

O'REILLY[®]

Artificial Intelligence Conference

PRESENTED WITH  intel AI

Using deep learning and time-series forecasting to reduce transit delays

Mark Ryan mryan@ca.ibm.com

Alina Zhang alina.li.zhang@gmail.com

theaiconf.com



Introduction

- Starting with a publicly-available, real-world structured dataset on light-rail transit delays:
 - Explore data on delays
 - Prepare dataset
 - Train & assess deep learning model to predict delays
- Example of:
 - Applying deep learning to a medium-sized tabular dataset
 - Transforming ill-formed address information into map visualizations
 - Combining multiple data types (continuous, categorical and text) in a single deep learning model that incorporates embeddings

The problem: streetcar delays

- Toronto has the biggest network streetcar network in North America
- Advantages: greener / lower labour cost than buses; cheaper than subways
- Major disadvantage: streetcar delays trigger general gridlock
- Goal: Prevent gridlock by predicting and preventing streetcar delays



The solution: DL + TSF

- Apply standard analysis techniques and time series forecasting to analyze the data
- Apply data transformations to prepare it for training
- Generate Keras DL model based on categories of columns in the data (categorical, continuous, and text)
- Iteratively train and assess model to improve accuracy
- Create pipeline to encapsulate data preparation steps + model training

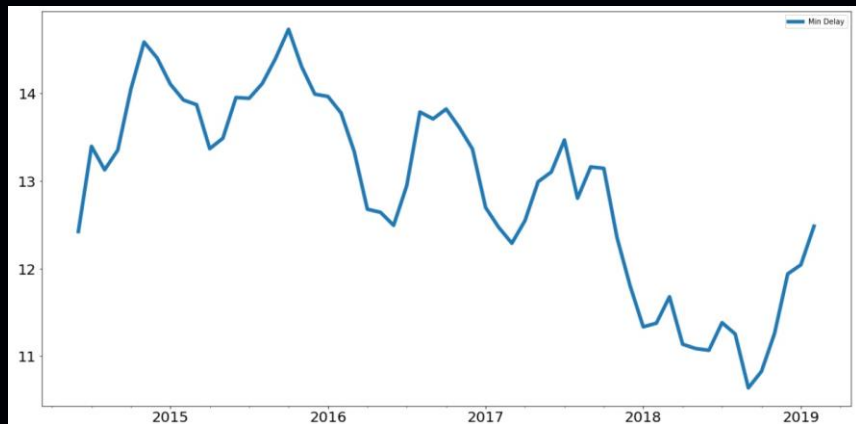
The dataset: overview

- ~70 k records with details of all streetcar delays since January 2014
- Messy data:
 - Invalid routes, vehicles, and direction of travel
 - Locations are free-form, inconsistent descriptions
- *Interesting, real-world dataset that demands serious effort to prepare for machine learning*

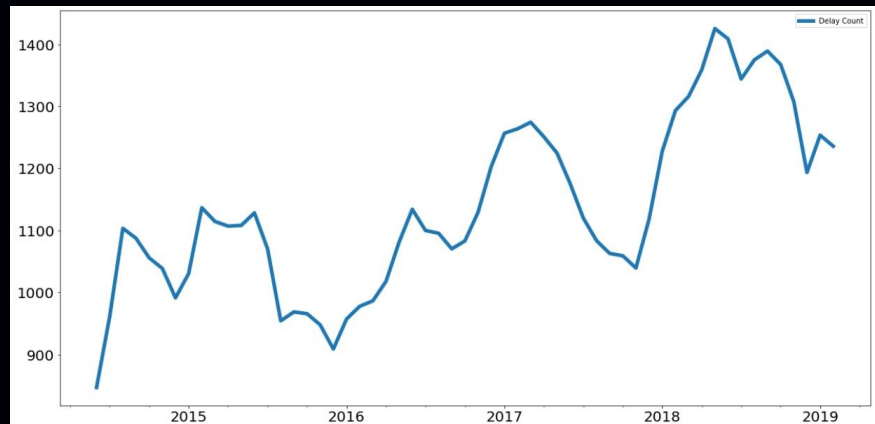
1	Report Date	Route	Time	Day	Location	Incident	Min Delay	Min Gap	Direction	Vehicle
2	2014-12-01	510	1:28:00 AM	Monday	Spadina and Oxford	Emergency Services	77	87	B/W	4124
3	2014-12-01	306	3:59:00 AM	Monday	Gerrard and Kingsmount Park Rd.	Investigation	41	71	W/B	4044
4	2014-12-01	512	5:02:00 AM	Monday	Exhibition Loop	Late Leaving Garage	8	16	W/B	4171
5	2014-12-01	504	5:36:00 AM	Monday	Queen and Roncesvalles	Late Leaving Garage	6	12	E/B	4233
6	2014-12-01	506	5:52:00 AM	Monday	Coxwell and Gerrard	Mechanical	4	8	E/B	4077

The dataset: variations by year

Delay duration: rolling average

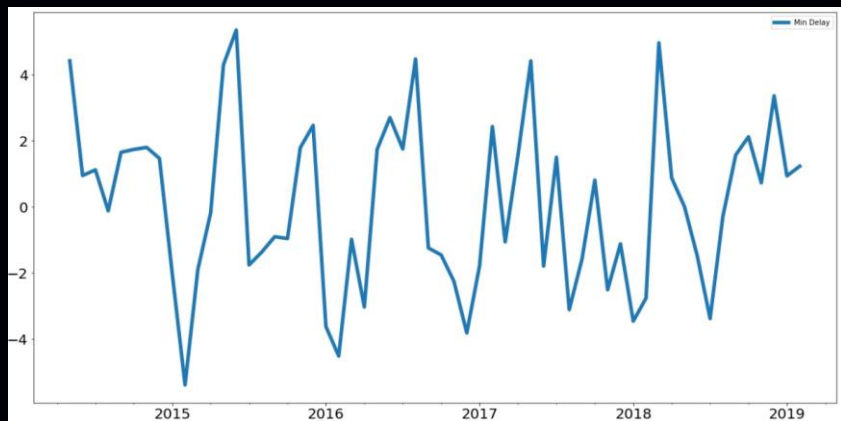


Delay count: rolling average

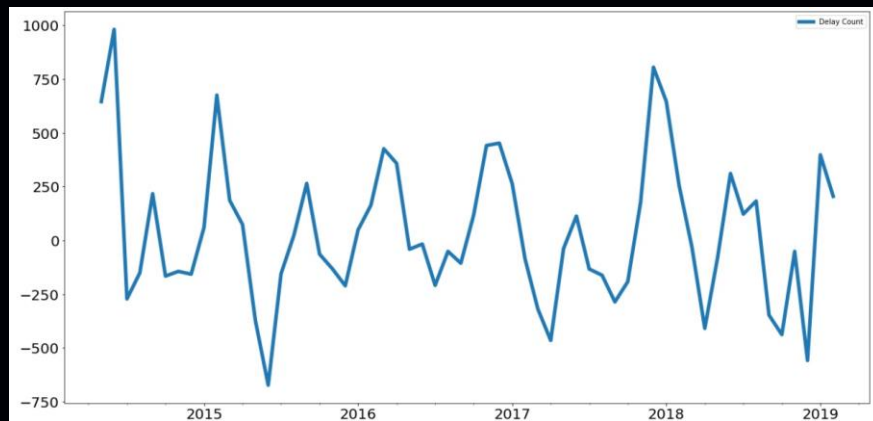


The dataset: seasonality analysis

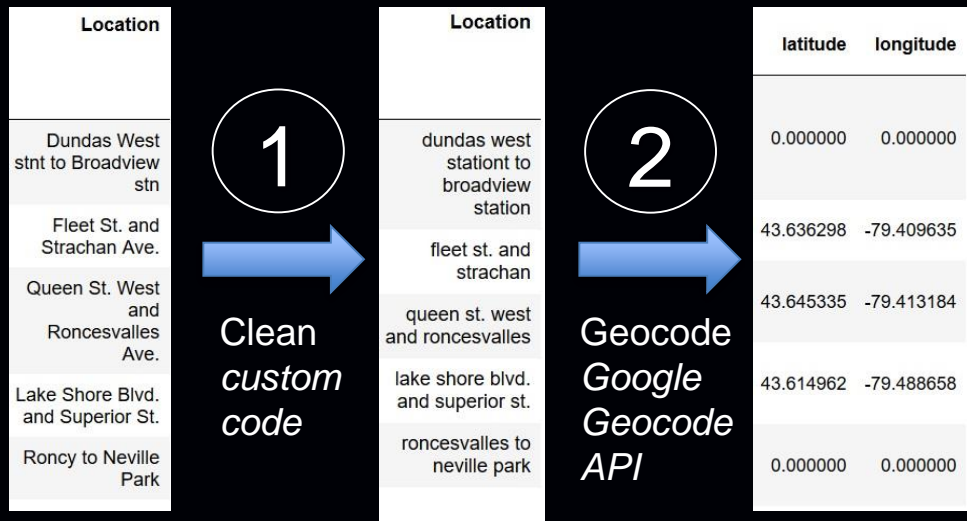
Delay duration seasonality



Delay count seasonality



The dataset: messy addresses



How to get from messy input addresses to latitude / longitude?

1. Clean up location values:

- Common casing
- Consistent order of street names
- Consistent terms ("stn" -> "station" ; "Roncy" -> "Roncesvalles")

2. Apply Google Geocode API:

- Batch call on cleaned up location values
- Parse returned JSON and save latitude and longitude values as new features

The dataset: visualize locations

Pixiedust: rough identification of invalid locations



Folium: zero in on hotspots



The dataset: visualize locations

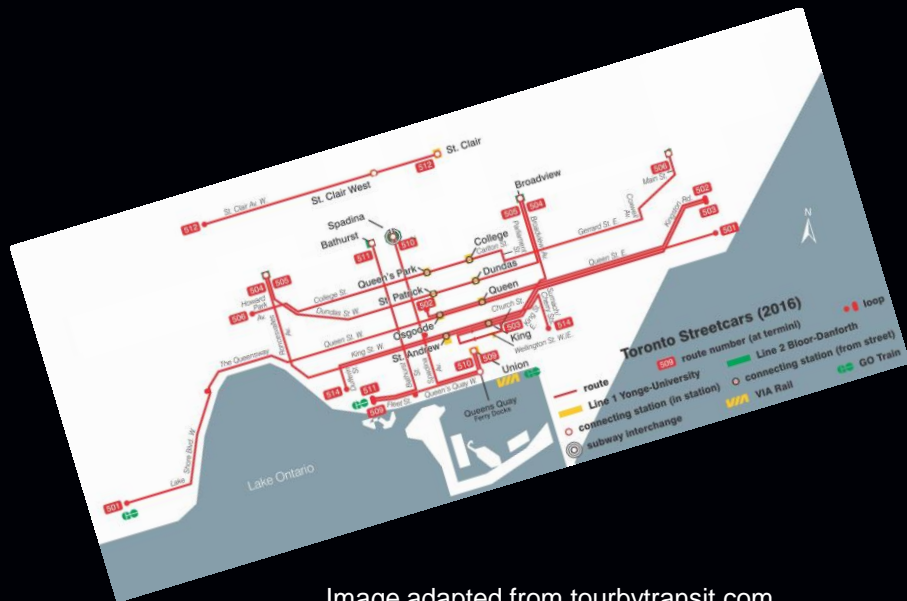
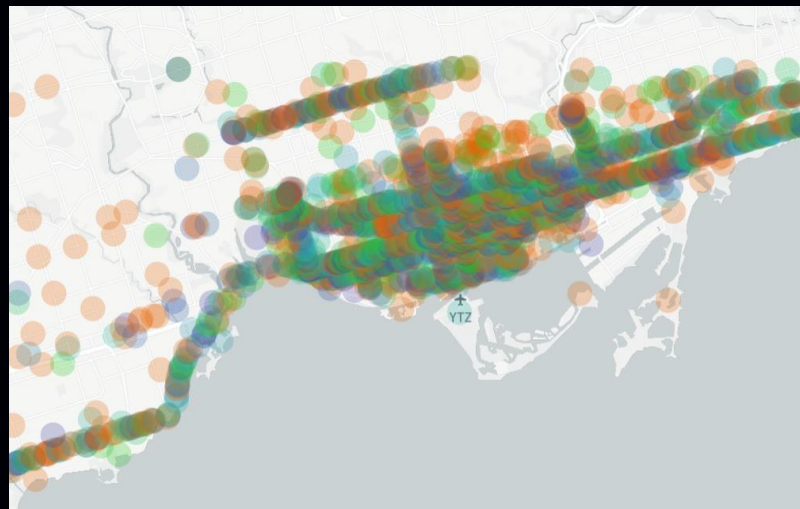


Image adapted from tourbytransit.com

The dataset: transformations

Report Date	Route	Time	Day	Location	Incident	Min Delay	Min Gap	Direction
2016-01-01 00:00:00	511	02:14:00	Friday	fleet st. and strachan	Mechanical	10.0	20.0	e
2016-01-01 00:00:00	301	02:22:00	Friday	queen st. west and roncesvalles	Mechanical	9.0	18.0	w
2016-01-01 00:00:00	301	03:28:00	Friday	lake shore blvd. and superior st.	Mechanical	20.0	40.0	e
2016-01-01 00:00:00	505	15:42:00	Friday	broadview station loop	Investigation	4.0	10.0	w
2016-01-01 00:00:00	504	15:54:00	Friday	broadview and queen	Mechanical	6.0	12.0	e



Report Date	Route	Time	Day	Location	Incident	Min Delay	Min Gap	Direction	...	longitude	x	y	z
2017-09-04 00:00:00	13	18:52:00	1	old weston road and st.clair	[1]	27.0	54.0	5	...	-79.463024	0.925917	1.065716	0.942998
2016-07-01 00:00:00	4	21:18:00	0	connaught and queen	[1]	8.0	16.0	5	...	-79.322360	1.085382	0.939566	0.968273
2016-03-24 00:00:00	7	06:48:00	4	boustead and roncesvalles	[1]	6.0	12.0	4	...	-79.451723	0.933501	1.049577	1.002038
2015-08-10 00:00:00	13	13:56:00	1	roncesvalles yard	[5, 6, 7]	4.0	8.0	2	...	-79.449050	0.934383	1.044788	1.023267
2017-10-04 00:00:00	5	10:42:00	6	queen and river	[1]	10.0	20.0	5	...	-79.356530	1.045211	0.968827	0.990846

- Replace categorical values with numeric IDs
- Tokenize text values
- For training model, replace latitude and longitude values with x, y, z normalizations

The model: layers by category

- Categorize columns in the dataset:
 - Continuous: length of delay
 - Categorical: route, vehicle, direction, time of day, day of week
 - Text: description of incident
 - Spatial: location
- Automatically build a simple Keras model:
 - Build model by iterating through columns by type
 - layers for each column type have distinct characteristics (e.g. GNU for text, embeddings for text and categorical columns)
 - As long as columns are categorized correctly, the model automatically adapts to new schemas / additional columns

The model: layers by category

```
# define layers for categorical columns
for col in collist:
    catinputs[col] = Input(shape=[1],name=col)
    inputlayerlist.append(catinputs[col])
    embeddings[col] = (Embedding(max_dict[col],catemb) (catinputs[col]))
    # batchnorm all
    embeddings[col] = (BatchNormalization() (embeddings[col]))
    collistfix.append(embeddings[col])

# define layers for text columns
if includetext:
    for col in textcols:
        print("col",col)
        textinputs[col] = Input(shape=[X_train[col].shape[1]], name=col)
        print("text input shape",X_train[col].shape[1])
        inputlayerlist.append(textinputs[col])
        textembeddings[col] = (Embedding(textmax,textemb) (textinputs[col]))
        textembeddings[col] = (BatchNormalization() (textembeddings[col]))
        textembeddings[col] = Dropout(dropout_rate) ( GRU(16,kernel_regularizer=l2(12_lambda)) (textembeddings[col]))
        collistfix.append(textembeddings[col])
        print("max in the midst",np.max([np.max(train[col].max()), np.max(test[col].max())])+10)
    print("through loops for cols")

# define layers for continuous columns
for col in continuouscols:
    continputs[col] = Input(shape=[1],name=col)
    inputlayerlist.append(continputs[col])
```

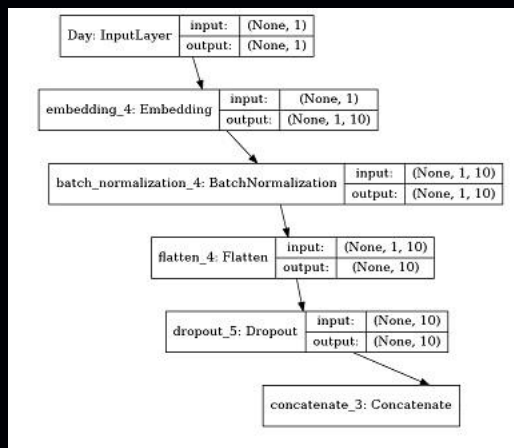
} Build layers for categorical columns

} Build layers for text columns

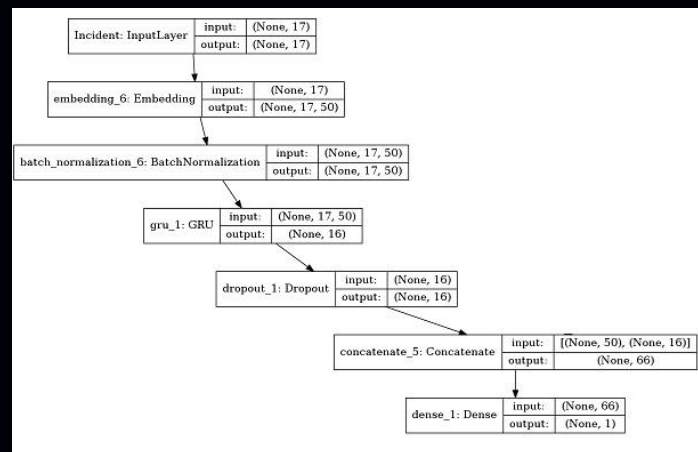
} Build layers for continuous columns

The model: layers by category

Categorical



Text



The model: parameters

- Hyperparameters:
 - `learning_rate = 0.001`
 - `dropout_rate = 0.0003`
 - `l2_lambda = 0.0003`
 - `loss_func = "binary_crossentropy"`
- Model-specific parameters:
 - `targetthresh = 5.0` *threshold for delay duration*
 - `targetcontinuous = False` *switch for predicting delay duration (vs whether delays are greater than the threshold)*

The model: accuracy

- Experiment 1: predict whether a delay over a given threshold (e.g. 5 or 10 minutes) occurs:
 - High-water mark accuracy: 75%
- Experiment 2: predict whether a delay occurs given all features:
 - Generate augmented dataset with entries for all hour / route / direction combinations
 - Predict whether a delay will occur for a given set of conditions
 - High-water mark accuracy: 74%

Lessons learned & best practices

- Lessons learned:
 - Use a methodical approach for hyperparameter selection
 - Plan for the pipeline from the start
 - Don't assume which features are categorical – evaluate first
 - Don't underestimate the effort required to convert free-form addresses to latitude and longitude
- Best practices:
 - Building the model from the start to make it easy to add and drop features
 - Saving intermediate datasets as pickled dataframes worked well for this size of dataset, saved time, and made it easier to keep the code organized
 - Once we had had longitude and latitude values they were very useful



Next steps

- Assess model on latest delay data from 2019
- Complete pipeline to simplify scoring
- Simple web deployment to allow scoring of single data points
- Complete deployment to allow scoring of batch data points
- Explore embeddings to find implicit groupings in the data

Code and data

Repo with code and associated material:

https://github.com/ryanmark1867/ai_conference_june_2019

Original data source: <https://www.toronto.ca/city-government/data-research-maps/open-data/open-data-catalogue/#e8f359f0-2f47-3058-bf64-6ec488de52da>

Additional related content: https://medium.com/@markryan_69718