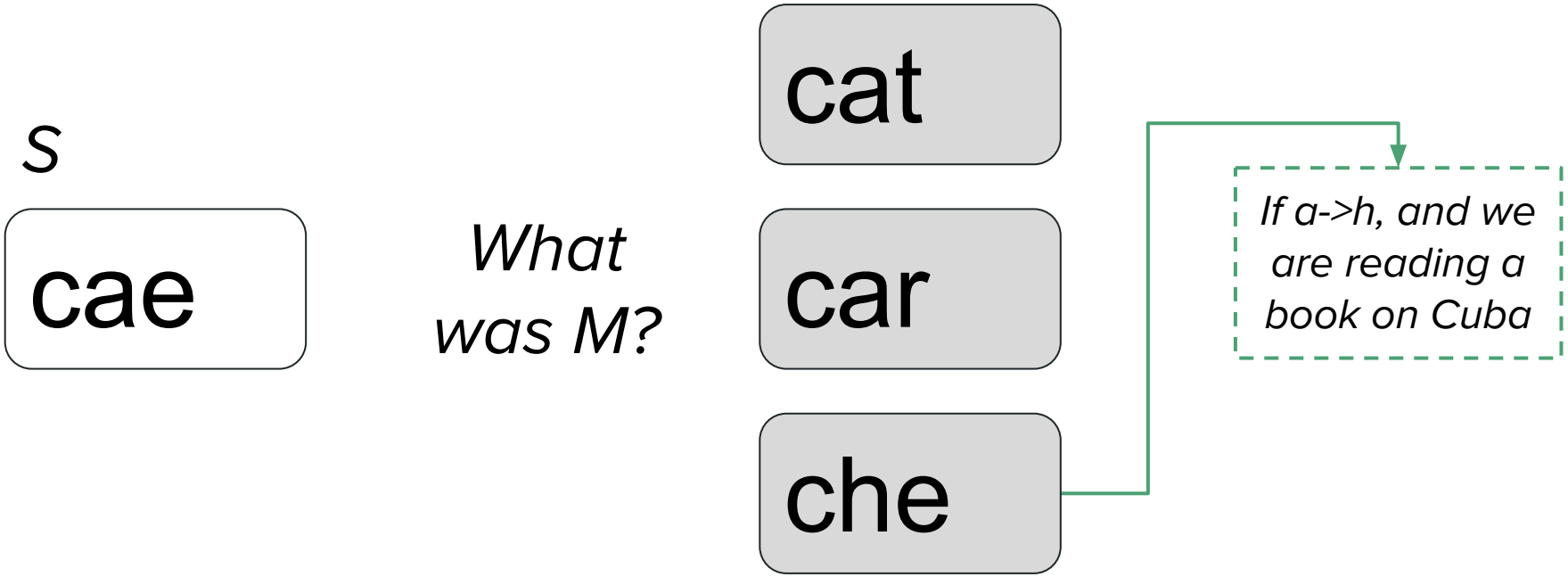
A noisy channel model



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**.

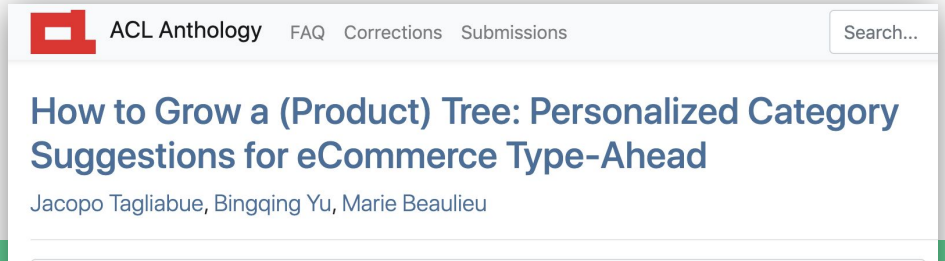
A noisy channel model - Homework hints

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 live 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 live 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 teach 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

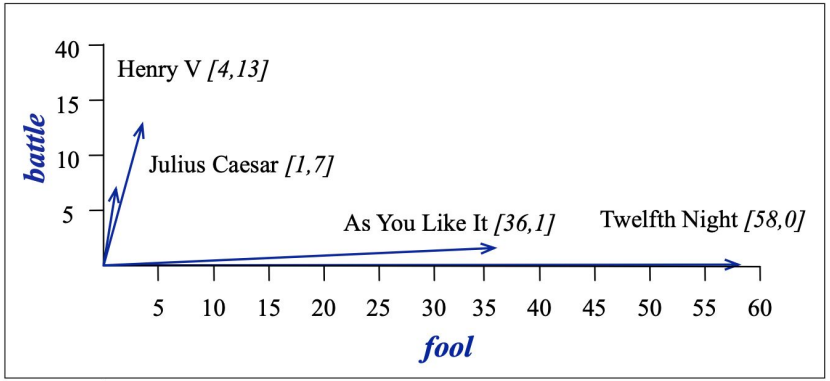


Figure 6.4 A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words *battle* and *fool*. The comedies have high values for the *fool* dimension and low values for the *battle* dimension.

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)?

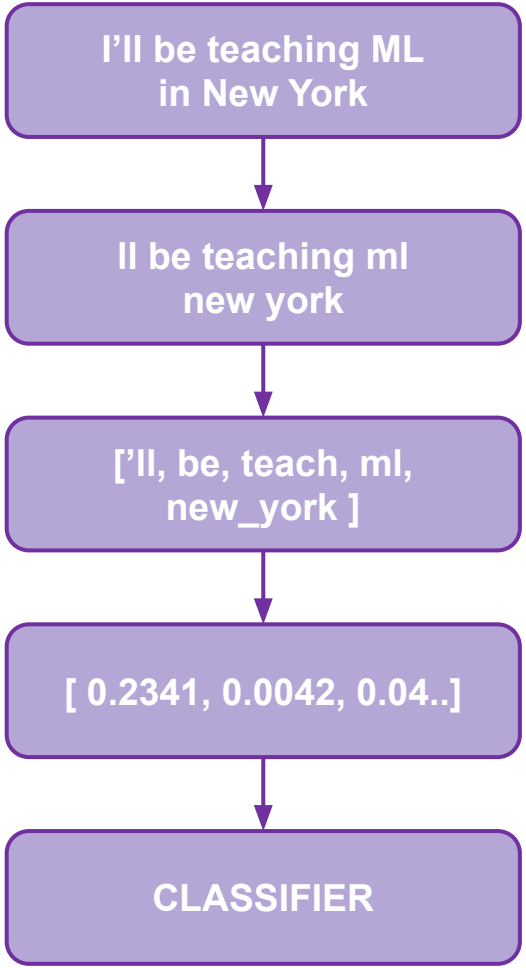
See the notebook on text classification for hand-on experience and tips on vectorization!

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!

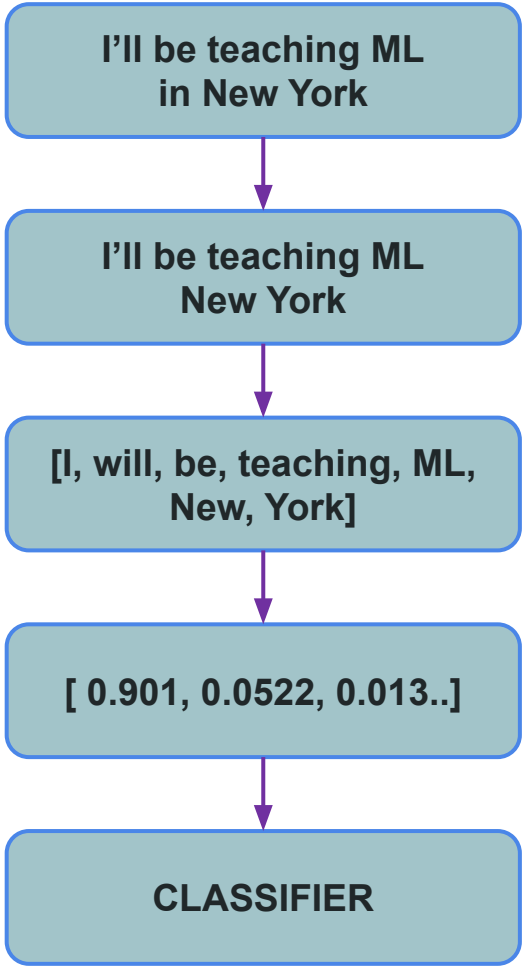
Text classification flow

- 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.

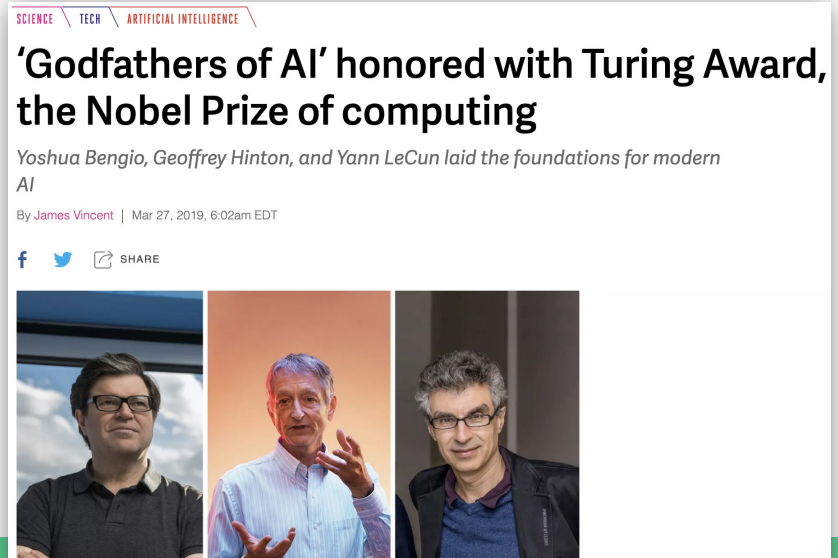
PRACTICE

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

Going neural

ML at the time of Deep Learning

- There is no doubt that the recent wave of interest in AI has been driven by the so-called “**deep learning revolution**”.

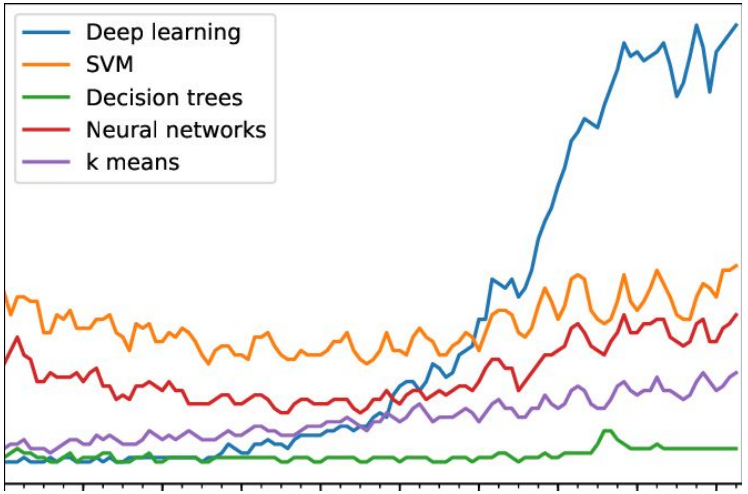

SCIENCE | TECH | ARTIFICIAL INTELLIGENCE

‘Godfathers of AI’ honored with Turing Award, the Nobel Prize of computing

Yoshua Bengio, Geoffrey Hinton, and Yann LeCun laid the foundations for modern AI

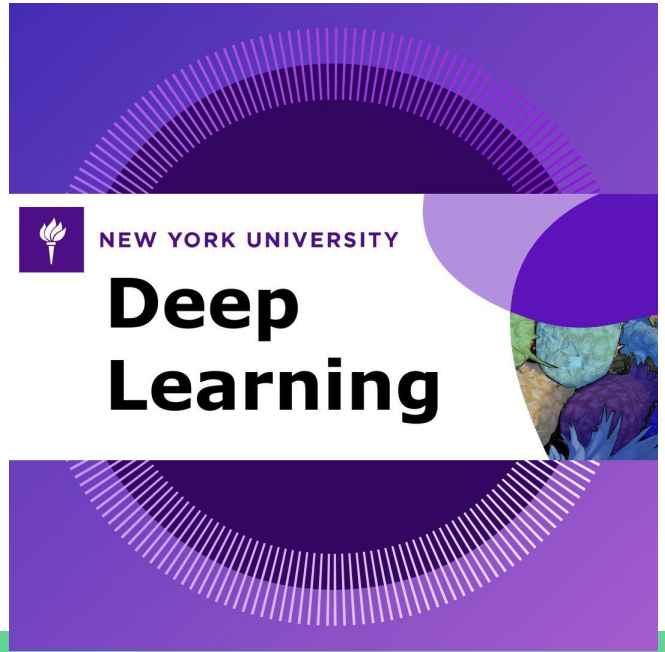
By James Vincent | Mar 27, 2019, 6:02am EDT

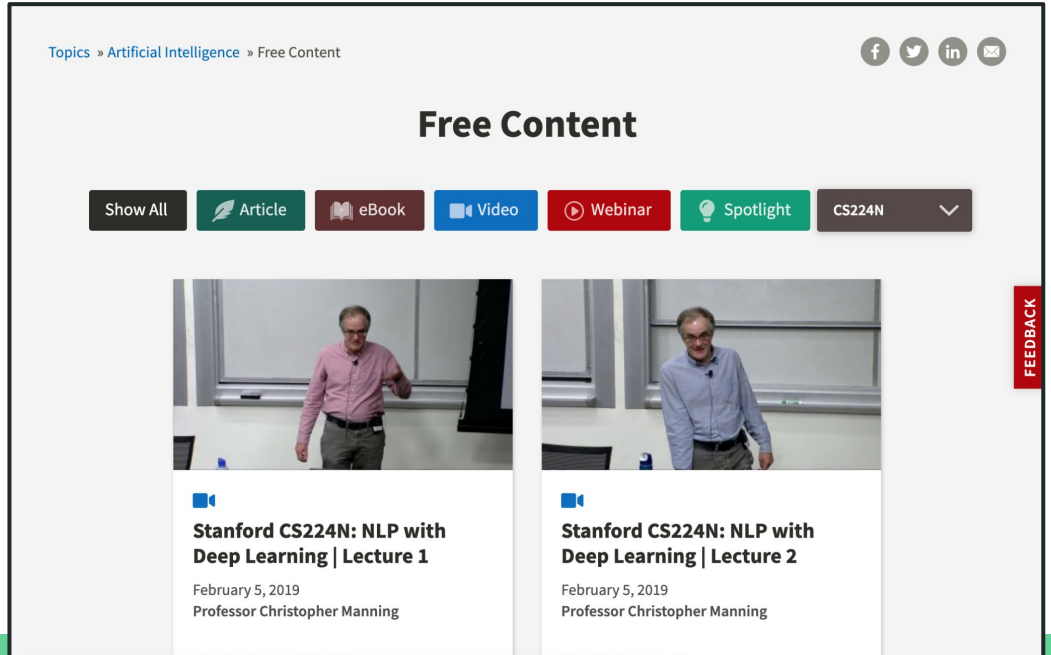
SHARE



Bonus: Internet is an amazing place!

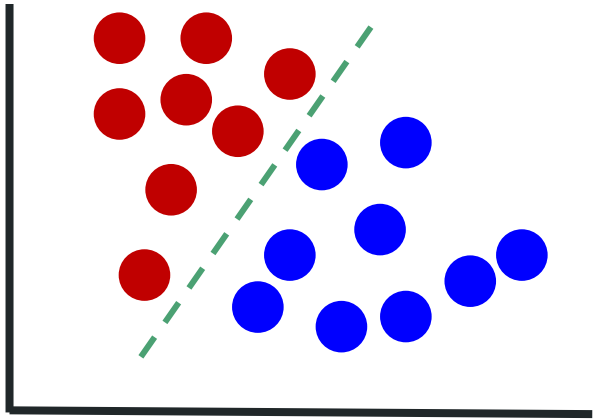
- NYU DL course
- Stanford NLP course





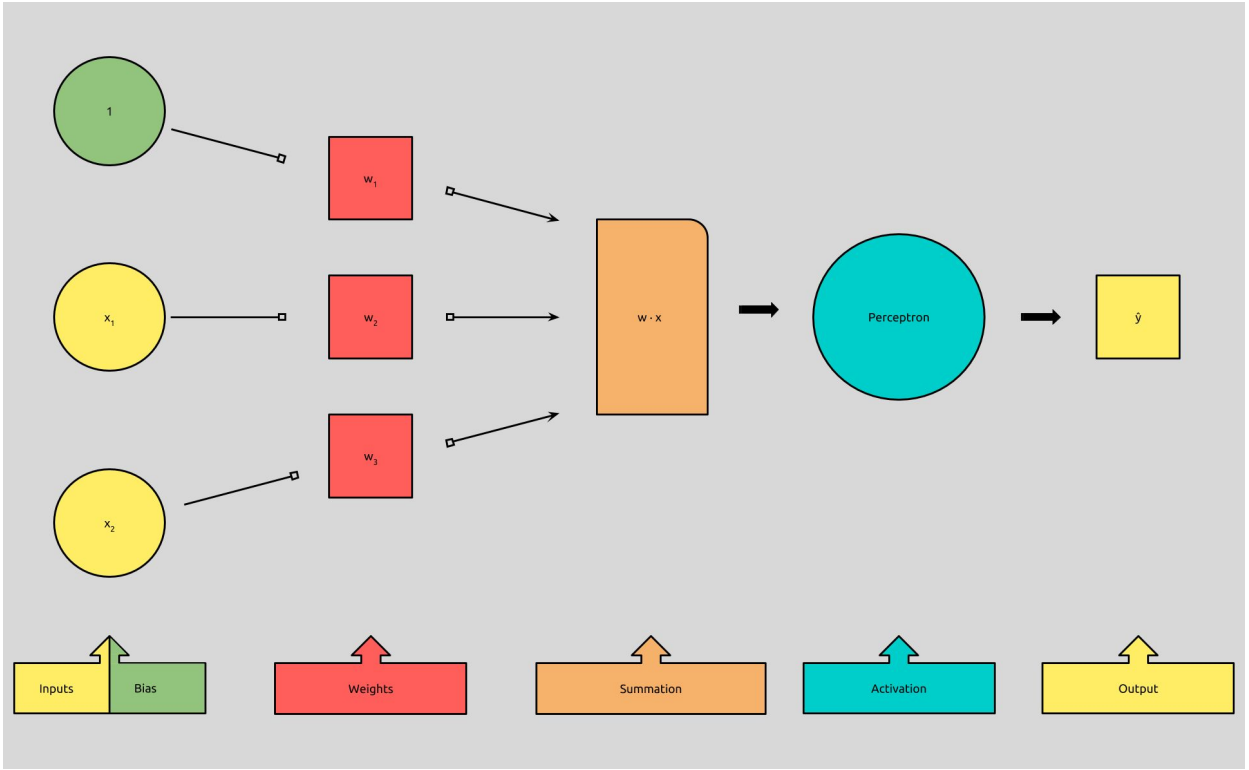
Blast from the past (1957): the perceptron

- A simple classification problem (red vs blue): how do find the decision boundary?



Blast from the past (1957): the perceptron

- Inputs, x_1, x_2
- Bias, **1**
- Weights, w_1, w_2, w_3
- Activation
- Result



Blast from the past (1957): the perceptron

- **Q1**: how do we make a decision (“forward pass”), given inputs and weights?

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$$

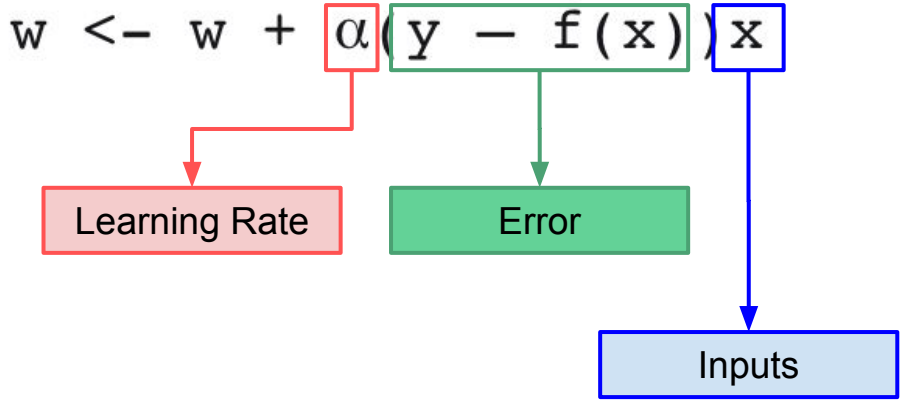
A threshold function: if the dot product of weights and inputs + bias > 0, the output is 1 (0 otherwise).

Example:

- Inputs = 1, 1, Weights: 0,0
- Bias = 1
- dot(input, weights) = 0
- Dot + bias = 1

Blast from the past (1957): the perceptron

- **Q2**: how do we learn the weights, based on a training set (“backward”)?



We calculate the prediction error, multiply by a learning rate and by the inputs, and update the weights.

Example:

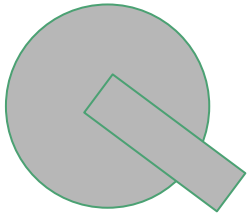
- Inputs = 0, 0, Weights: 0,0, Bias = 1
- Prediction = 1, Target = 0
- 0 = 0 + alpha(0 - 1) * 0

Blast from the past (1957): the perceptron

- Perceptrons can only learn linearly separable functions (Q: can they learn XOR?)
- Deep neural networks (that is networks in which several layers of neurons are stacked together, between the input layer and the output) are the distant cousins of the original perceptron, and have been proven to be able to learn extremely complex non-linear boundaries.
- While NNs are significantly more complex, they share the same basic building properties:
  a. We need an activation function, which may be a bit more complex (e.g. ReLU), but still determines the output of neurons based on inputs and weights;
  b. We need a learning procedure, which may be a bit more complex (backprop + gradient descent), but still determines how to iteratively adjust weights based on how “off” is the prediction with respect to the true label.

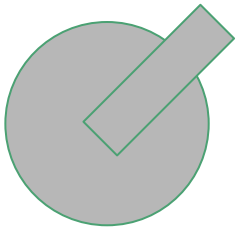
Back-propagation in the shower

1



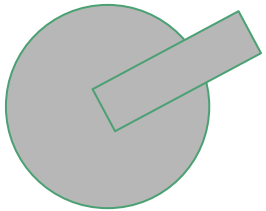
A bit too cold!

2



A bit too hot!

3



Great!

Back to NLP: from words to vectors (remember?)

- **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!

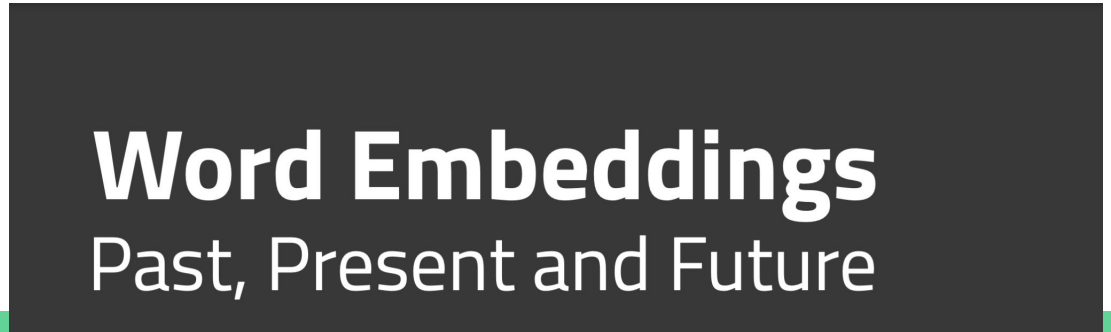
Option #1: count vectorizer (big and sparse!)

Option #2: TF IDF vectorizer (big and sparse!)

Option #3: word2vec (small and dense!)

Word vectors in a *prediction* task

- (Philosophical) distributional hypothesis: “words that appear in similar contexts have similar meanings”
- Computational hypothesis: learn a classifier that given a target word (e.g. cat) tell me how likely is that a context word appear next to it (Germany, furry), *and use the learned weights as the word vector.* Since the weights will adjust during training to make sure similar words have high probability, their vectors will be close in the resulting embedding space.



Word vectors in a *prediction* task

- CORPUS: “The furry cat is on the mat”
- WINDOW LENGTH: 2
- TARGET: “cat”
- INPUT PREPARATION, positive and negative samples

Target	Context	Label
cat	furry	1
cat	the	1
cat	is	1
cat	on	1

Target	Context	Label
cat	Berlin	0
cat	Jacopo	0
cat	ciao	0
cat	table	0

Word vectors in a *prediction* task

- CORPUS: “The furry cat is on the mat”
- WINDOW LENGTH: 2
- TARGET: “cat”
- INPUT PREPARATION, positive and negative samples (a=0.75)

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_{w'} count(w')^\alpha}$$

Target	Context	Label
cat	furry	1
cat	the	1
cat	is	1
cat	on	1

Target	Context	Label
cat	Berlin	0
cat	Jacopo	0
cat	ciao	0
cat	table	0

Word vectors in a *prediction* task

- Now that we have input examples (positive and negative), let's define our learning objective:
  - We want to maximize the similarity of (t,c) drawn from the positive examples
  - We want to minimize the similarity of (t,c) drawn from the negative examples

$$L(\theta) = \sum_{(t,c)\in+} \log P(+|t,c) + \sum_{(t,c)\in-} \log P(-|t,c)$$

Word vectors in a *prediction* task

- Now that we have input examples (positive and negative), let's define our learning objective:
  - We want to maximize the similarity of (t,c) drawn from the positive examples
  - We want to minimize the similarity of (t,c) drawn from the negative examples

Dot product

Sigmoid

$$= \log \sigma(c \cdot t) + \sum_{i=1}^{k} \log \sigma(-n_i \cdot t)$$

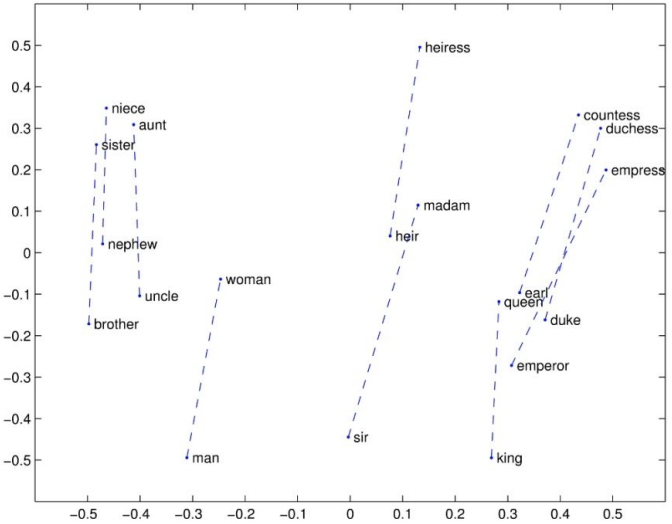
$$= \log \frac{1}{1+e^{-c \cdot t}} + \sum_{i=1}^{k} \log \frac{1}{1+e^{n_i \cdot t}}$$

“Talk is cheap, show me the code”

- Some pretty cool features of word2vec:
  - Vectors are good for downstream models: if we use 100-dim embeddings as features, a classifier can just learn 100 weights to represent a function.
  - Learning is “self-supervised”, as any text we can find on the internet can be turned into positive/negative pairs *without manual intervention*.
  - The vector space encodes automatically similarities and analogies.
  - Vectors are “*portable*”: they can be learned on Wikipedia and applied to finance news.
- The window size influences which “similarity” we care about:
  - Shorter windows leads to representations that are syntactically and semantically similar: “cat” will be very similar to “dog” for example - same topic (pet), same part of speech (noun)
  - Long context windows leads to representations that are topically related but not necessarily syntactically equivalent, such as “cat” and “furry” for example.
- We are going to drive these points home by looking at a practical implementation using the fantastic Python library Gensim - check the notebook code!

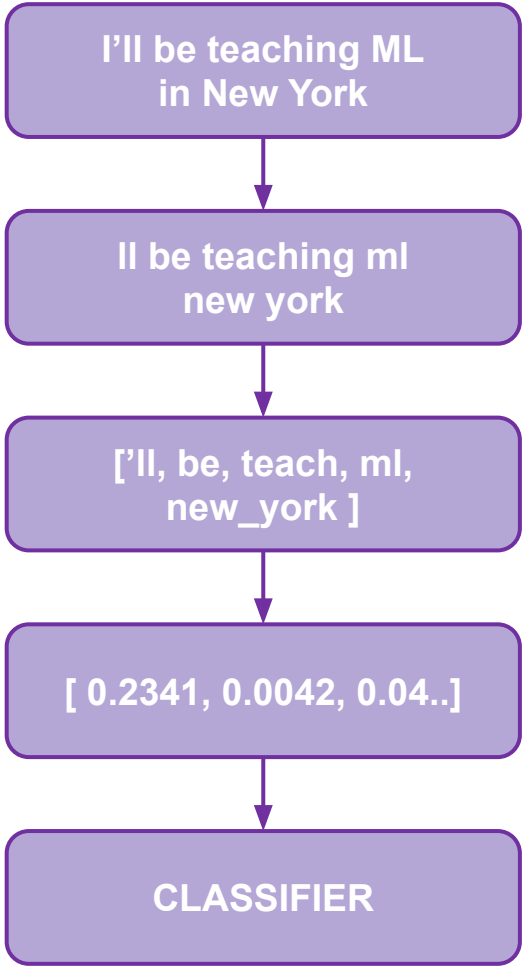
Doing algebra with words

- word2vec tend to capture well similarity between words and some analogical relations.
- In particular, once you have a well-trained embedding space, the offsets between vector embeddings can be used to solve analogies such as: “man : king = women : ?” (*queen*)
- This is possible since the result of vector(‘king’) - vector(‘man’) + vector(‘woman’) is a vector close to vector(‘queen’).



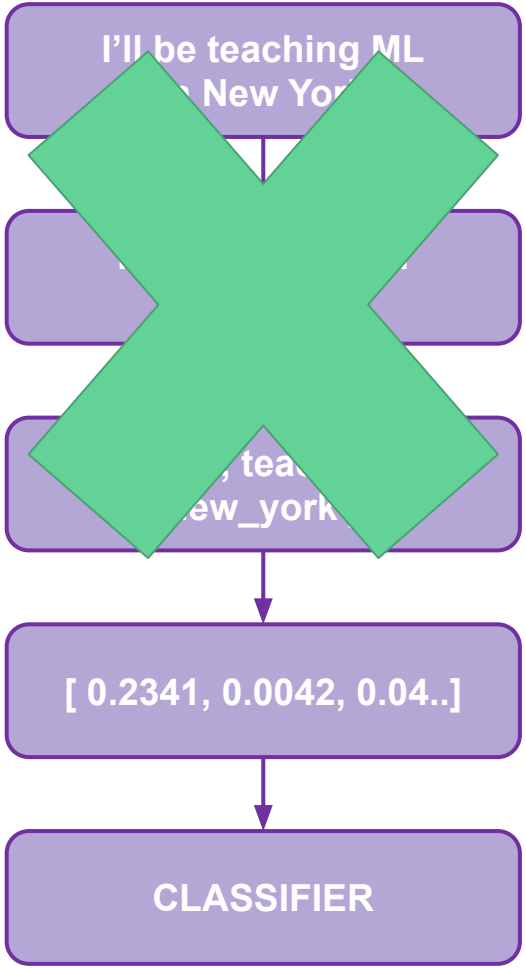
Text classification flow (again)

- Pre-processing: casing (?), punctuation (?), stop words (?).
- Tokenization: split text in tokens { stemming (?), lemmatization (?), phrases (?) }.
- Vectorization: apply a vectorization technique -> **word2vec**
- Modelling (training and testing): train a classifier model over text vectors - this is equivalent to your previous experience with scikit-like APIs.



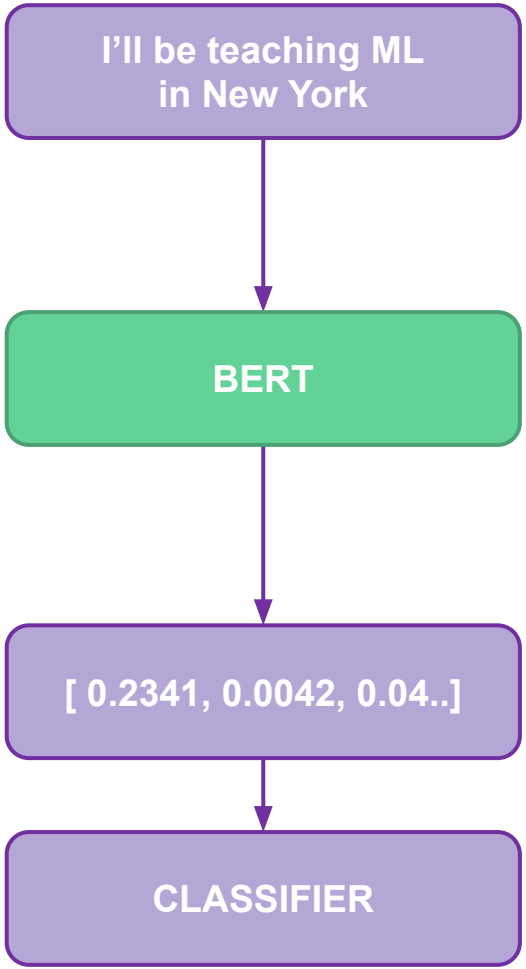
Text classification with BERT (teaser)

- Download a large pre-trained model, such as BERT.
- Vectorization: transform text into dense vectors.
- Feed vectors to a neural architecture for classification.



Text classification with BERT (teaser)

- Download a large pre-trained model, such as BERT.
- Vectorization: transform text into dense vectors.
- Feed vectors to a neural architecture for classification.



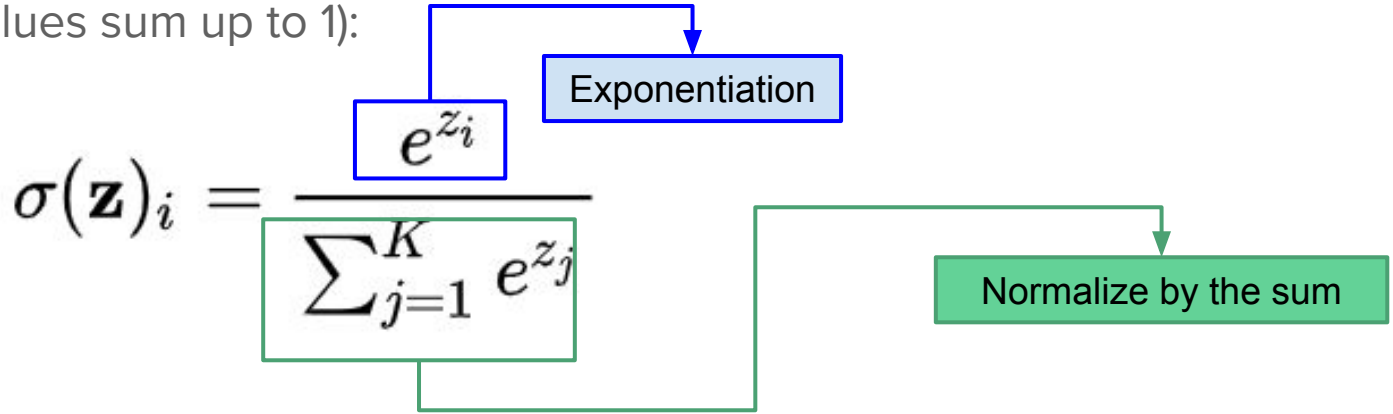
The softmax function (teaser)

- How do we go from a set of values (floats) outputted by neurons to a classification decision?
- The last activation function for classification is typically the *softmax* function, which takes a vector of floats and converts it into a vector of probabilities (i.e. values sum up to 1):

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

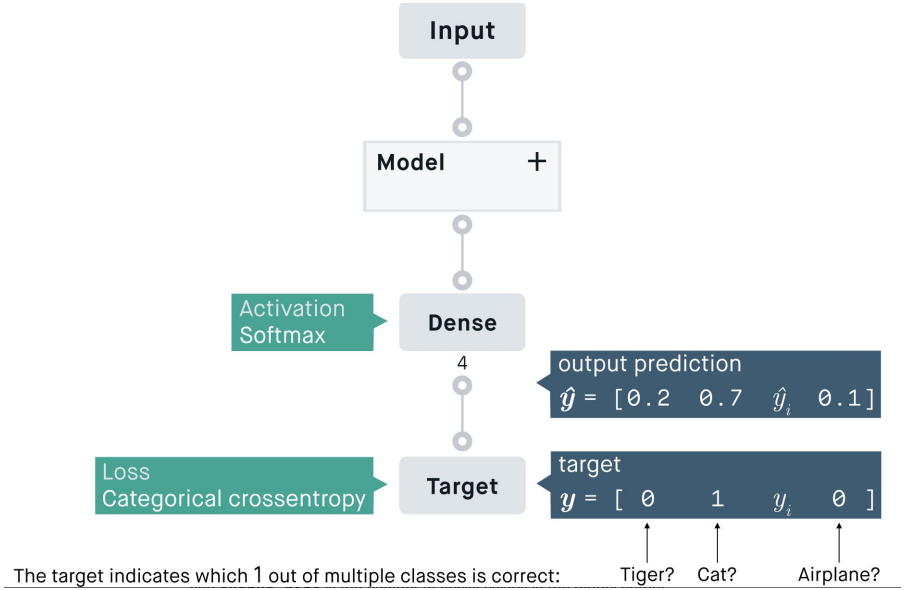
The softmax function (teaser)

- How do we go from a set of values (floats) outputted by neurons to a classification decision?
- The last activation function for classification is typically the *softmax* function, which takes a vector of floats and converts it into a vector of probabilities (i.e. values sum up to 1):



Cross-entropy loss (teaser)

- Given the output of a softmax and a true label for a classification problem, how do we define our loss?
- The typical choice is what is known as categorical cross-entropy:
  - "*The cross-entropy error function instead of the sum-of-squares for a classification problem leads to faster training as well as improved generalization*"



$$\text{Loss} = -\sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i$$

More readings

Word embeddings

- Skip-gram with negative sampling - the algorithm we explained - is one among many ways of getting word embeddings: another popular choice is GloVe.
- Piero Molino's fantastic slides (and lecture) on word2vec, and, more generally, the link between ideas in linguistics and philosophy and modern distributional methods in NLP.

Deep Neural Networks

- Our explanations of NN has been comically short, and mostly useful to introduce basic intuition. There are many resources for a proper deep learning introduction: in particular, the fantastic book by Keras creator, the deep learning book, and Fast AI course are excellent places to start.