Managing ML Pipelines: Feature Stores and the Coming Wave of Embedding Ecosystems

VLDB 2021
Speakers

Laurel Orr  
Stanford

Atindriyo Sanyal  
Uber AI

Xiao Ling  
Apple

Megan Leszczynski  
Stanford

Karan Goel  
Stanford
Modern ML Pipelines

ML pipelines help engineers build and deploy models

**Standardization**  **Reproducibility**  **Easier to Maintain**

---

**H2O AutoML**
Scalable AutoML in H2O-3 Open Source

---

**Overton**

- Supervision Data
- Schema
  - Payloads + Tasks
  - (specified once)

**Actions**
- Add/augment slices
- Add labeling functions
- Add synthetic examples

**Overton**
- Combine Supervision
- Train & Tune Models
- Create Deployable Model

**Fine-grained quality reports**
- task 1  task 2
  - slice 1  
  - slice 2  
  - slice 3  
  - slice 4  

---

**LUDWIG**
v0.4
Engineer Workflow Today

STORE and MANAGE DATA

Feature / Data Engineering

BUILD MODEL and TRAIN

Model Testing / Deployment

DEPLOY and MONITOR

Weights & Biases

Monitor

kafka

TensorFlow

PyTorch

HDFS

cassandra
Engineer Workflow of Yesteryear (< 2017-8)

STORE and MANAGE DATA

“Pipeline Jungle*”

DEPLOY and MONITOR

The “Pipeline Jungle” Experience

Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

The “Pipeline Jungle” Experience

The challenges to deploying a model:

- One-off feature definitions
- Lack of reproducibility
- Inconsistent storage
- No standard evaluations and testing
- Difficult to detect and recover from errors
- ...

Popular Product Feature

"More than 4.5 stars"

Recommendator Models across Teams

Product Recommender Model

"More than 5K purchases"

Product Recommender Model

"80% positive sentiment reviews"

Product Recommender Model
Feature Store Solution

Systems to build, deploy, and monitor ML pipelines with special focus on *feature management*.
Enter Self-Supervision

Paradigm where models learn embedding representations of underlying training data *without* needed manually labels.
Self-Supervision Example: Transformers and MLM

Learn word embeddings by train a language model to predict a masked word in a given context.

Word embeddings encode contextual information.
Enter Self-Supervision

Paradigm where models learn embedding representations of the underlying training data *without* manual labels.

Embeddings are then used in downstream models.
Self-Supervision Example: Transformers

Learn word embeddings by train a language model to predict a masked word in a given context.

Word embeddings encode contextual information.
Recall Feature Store Solution

STORE and MANAGE DATA

Feature / Data Engineering → Feature Store

FEATURE MANAGEMENT

Model Training / Deployment

DEPLOY and MONITOR

Weights & Biases

Monitor
Embedding Ecosystems

Self-supervised embeddings, models that train them, and downstream systems that use them.

PART 2

STORE and MANAGE DATA

- Reduction in engineer effort

EMBEDDING MANAGEMENT

- Higher quality downstream systems

DEPLOY and MONITOR

- One embedding for multiple tasks

Weights & Biases

Monitor

Embedding Training

Model Training / Deployment

Embedding Store
FeatureStores
Engineer Workflow of Yesteryear (< 2017-8)

“Pipeline Jungle*”

Lack of Feature Management

- STORE and MANAGE DATA
- DEPLOY and MONITOR

"Pipeline Jungle*"

- 0 Feature Quality observability
  - 0 cataloging, monitoring of Features across the organization.
- Data Sources [Batch, Streaming, Realtime]
- Training SDKs
- Serving SDKs
- Pipelines

Uber
Lack of Feature Management

- Bespoke one-off systems
- Repeated work
- Hard to manage streaming inputs
- Difficult to maintain systems and correct errors
- Training Serving skew

“Pipeline Jungle*”

STORE and MANAGE DATA

DEPLOY and MONITOR

Data Sources [Batch, Streaming, Realtime]

Training SDKs

Serving SDKs

Trained Model

Deployed Model

Deploy and Monitor Served Models

pipelines
Lack of Feature Management

- **Days/Weeks to make data ML ready**
  - Materializing Features from various data sources.
  - Duplicating code while materializing in training & serving
  - No guarantees of training-serving parity

- **Near 0 monitoring of Features**
  - No Feature health metrics out of the box (due to the various sources problem)
  - No online-offline parity monitoring, leading to models performing poorly
  - No feature drift monitoring
  - No idea about Feature impact on a model

- **High latency, unreliable Feature serving in production models at scale**
  - Poor Model latencies leading to bad user experience.
  - No dedicated dynamic resource allocation for feature engineering
  - Multiple RPC calls at high throughputs to fetch features dramatically increasing latencies
Feature Stores

STORE and MANAGE DATA

FEATURE STORE

Quick Feature Authoring
  - Feature Reuse
  - Unified Feature Warehouse Management
  - Feature Retrieval at Scale
  - Model-level Data Quality Alerting
  - Feature Quality Monitoring

DEPLOY and MONITOR

Trained Model
  - Store and Manage Trained Models

Deployed Model
  - Deploy and Monitor Served Models
Use case - ETA of an Uber EATS Order

Key ML Features

- How large is the order? \(\text{order\_size}\)
- How busy is the restaurant? \(\text{n\_meal}\)
- How quick is the restaurant? \(\text{meal\_preptime}\)
- How busy is the traffic? \(\text{n\_busy}\)
Palette Feature Store: Workflow

- Search for features
  - by feature_name
  - by entity (e.g. eater_features)
  - by type (e.g. categorical_features)
  - by models (e.g. features used in eta_prediction_model)
  - or any combination ...
Palette Feature Store: Workflow

- Onboard Features and Author Pipelines
  - Metadata driven onboarding process
  - Feature Pipelines automatically created
  - Immediately available for consumption during Training & Inferencing
Palette Feature Store: Workflow

Onboard Features → Author Feature Pipelines → Train Your Model → Deploy Your Model

```python
dsl1 = DSLEstimator(lambdas={
    "region_id": regionId("@palette:restaurant:realtime_feature:lat:r_id", "@palette:restaurant:realtime_feature:lat:r_id")
})

dsl2 = DSLEstimator(lambdas={
    "prep_time": nFill(nVal("@palette:restaurant:batch_features:prep_time:r_id"),
    "n_meal": nFill(nVal("@palette:restaurant:realtime_features:n_nean:r_id"),
    "n_order": nFill(nVal("@basis:n_order"),
    "n_busy": nFill(nVal("@palette:restaurant:service_feature:n_busy:region_id"))
})

ml_pipeline = MLPipeline(dsl1, dsl2)
model = ml_pipeline.fit(basis_dataframe)
michelangelo_api.deploy(model)
```
Palette Feature Store: Workflow

- Monitor Feature Metrics
  - Training-Serving **Skew**
  - Feature **Drift**
  - Feature **Importance<>Drift** correlation
  - Feature Quality (**Freshness, Consistency, Null Rate**)
Feature Store (Palette) Lifecycle

**Feature Preparation**
Batch & Streaming ETLs

**Feature Storage**
- Historical & Near Real-Time
- Curated & Crowd-sourced
- Metadata
- Scalable offline access
- Scalable online access
- Online/Offline data parity

**Feature Discovery**
Sharing across Models
Automatic feature selection

**Feature Monitoring**
Data Quality reporting

**Feature Transforms**
Model specific transforms
Feature Stores in an End to End ML Platform
Feature Stores in an End to End ML Platform
Palette Feature Store Organization

Organized as **entity : feature_group : feature : join_key**

*e.g. restaurant : order_history : meal_preptime : restaurant_uuid*

- **Feature Store Abstractions:**
  - **Entity**: A Top Level Business Unit (e.g. *eater, courier, restaurant*)
  - **Feature Group**: Group of Features commonly used together (e.g. *order_history*)
  - **Feature**: The Feature (e.g. *meal_preptime, n_meal, sum_orders_1week*)
  - **Entity Key**: The UUIDs of the entities (e.g. *eater_uuid, restaurant_uuid*)

- **Bring** your join keys or UUIDs
- **Join** together cross-entity Feature sets with minimal code
- **Train** on historical Feature values
- **Serve** the latest, most accurate values of Features at Low Latency
- Backed by a dual datastore system (training & serving)
- Get Training Serving parity out of the box
Feature Types in Palette (Michelangelo)

- **Batch Features**: Features calibrated on **historical data**
  Generated via offline **batch jobs**
  **Auto dispersed** for model inferencing
  E.g. `meal_preptime` (average prep time of historical orders)

- **Near Real Time Features**: Features calibrated on **streaming data**
  Generated via **near real-time streaming jobs** (Flink, AthenaX)
  Auto dispersed for model training
  E.g. `n_meal` (how busy is the restaurant)

- **RPC Features**: Features retrieved via 3P APIs
  Features calibrated on 3P API calls
  Calculated at run time and served to models directly
  Auto dispersed for model training
  E.g. `location_geohash` (current geohash location of the courier)
Computing Batch Features

- Computed using **Historical Data**
- Not time sensitive
- Ingested from Hive Queries or Spark Jobs
- Aggregates over days/weeks
- E.g. `meal_preptime`
Computing Near Real Time Features

- Signals generated **seconds ago**
- Write Flink SQL to perform real-time aggregations
- Materialize to the online store
- Auto ETL and Backfill to the offline store
- E.g. `n_busy` (How busy is the restaurant)
  - Kafka event streams
  - Perform **Real-Time aggregations**
Computing RPC Features

- Signals generated **in real-time**
- Make RPC calls to Fetch Features behind the scenes
- Auto ETL and Backfill to the offline store
- E.g. lat/long:
  - Fetched via HTTP calls
Feature Extraction & Transformation

- Michelangelo Transformer
  - `transform()` and `scoreInstance()`
  - ML Readable / Writable
  - Extension of the Spark Transformer Framework
  - Parity across Spark and Spark-less environments
    - UDFs / DSLs
    - In-house unit testing framework for parity

- Feature Store APIs as Transformers
  - Feature Engineering as an integral part of the ML Pipeline
Michelangelo Feature Store APIs as Spark Transformers

```python
palette_tx1 = PaletteTransformer( {
    "nMeal": "@palette:restaurant:realtime_feature:nMeal:r_id",
    "prep_time": "@palette:restaurant:batch_feature:prep_time:r_id",
    "lat": "@palette:restaurant:realtime_feature:lat:r_id",
    "long": "@palette:restaurant:realtime_feature:long:r_id",
})
```

- Instantiate Palette Transformer with Feature expressions
- Create a pipeline with one or more stages of estimators and transformers
- `model = pipeline.fit()`
- Evaluate your model via `transform()`
- Score your model via `scoreInstance()`
## DSLs: Feature Manipulation / Imputation

- Write expressions to define Transformations
- Pre-compiled Scala code execution at runtime
- Example Michelangelo code:

```scala
dsl_est1 = DSLEstimator(lambdas=
   
   "region_id": regionId("restaurant:fg:lat:r_id", "restaurant:fg:long:r_id")
)

dsl_est2 = DSLEstimator(lambdas=

   "prep_time": nFill(nVal("restaurant:batch_fg:prep_time:r_id"), -1),
   "n_meal": nFill(nVal("restaurant:realtime_fg:n_meal:r_id"), -1),
   "order_size": nVal("basis:order_size"),
   "n_busy": nFill(nVal("restaurant:service_fg:n_busy:region_id"), -1)

)
```

*fg: feature_group*
Uber EATS Transformation Example

Computation Order

- n_meal:restaurant_id -> n_meal
- meal_preptime:restaurant_id -> meal_preptime -> DSL
- busy_scale: restaurant_id -> lat, long -> regionId(lat, long) -> busy_scale

Palette Transformer
  ld -> n_meal
  ld -> meal_preptime
  ld -> lat, long

DSL Transformer
  Lat, long -> region_id

Palette Transformer
  region_id -> n_busy

DSL Transformer
  impute(n_meal)
  impute(meal_preptime)

Training: transform()
Serving: score_instance()
Feature Store Results & Takeaways

- **Democratized Usage**: 20K+ Features used across 8K+ production models
- **Model development times** reduced from days to hours
- **Multi Modality Support**: Batch, Realtime and RPC Features with online and offline parity
- **Offline scalability**: Joins across billions of rows
- **Online serving latency**: Parallel IO, fast storage with caching
- **Feature Transformers**: Setup chains of transformations at training/serving time
Embedding Ecosystems
Managing ML Pipelines: Feature Stores and the Coming Wave of Embedding Ecosystems

Xiao Ling | VLDB Tutorial 2021
Recap: Self-Supervision Embeddings

Used in many different downstream systems

Downstream systems require less supervised data and provide a quality lift compared to hand-tuned predecessors.
Recap: Embedding Ecosystems

New age of feature store systems manage pretrained embeddings downstream systems use them as inputs.
Grounding Use Case: Named Entity Disambiguation

Map “strings to things” in a knowledge base.

Key part of assistant, search, and information extraction
The Long Tail of Entities

HEAD
Washington, DC
Q61
>1k examples in training

TORSO
Chevrolet, Corvette
Q56166
(10, 1000)

TAIL
Reddish Potato Beetle
Q14934552
[1, 10]

UNSEEN
Sauce! by XXXTENTACION
Q???
0 example

Popular
IR and BERT/IR work great for the HEAD.

Rare
The majority of entities are rare!
13% entities have Wikipedia page.
< 1% of songs in Wikidata!
#1 Tail Scalability Challenge

Large number of patterns needed to resolve the tail, making it difficult to scale a system that can learn the patterns.

90 million entities in Wikidata -> 90*100 million examples for 60 F1

Subtle reasoning clues are needed for the tail! (+40 F1 points by encoding these reasoning patterns)

Bootleg: Chasing the Tail with Self-Supervised Named Entity Disambiguation. Orr et al, CIDR 2021
Entity Embeddings in Downstream Applications

Experiment on the entity linking task in an existing Q&A system

- With and without Bootleg-learned entity embeddings
- The entity embeddings significantly improve F1 by a relative 8%
  Also, a relative 8% improvement on tail entities!
#2 Memory Usage

Embeddings linearly grow per number of entities

- 128d float32 x 5M ~ 2.4 Gb (English Wikipedia)
- 128d float32 x 96M ~ 46.08 Gb (Wikidata)

It requires larger and larger servers over time 💲 DBHelper

- More computation affects service latency 💷licting

Hard to fit on device! 📦
Memory Usage
Memory can be saved w/o a big quality sacrifice

Only keep the top k entity embeddings (i.e., compression ratio 100 - k)
- Uses a random UNK entity embedding
- Less memory-heavy signals remain

F1 only drops by 0.8 overall
- Memory drops from 5.2GB to 0.3GB!
#3 Embeddings in i18n languages

Embeddings work on other languages

Challenges

- Lack of equally abundant resources in English
- Memory usage increases the size of embeddings by the num of languages
Multilingual Entity Embeddings

Botha et al. 2020 proposed to train multilingual entity embeddings

- Memory usage doesn’t grow with the number of languages
- Entity embeddings trained from resources across languages
- Enabled by a multilingual language model

Entity Linking in 100 Languages. Botha et al., EMNLP 2020
#4 Embedding Stability

Entity embeddings are self-supervised from Wikipedia

- 20k new articles / month

Updating the model is hard

- Retraining entity embeddings takes hours, even days
  Also, need to retrain *each* downstream application!

- Previous correct prediction might change!
#5 Model Evaluation and Monitoring

Are the embeddings
- sensitive to questions?
- vulnerable to attacks?
- biased to entities popular in one country?, Etc...

Is the downstream application affected by updated embeddings?
- What about 10s or 100s downstream apps?
- How to enable safe regular model updates?

Summary of Challenges

#1 Long-tail of entities
#2 Memory usage
#3 Multi-lingual embeddings
#4 Embedding stability
#5 Model monitoring

*Bold* will be discussed in the following sections
Self-Supervised Training Data: The Challenge of the Long Tail
Grounding Use Case: Named Entity Disambiguation

Map “strings to things” in a knowledge base.

How tall is Lincoln?

Key part of assistant, search, and information extraction
Tail Challenge

Subtle reasoning clues are needed for the tail!
(+40 F1 points by encoding these reasoning patterns)

Impossible to scale the data to memorize all patterns needed for rare entities
Bootleg: Tackles the Tail with Structural Knowledge

Key Idea: reasoning over type and relationship signals can resolve unseen entities.

Disambiguation Inputs and Outputs

**Input:** Sentence

Where is **Lincoln** in **Logan County**?

**Output:** Entities

- Lincoln, IL
- Logan County, IL

**Extract Candidates**

- Lincoln, IL
- Lincoln, NE
- Abraham Lincoln
- Logan County, IL
- Logan County, OK
- Logan County, OH

**Entity Profiles**

- Logan County, IL
  - id: "Q292973" , name: “Logan County, IL”
  - types: [“county”, “geographic_loc”],
  - relations: ["<capital-of", "Q457134"],
  - "<named-after", "Q169067"]

**Entity Payload**

- entity payload

**Disambiguate**
Reasoning over Relationships

David and Victoria Mitchell added spice to their marriage

Victoria Mitchell (poker player, writer)

David Mitchell

Victoria Mitchell (runner)

Love Story by Taylor Swift

Love Story by Andy Williams

Love Story by Williams

Spouses

David Mitchell

Love Story is the second-best policy and other suits to lose by
Reasoning over Types

How tall is Lincoln?

What is the cheapest Lincoln?

How many people are in Lincoln?

People have heights, not places or brands

Brands have prices, not places or people

Places have populations, not people or brands
Bootleg: Tackles the Tail with Structural Knowledge

**Key Idea:** reasoning over *type* and *relationship* signals can resolve unseen entities.

**Implementation:** use *embeddings* to teach a model to reason over types and relationships.

Disambiguation Inputs and Outputs

**Input**: Sentence

Where is **Lincoln** in **Logan County**?

**Output**: Entities

Lincoln, IL  Logan County, IL

Disambiguate

Entity Profiles

Extract Candidates

- Lincoln, IL
- Lincoln, NE
- Abraham Lincoln
- Logan County, IL
- Logan County, OK
- Logan County, OH

Entity Payload

```json

id: "Q292973", name: "Logan County, IL", types: ["county", "geographic_loc"], relations: ["<capital-of", "Q457134"], <named-after>, "Q169067"

```

**entity payload**
Using Embeddings to Encode Signals

For each candidate, we use the entity profile to extract (learned) embeddings.
For each candidate, we use the entity profile to extract (learned) embeddings.
Using Embeddings to Encode Signals

Relation Embedding

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>child</td>
<td></td>
</tr>
<tr>
<td>capitol-of</td>
<td></td>
</tr>
<tr>
<td>founder</td>
<td></td>
</tr>
<tr>
<td>named-after</td>
<td>red</td>
</tr>
<tr>
<td>borders</td>
<td></td>
</tr>
<tr>
<td>league</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Logan County, IL

{ id: "Q292973", name: "Logan County, IL", types: ["county", "geographic_loc"], relations: [<"capital-of", "Q457134">, <"named-after", "Q169067"] }

entity

relation

Logan County, IL
Using Embeddings to Encode Signals

The entity payload has embeddings mapping for each structural resource.
Where is Lincoln in Logan County?

Simplest architecture that supports reasoning over types and relations.
Bootleg: Tail Performance

*On the head, BERT-based baseline performs ~ 5 F1 points of Bootleg.*
*On the tail, Bootleg outperforms baseline by > 40 F1 points!*

<table>
<thead>
<tr>
<th>Evaluation Set</th>
<th>BERT NED Baseline</th>
<th>Bootleg</th>
<th># Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>85.9</td>
<td>91.3</td>
<td>4,066K</td>
</tr>
<tr>
<td>Torso Entities</td>
<td>79.3</td>
<td>87.3</td>
<td>1,912K</td>
</tr>
<tr>
<td>Tail Entities</td>
<td>27.8</td>
<td>69.0</td>
<td>163K</td>
</tr>
<tr>
<td><strong>Unseen Entities</strong></td>
<td><strong>18.5</strong></td>
<td><strong>68.5</strong></td>
<td><strong>10K</strong></td>
</tr>
</tbody>
</table>

Performance results on Wikipedia dataset.
Bootleg: Industrial Performance

Included Bootleg embeddings into an Overton production task answering millions of users’ factoid queries. We report relative lift.

<table>
<thead>
<tr>
<th>Evaluation Set</th>
<th>English</th>
<th>Spanish</th>
<th>French</th>
<th>German</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Entities</td>
<td>1.08</td>
<td>1.03</td>
<td>1.02</td>
<td>1.00</td>
</tr>
<tr>
<td>Tail Entities</td>
<td>1.08</td>
<td>1.17</td>
<td>1.05</td>
<td>1.03</td>
</tr>
</tbody>
</table>
Using Bootleg Downstream: SoTA on the TACRED Benchmark

**Goal:** extract the relationship between a subject and object pair.

**Gold relation: per:employee_of**

Mays worked with several other companies aside from Media Enterprises in his career.

**Micro-Avg. F1 on TACRED Revised test dataset:**

<table>
<thead>
<tr>
<th>Model</th>
<th>Test F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpanBERT</td>
<td>78.0</td>
</tr>
<tr>
<td>KnowBERT</td>
<td>79.3</td>
</tr>
<tr>
<td>Bootleg+SpanBERT</td>
<td>80.2 (SoTA)</td>
</tr>
</tbody>
</table>

Zhang et al., 2017 and Hennig et al., 2020.

**Bootleg resolves errors made in by the prior SoTA model.**

**Gold relation: per:cause_of_death**

Vincent Astor, like Marshall, died unexpectedly of a heart attack in 1959 ... 

WikiData relation: ['cause of death']

<table>
<thead>
<tr>
<th>SpanBERT</th>
<th>no_relation</th>
<th>✗</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bootleg</td>
<td>per:cause_of_death</td>
<td>✔</td>
</tr>
</tbody>
</table>

Leveraging type and relation information downstream

**Gold relation: org:alternate_names**

The International Water Management Institute or IWMI study said both ...

WikiData same entity

<table>
<thead>
<tr>
<th>SpanBERT</th>
<th>no_relation</th>
<th>✗</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bootleg</td>
<td>org:alternate_names</td>
<td>✔</td>
</tr>
</tbody>
</table>

Understand that sub-strings relate to the same entity
Self-Supervised Data Take Away

Self-supervised data does not well represent tail distributions -> embeddings may not be high quality for rare entities

Solution: merged unstructured data with structured knowledge that can generalize to the tail.
Embedding Management: Stability
Problem Setting: Embedding Store

New embeddings require downstream tasks to be retrained!
Why do embeddings need to be retrained?

- Learn new entities
- Leverage new activity
- Understand new words

Model freshness is necessary for user satisfaction in many products.

[1] https://about.instagram.com/blog/engineering/designing-a-constrained-exploration-system
Google retrained their app store Google Play models every day, and Facebook retrained search models every hour.

But model training can be unstable...

Prediction churn

Challenges of Instability

Debugging

Consistent user-experience

Model dependencies

Research reliability
Problem Setting: Embedding Store

How does the embedding instability propagate to downstream tasks?
Outline

• Downstream instability definition

• Stability-space tradeoff

• Measuring embedding quality with distance measures
Definition: Downstream Instability

Downstream instability = % prediction disagreement between models trained on a pair of embeddings
Embedding Hyperparameters that Impact Storage

**Dimension**

- # features / word

**Precision**

- # bits / feature

**Embedding Size**

= Uniform Quantization

<table>
<thead>
<tr>
<th>Uniform Quantization</th>
<th>32-bit</th>
<th>1-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval: [-0.1, 0.1]</td>
<td>0.04</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>-0.03</td>
<td>-0.1</td>
</tr>
<tr>
<td></td>
<td>-0.08</td>
<td>-0.1</td>
</tr>
</tbody>
</table>

Stability-Space Tradeoff for Word Embeddings

**Sentiment Analysis**

**NER**

**SST-2**

**CoNLL-2003**

Embedding Size

Downstream Instability
Goal: Embedding Distance Measure for Instability

The measure should relate the **distance between the embeddings** to the **downstream instability**.
Embedding Distance Measures

- k-NN measure [1,2,3]
- Semantic displacement (SD) [4]
- PIP loss [5]
- Eigenspace overlap (EO) [6]
- Eigenspace instability measure (EIS) [7]

Using Embedding Distance Measures to Minimize Downstream Instability

**k-NN measure** and **theoretically grounded EIS measure** outperform other measures for selecting embeddings to **minimize downstream instability**.
Stability Takeaways

- Defined downstream instability with respect to embeddings
- Stability-space tradeoff (precision, dimension)

- Measuring embedding quality with embedding distance measures
  - EIS and k-NN measures select embeddings with lower downstream instability
Closing the Loop of Model Development: Monitoring and Patching
Monitoring and Patching

**Embeddings need to be updated:** distribution shift, changing needs

**Monitor (when to update)**
Evaluate and track distribution shift

**Patch (how to update)**
Fix bugs and improve performance
Remember: Embedding Store

Important: update embeddings not downstream models → changes propagate down to models!
Crucial Bottleneck: Evaluation

Can’t monitor and patch embeddings without evaluation

**Downstream Task**
- Named Entity Linking (NEL)
- Question Answering
- Sentiment Analysis
- Relation Extraction

Evaluate model errors

Monitor

Patch

Embedding Store
Crucial Bottleneck: Evaluation

Can’t monitor and patch embeddings without evaluation

Many Evaluation Strategies
- Critical data slices
- Bias / fairness concerns
- Sensitivity to perturbations
- Invariance to transformations
- and more!

Shift towards **fine-grained evaluation** with new tools (e.g. Robustness Gym, Dynabench)
**Tool: Robustness Gym**

Consolidates different evaluation strategies (slices, transformations) and metrics

<table>
<thead>
<tr>
<th>Evaluation Strategies</th>
<th>Metrics</th>
<th>Accuracy</th>
<th>F1</th>
<th>Class Dist</th>
<th>Pred Dist</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Constituency Tree Overlap (McCoy, 2019)</td>
<td></td>
<td>90.2</td>
<td>89.7</td>
<td>20</td>
<td>39</td>
<td>41</td>
</tr>
<tr>
<td>High Constituency Tree Overlap (McCoy, 2019)</td>
<td></td>
<td>93.2</td>
<td>92.2</td>
<td>53</td>
<td>24</td>
<td>23</td>
</tr>
<tr>
<td>Negation @ hypothesis (Naik, 2018)</td>
<td></td>
<td>90.8</td>
<td>86.0</td>
<td>22</td>
<td>17</td>
<td>61</td>
</tr>
<tr>
<td>Negation @ premise (Naik, 2018)</td>
<td></td>
<td>79.5</td>
<td>79.5</td>
<td>31</td>
<td>38</td>
<td>31</td>
</tr>
<tr>
<td>Possessive Preposition @ hypothesis (Chen, 2020)</td>
<td></td>
<td>90.9</td>
<td>90.9</td>
<td>39</td>
<td>34</td>
<td>27</td>
</tr>
<tr>
<td>Quantifier @ hypothesis (Chen, 2020)</td>
<td></td>
<td>88.2</td>
<td>88.3</td>
<td>38</td>
<td>34</td>
<td>28</td>
</tr>
<tr>
<td>Temporal Preposition @ hypothesis (Chen, 2020)</td>
<td></td>
<td>87.7</td>
<td>86.0</td>
<td>13</td>
<td>61</td>
<td>25</td>
</tr>
<tr>
<td>Low Lexical Overlap (McCoy, 2019)</td>
<td></td>
<td>90.5</td>
<td>89.6</td>
<td>20</td>
<td>33</td>
<td>47</td>
</tr>
<tr>
<td>High Lexical Overlap (McCoy, 2019)</td>
<td></td>
<td>92.7</td>
<td>91.9</td>
<td>52</td>
<td>29</td>
<td>19</td>
</tr>
<tr>
<td>BAE (Garg, 2019)</td>
<td></td>
<td>80.3</td>
<td>78.4</td>
<td>13</td>
<td>58</td>
<td>29</td>
</tr>
<tr>
<td>Easy Data Augmentation (Wei, 2019)</td>
<td></td>
<td>82.3</td>
<td>82.2</td>
<td>34</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>Keyboard Character Errors (Ma, 2019)</td>
<td></td>
<td>65.8</td>
<td>65.4</td>
<td>34</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>Synonym Substitution (Ma, 2019)</td>
<td></td>
<td>75.4</td>
<td>75.1</td>
<td>34</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>SNLI (Bowman, 2015)</td>
<td></td>
<td>90.9</td>
<td>90.9</td>
<td>33</td>
<td>33</td>
<td>34</td>
</tr>
</tbody>
</table>

**Example:** BERT embeddings are sensitive to character errors
Tool: Robustness Gym

Consolidates many different evaluation types (subpopulations, transformations) and metrics

Example: BERT embeddings are sensitive to character errors

Emerging questions
- Discovering important failure modes automatically
- Understanding knowledge captured by an embedding

Metrics

<table>
<thead>
<tr>
<th>Evaluation</th>
<th>Accuracy</th>
<th>F1</th>
<th>Class Dist</th>
<th>Pred Dist</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Constituency Tree Overlap</td>
<td>90.2</td>
<td>89.7</td>
<td>20</td>
<td>39</td>
<td>41</td>
</tr>
<tr>
<td>High Constituency Tree Overlap</td>
<td>93.2</td>
<td>92.2</td>
<td>53</td>
<td>24</td>
<td>23</td>
</tr>
<tr>
<td>Negation @ hypothesis (Naik, 2018)</td>
<td>90.8</td>
<td>86.0</td>
<td>22</td>
<td>17</td>
<td>61</td>
</tr>
<tr>
<td>Negation @ premise (Naik, 2018)</td>
<td>79.5</td>
<td>79.5</td>
<td>31</td>
<td>38</td>
<td>31</td>
</tr>
<tr>
<td>Possessive Pronoun @ hypothesis (Chen, 2020)</td>
<td>90.0</td>
<td>90.9</td>
<td>34</td>
<td>34</td>
<td>27</td>
</tr>
<tr>
<td>Subject @ hypothesis (Chen, 2020)</td>
<td>88.2</td>
<td>86.3</td>
<td>38</td>
<td>34</td>
<td>28</td>
</tr>
<tr>
<td>High Lexical Overlap (McCoy, 2019)</td>
<td>89.7</td>
<td>91.5</td>
<td>13</td>
<td>61</td>
<td>25</td>
</tr>
<tr>
<td>BAE (Garg, 2019)</td>
<td>80.5</td>
<td>78.4</td>
<td>33</td>
<td>33</td>
<td>47</td>
</tr>
<tr>
<td>Easy Data Augmentation (Wei, 2019)</td>
<td>82.3</td>
<td>82.2</td>
<td>34</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>Keyboard Character Errors (Ma, 2019)</td>
<td>65.8</td>
<td>65.4</td>
<td>34</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>Synonym Substitution (Ma, 2019)</td>
<td>75.4</td>
<td>75.1</td>
<td>34</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>SNLI (Bowman, 2015)</td>
<td>90.9</td>
<td>90.9</td>
<td>34</td>
<td>33</td>
<td>33</td>
</tr>
</tbody>
</table>
Important Evaluation Strategy: Slice-Based Evaluation

A type of fine-grained evaluation

→ Measure fine-grained performance on critical subpopulations (filtering)

**Example:**

short passages (< 50 words) in a text dataset
Most Named Entity Linking systems are poor on rare entities
Evaluation over Time: **Monitoring**

Continually evaluate as the world changes

- **Train set**
- **Validation set**

Source data distribution \((x, y) \sim \mathcal{P}_s\)

Labeled
Evaluation over Time: **Monitoring**

Continually evaluate as the world changes

```
Model
```

```
Train set  Validation set
```

```
Source data distribution $(x, y) \sim \mathcal{P}_s$
Labeled
```

```
Target data distribution $(x, y) \sim \mathcal{P}_t$
```

Distribution shift
Evaluation over Time: **Monitoring**

Continually evaluate as the world changes

Need to monitor model performance on unlabeled data

Train set | Validation set
---|---
Source data distribution \((x, y) \sim \mathcal{P}_s\) | Labeled

Model

Target data distribution \((x, y) \sim \mathcal{P}_t\) | Unlabeled

Learn | Evaluate

Deploy

Distribution shift

Evaluate?
Approach: Importance Weighting

**Estimate metrics on incoming data**

Upweight examples in our dev set more likely to be seen in the future

**Theoretical Foundations**

Density ratio estimation (Sugiyama, 2012)

**Recent work:** accurately estimate performance with slice-based evaluation + importance weighting

Mandoline: Slice-based reweighting framework

**Slice**: user-defined grouping of data $y(x) \in \{-1, 1\}$
Mandoline: Slice-based reweighting framework

**Slice**: user-defined grouping of data $y(x) \in \{-1, 1\}$

- **negation**: contains `not`, `n't`
- **male pronoun**: contains `he`, `him`
- **strong sentiment**: contains `love`, `adore`

### (Source) Labeled Validation Set

<table>
<thead>
<tr>
<th>Sentence</th>
<th>Slices</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>I love eating ice-cream.</td>
<td>-1</td>
<td>✔</td>
</tr>
<tr>
<td>He loved walking on the beach.</td>
<td>-1</td>
<td>✔</td>
</tr>
<tr>
<td>He didn’t like drinking coffee.</td>
<td>1</td>
<td>✗</td>
</tr>
</tbody>
</table>

Source Accuracy: 91%

Mandoline: Slice-based reweighting framework

**Slice**: user-defined grouping of data $y(x) \in \{-1, 1\}$

<table>
<thead>
<tr>
<th>(Source) Labeled Validation Set</th>
<th>Slices</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>I love eating ice-cream.</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>He loved walking on the beach.</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>He didn’t like drinking coffee.</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(Target) Unlabeled Test Set</th>
<th>Slices</th>
</tr>
</thead>
<tbody>
<tr>
<td>He does not love eating scones.</td>
<td>1 1 1</td>
</tr>
<tr>
<td>He loves taking risks.</td>
<td>-1 1 1</td>
</tr>
<tr>
<td>She likes drinking coffee.</td>
<td>-1 -1 -1</td>
</tr>
</tbody>
</table>

Source Accuracy: 91%

Mandoline: Slice-based reweighting framework

**Slice**: user-defined grouping of data \( y(x) \in \{-1, 1\} \)

- **negation** contains *not*, *n’t*
- **male pronoun** contains *he*, *him*
- **strong sentiment** contains *love*, *adore*

### (Source) Labeled Validation Set

<table>
<thead>
<tr>
<th>Sentence</th>
<th>Slices</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>I love eating ice-cream.</td>
<td>-1</td>
<td>✓</td>
</tr>
<tr>
<td>He loved walking on the beach.</td>
<td>-1</td>
<td>✓</td>
</tr>
<tr>
<td>He didn’t like drinking coffee.</td>
<td>1</td>
<td>✗</td>
</tr>
</tbody>
</table>

**Source Accuracy**: 91%

### (Target) Unlabeled Test Set

<table>
<thead>
<tr>
<th>Sentence</th>
<th>Slices</th>
</tr>
</thead>
<tbody>
<tr>
<td>He does not love eating scones.</td>
<td>1</td>
</tr>
<tr>
<td>He loves taking risks.</td>
<td>-1</td>
</tr>
<tr>
<td>She likes drinking coffee.</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Target Accuracy**: 84%

Mandoline: Slice-based reweighting framework

**Slice**: user-defined grouping of data $y(x) \in \{-1, 1\}$

**Takeaways**
- Monitor any model: importance weighting
- Add domain knowledge (slices) to improve monitoring

Embedding Model **Patching**

Once errors are identified, need to retrain or update embeddings

**Many Approaches**

<table>
<thead>
<tr>
<th>Data-Centric</th>
<th>Model-Centric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Augmentation</td>
<td>Training Paradigm</td>
</tr>
<tr>
<td>Data Collection</td>
<td>Training Algorithm</td>
</tr>
<tr>
<td>Active Sampling</td>
<td>Features</td>
</tr>
<tr>
<td>Weak Labeling</td>
<td>Architecture</td>
</tr>
<tr>
<td>Data Preprocessing</td>
<td></td>
</tr>
</tbody>
</table>

Named Entity Linking

map “strings” to “things” in a knowledge base like Wikipedia

A correct NEL is required for the downstream system!
End to End Example: Named Entity Linking

**Repurposing** Bootleg NEL system to patch errors for sports QA

Sports QA: prefer if the model predicted the national sports team instead of the country!

End to End Example: Named Entity Linking

**Repurposing** Bootleg NEL system to patch errors for sports QA
End to End Example: Named Entity Linking

**Repurposing** Bootleg NEL system to patch errors for sports QA

<table>
<thead>
<tr>
<th>Subpop.</th>
<th>Gold Label</th>
<th>Pred. Label</th>
<th>Size (Off-The-Shelf → Patched)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Team</td>
<td>Country</td>
<td>151 → 106</td>
<td>3393 → 3447 (↑)</td>
</tr>
<tr>
<td>Team</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

25% absolute accuracy improvement in sports-related errors

Embedding Model **Patching**

Once errors are identified, need to retrain or update embeddings

### Many Approaches

<table>
<thead>
<tr>
<th>Data-Centric</th>
<th>Model-Centric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Paradigm</td>
<td>Training Algorithm</td>
</tr>
<tr>
<td>Training Algorithm</td>
<td>Features</td>
</tr>
<tr>
<td>Architecture</td>
<td></td>
</tr>
</tbody>
</table>

**New area of research!**
- Incremental and targeted embedding updates
- Backwards compatibility for updated embeddings e.g. stability
- Data-centric vs. model-centric updates
- Sample efficiency and efficacy of approaches
- Time-to-update and optimal cadence

Future Directions
Embeddings as First Class Citizens

What is the right system for embedding management in ML pipelines?

- Search
  What set of embeddings are best for a specific task?

- Provenance
  What data had the most “impact” on these embeddings?

- Quality
  What are the right metrics/probes for embedding quality?

- Embedding A

- Embedding B

  \[(x_1, y_1), \ldots, (x_n, y_n)\]
End-to-End Model Patching

How can we automate and provide guidance for embedding patching?

What are the current failure modes?
What data engineering strategy to use?
How do I update my models efficiently?

Data Augmentation
Data Collection
Active Sampling
Weak Labeling
Data Preprocessing

Embeddings $t$

Embeddings $t+1$
Interactive Machine Learning

How can we facilitate human interaction with model training and evaluation data?

- Construct Data
- Measure
- Monitor
- Maintain

How can we support the entire lifecycle?

How to integrate multiple modalities?

Systems to store and manage models and data?

https://github.com/robustness-gym/meerkat