

INNOVATE

ONLINE CONFERENCE

MACHINE LEARNING
AND AI EDITION



Simplifying time-series forecasting and real-time personalization

Danilo Poccia
Principal Evangelist, Serverless
Amazon Web Services

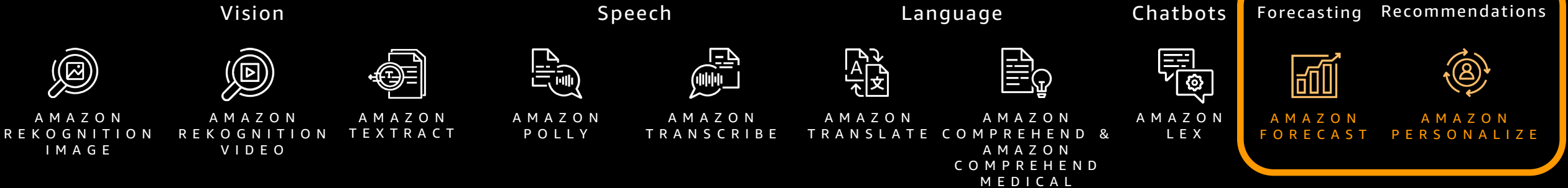
 @danilop

Agenda

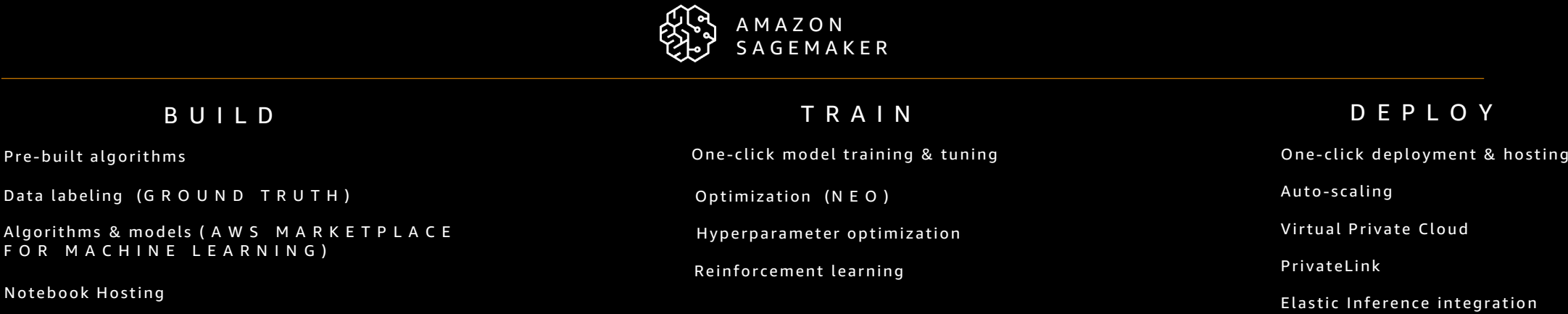
- Time-series forecasting
- Amazon Forecast: Introduction & demo
- Real-time personalization & recommendation
- Amazon Personalize: Introduction & demo
- Takeaways

The Amazon ML stack: Broadest & deepest set of capabilities

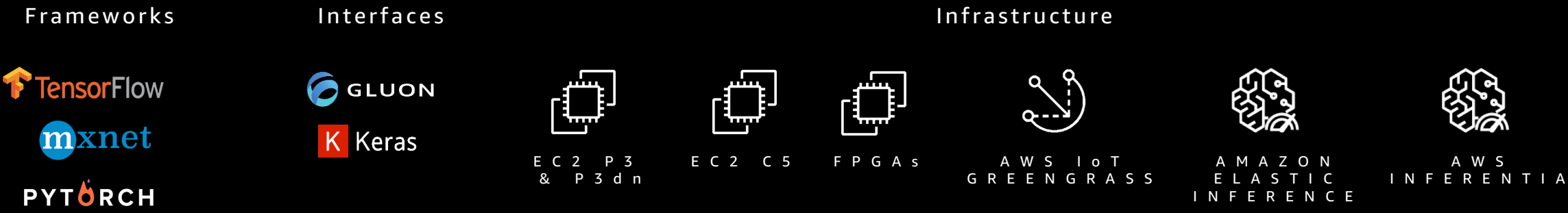
AI SERVICES



ML SERVICES



ML FRAMEWORKS & INFRASTRUCTURE





Time-series forecasting

Forecasting

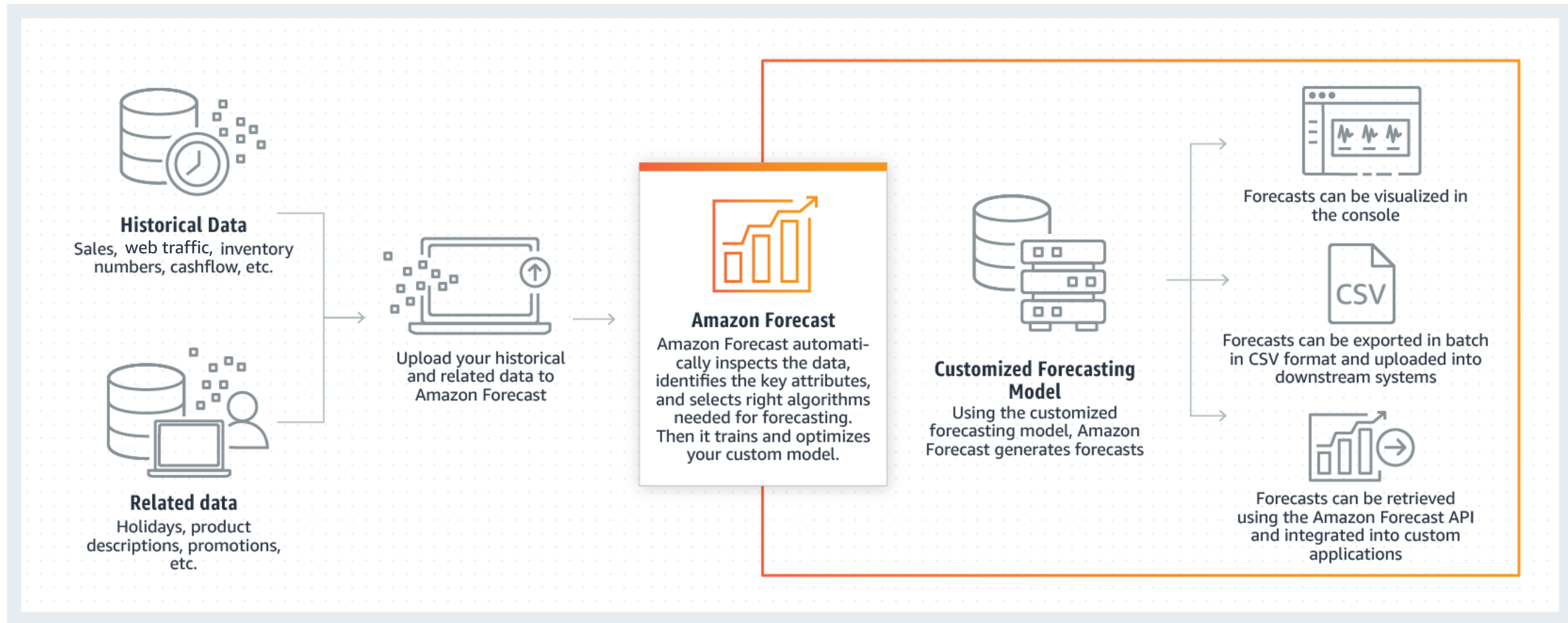


Product demand planning

Financial planning

Resource planning

Amazon Forecast



Amazon Forecast workflow

1. Create related datasets and a dataset group
2. Get training data
 - Import historical data to the dataset group
3. Train a predictor (trained model) using an algorithm or AutoML
4. Evaluate the predictor version using metrics
5. Create a forecast (for every item in the dataset group)
6. Retrieve forecasts for users

How Amazon Forecast works

- Dataset groups
- Datasets
 - TARGET_TIME_SERIES – (item_id, timestamp, demand) – demand is required
 - RELATED_TIME_SERIES – (item_id, timestamp, price) – no demand
 - ITEM_METADATA – (item_id, color, location, genre, category, ...)
- Predictors
- Forecasts

Dataset domains

Domain	For
RETAIL	Retail demand forecasting
INVENTORY_PLANNING	Supply chain and inventory planning
EC2_CAPACITY	Forecasting Amazon EC2 capacity
WORK_FORCE	Workforce planning
WEB_TRAFFIC	Estimating future web traffic
METRICS	Forecasting metrics, such as revenue and cash flow
CUSTOM	All other types of time-series forecasting

TARGET_TIME_SERIES dataset

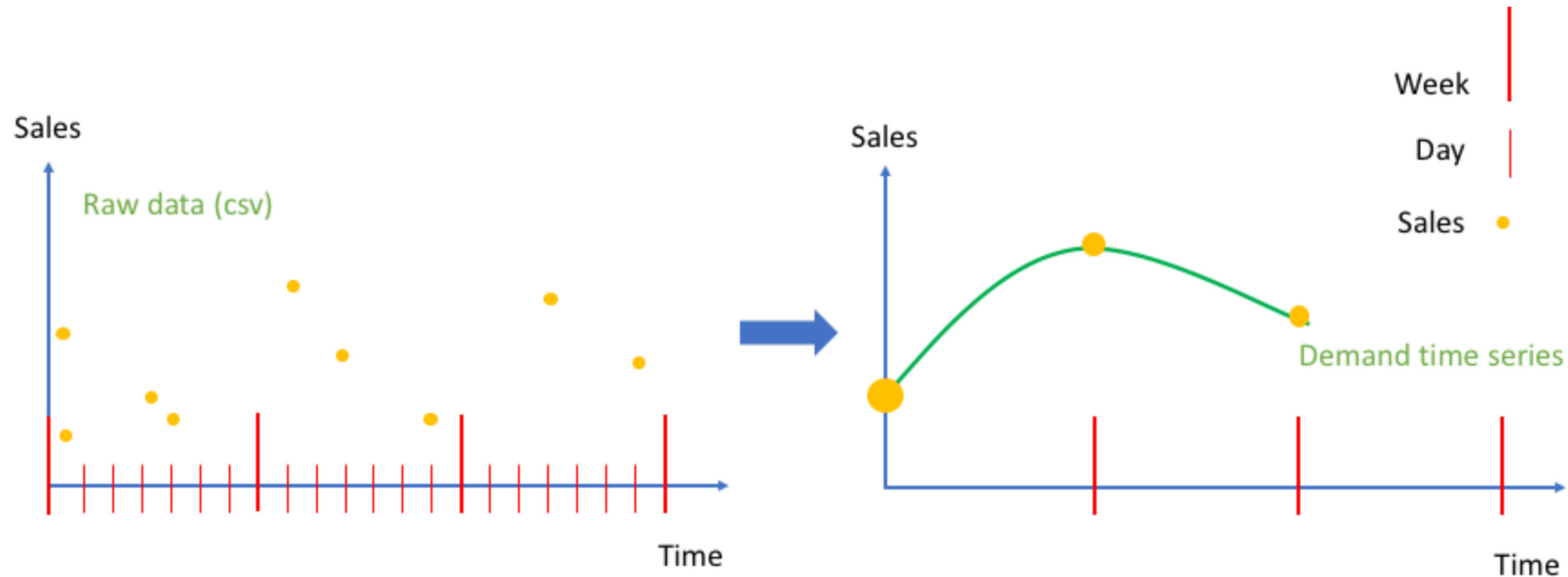
timestamp	item_id	store	demand
2019-01-01	socks	NYC	25
2019-01-05	socks	SFO	45
2019-02-01	shoes	ORD	10
...			
2019-06-01	socks	NYC	100
2019-06-05	socks	SFO	5
2019-07-01	shoes	ORD	50

Dataset schema

```
{
  "attributes": [
    {
      "attributeName": "timestamp",
      "attributeType": "timestamp",
    },
    {
      "attributeName": "item_id",
      "attributeType": "string",
    },
    {
      "attributeName": "store",
      "attributeType": "string",
    },
    {
      "attributeName": "demand",
      "attributeType": "float",
    }
  ]
}
```

← "YYYY-MM-DD hh:mm:ss"

Data alignment



Data is automatically aggregated by forecast frequency, for example, hourly, daily, or weekly.

RELATED_TIME_SERIES dataset

timestamp	item_id	store	price
2019-01-01	socks	NYC	10
2019-01-02	socks	NYC	10
2019-01-03	socks	NYC	15
...			
2019-01-05	socks	SFO	45
2019-06-05	socks	SFO	10
2019-07-11	socks	SFO	30
...			
2019-02-01	shoes	ORD	50
2019-07-01	shoes	ORD	75
2019-07-11	shoes	ORD	60

Algorithms

Algorithm	What
ARIMA	Autoregressive integrated moving average (ARIMA) is a commonly used local statistical algorithm for time-series forecasting
DeepAR+	A supervised learning algorithm for forecasting scalar (one-dimensional) time series using recurrent neural networks (RNNs); supports hyperparameter optimization (HPO)
ETS	Exponential smoothing (ETS) is a commonly used local statistical algorithm for time-series forecasting
NPTS	Non-parametric time series (NPTS) is a scalable, probabilistic baseline forecaster algorithm; NPTS is especially useful when the time series is intermittent (or sparse, containing many 0s) and bursty
Prophet	A popular local Bayesian structural time series model

DeepAR algorithm

arXiv:1704.04110v3 [cs.AI] 22 Feb 2019

DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks

David Salinas, Valentin Flunkert, Jan Gasthaus
Amazon Research
Germany
<dsalina,flunkert,gasthaus@amazon.com>

Abstract

Probabilistic forecasting, i.e. estimating the probability distribution of a time series' future given its past, is a key enabler for optimizing business processes. In retail businesses, for example, forecasting demand is crucial for having the right inventory available at the right time at the right place. In this paper we propose DeepAR, a methodology for producing accurate probabilistic forecasts, based on training an auto-regressive recurrent network model on a large number of related time series. We demonstrate how by applying deep learning techniques to forecasting, one can overcome many of the challenges faced by widely-used classical approaches to the problem. We show through extensive empirical evaluation on several real-world forecasting data sets accuracy improvements of around 15% compared to state-of-the-art methods.

1 Introduction

Forecasting plays a key role in automating and optimizing operational processes in most businesses and enables data driven decision making. In retail for example, probabilistic forecasts of product supply and demand can be used for optimal inventory management, staff scheduling and topology planning [13], and are more generally a crucial technology for most aspects of supply chain optimization.

The prevalent forecasting methods in use today have been developed in the setting of forecasting individual or small groups of time series. In this approach, model parameters for each given time series are independently estimated from past observations. The model is typically manually selected to account for different factors, such as autocorrelation structure, trend, seasonality, and other explanatory variables. The fitted model is then used to forecast the time series into the future according to the model dynamics, possibly admitting probabilistic forecasts through simulation or closed-form expressions for the predictive distributions. Many methods in this class are based on the classical Box-Jenkins methodology [3], exponential smoothing techniques, or state space models [11, 19].

In recent years, a new type of forecasting problem has become increasingly important in many applications. Instead of needing to predict individual or a small number of time series, one is faced with forecasting thousands or millions of related time series. Examples include forecasting the energy consumption of individual households, forecasting the load for servers in a data center, or forecasting the demand for all products that a large retailer offers. In all these scenarios, a substantial amount of data on past behavior of similar, related time series can be leveraged for making a forecast for an individual time series. Using data from related time series not only allows fitting more complex (and hence potentially more accurate) models without overfitting, it can also alleviate the time and labor intensive manual feature engineering and model selection steps required by classical techniques.

In this work we present DeepAR, a forecasting method based on autoregressive recurrent networks, which learns such a *global* model from historical data of *all time series* in the data set. Our method

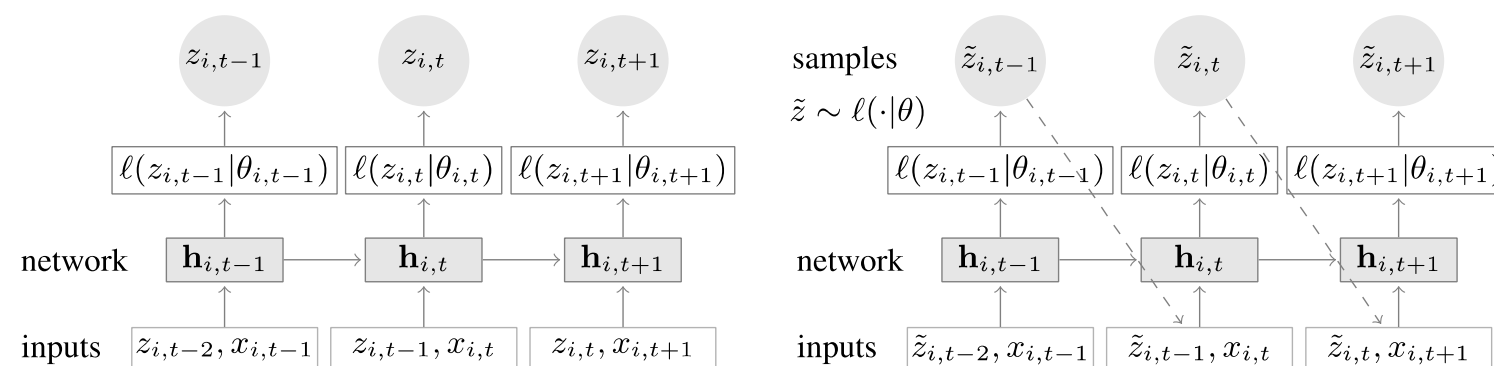
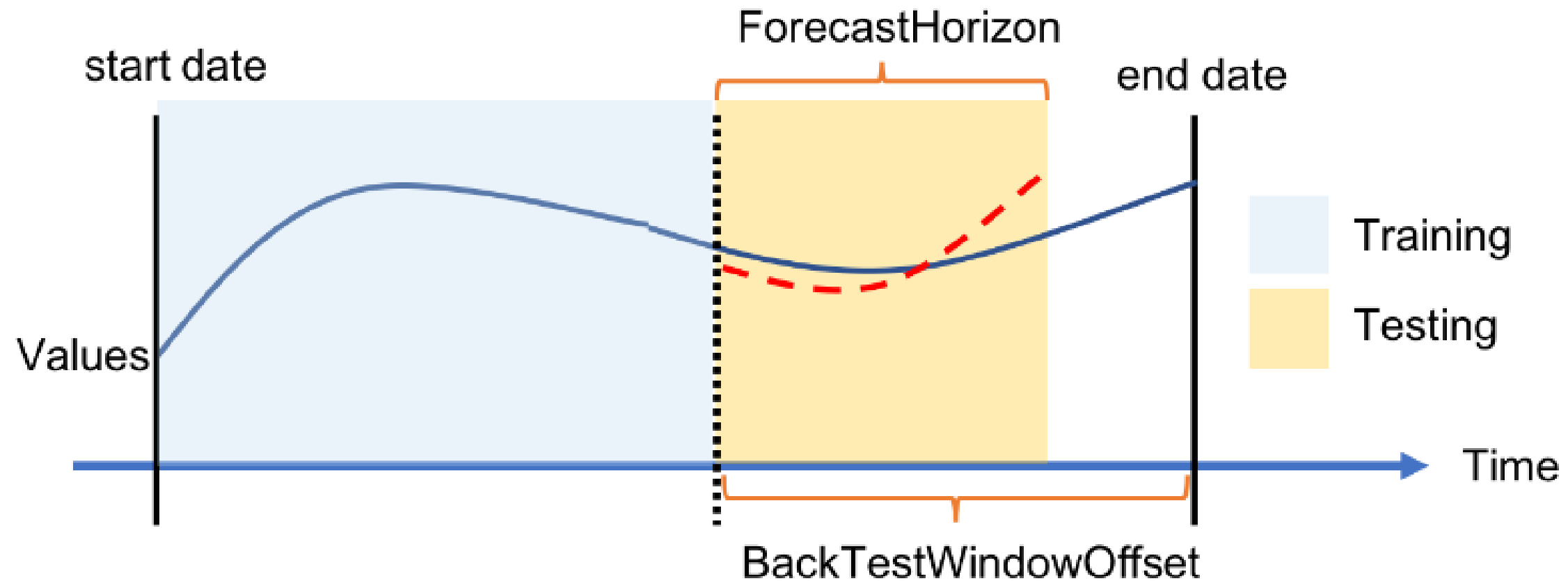


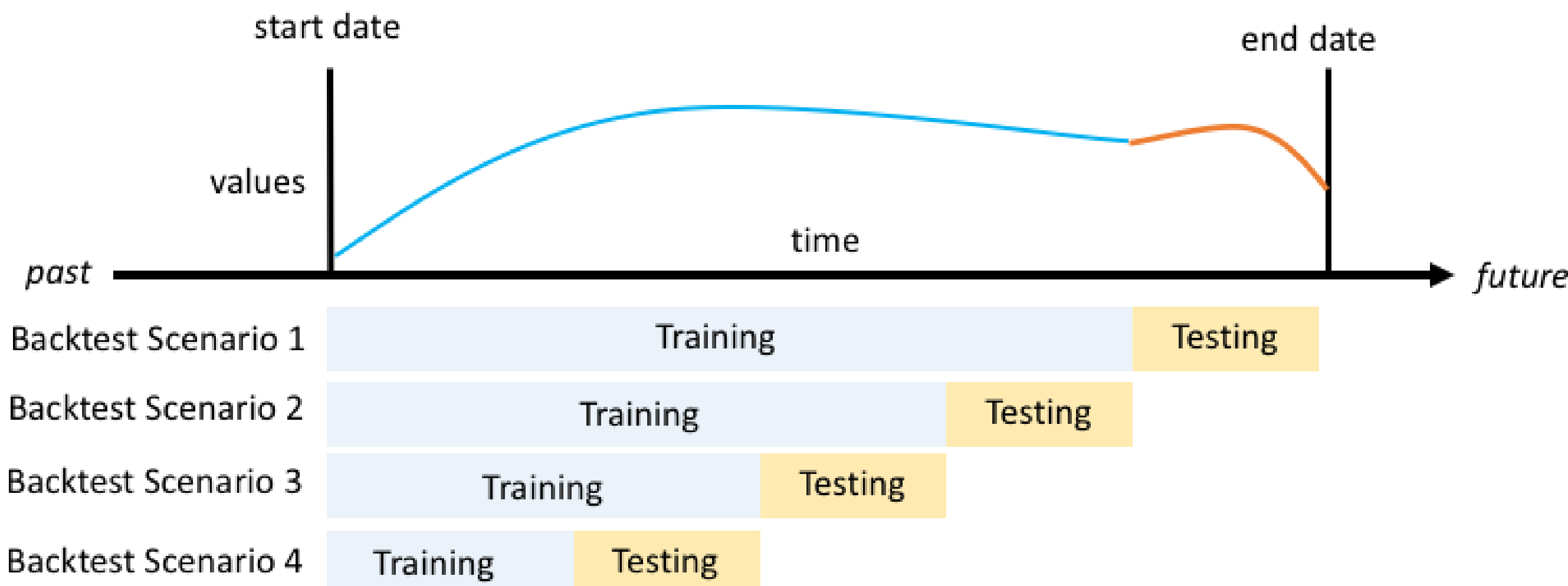
Figure 2: Summary of the model. Training (left): At each time step t , the inputs to the network are the covariates $x_{i,t}$, the target value at the previous time step $z_{i,t-1}$, as well as the previous network output $h_{i,t-1}$. The network output $h_{i,t} = h(h_{i,t-1}, z_{i,t-1}, x_{i,t}, \Theta)$ is then used to compute the parameters $\theta_{i,t} = \theta(h_{i,t}, \Theta)$ of the likelihood $\ell(z|\theta)$, which is used for training the model parameters. For prediction, the history of the time series $z_{i,t}$ is fed in for $t < t_0$, then in the prediction range (right) for $t \geq t_0$ a sample $\hat{z}_{i,t} \sim \ell(\cdot|\theta_{i,t})$ is drawn and fed back for the next point until the end of the prediction range $t = t_0 + T$ generating one sample trace. Repeating this prediction process yields many traces representing the joint predicted distribution.

<https://arxiv.org/abs/1704.04110>

Training using a BackTestWindow



Training & testing



Predictor metrics

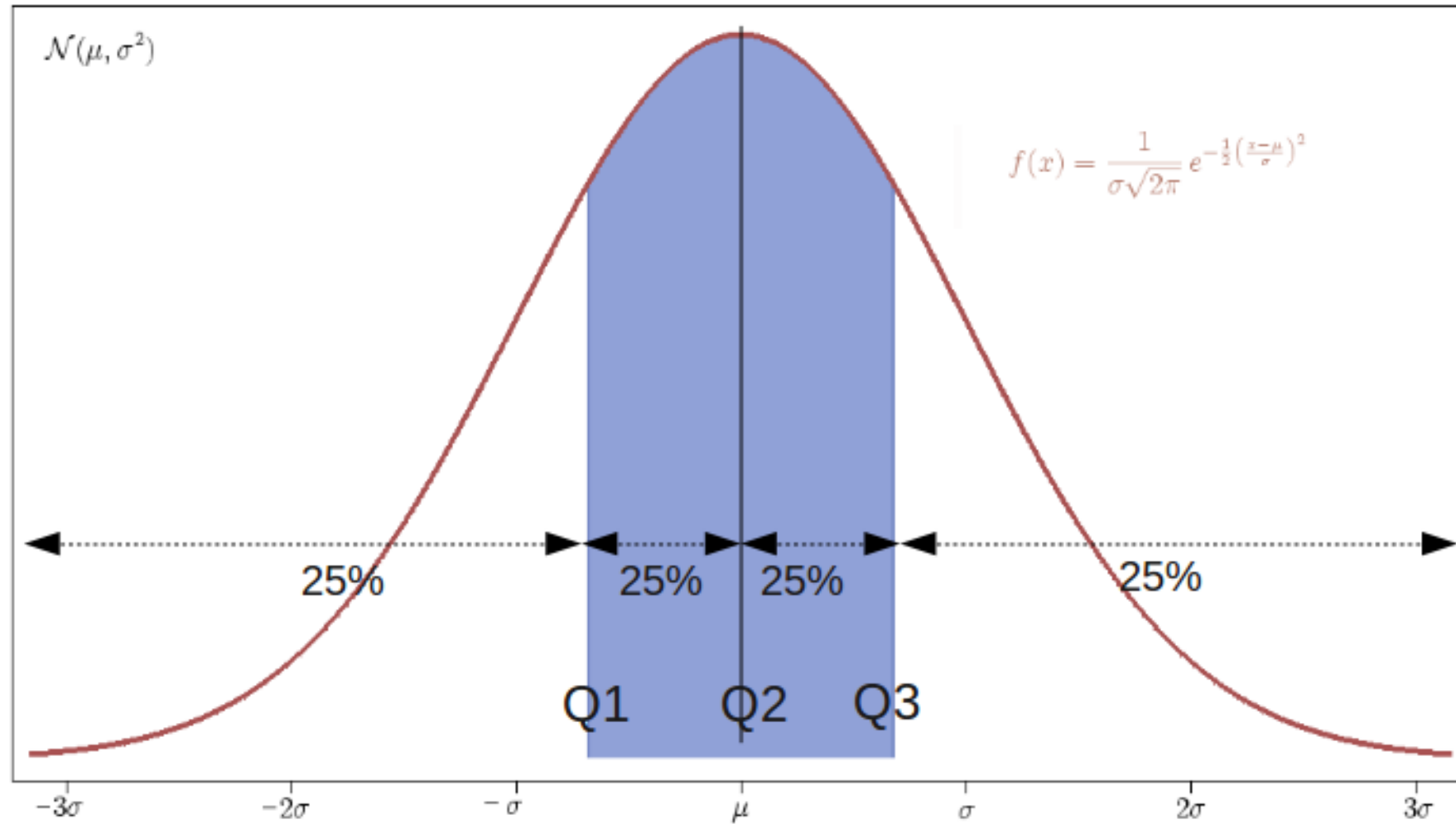
	Predictor name ▼	Training status ▼	wQL[0.5] ▼	wQL[0.9] ▼	wQL[0.1] ▼
<input type="radio"/>	elec_predictor	✔ Active	0.1814	0.1333	0.0414

$$\text{RMSE} = \sqrt{\frac{1}{nT} \sum_{i,t} (\hat{y}_{i,t} - y_{i,t})^2} \quad \text{Root Mean Square Error}$$

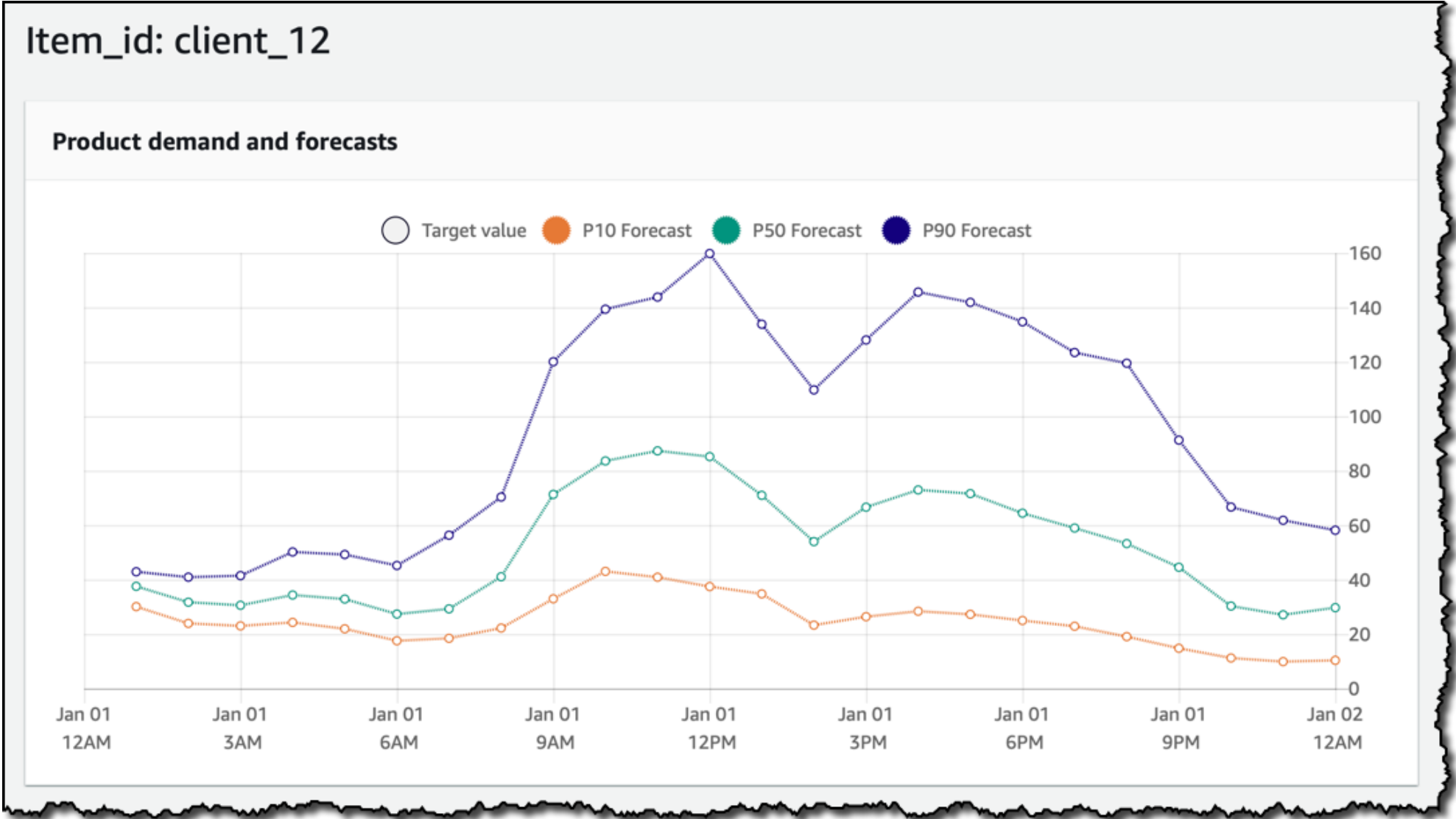
$$\text{wQuantileLoss}[\tau] = 2 \frac{\sum_{i,t} [\tau \max(y_{i,t} - q_{i,t}^{(\tau)}, 0) + (1 - \tau) \max(q_{i,t}^{(\tau)} - y_{i,t}, 0)]}{\sum_{i,t} |y_{i,t}|}$$

$$\text{wQuantileLoss}[0.5] \longrightarrow \text{MAPE} = \frac{\sum_{i,t} |\hat{y}_{i,t} - y_{i,t}|}{\sum_{i,t} |y_{i,t}|} \quad \text{Mean Absolute Percentage Error}$$

Predictor metrics: Quantiles



Getting a forecast: Interpreting P-numbers





Demo: Amazon Forecast

Amazon Forecast examples & notebooks



<https://github.com/aws-samples/amazon-forecast-samples>



Real-time personalization & recommendation

Personalization & recommendation

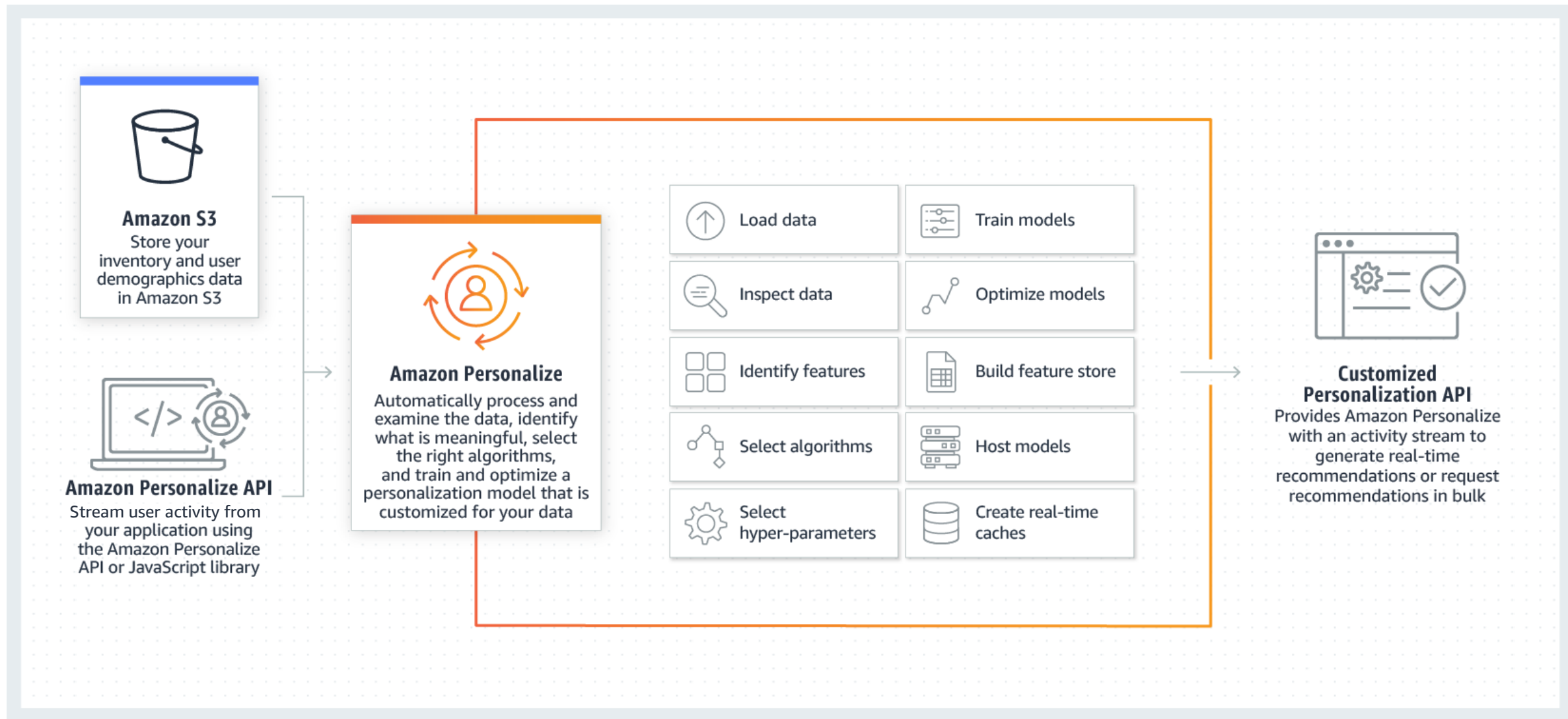


Personalized recommendations

Personalized search

Personalized notifications

Amazon Personalize



Amazon Personalize workflow

1. Create related datasets and a dataset group
2. Get training data
 - Import historical data into the dataset group
 - Record live events to the dataset group
3. Create a solution version (trained model) using a recipe or AutoML
4. Evaluate the solution version using metrics
5. Create a campaign (deploy the solution version)
6. Provide recommendations for users

How Amazon Personalize works

- Dataset groups
- Datasets
 - Users: Age, gender, or loyalty membership
 - Items: Price, type, or availability
 - Interactions: Between users and items
- User events
- Recipes and solutions
- Metrics
- Campaigns
- Recommendations

Dataset schemas

Dataset Type	Required Fields	Reserved Keywords
Users	USER_ID (string) 1 metadata field	
Items	ITEM_ID (string) 1 metadata field	
Interactions	USER_ID (string) ITEM_ID (string) TIMESTAMP (long)	EVENT_TYPE (string) EVENT_VALUE (string)



Training data

userId	movieId	timestamp
1	1	964982703
1	3	964981247
1	6	964982224
2	47	964983815
2	50	964982931
2	70	964982400
...		

Training data schema: Users

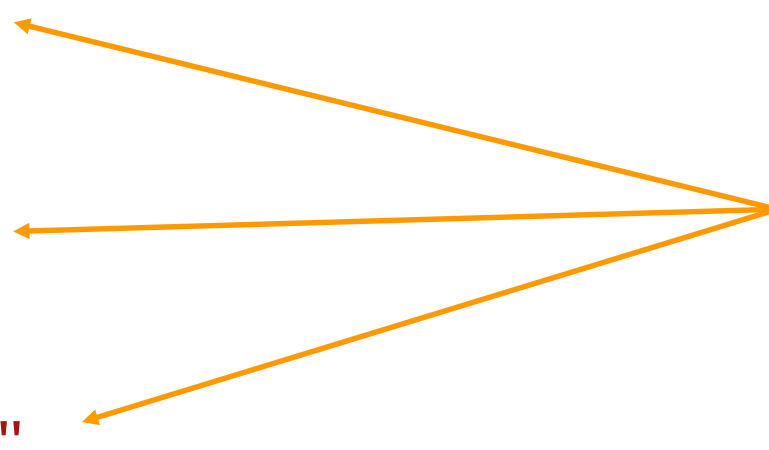
```
{  
  "type": "record",  
  "name": "Users",  
  "namespace": "com.amazonaws.personalize.schema",  
  "fields": [  
    {  
      "name": "USER_ID",  
      "type": "string"  
    },  
    {  
      "name": "AGE",  
      "type": "int"  
    },  
    {  
      "name": "GENDER",  
      "type": "string",  
      "categorical": true  
    }  
  ],  
  "version": "1.0"  
}
```

For categories, like genre

Training data schema: Interactions

```
{
  "type": "record",
  "name": "Interactions",
  "namespace": "com.amazonaws.personalize.schema",
  "fields": [
    {
      "name": "USER_ID",
      "type": "string"
    },
    {
      "name": "ITEM_ID",
      "type": "string"
    },
    {
      "name": "TIMESTAMP",
      "type": "long"
    }
  ],
  "version": "1.0"
}
```

An interaction between a user and an item at a specific point in time



Using EVENT_TYPE and EVENT_VALUE fields

```
{
  "type": "record",
  "name": "Interactions",
  "namespace": "com.amazonaws.personalize.schema",
  "fields": [
    {
      "name": "USER_ID",
      "type": "string"
    },
    {
      "name": "ITEM_ID",
      "type": "string"
    },
    {
      "name": "EVENT_TYPE",
      "type": "string"
    },
    {
      "name": "EVENT_VALUE",
      "type": "float"
    },
    {
      "name": "TIMESTAMP",
      "type": "long"
    }
  ],
  "version": "1.0"
}
```

You can filter events based on the type (ex., "purchase," "click," or "wishlist") or a value (ex., a rating or the paid price)

Using categorical data

You can include more than one category in the training data using the “vertical bar” character, also known as “pipe”:

```
ITEM_ID, GENRE  
item_123, horror | comedy
```

Multiple categories



Solution metrics

```
{
  "solutionVersionArn": "arn:aws:personalize:...",
  "metrics": {
    "arn:aws:personalize:...": {
      "coverage": 0.27,
      "mean_reciprocal_rank_at_25": 0.0379,
      "normalized_discounted_cumulative_gain_at_5": 0.0405,
      "normalized_discounted_cumulative_gain_at_10": 0.0513,
      "normalized_discounted_cumulative_gain_at_25": 0.0828,
      "precision_at_5": 0.0136,
      "precision_at_10": 0.0102,
      "precision_at_25": 0.0091
    }
  }
}
```

With the exception of coverage,
higher is better



Evaluating a solution version

Metric	How	When
coverage	The proportion of unique recommended items from all queries out of the total number of unique items in the training data.	
mean_reciprocal_rank_at_25	The mean of the reciprocal ranks of the first relevant recommendation out of the top 25 recommendations over all queries.	This metric is appropriate if you're interested in the single highest-ranked recommendation.
normalized_discounted_cumulative_gain_at_K	Discounted gain assumes that recommendations lower on a list of recommendations are less relevant than higher recommendations. Therefore, each recommendation is given a lower weight by a factor dependent on its position.	This metric rewards relevant items that appear near the top of the list because the top of a list usually draws more attention.
precision_at_K	The number of relevant recommendations out of the top K recommendations divided by K.	This metric rewards precise recommendations of the relevant items.



Recording live events: Getting a tracking ID

```
import boto3

personalize = boto3.client('personalize')

response = personalize.create_event_tracker(
    name='MovieClickTracker',
    datasetGroupArn='arn:aws:personalize:...'
)

print(response['eventTrackerArn'])
print(response['trackingId'])
```

Recording live events: Event-interactions dataset

```
{
  "datasets": [
    {
      "name": "ratings-dsgroup/EVENT_INTERACTIONS",
      "datasetArn": "arn:aws:personalize:...",
      "datasetType": "EVENT_INTERACTIONS",
      "status": "ACTIVE",
      "creationDateTime": 1554304597.806,
      "lastUpdatedDateTime": 1554304597.806
    },
    {
      "name": "ratings-dataset",
      "datasetArn": "arn:aws:personalize:...",
      "datasetType": "INTERACTIONS",
      "status": "ACTIVE",
      "creationDateTime": 1554299406.53,
      "lastUpdatedDateTime": 1554299406.53
    }
  ],
  "nextToken": "...
}
```

New dataset created automatically for the tracking events

Recording live events: PutEvents operation

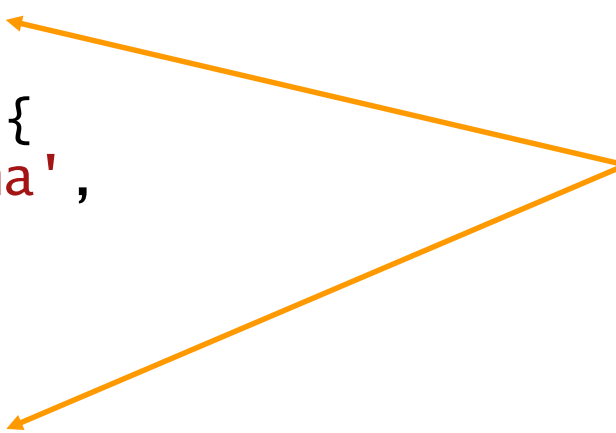
```
import boto3

personalize_events = boto3.client(service_name='personalize-events')

personalize_events.put_events(
    trackingId = 'tracking_id',
    userId= 'USER_ID',
    sessionId = 'session_id',
    eventList = [{
        'sentAt': TIMESTAMP,
        'eventType': 'EVENT_TYPE',
        'properties': "{\\"itemId\\": \\"ITEM_ID\\"}"
    }]
)
```

More advanced PutEvents operation

```
personalize_events.put_events(  
    trackingId = 'tracking_id',  
    userId= 'user555',  
    sessionId = 'session1',  
    eventList = [{  
        'eventId': 'event1',  
        'sentAt': '1553631760',  
        'eventType': 'like',  
        'properties': json.dumps({  
            'itemId': 'choc-panama',  
            'eventValue': 'true'  
        })  
    }, {  
        'eventId': 'event2',  
        'sentAt': '1553631782',  
        'eventType': 'rating',  
        'properties': json.dumps({  
            'itemId': 'movie_ten',  
            'eventValue': '4',  
            'numRatings': '13'  
        })  
    }]  
)
```



Multiple events
with more data

Recording live events with AWS Amplify

```
import { Analytics, AmazonPersonalizeProvider } from 'aws-amplify';
Analytics.addPluggable(new AmazonPersonalizeProvider());
// Configure the plugin after adding it to the Analytics module
Analytics.configure({
  AmazonPersonalize: {
    // REQUIRED - The trackingId to track the events
    trackingId: '<TRACKING_ID>',

    // OPTIONAL - Amazon Personalize service region
    region: 'XX-XXXX-X',

    // OPTIONAL - The number of events to be deleted from the buffer when flushed
    flushSize: 10,

    // OPTIONAL - The interval in ms to perform a buffer check and flush if necessary
    flushInterval: 5000, // 5s
  }
});
```

Using Amazon Personalize

Recording live events with AWS Amplify

```
Analytics.record({  
  eventType: "Identify",  
  properties: {  
    "userId": "<USER_ID>"  
  }  
}, 'AmazonPersonalize');
```

Send events from the browser

```
Analytics.record({  
  eventType: "<EVENT_TYPE>",  
  userId: "<USER_ID>", (optional)  
  properties: {  
    "itemId": "<ITEM_ID>",  
    "eventValue": "<EVENT_VALUE>"  
  }  
}, "AmazonPersonalize");
```

<https://aws-amplify.github.io/docs/js/analytics>

Using predefined recipes

Recipe type	API	userId	itemId	inputList
USER_PERSONALIZATION	GetRecommendations	required	optional	N/A
PERSONALIZED_RANKING	GetPersonalizedRanking	required	N/A	list of itemId's
RELATED_ITEMS	GetRecommendations	not used	required	N/A



Predefined USER_PERSONALIZATION Recipes

Recipe	How	When	AutoML	Metadata
HRNN	A hierarchical recurrent neural network, which can model the temporal order of user-item interactions.	Recommended when user behavior is changing with time (the evolving intent problem).	✓	
HRNN-Metadata	HRNN with additional features derived from contextual metadata (Interactions dataset), along with user and item metadata (Users and Items datasets).	Performs better than non-metadata models when high-quality metadata is available. Can involve longer training times.	✓	✓
HRNN-Coldstart	Similar to HRNN-metadata with personalized exploration of new items.	Recommended when frequently adding new items to a catalog and you want the items to immediately show up in recommendations.	✓	✓
Popularity-Count	Calculates popularity of items based on a count of events against that item in the user-item interactions dataset.	Use as a baseline to compare other user-personalization recipes.		

Hierarchical recurrent neural networks (HRNNs)

Personalizing Session-based Recommendations with Hierarchical Recurrent Neural Networks

Massimo Quadrana
Politecnico di Milano, Milan, Italy
massimo.quadrana@polimi.it

Balázs Hidasi
Gravity R&D, Budapest, Hungary
balazs.hidasi@gravityrd.com

Alexandros Karatzoglou
Telefonica Research, Barcelona, Spain
alexk@tid.es

Paolo Cremonesi
Politecnico di Milano, Milan, Italy
paolo.cremonesi@polimi.it

ABSTRACT

Session-based recommendations are highly relevant in many modern on-line services (e.g. e-commerce, video streaming) and recommendation settings. Recently, Recurrent Neural Networks have been shown to perform very well in session-based settings. While in many session-based recommendation domains user identifiers are hard to come by, there are also domains in which user profiles are readily available. We propose a seamless way to personalize RNN models with cross-session information transfer and devise a Hierarchical RNN model that relays end evolves latent hidden states of the RNNs across user sessions. Results on two industry datasets show large improvements over the session-only RNNs.

CCS CONCEPTS

• Information systems → Recommender systems; • Computing methodologies → Neural networks;

KEYWORDS

recurrent neural networks; personalization; session-based recommendation; session-aware recommendation

1 INTRODUCTION

In many online systems where recommendations are applied, interactions between a user and the system are organized into sessions. A session is a group of interactions that take place within a given time frame. Sessions from a user can occur on the same day, or over several days, weeks, or months. A session usually has a goal, such as finding a good restaurant in a city, or listening to music of a certain style or mood.

Providing recommendations in these domains poses unique challenges that until recently have been mainly tackled by applying conventional recommender algorithms [10] on either the last interaction or the last session (session-based recommenders). Recurrent Neural Networks (RNNs) have been recently used for the purpose of session-based recommendations [7] outperforming item-based

methods by 15% to 30% in terms of ranking metrics. In *session-based recommenders*, recommendations are provided based solely on the interactions in the current user session, as user are assumed to be anonymous. But in many of these systems there are cases where a user might be logged-in (e.g. music streaming services) or some form of user identifier might be present (cookie or other identifier). In these cases it is reasonable to assume that the user behavior in past sessions might provide valuable information for providing recommendations in the next session.

A simple way of incorporating past user session information in session-based algorithm would be to simply concatenate past and current user sessions. While this seems like a reasonable approach, we will see in the experimental section that this does not yield the best results.

In this work we describe a novel algorithm based on RNNs that can deal with both cases: (i) *session-aware recommenders*, when user identifiers are present and propagate information from the previous user session to the next, thus improving the recommendation accuracy, and (ii) *session-based recommenders*, when there are no past sessions (i.e., no user identifiers). The algorithm is based on a Hierarchical RNN where the hidden state of a lower-level RNN at the end of one user session is passed as an input to a higher-level RNN which aims at predicting a good initialization (i.e., a good context vector) for the hidden state of the lower RNN for the next session of the user.

We evaluate the Hierarchical RNNs on two datasets from industry comparing them to the plain session-based RNN and to item-based collaborative filtering. Hierarchical RNNs outperform both alternatives by a healthy margin.

2 RELATED WORK

Session-based recommendations. Classical CF methods (e.g. matrix factorization) break down in the session-based setting when no user profile can be constructed from past user behavior. A natural solution to this problem is the item-to-item recommendation approach [11, 16]. In this setting an item-to-item similarity matrix is precomputed from the available session data, items that are often clicked together in sessions are deemed to be similar. These similarities are then used to create recommendations. While simple, this method has been proven to be effective and is widely employed. Though, these methods only take into account the last click of the user, in effect ignoring the information of the previous clicks.

s_1

s_2

$i_{1,3}$

$i_{2,1}$

$i_{2,2}$

$i_{2,3}$

$i_{2,4}$

c_0



c_1

user representation
propagation

c_2

<https://arxiv.org/abs/1706.04148>



Predefined PERSONALIZED_RANKING recipes

Recipe	How	When	AutoML	Metadata
Personalized-Ranking		Use this recipe when you're personalizing the results for your users, such as personalized reranking of search results or curated lists.		

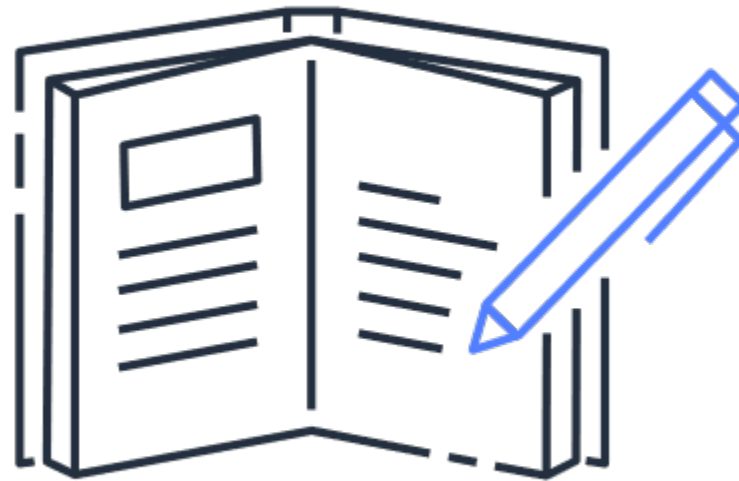
Predefined RELATED_ITEMS recipes

Recipe	How	When	AutoML	Metadata
SIMS	<p>Item-to-item similarities (SIMS) is based on the concept of collaborative filtering.</p> <p>It generates items similar to a given item based on co-occurrence of the item in user history in the user-item interaction dataset.</p> <p>In the absence of sufficient user behavior data for an item, or if the specified item ID is not found, the algorithm returns popular items as recommendations.</p>	<p>Use for improving item discoverability and in detail pages.</p> <p>Provides fast performance.</p>		



Demo: Amazon Personalize

Amazon Personalize examples & notebooks



<https://github.com/aws-samples/amazon-personalize-samples>



Takeaways

Takeaways

- Forecasting and personalization can help improve your business efficiency
- Amazon Forecast provides accurate time-series forecasting
- Amazon Personalize provides real-time personalization and recommendations
- They are both based on the same technology used at Amazon.com and don't require machine learning expertise to be used

Links

- Blogs

- <https://aws.amazon.com/blogs/aws/amazon-forecast-time-series-forecasting-made-easy/>
- <https://aws.amazon.com/blogs/aws/amazon-forecast-now-generally-available/>
- <https://aws.amazon.com/blogs/aws/amazon-personalize-real-time-personalization-and-recommendation-for-everyone/>
- <https://aws.amazon.com/blogs/aws/amazon-personalize-is-now-generally-available/>

- Examples & Notebooks

- <https://github.com/aws-samples/amazon-forecast-samples>
- <https://github.com/aws-samples/amazon-personalize-samples>

Links

- Training algorithms
 - DeepAR – <https://arxiv.org/abs/1704.04110>
 - HRNN – <https://arxiv.org/abs/1706.04148>
- Evaluating performance of a trained model
 - https://en.wikipedia.org/wiki/Mean_absolute_percentage_error (MAPE)
 - https://en.wikipedia.org/wiki/Quantile_regression
 - https://en.wikipedia.org/wiki/Mean_reciprocal_rank
 - https://en.wikipedia.org/wiki/Discounted_cumulative_gain



Thank you!

Danilo Poccia
Principal Evangelist, Serverless
Amazon Web Services

 @danilop

INNOVATE | MACHINE LEARNING
ONLINE CONFERENCE AND AI EDITION