Testing ML Systems

Things that may be obvious in retrospect (but perhaps not the first time around)

Deep Dream

Talk through example

- Key points to cover
- Insight behind Deep Dream
- Gradient Ascent
- Functional API

Also, midterms look good!

- Returned tonight, we'll curve and post grades as soon as we can.

Administrative stuff

Guest lecture next week

- Reinforcement learning
- Gabriela Tavares, an engineer on search, received her PhD from Caltech.
- Fundamentals of RL & an intro to deep RL.

Today's agenda

Testing

Two helpful papers

- Hidden Technical Debt in Machine Learning Systems
- What's your ML test score?

I love this style of writing, simple collections of useful advice.

Structured data

- Feature columns (now in Keras)
- "Wide and Deep" models for ranking

Used heavily in production with structured data

Powers Google Play.

A lot of testing is obvious in retrospect

- Many of the best practices in this talk aren’t rocket science.
- They’re learned from assorted mistakes over the years.
- One of the best ways to avoid those in your own work is just having an awareness of these topics.

Why test?

Why test?

The goal is not to add new functionality, but to **enable future improvements**, reduce errors, and improve maintainability.

- This one is important. Most of the time, you will be working with large codebases written by other people over a long time. They will be too large to understand (it could take months before you have a firm grasp on everything).

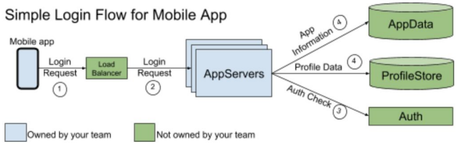
- How can you know your changes won't break anything?

Testing on the Toilet

What Makes a Good End-to-End Test?

An end-to-end test tests your entire system from one end to the other, treating everything in between as a black box. **End-to-end tests can catch bugs that manifest *across* your entire system**. In addition to unit and integration tests, they are a critical part of a balanced testing diet, providing confidence about the health of your system in a near production state. Unfortunately, end-to-end tests are **slower, more flaky, and more expensive to maintain** than unit or integration tests. Consider carefully whether an end-to-end test is warranted, and if so, how best to write one.

Let's consider how an end-to-end test might work for the following "login flow":



In order to be cost effective, an end-to-end test should **focus on aspects of your system that cannot be reliably evaluated with smaller tests**, such as resource allocation, concurrency issues and API compatibility. More specifically:

- **For each important use case, there should be one corresponding end-to-end test.** This should include one test for each important class of error. The goal is the keep your total end-to-end count low.
- Be prepared to **allocate at least one week a quarter per test to keep your end-to-end tests stable** in the face of issues like slow and flaky dependencies or minor UI changes.
- **Focus your efforts on verifying overall system behavior instead of specific implementation details**; for example, when testing login behavior, verify that the process succeeds independent of the exact messages or visual layouts, which may change frequently.
- **Make your end-to-end test easy to debug** by providing an overview-level log file, documenting common test failure modes, and preserving all relevant system state information (e.g.: screenshots, database snapshots, etc.).

End-to-end tests also come with some important caveats:

- System components that are owned by other teams may change unexpectedly, and break your tests. This increases overall maintenance cost, but can highlight incompatible changes
- **It may be more difficult to make an end-to-end test fully hermetic**; leftover test data may alter future tests and/or production systems. Where possible keep your test data ephemeral.
- An end-to-end test often necessitates multiple test doubles (fakes or stubs) for underlying dependencies; they can, however, have a high maintenance burden as they drift from the real implementations over time.

More information, discussion, and archives:

testing.googleblog.com

Copyright Google Inc. Licensed under a Creative Commons Attribution–ShareAlike 4.0 License (http://creativecommons.org/licenses/by-sa/4.0/).

Testing on the Toilet

Keep Cause and Effect Clear

Can you tell if this test is correct?

```
@Test public void testIncrement_existingKey() {
  assertEquals(9, tally.get("key1"));
}
```

It's impossible to know without seeing how the `tally` object is set up:

```
private final Tally tally = new Tally();
@Before public void setUp() {
   tally.increment("key1", 8);
   tally.increment("key2", 100);
   tally.increment("key1", 0);
   tally.increment("key1", 1);
}
// 200 lines away
@Test public void testIncrement_existingKey() {
  assertEquals(9, tally.get("key1"));
}
```

The problem is that the modification of `key1`'s values *occurs 200+ lines away from the assertion.* Otherwise put, **the *cause* is hidden far away from the *effect*.**

Instead, **write tests where the effects immediately follow the causes**. It's how we speak in natural language: "If you drive over the speed limit *(cause)*, you'll get a traffic ticket *(effect)*." Once we group the two chunks of code, we easily see what's going on:

```
private final Tally tally = new Tally();
@Test public void testIncrement_newKey() {
  tally.increment("key", 100);
  assertEquals(100, tally.get("key"));
}
@Test public void testIncrement_existingKey() {
  tally.increment("key", 8);
  tally.increment("key", 1);
  assertEquals(9, tally.get("key"));
}
@Test public void testIncrement_incrementByZeroDoesNothing() {
  tally.increment("key", 8);
  tally.increment("key", 0);
  assertEquals(8, tally.get("key"));
}
```

This style may require a bit more code. Each test sets its own input and verifies its own expected output. **The payback is in more readable code and lower maintenance costs.**

More information, discussion, and archives:

testing.googleblog.com

Copyright Google Inc. Licensed under a Creative Commons Attribution–ShareAlike 4.0 License (http://creativecommons.org/licenses/by-sa/4.0/).

Testing, as we usually think about it

An important and interesting field. I think it's often undervalued and (incorrectly!) seen as boring.

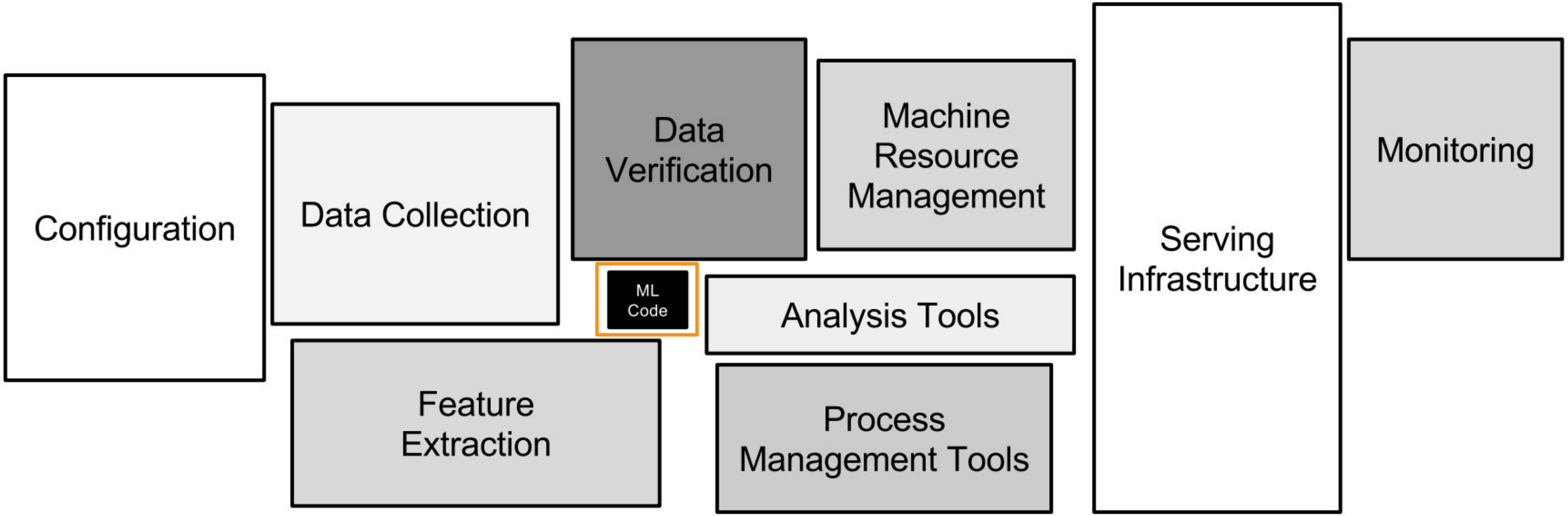
- Unit tests
- Regression tests
- Integration tests, mocks, stubs.
- End-to-end tests

Only a small fraction of real-world ML systems is composed of the code for the model, as shown by the small black box in the middle.

Defining and tuning an accurate model.

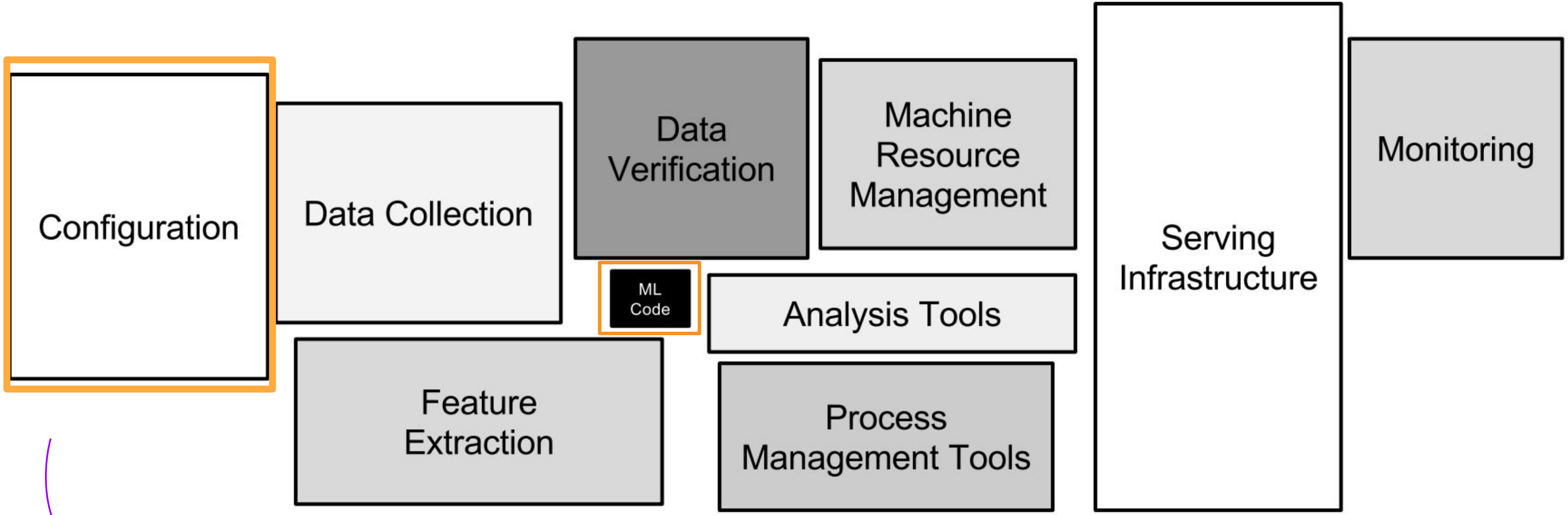
[Hidden Technical Debt in Machine Learning Systems]

Only a small fraction of real-world ML systems is composed of the code for the model, as shown by the small black box in the middle.



[Hidden Technical Debt in Machine Learning Systems]

Only a small fraction of real-world ML systems is composed of the code for the model, as shown by the small black box in the middle.



Should be checked into repo and treated as code (w/ code reviews for changes).

[Hidden Technical Debt in Machine Learning Systems]

Hidden Technical Debt in Machine Learning Systems

Another one of those excellent papers that few people noticed until recently.

Developing and deploying ML systems

- ?

Maintaining them over time

- ?

[Hidden Technical Debt in Machine Learning Systems]

Hidden Technical Debt in Machine Learning Systems

Another one of those excellent papers that few people noticed until recently.

Developing and deploying ML systems

- Fast and cheap.

Maintaining them over time

- Difficult and expensive (many people, many code changes)

Hidden Technical Debt in Machine Learning Systems

Technical debt

Another phrase I like: “mortgaging the future”.

May be paid down by...

- Refactoring code, improving unit tests, **deleting dead code**, reducing dependencies, tightening, APIs, and improving documentation.

Deferring such payments result in compounding costs.

- **Hidden debt is dangerous because it compounds silently.**

Knight Capital

$460M loss + $12M fine

Bug in an automated system caused trades losing $472 million in 45 minutes.

- An obsolete code path designed to be used in an experimental environment was activated in production, causing unexpected behavior.

Quick discussion

- In retrospect, how would you fix the above?

https://www.sec.gov/news/press-release/2013-222

Knight Capital

$460M loss + $12M fine

Bug in an automated system caused trades losing $472 million in 45 minutes.

- An obsolete code path designed to be used in an experimental environment was activated in production, causing unexpected behavior.

Quick discussion

- In retrospect, how would you fix the above?

Should have been removed from codebase

https://www.sec.gov/news/press-release/2013-222

Knight Capital

“During the first 45 minutes after the market opened on August 1, Knight Capital’s router rapidly sent more than 4 million orders into the market when attempting to fill just 212 customer orders.”

Quick discussion

- In retrospect, how would you fix the above?

Knight Capital

"During the first 45 minutes after the market opened on August 1, Knight Capital's router rapidly sent more than 4 million orders into the market when attempting to fill just 212 customer orders."

Quick discussion

- In retrospect, how would you fix the above?

- Enforce limits on your systems.

Testing in ML is much harder

Why?

Testing in ML is much harder

Data influences behavior

- We're using ML exactly when it's difficult or impossible to write software logic to product the desired behavior *without* depending on data.

A number of the following ideas may feel obvious in retrospect. They're important to be aware of.

Testing in ML is much harder

Data influences behavior

- We’re using ML exactly when it's difficult or impossible to write software logic to product the desired behavior *without* depending on data.

- This makes testing components of ML systems in isolation difficult: a small change in data can have large downstream effects.

- **Changing Anything Changes Everything.**

A number of the following ideas may feel obvious in retrospect. They’re important to be aware of.

What could go wrong with this example?

Anti-pattern

In order to serve an embedding trained with an Estimator, you can send out the lower dimensional representation of your categorical variable along with your normal prediction outputs. Embedding weights are saved in the SavedModel, and one option is to share that file itself. Alternatively, you can serve the embedding on demand to clients of your machine learning team—which may be more maintainable,because those clients are now only loosely coupled to your choice of model architecture. They will get an updated embedding every time your model is replaced by a newer, better version.

Simple example: cascades

Say model m_a is used as an input to m_b

- Updating m_a changes the distribution of inputs to m_b causing unexpected behavior.

- Designers of m_b may not have been aware m_a changed.

Entanglement

ML systems mix signals together

- Consider a system that takes features $\mathbf{x_1}$, …, $\mathbf{x_2}$ as input

If we change the distribution of $\mathbf{x_1}$ in the data, the importance or weights on the remaining features may all change.

Data should be versioned.

Entanglement

ML systems mix signals together

- Consider a system that takes features $\mathbf{x_1}$, …, $\mathbf{x_2}$ as input

If we change the distribution of $\mathbf{x_1}$ in the data, the importance or weights on the remaining features may all change.

- Likewise, adding a new feature $\mathbf{x_{n+1}}$ can cause similar changes.

- As can removing a feature.

Data -- and services that provide features -- should be versioned.

Entanglement is true for hyperparameters as well

None of these are independent

- # of layers
- # of units
- epochs
- batch size
- optimizer
- optimizer settings
- weight initialization
- The version of your library (probably)
- etc

Undeclared Consumers

Often, a prediction from a ML model is made widely accessible, either at runtime or by writing to logs later consumed by other systems.

- **Quick discussion**: what can go wrong if you consume data from one system's output, without them knowing about it?

Undeclared Consumers

Often, a prediction from a ML model is made widely accessible, either at runtime or by writing to logs later consumed by other systems.

- Without access controls, some of these consumers may be undeclared, silently using the output of a given model.
- Creates a hidden dependency on the model elsewhere in the stack.
- Changes to the model will have downstream consequences - often in ways that are poorly understood and hard to detect.

This can be addressed with access restrictions or SLAs.

Unstable Data Dependencies

A model $\mathbf{m_1}$ uses pretrained embeddings as input.

- Engineering ownership of the embeddings changes. Improvements are made to increase their accuracy.

- Has an unintended consequence on $\mathbf{m_1}$, which is now miscalibrated.

This can be addressed by <u>versioning data</u> - which is the approach taken by TF Hub.

TensorFlow Hub

https://tfhub.dev/google/universal-sentence-encoder/1

TensorFlow Hub

USER GUIDE

universal-sentence-encoder

Version

1

Module type

Text embedding

Network

DAN

Language

English

Publisher

Google

Encoder of greater-than-word length text trained on a variety of data.

Version

1

Warning: An updated version of this module is available at https://tfhub.dev/google/universal-sentence-encoder/2 .

Overview

The Universal Sentence Encoder encodes text into high dimensional vectors that can be used for text classification, semantic similarity, clustering and other natural language tasks.

The model is trained and optimized for greater-than-word length text, such as sentences, phrases or short paragraphs. It is trained on a variety of data sources and a variety of tasks with the aim of dynamically accommodating a wide variety of natural language understanding tasks. The input is variable length English text and the output is a 512 dimensional vector. We apply this model to the STS

Use this module

Copy URL

Open Colab notebook

Underutilized Data Dependencies

This one is less obvious.

In code, underutilized dependencies are packages that are mostly unneeded.

- In ML, underutilized data dependencies are input signals that provide little incremental benefit.

- These can be removed with little to no determinant.

- There is a cost to keeping them.

Quick discussion: what's the cost?

Example

E.g., ICD-9 or ICD-10.

EHR setting. A hospital is updating from old procedure codes to new ones.

- To ease transition, new codes are added to system alongside the old ones.
- All records have both codes.
- A year later, the database used to look up the old codes is shutdown.

New records now only have the new code. **What happens?**

Example

E.g., ICD-9 or ICD-10.

EHR setting. A hospital is updating from old procedure codes to new ones.

- To ease transition, new codes are added to system alongside the old ones.
- All records have both codes.
- A year later, the database used to look up the old codes is shutdown.

New records now only have the new code.

- **This will not be a good day.**

More examples

Legacy Features

- A feature F is included in a model early in its development. Over time, F is made redundant by new features, but not removed.

More examples

Bundled Features

- A group of features is evaluated and found to be beneficial.

- Because of deadline pressures, all the features in the group are added to the model, possibly including features with little or no value.

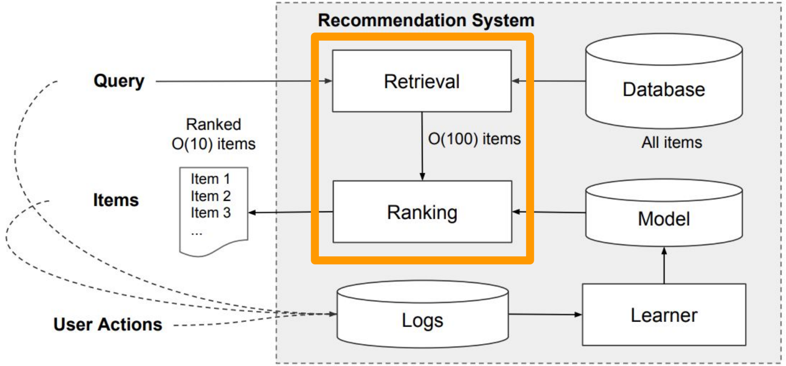
More examples

Epsilon-features

- Tempting to improve model accuracy…
- ... even when the accuracy gain is small or when the complexity of a new feature might be high.

Feedback loops

- The ranker's output impacts metrics...
- ... which are recorded in logs.
- … which are used to train the retriever.
- … which provides data for the ranker.



Tip: place action limits on systems designed to take actions in the real world

- E.g., maximum # of emails that can be marked as spam.
- Trigger an alert if reached.

Tip: monitor distribution of predicted labels

- Should roughly equal that from the training data.
- Divergence might signal a bug or a condition that needs attention.

Continued

What's your ML Test Score?

Also great work.

A collection of best practices learned from experience

Here are a few of my favorites.

- Definitely read the rest on your own.

Hidden Technical Debt in Machine Learning Systems

Test that preprocessing code is **identical** between training and serving

- Ideally, use the same codepath.
- Challenging w/ different languages and systems.
- E.g., vectorizing text in Keras (Python) when **deploying your model in a webpage** (with TensorFlow.js).
- Or, more commonly, when reusing a model developed by someone else with a messy, spaghetti code data preprocessing pipeline.

Test the relationship between offline proxy metrics and the actual impact metrics

- Improving accuracy sounds good! But, does it matter?

- If you tell your boss Sarah you've improved the Ranker's accuracy by 7%, what will she say?

Test the relationship between offline proxy metrics and the actual impact metrics

- Improving accuracy sounds good! But, does it matter?
- If you tell your boss Sarah you've improved the Ranker's accuracy by 7%, what will she say?
- How do the metrics we measure in our model correspond w/ metrics we care about in the **real world** (e.g., user satisfaction)?

Quick discussion:

- How can you measure this?

Test the effect of model staleness

- If predictions are based on a model trained yesterday versus last week versus last year, what is the impact on the live metrics of interest?

Quick discussion

- How often do you think a large software company retrains their recommendation models?

- Every… minute? Hour? Day? Week? Month? Year?

Test against a simpler model as a baseline

- E.g., a linear model with a few features.

- Use this to confirm a more complex model and more features are worth it.

Test on data slices

- Say your model is very accurate on the entire dataset.
- Are there any segments of users for whom it's not?
- In addition to testing on the entire dataset, consider testing on subsets (be careful not to introduce bias here).

Canary

- Deploy new models to a small number of users (say, 2%) before scaling up.

Rollbacks

- Build and test reliable infrastructure to support rolling back to previous versions of your model in production in case of problems.
- This is hard, but important, and will help you sleep at night.

Structured data

The world is not a Kaggle competition

Perception problems on Kaggle

- Almost always won by DL

Structured data problems on Kaggle

- Almost always won by tree-based models

The world is <u>not</u> a Kaggle competition

Perception problems on Kaggle

- Almost always won by DL

Structured data problems on Kaggle

- Almost always won by tree-based models

How can we go further? What can we do that's new?

Challenges with structured data

As usual, it's not modeling! What's hard here?

- **Quick discussion: where does most of your time go when trying to classify structured data?**

- Imagine you're working with medical records collected by several institutions and hundreds of providers.

Challenges with structured data

As usual, it's not modeling! What's hard here?

- **Quick discussion: where does most of your time go when trying to classify structured data?**

- Imagine you're working with medical records collected by several institutions and hundreds of providers.

Data cleaning. Interesting question: is this as necessary with DL?

Two of my favorite libraries. How do they fare on the structured data example in TensorFlow?

github.com/tensorflow/models/tree/master/official/wide_deep

Trees are hard to beat

- For most typical business problems, random forests in scikit-learn are all you need.

Always use a tree-based model as a strong baseline. For most business problems, it's probably the best solution.

DL creates new opportunities

- Ability to **combine structured + unstructured data** in a single model.

Quick discussion:

- Any ideas? How might this look in a medical domain?

Trees are hard to beat

- For most typical business problems, random forests in scikit-learn are all you need.

Always use a tree-based model as a strong baseline. For most business problems, it's probably the best solution.

DL creates new opportunities

- Ability to **combine structured + unstructured data** in a single model.

Quick discussion:

- Any ideas? How might this look in a medical domain?

Imagine training a model on structured data (age, blood glucose, bp, etc) + the complete written notes(!) and/or + images(!) and/or + raw output of EKGs, etc.

Trees are hard to beat

- For most typical business problems, random forests in scikit-learn are all you need.

Always use a tree-based model as a strong baseline. For most business problems, it's probably the best solution.

DL creates new opportunities

- Ability to **combine structured + unstructured data** in a single model.

- **Less need for data cleaning** (instead of mapping terms to a canonical format, use an embedding to learn the relationships)

Trees are hard to beat

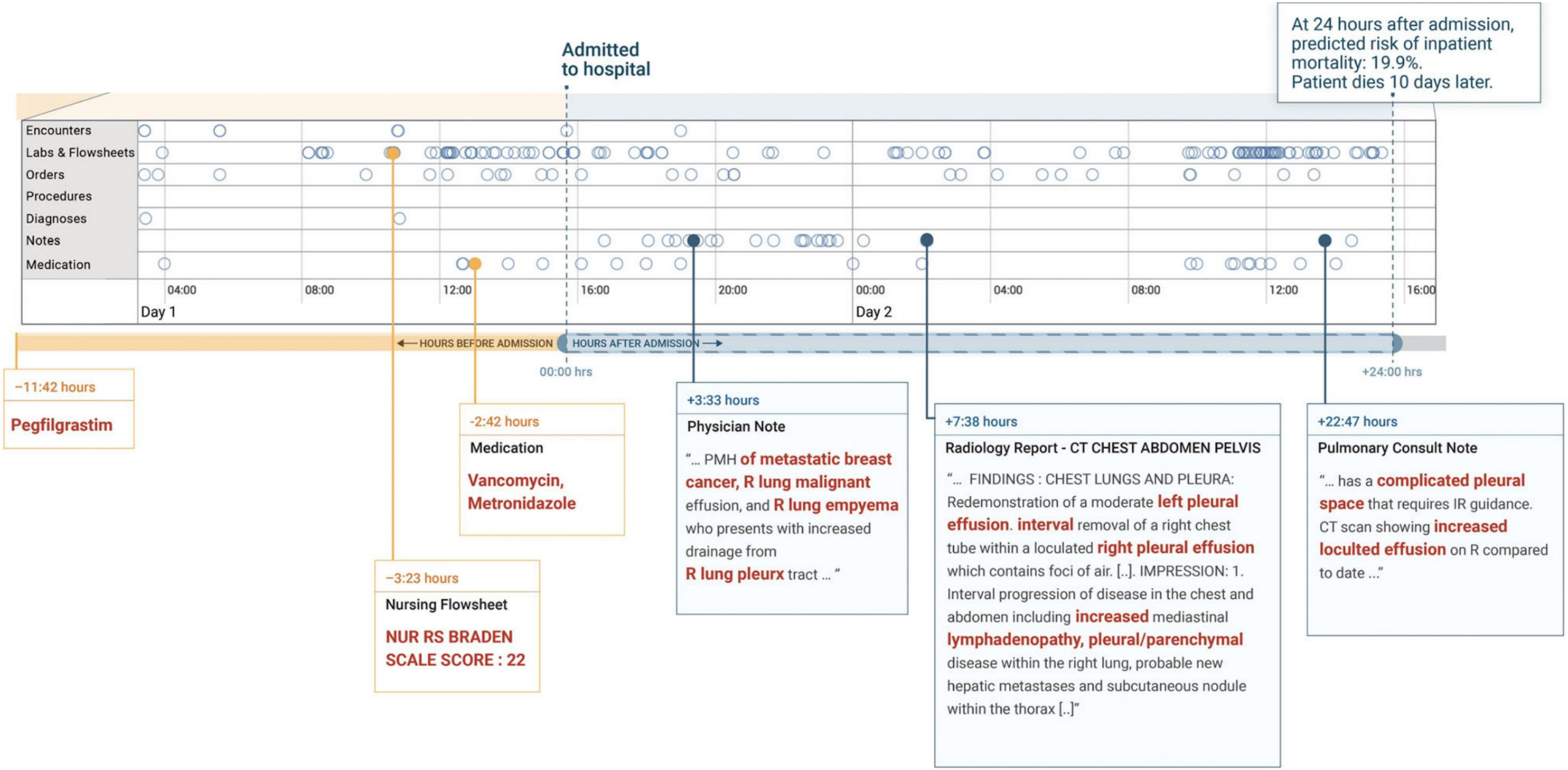
- For most typical business problems, random forests in scikit-learn are all you need.

Always use a tree-based model as a strong baseline. For most business problems, it's probably the best solution.

DL creates new opportunities

- Ability to **combine structured + unstructured data** in a single model

- **Potentially, less need for data cleaning** (instead of mapping terms to a canonical format, use an embedding to learn the relationships)



[Scalable and accurate deep learning for electronic health records]

Dataset contained

- **Structured data**: Patient demographics, provider orders, diagnoses, procedures, medications, laboratory values, vital signs, and flowsheet data.

- **Unstructured data**: Free-text medical notes

What's interesting:

- Temporal sequence
- **Minimal data cleaning**

Datasets were mapped to a high-level representation of healthcare data, but each individual site's idiosyncratic codings were left unchanged (e.g., shorthand not corrected in structured data attributes).

A few notes for people reading at home. There are a couple of areas where DL on structured data fails, and few new opportunities.

- Ability to process structured data w/ less data cleaning.
- Ability to combine structured and unstructured data

If you're working on a business problem (say, a 10^2 to 10^5 rows of structured data in a CSV file): use a tree-based model (at a minimum as a strong baseline, and probably a best solution).

If your problem has 10^6 to **10^9 rows**, and/or unstructured data as well -> consider DL.

Not uncommon, Google Play Store was trained on 500 x 10^9 examples in 2016.

More importantly, opportunities abound for research in avoiding data cleaning.

[Scalable and accurate deep learning for electronic health records]

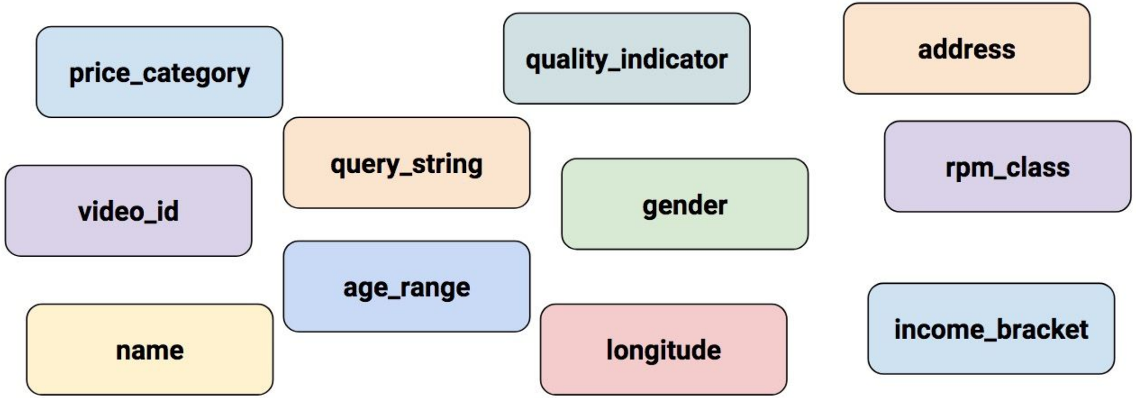
Facets and structured data walkthrough

Feature columns

Latest code

https://www.tensorflow.org/alpha/tutorials/keras/feature_columns

Say you have a bunch of structured data in a CSV



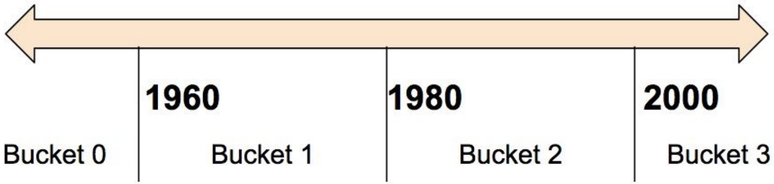
Feature columns describe how to treat each column

Numeric columns

```
# Raw input to a numeric column.

year_feature = tf.feature_column.numeric_column(key="Year")
```

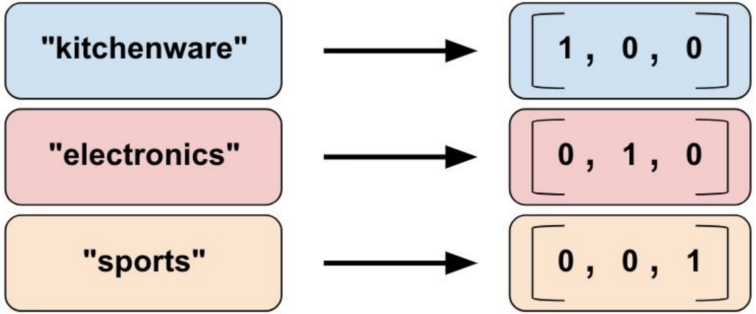
Bucketized columns

```
# Raw input to a numeric column.
year_feature = tf.feature_column.numeric_column(key="Year")


# Then, bucketize the numeric column on the years 1960, 1980, and 2000.
bucketized_year = tf.feature_column.bucketized_column(
    source_column = year_feature,
    boundaries = [1960, 1980, 2000])
```



Categorical columns

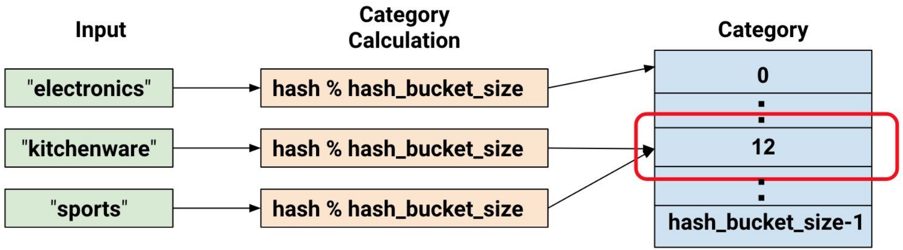
```
# Create a categorical feature by mapping the input to one of
# the elements in the vocabulary list.
category_feature =
    tf.feature_column.categorical_column_with_vocabulary_list(
        key="category",
        vocabulary_list=["kitchenware",
                         "electronics",
                         "sports"])
```



Hashed columns

```
# Create a hashed feature column
hashed_feature_column =
    tf.feature_column.categorical_column_with_hash_bucket(
        key = "category",
        hash_bucket_size = 100) # The number of hash buckets
```

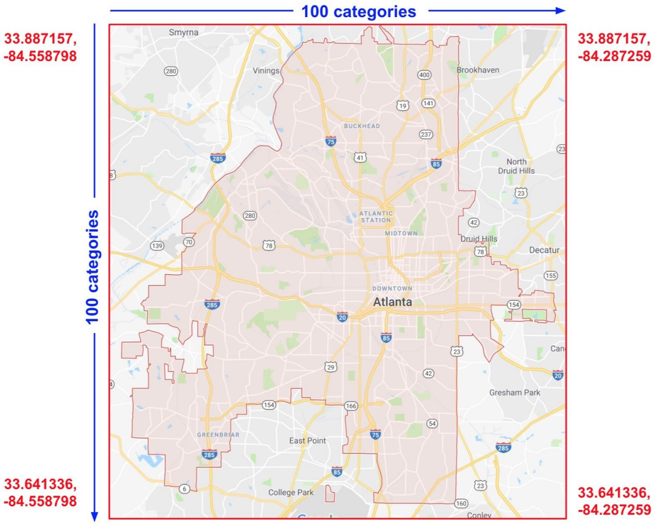
Collisions happen ¯_(ツ)_/¯



Crossed columns

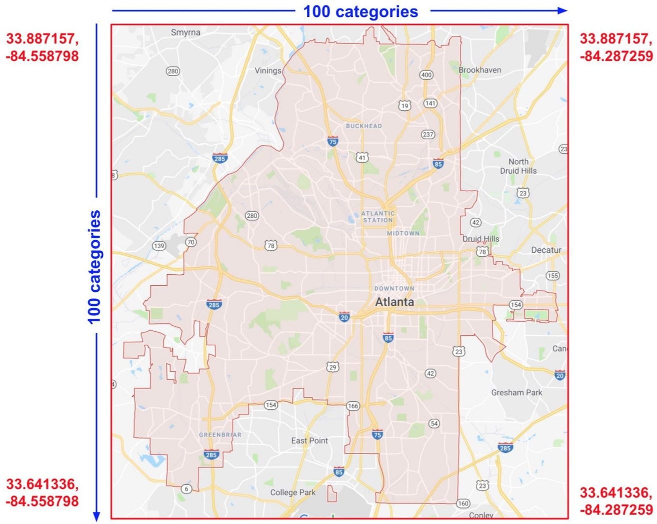
```
# Bucketize latitude and longitude

latitude_fc = tf.feature_column.bucketized_column(

    tf.feature_column.numeric_column('latitude'),

    list(atlanta.latitude.edges))

longitude__fc = tf.feature_column.bucketized_column(

    tf.feature_column.numeric_column('longitude'),

    list(atlanta.longitude.edges))
```

Feature crosses enable a linear model to learn separate weights for each combination of features.



Crossed columns

```
# Cross the columns, using 5000 hash bins.
crossed__fc = tf.feature_column.crossed_column(
    [latitude_fc, longitude_fc], 5000)

# Produces a grid of features, e.g.,
(0,0),  (0,1), ... , (0,99)
...      ...             ...
(99,0), (99,1), ... , (99, 99)
```

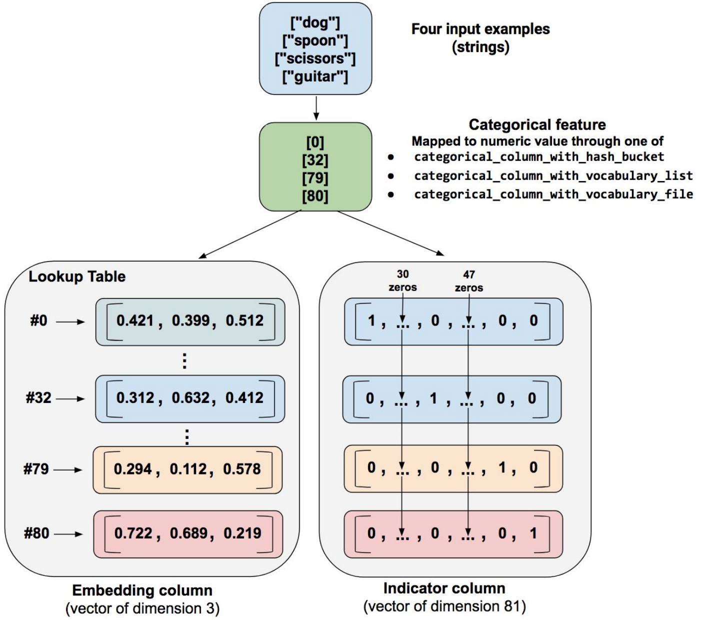


Feature crosses enable a linear model to learn separate weights for each combination of features.

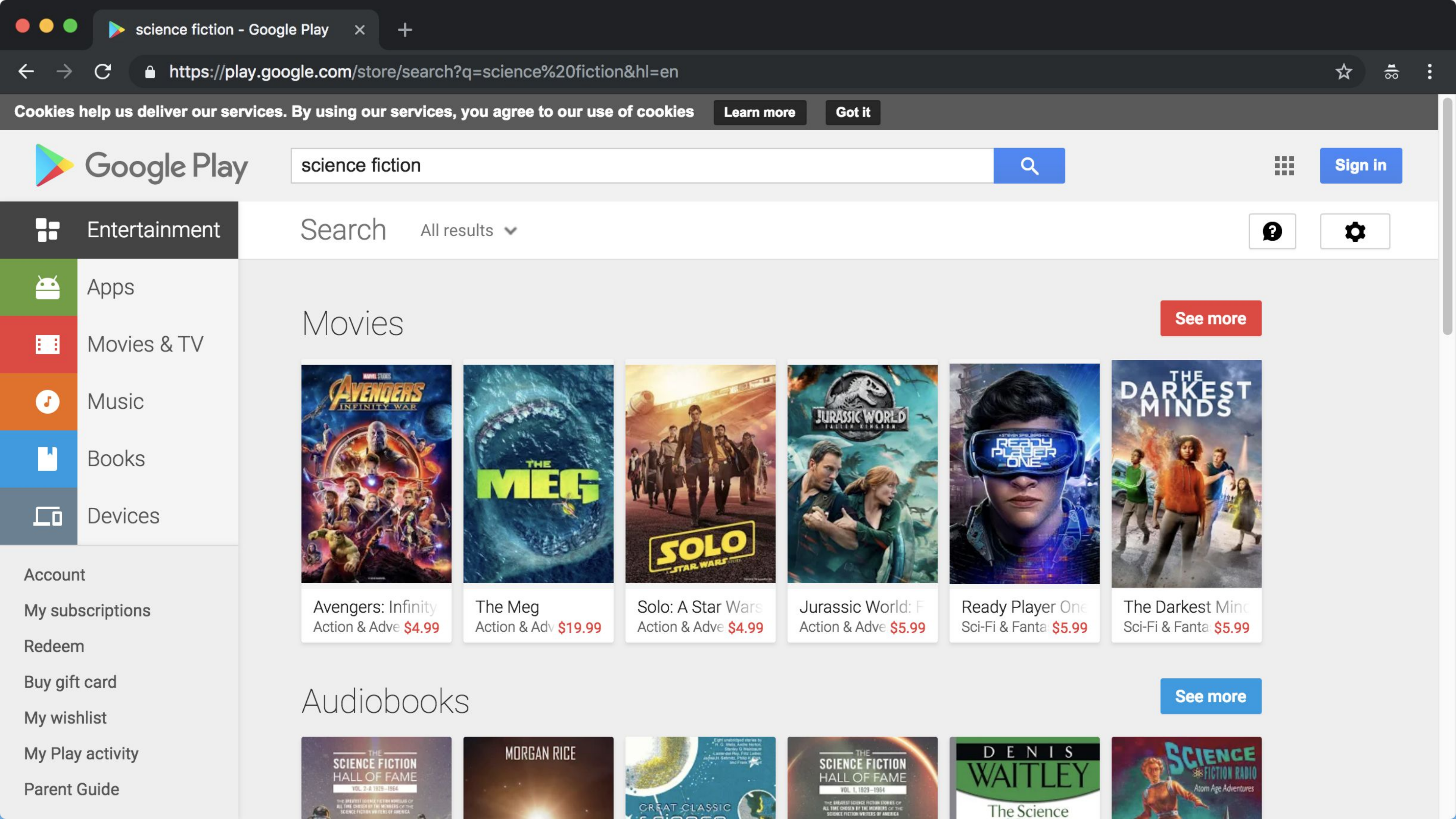
Embedding columns

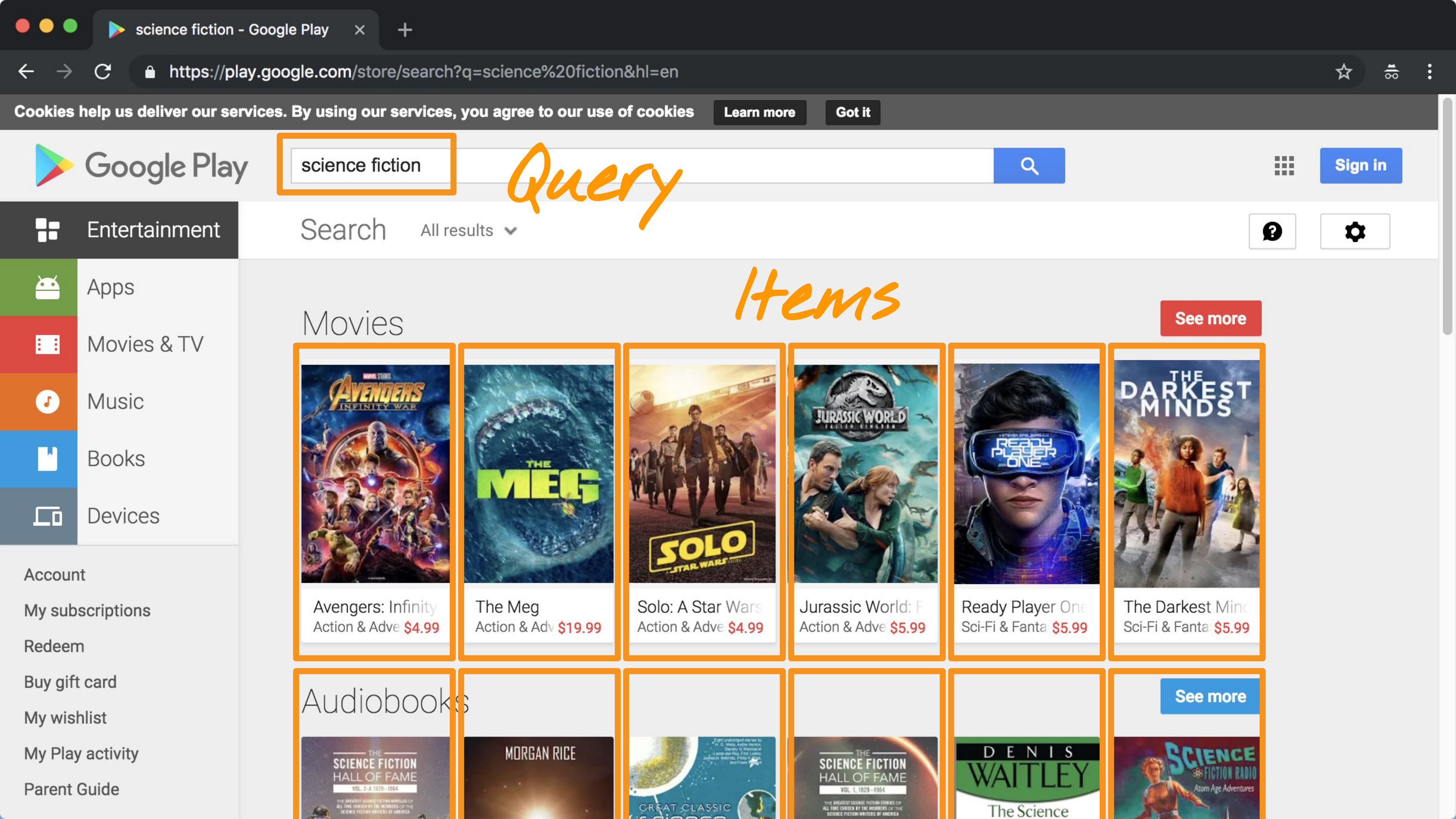
```
categorical_column = ... # Create any categorical column


# Represent the categorical column as an embedding column.

embedding_fc = tf.feature_column.embedding_column(

    categorical_column=categorical_column,

    dimension=embedding_dimensions)
```

*Rule of thumb: a good starting point for the embedding dimension is 0.25 * vocab size*



Recommendation systems


science fiction - Google Play
https://play.google.com/store/search?q=science%20fiction&hl=en
Cookies help us deliver our services. By using our services, you agree to our use of cookies
Learn more
Got it
Google Play
science fiction
Sign in
Entertainment
Apps
Movies & TV
Music
Books
Devices
Account
My subscriptions
Redeem
Buy gift card
My wishlist
My Play activity
Parent Guide
Search
All results
Movies
See more
Avengers: Infinity
Action & Adve $4.99
The Meg
Action & Adv $19.99
Solo: A Star Wars
Action & Adve $4.99
Jurassic World: F
Action & Adve $5.99
Ready Player One
Sci-Fi & Fanta $5.99
The Darkest Minc
Sci-Fi & Fanta $5.99
Audiobooks
See more
THE SCIENCE FICTION HALL OF FAME
MORGAN RICE
GREAT CLASSIC
THE SCIENCE FICTION HALL OF FAME
DENIS WAITLEY
The Science
SCIENCE FICTION RADIO
Atom Age Adventures


science fiction - Google Play
https://play.google.com/store/search?q=science%20fiction&hl=en
Cookies help us deliver our services. By using our services, you agree to our use of cookies
Learn more
Got it
Google Play
science fiction
Query
Sign in
Entertainment
Search
All results
Apps
Movies & TV
Music
Books
Devices
Account
My subscriptions
Redeem
Buy gift card
My wishlist
My Play activity
Parent Guide
Items
Movies
See more
Avengers: Infinity
Action & Adve $4.99
The Meg
Action & Adv $19.99
Solo: A Star Wars
Action & Adve $4.99
Jurassic World: F
Action & Adve $5.99
Ready Player One
Sci-Fi & Fanta $5.99
The Darkest Minc
Sci-Fi & Fanta $5.99
Audiobooks
See more

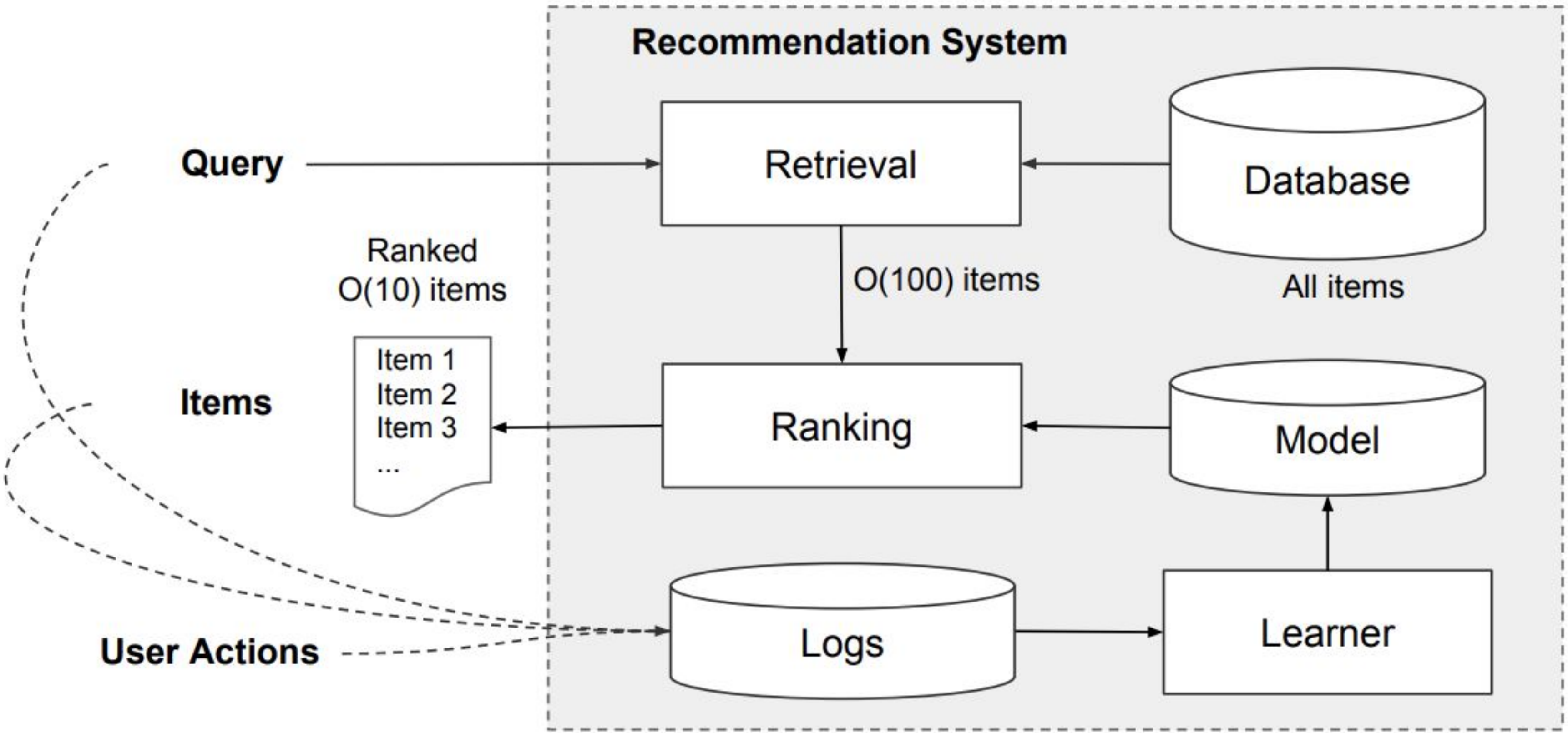
Large-scale recommendation systems

[Google Play store]

- Millions of users
- Millions of items

Given a query (“science fiction”) return a list of apps / movies / books / etc.

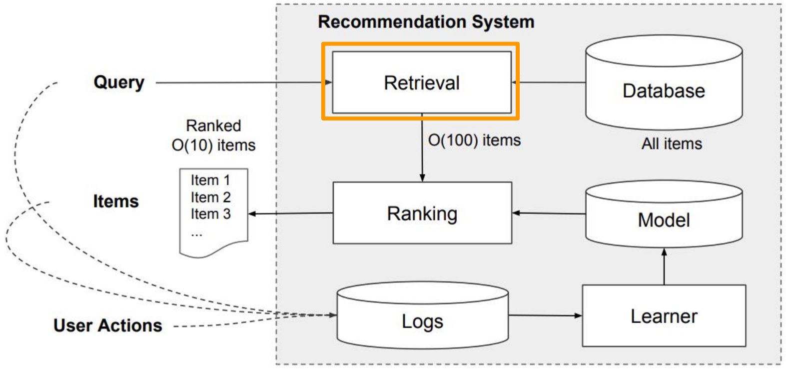
- Latency requirements: ~10 milliseconds
- Intractable to score the query against every app


Recommendation System
Query
Retrieval
Database
Ranked
O(10) items
O(100) items
All items
Item 1
Item 2
Item 3
...
Items
Ranking
Model
Logs
Learner
User Actions

Retrieval

Quickly produces a rough list of items that match the query

- Uses a combination of machine-learned models.
- And manually defined rules.
- Basic features.



Ranking

Sorts the items by the probability a user will take an action.

- Uses a combination of machine-learned models and manually defined rules.

User features

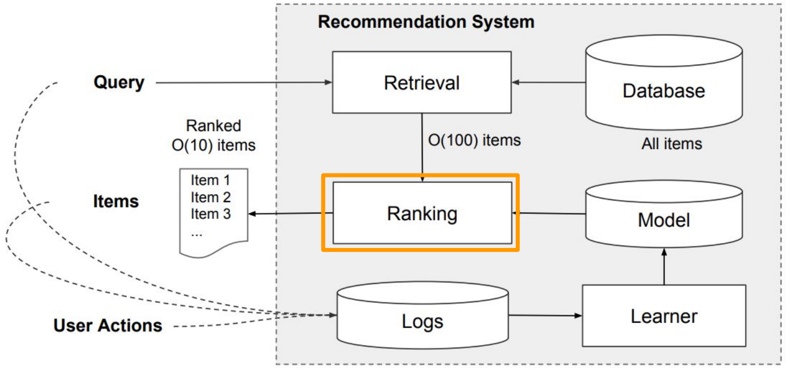
Country, language, demographics, etc.

Contextual features

Device type, hour of the day, day of the week, etc.

Item features

Movie title, price, historical stats, etc.



Memorization and generalization

Memorize rare queries

- A small number of users may be interested in obscure apps / movies / etc.

Generalize

- Learn similarities between groups of users / groups of items / and use those to recommend related content.

Aside

Quick discussion

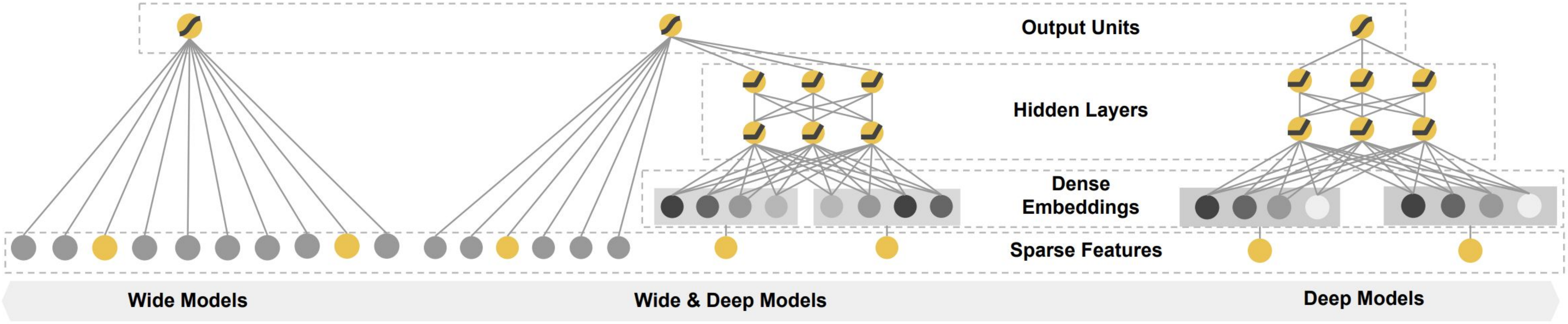
- How often do you think a large software company might retrain their recommendation models, and deploy a fresh copy in production?

- Once a [year, month, week, day, hour, minute?]

Wide and Deep

Facets demo

https://pair-code.github.io/facets/



A simple idea that's been successful in practice for the ranking part of this story.

Mayyyyyybe a bit oversold, FYI.

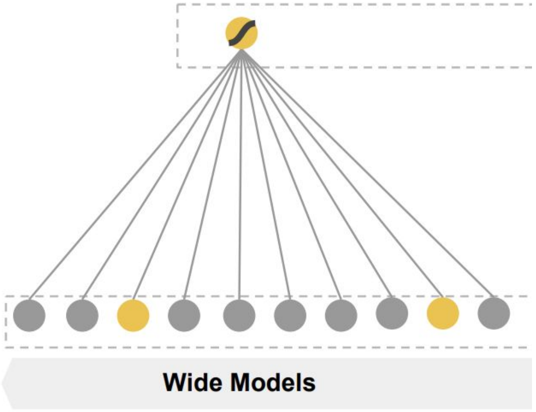
Memorization

Pros:

- Linear (or wide) models are simple, interpretable, fast to train.
- Feature importances? Print out weights after training, sort descending.

Cons:

- Cannot learn interactions between features.

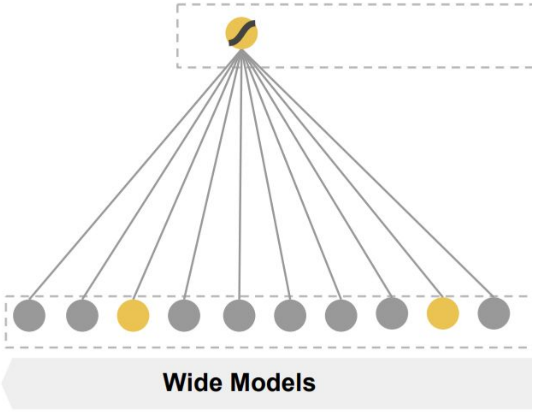


Quick discussion: which of the feature columns that we've seen so far can be used to help a wide model learn interactions between features?

Memorization

Feature crosses

- ANDs two input features into one.
- E.g., create a new input feature which is true if "AND(gender=female, language=en)"

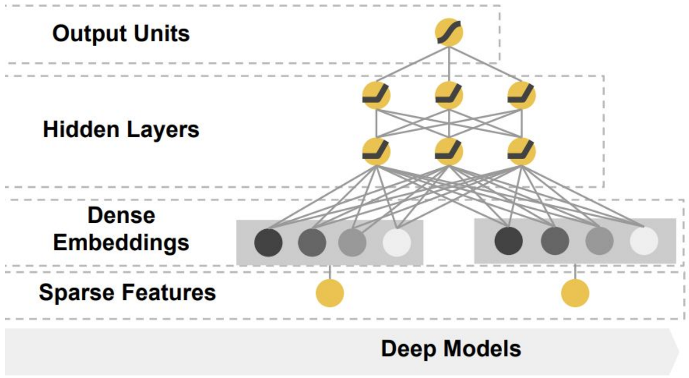


Generalization

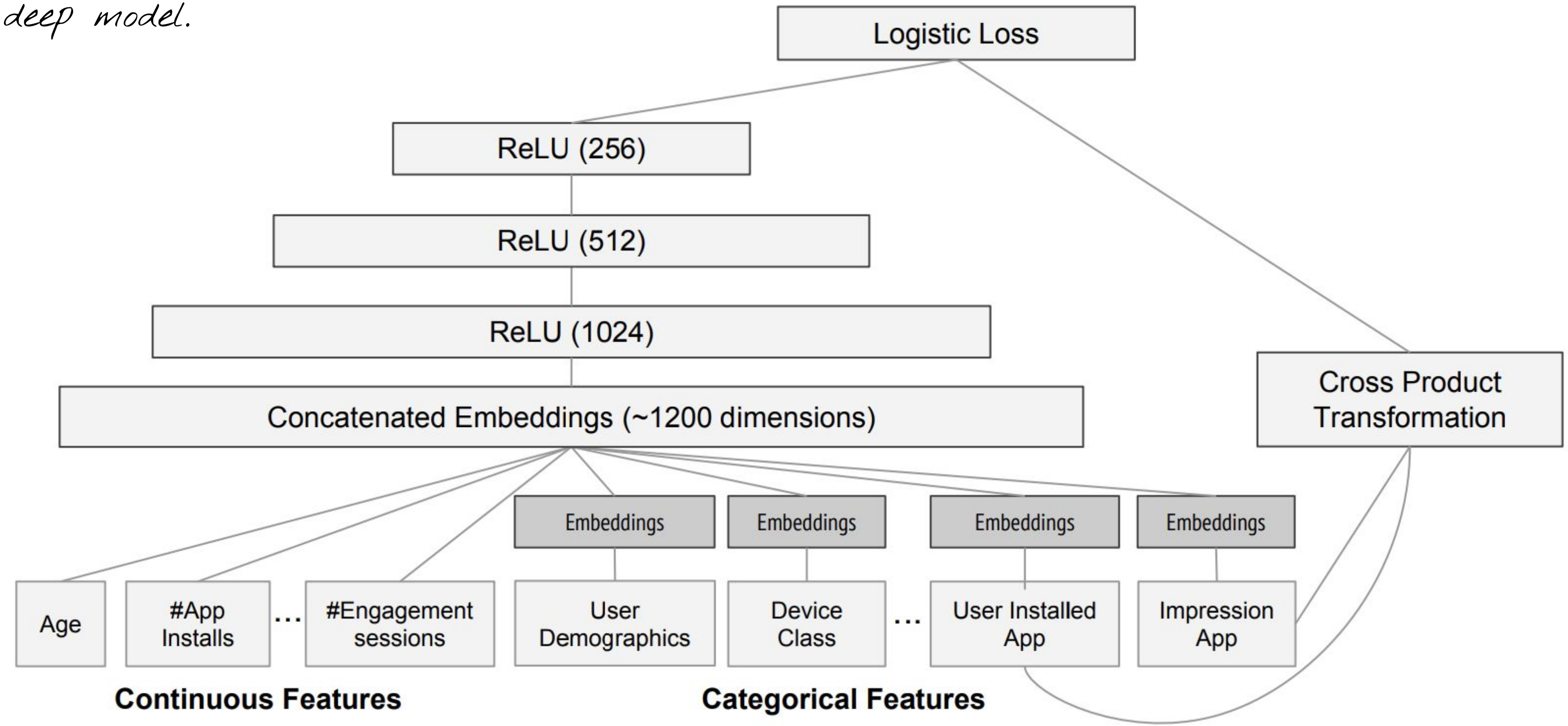
Embeddings

- Spare categorical features (e.g., “language=en”) are converted into embeddings.
- Dimensionality ranges between 10 and 100.

Result, model learns groups of users with similar behavior.



Also notice this is not a super deep model.



Trained on 500 x 10^9 examples

- yes you can work in projects for the class project
    - If working in a group, your project should be a bit more substantial than folks doing it alone.
- How to get started:
- build training set locally
    - Install open slide to read images (c library + python interface)
    - Slide pixel box along training images
    - At each step
        - Extract an image (300x300 region) and save it to disk
        - What's the label? Is it positive or negative? To find that out, check the annotations (see the starter code on Colab). Look at the same region in the annotation, if any pixels there are 1 (double check if that's right), you know there are cancerous cells and the image you just extracted is a positive example. Otherwise, it's negative.
    - Once you've extracted the training data, you now have a binary image classiification problem.
    - Either train a model locally, or upload your training data to the web and train in Colab.
    - Start small! No need to use multiple models or zoom levels -- the goal is to build a working proof of concept
- upload or train locally
- After you have a trained model, use it to make predictions

Reading

Reading

- Hidden Technical Debt in Machine Learning Systems
- What's your ML test score? A rubric for ML production systems
- Wide & Deep Learning for Recommender Systems