Real-world ML use cases

Piero Molino

Agenda

• NLP models for Customer Support
• Model retraining strategies
• Gran Neural Networks for dish/restaurant recommendation
• Learning from recommender system deployment
• Lessons learned from real-world data collection
Customer Obsession Ticket Assistant

Improving Uber Customer Support with Natural Language Processing and Deep Learning

Piero Molino | AI Labs
Huaixiu Zheng | Applied Machine Learning
Yi-Chia Wang | Applied Machine Learning
Main Takeaways

COTA v1: classical NLP + ML models
  ○ Faster and more accurate customer care experience
  ○ Million $ of saving while retaining customer satisfaction

COTA v2: deep learning models
  ○ Experiments with various deep learning architectures
  ○ 20-30% performance boost compared to classical models
COTA Blog Post and followup, KDD paper

COTA: Improving Uber Customer Care with NLP & Machine Learning

By Huaixiu Zheng, Yi-Chia Wang, & Piero Molino

January 3, 2018
Motivation and Solution

Complexity of Customer support @Uber

COTA v1: Traditional ML / NLP Models

Multi-class Classification vs Ranking

COTA v2: Deep Learning Models

Deep learning architectures

COTA v1 vs COTA v2
Agenda

Motivation and Solution

Complexity of Customer support @Uber

COTA v1: Traditional ML / NLP Models

Multi-class Classification vs Ranking

COTA v2: Deep Learning Models

Deep learning architectures

COTA v1 vs COTA v2
What is the challenge?
As Uber grows, so does our volume of support tickets.

Millions of tickets from riders / drivers / eaters per week

Thousands of different types of issues users may encounter
Uber Support Platform

1. User
   - Select Flow Node
   - Write Message

2. Response
   - Write response using a Reply Template
   - Select Action

3. Contact Ticket
   - Lookup info & Policies

4. CSR
   - Select Contact Type
What is the challenge?

And it is not easy to solve a ticket

1000+ types in a hierarchy depth: 3~6

10+ actions (adjust fare, add appeasement, …)

1000+ reply templates
Motivation and Solution

Complexity of Customer support @Uber

COTA v1: Traditional ML / NLP Models

Multi-class Classification vs Ranking

COTA v2: Deep Learning Models

Deep learning architectures

COTA v1 vs COTA v2
COTA v1: Suggested Resolution

Machine learning models recommending the 3 most relevant solutions

SUGGESTED CONTACT TYPES
- Driver > Account > Unable to sign in or go online > Account inactive
- Driver > Account > Profile > Unsubscribe > SMS or Text
- Driver > Account > Vehicles > Edit vehicle class

Reorder actions in relevance
Surface top-3 most-relevant reply templates
COTA v1 Model Pipeline

**Features**
- User Information
- Trip Information
- Ticket Metadata
- Ticket Message

**Preprocessing**
- Tokenization
- Lowercasing
- Stopword removal
- Lemmatization

**Feature Engineering**
- TF-IDF
- LSA

**ML Algorithm**
- Multiclass Classification

**Predictions**
- Top 3 Contact Types
- Top 3 Reply Templates

**Multiclass classification**

Information about the contact type or reply, obtained from all tickets belonging to it

**For Each Class**
- Binary Classification

**Pointwise ranking**
From Classification to Ranking

Multi-class Classification

<table>
<thead>
<tr>
<th>Tickets Features</th>
<th>Label (CT1, CT2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1 features</td>
<td>CT1</td>
</tr>
<tr>
<td>t2 features</td>
<td>CT2</td>
</tr>
</tbody>
</table>

Pointwise Ranking

<table>
<thead>
<tr>
<th>Tickets Features</th>
<th>Type Features</th>
<th>Sim (t, CT)</th>
<th>Label (0, 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1 features</td>
<td>CT1 features</td>
<td>0.8</td>
<td>1</td>
</tr>
<tr>
<td>t1 features</td>
<td>CT2 features</td>
<td>0.1</td>
<td>0</td>
</tr>
<tr>
<td>t2 features</td>
<td>CT1 features</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td>t2 features</td>
<td>CT2 features</td>
<td>0.7</td>
<td>1</td>
</tr>
</tbody>
</table>

Ranking allows us to include **features of candidate types** and **similarity features** between a ticket and a candidate type.

Model: **Random Forest** with hyperparameters optimized through **grid search**.
Performance Comparison

6% absolute (10% relative) improvement

Hits@3: any of the top 3 suggestions is selected by CSRs
Motivation and Solution

Complexity of Customer support @Uber

COTA v1: Traditional ML / NLP Models
  Multi-class Classification vs Ranking

COTA v2: Deep Learning Models
  Deep learning architectures

COTA v1 vs COTA v2
COTA v2: Deep Learning Architecture

Generic architecture, reusable in many different applications. We are considering open-sourcing it!
Key contributors
Travis Addair
Yaroslav Dudin
Sai Sumanth Miryala
Jim Thompson
Avanika Narayan
Ivaylo Stefanov
John Wahba
Doug Blank
Patrick von Platen
Carlo Grisetti
Chris Van Pelt
Boris Dayma

Documentation
http://ludwig.ai

Repository
http://github.com/uber/ludwig

Blogpost
http://eng.uber.com/introducing-ludwig
http://eng.uber.com/ludwig-v0-2/
http://eng.uber.com/ludwig-v0-3/

White paper
COTA v2: Text Encoding Models

Char CNN

Char Seq

6 x 1D Conv

2 x FC

Vector

Word CNN

Word Seq

1D Conv width 2

Conv width 3

Conv width 4

Conv width 5

2 x FC

Vector

Char / Word RNN

Char / Word Seq

2 x RNN

2 x FC

2 x FC

Vector

Char / Word CNN RNN

Char / Word Seq

3 x Conv

2 x RNN

2 x FC

Vector
Which text encoder?
Hyperparameter search for contact type classification

<table>
<thead>
<tr>
<th>Model</th>
<th>Validation accuracy</th>
<th>Training time per epoch in minutes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CharCNNRNN opt</td>
<td>0.4805</td>
<td>35</td>
</tr>
<tr>
<td>WordCNN opt</td>
<td>0.4733</td>
<td>4</td>
</tr>
<tr>
<td>WordRNN opt</td>
<td>0.4713</td>
<td>17</td>
</tr>
<tr>
<td>WordCNNRNN opt</td>
<td>0.4615</td>
<td>12</td>
</tr>
<tr>
<td>CharCNN opt</td>
<td>0.4598</td>
<td>5</td>
</tr>
<tr>
<td>WordCNN</td>
<td>0.4733</td>
<td>4</td>
</tr>
<tr>
<td>WordRNN opt</td>
<td>0.4713</td>
<td>17</td>
</tr>
<tr>
<td>WordCNNRNN opt</td>
<td>0.4615</td>
<td>12</td>
</tr>
<tr>
<td>CharCNN opt</td>
<td>0.4598</td>
<td>5</td>
</tr>
</tbody>
</table>

WordCNN is the best compromise between performance and speed 20%+ over Random Forest used in COTA v1 and ~10x faster than CharCNNRNN
Sequence Model for Type Selection

Predict the sequence of nodes instead of leaf node

Example: **Driver** > **Trips** > **Pickup and drop-off issues** > **Cancellation Fee** > **Driver Cancelled**

Use a Recurrent Decoder to predict **sequences of nodes** in the contact type tree

Pick the last class before `<eos>` as prediction

Model makes **more reasonable mistakes**
Final Architecture
Multi-task sequential learning

Text features
e.g. message

Categorical features
e.g. flow node

Numerical features
e.g. trip fare

Binary features
e.g. is completed

Convolution layers
Embedding layer
Batch-norm layer

FC + Dropout layers

Recurrent Decoder
Softmax layer

TYPE
REPLY

Train

TYPE
REPLY
ground-truth

Test

TYPE
REPLY
predicted
Effect of Adding Dependencies Between Tasks

Adding the dependency from Type to Reply improves accuracy.

It also improves a lot the coherence between the two models, increasing combined accuracy consistently.

Combined accuracy computed requiring both Type and Reply model to be correct at the same time.
Outline

Motivation and Solution

Complexity of Customer support @Uber

COTA v1: Traditional ML / NLP Models

Multi-class Classification vs Ranking

COTA v2: Deep Learning Models

Deep learning architectures

COTA v1 vs COTA v2
COTA v2 is consistently more effective than COTA v1 on all metrics for both models.

The combined accuracy in particular shows an absolute $\sim +9\%$ (relative $\sim +20\%$).
COTA v1 vs. COTA v2 A/B Test
COTA v2 is 20-30% more accurate than COTA v1 in online A/B tests.

COTA v1 reduces handling time of ~8%, while COTA v2 provides an additional ~7% reduction, more than ~15% overall reduction.

Statistically significant customer satisfaction improvement.
Threshold on Type Model Confidence

[type model diagram]

[type wordCNN model diagram]
Threshold on Both Models’ Confidence
Coverage vs. Maximum Accuracy

- 95% accuracy → 10% coverage
- 90% accuracy → 20% coverage
Conclusions

Using NLP & ML COTA makes customer care experience **faster** and **more accurate** while saving Uber millions of $.

Moving from traditional to deep learning models, we observe a substantial **performance boost** (up to 30%).

Using intelligent suggestions we were able to **reduce ticket handling time** without impacting customer satisfaction.
Model degradation

**Distribution shift** in the real world

- Bugs get solved, probability of a issue type can decrease
- New products can be added (UberPool) so new issue types appear

**Older data becomes noise**

- We often talk about distribution shift in the test set, but the test set of a month ago is the training set now
Retraining Strategy

Dealing with distribution shift is an open research topic.

In practice in most cases the safest route is just retraining the model.

But...

- How often to retrain?
- What triggers retraining?
- With how much data?
Offline simulation: time-based split
Offline simulation: split in weeks

<table>
<thead>
<tr>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dataset</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
</tr>
<tr>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Test</td>
</tr>
<tr>
<td>Train</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
</tr>
</tbody>
</table>
Offline simulation

<table>
<thead>
<tr>
<th>Train</th>
<th>Train</th>
<th>Train</th>
<th>Train</th>
<th>Train</th>
<th>Train</th>
<th>Train</th>
<th>Train</th>
<th>Test</th>
<th>Test</th>
<th>Test</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
</tr>
<tr>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
</tr>
<tr>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
</tr>
<tr>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
</tr>
</tbody>
</table>
# Offline simulation

<table>
<thead>
<tr>
<th>Train</th>
<th>Train</th>
<th>Train</th>
<th>Train</th>
<th>Train</th>
<th>Train</th>
<th>Train</th>
<th>Train</th>
<th>Train</th>
<th>Test</th>
<th>Test</th>
<th>Test</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
</tr>
<tr>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
</tr>
<tr>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
</tr>
<tr>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Train</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
</tr>
</tbody>
</table>

...
Offline simulation

Time

Jan Feb Mar Apr

Dataset

Train Train Train Train Train Test Test
Train Train Train Train Train Test Test
Train Train Train Train Train Test Test
Train Train Train Train Train Test Test
Train Train Train Train Train Test Test
Train Train Train Train Train Test Test
Retraining Strategy

How often to retrain?

With how much data?
Online Retraining

What triggers retraining?

Used learnings from offline simulation

Retrained **when performance dropped** below performance on the test set at training original training time - 8% (relative)

Retrained with **1.5 months** of training data, as we learned from the offline simulation that more was detrimental to performance
COTA Team
Cross-functional collaboration

AI Labs
Applied Machine Learning
Customer Obsession
Michelangelo
Sensing and and Perception
Enhancing Recommendations on Uber Eats with Graph Convolutional Networks

Ankit Jain/Piero Molino

ankit.jain/piero@uber.com
Agenda

1. Graph Representation Learning
2. Dish Recommendation on Uber Eats
3. Graph Learning on Uber Eats
Graph data

Social networks

Linked Open Data

Biomedical networks

Information networks

Internet

Networks of neurons
Tasks on graphs

Node classification
   Predict a type of a given node

Link prediction
   Predict whether two nodes are linked

Community detection
   Identify densely linked clusters of nodes

Network similarity
   How similar are two (sub)networks
Learning framework

Define an encoder mapping from nodes to embeddings

Define a node similarity function based on the network structure

Optimize the parameters of the encoder so that:

\[ \text{similarity}(u, v) \approx z_v^\top z_u \]
Shallow encoding

Simplest encoding approach: encoder is just an embedding-lookup

Algorithms like Matrix Factorization, Node2Vec, Deepwalk fall in this category

\[ Z = \text{embedding matrix} \]

embedding vector for a specific node

One column per node

Embedding size
Shallow encoding limitations

$O(|V|)$ parameters are needed, every node has its own embedding vector.

Either not possible or very time consuming to generate embeddings for nodes not seen during training.

Does not incorporate node features.
Graph Neural Network

**Key Idea:** To obtain node representations, use a neural network to aggregate information from neighbors recursively by limited Breadth-First Search (BFS).
**Inductive capability**

In many real applications new nodes are often added to the graph.

Need to generate embeddings for new nodes without retraining.

Hard to do with shallow methods.

- **train with snapshot**
- **new node arrives**
- **generate embedding for new node**
Dish Recommendation on Uber Eats
Suggested Dishes

Recommended Dishes Carousel

Picked for You

- **Kofta Curry**
  - Vegetable dumplings cooked with herbs and spices in a creamy sauce.
  - $9.99
  - You've ordered this before

- **Roti**
  - Two pieces. Served with raita and pickles. Traditional whole wheat Indian bread.
  - $1.50
  - Vegan
Graph Learning in Uber Eats
Bipartite graph for dish recommendation

Users connected to dishes they have ordered in the last M days

Weights are frequency of orders

Graph properties

Graph is dynamic: new users and dishes are added every day

Each node has features, e.g. word2vec of dish names
Max Margin Loss

For dish recommendation we care about **ranking**, not actual similarity score

Max Margin Loss:

\[
L = \sum_{(u,v) \in E} \max(0, -z_u z_v + z_u z_n + \Delta)
\]

- **positive pair**
- **negative sample**
- **margin**
New loss with Low Rank Positives

\[ L = \sum_{(u,v) \in E} \alpha_n \max(0, -z_u z_v + z_u z_n + \Delta_n) + \alpha_l \max(0, -z_u z_v + z_u z_l + \Delta_l) \]

\[ \Delta_l < \Delta_n \]
**Weighted pool aggregation**

Aggregate neighborhood embeddings based on edge weight

\[
\text{AGG} = \sum_{u \in N(v)} w(u, v) Q h_u^{k-1}
\]

\( Q \) denotes a fully connected layer
Offline evaluation

Trained the downstream Personalized Ranking Model using graph node embeddings

~12% improvement in test AUC over previous production model

<table>
<thead>
<tr>
<th>Model</th>
<th>Test AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous production model</td>
<td>0.784</td>
</tr>
<tr>
<td>With graph embeddings</td>
<td>0.877</td>
</tr>
</tbody>
</table>
Graph learning cosine similarity is the top feature in the model.
Online evaluation

Ran a A/B test of the Recommended Dishes Carousel in San Francisco

**Significant** uplift in Click-Through Rate with respect to the previous production model

**Conclusion:** Dish Recommendations with graph learning features are live in San Francisco, soon everywhere else
Serving

Training Pipeline Step 2

Training Pipeline Step 1

Data Pipeline Step 1

Source Tables

Versioned nodes and edges

Daily Ingestion

Date - k

Data Pipeline Step 2

Collapsed graph with latest nodes and edges

Date

Data Pipeline Step 3

Partitioned city graph using Cypher

Past Date

Data Pipeline Step 4

NetworkX graph for model training & embedding generation

GNN Model Training

Node Embeddings

Personalized Ranker Model Training

Ranker Model

Personalized Ranker Online Recommendation

Training Pipeline Step 1

Training Pipeline Step 2

Serving
More Resources

Uber Eng Blog Post

Learn better representation in data scarcity regimes like small/new cities through meta-learning [NeurIPS Graph Representation Learning Workshop 2019]
Learnings

In complex data pipelines, **the model isn't always the bottleneck**

- Graph processing was more expensive than model inference because of sheer size

Even when the model (or the data proc + model) is the bottleneck **you can often precompute and cache**

- Precomputed a big LRU cache of user-to-dish/restaurant similarities. It was recomputed entirely only when the model was updated and refreshed after user ordered
Learnings: online evaluation issues

Q: Despite big offline gains, only got small improvement in Click through rate and orders (still statistically significant and worth millions of dollars), why?

A:
Learnings: online evaluation issues

Q: Despite big offline gains, only got small improvement in Click through rate and orders (still statistically significant and worth millions of dollars), why?

A: Our recommendations where a small part of the UI, "favourite restaurants" and "Daily Deals" came always first in the UI and gathered most of clicks and orders. Bewre how you choose the denominator of your metrics!
Learnings: online evaluation issues

Q: Why is it hard to show big online gains in recommender systems in general?

A:
Q: Why is it hard to show big online gains in recommender systems in general?

A: If there's a model in production you are comparing against, you are likely using biased data for both training and prediction!
Learnings: data bias

**The world changes** (new restaurants and dishes) ->
ML lifecycle is a loop

**The user behavior changes** (now that my favorite pizza place is on the app, I start always ordering from there)

**Model deployment changes user behavior** (the items the model suggest influence your behavior)

**Biased training data and biased evaluation data**
Learnings: data bias

Q: How to collect unbiased data?

A:
Learnings: data bias

Q: How to collect unbiased data?

A: Complicated, one option is to show random recommendations to x% of users
Learnings: data bias

Q: What is the cost of collecting unbiased data?

A:
Learnings: data bias

Q: What is the cost of collecting unbiased data?

A: The likelihood of those users actually selecting those items is very low -> small positive data is collected, those users may not buy anything -> the company loses money!
Learnings: data bias

Q: What could be compromise solutions?

A:
Learnings: data bias

Q: What could be compromise solutions?

A: Show to users random predictions from within the top 100 predicted by the model. Data is still biased, but more likely to collect unexpected positive datapoints.
<table>
<thead>
<tr>
<th>Team</th>
<th>Ankit Jain</th>
<th>Isaac Liu</th>
<th>Ankur Sarda</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Piero Molino</td>
<td>Long Tao</td>
<td>Jimin Jia</td>
</tr>
<tr>
<td></td>
<td>Jan Pedersen</td>
<td>Nathan Berrebbi</td>
<td>Santosh Golecha</td>
</tr>
<tr>
<td></td>
<td>Ramit Hora</td>
<td>Alex Danilychev</td>
<td></td>
</tr>
</tbody>
</table>
Restaurant preparation time

The data generation process
Predict restaurant preparation time is useful, I can decide when to dispatch the driver to reduce wait! (If I can also predict when the driver will arrive)

- How do you know when a restaurant is done preparing?
- The driver can arrive early, in which case the preparation time is from initial order to order pickup
- If the driver arrives late, and the dish is already prepared, the order pickup time is a upper bound
Restaurant preparation time modeling

We tried training a model anyway using order pickup

**Huge variance** in the training data -> **Huge variance** in predictions!

Our model was **5min more accurate** than previous one, but with stddev +-10min!
Restaurant preparation time variance

Drilled into the data to understand the source of variance

Same restaurant, same day, same order, few minutes after -> **20min** prep time vs **2min** prep time

Why?

<table>
<thead>
<tr>
<th>Restaurant</th>
<th>Order</th>
<th>Day</th>
<th>Time</th>
<th>Prep Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>POD Thai</td>
<td>Pho Soup</td>
<td>Tuesday 2nd</td>
<td>19:10</td>
<td>20m</td>
</tr>
<tr>
<td>POD Thai</td>
<td>Pho Soup</td>
<td>Tuesday 2nd</td>
<td>19:15</td>
<td>2m</td>
</tr>
</tbody>
</table>
Restaurant preparation time new feature

Restaurants batch orders!

**Theory**: They prepare a big amount of soup when first ordered, the next soup order will take much less because they are already prepared

Added a feature in the model:

were items in the order ordered in the last $x$ minutes?

Improved predictions by 2min, reduced stddev by 1/3 (still a lot)
Restaurant preparation time moral

Went back to data collection, asked restaurants to notify us when the order was ready

Still noisy data (restaurants have no incentive to be precise, or they forget entirely), but better estimate

**Moral:** ML lifecycle is a loop and you can go back to the data collection process even after deployment, and iterate the process multiple times
What am I working on now

@Stanford with Chris Ré

**Ludwig**: declarative multimodal deep learning pipeline toolbox (no code needed, extensible, AutoML capabilities)

For a talk about Ludwig you can check my website [http://w4nderlu.st](http://w4nderlu.st) or the last Stanford MLSys Seminar Series episode [http://mlsys.stanford.edu](http://mlsys.stanford.edu)

Founded a company to make ML accessible to less technical people: AutoML + end-to-end platform built on Ludwig + secret spicy sauce!