The Alan Turing Institute

High performance, large scale regression

Alessandra Cabassi & Junyang Wang

The project

Alessandra Cabassi
University of Cambridge

Junyang Wang
Newcastle University

Sponsored by CRAY

- Close collaboration with Cray EMEA Research Lab.
- Use of the Urika®-GX agile analytics platform.

Goal

- Understand how well different, readily available, large-scale regression algorithms, software, and frameworks perform on distributed systems.
- Isolate computational and statistical performance issues.

Impact

- Provide guidelines to academics and industry on how to run regression algorithms on very large scale.
- Hopefully provide helpful insight to airplane companies.

Regression on Urika-GX
with Apache Spark and TensorFlow

Tools

Urika®-GX platform

- 14 nodes, 36 CPUs each.
- Ability to run multiple workloads concurrently, including Apache Spark™, Hadoop®.
- Interactive user interface with per node utilisation statistics.

Apache Spark™

- Fast, in-memory data processing engine with APIs to allow data workers to efficiently execute streaming, ML or SQL workloads.
- Libraries like MLib which includes popular statistics and ML methods
- Compatible with Apache Mesos, an architecture which manages nodes in a cluster.

TensorFlow™

- Open source software library for high performance numerical computation.
- Easy deployment of computation across a variety of platforms.
- APIs available in several languages.

TensorFlow, the TensorFlow logo and any related marks are trademarks of Google Inc.

Apache Spark

- Settings and resources in the cluster can be managed through Mesos inside Spark via Python API.

- Key settings are:

  Maximum number of cores ($coremax$)
  Number of cores per executor ($coresper$)
  Memory per executor ($memper$)

- Urika has 36 CPUs cores per node, 9 available nodes, and about 200gb of RAM available on each node. This makes for 324 total cores that are available.

- For fastest performance use all 324 cores, but if total memory exceeds around 1800gb Spark will reduce the number of cores as there isn't enough memory. So memory per executor should be kept below 200gb.

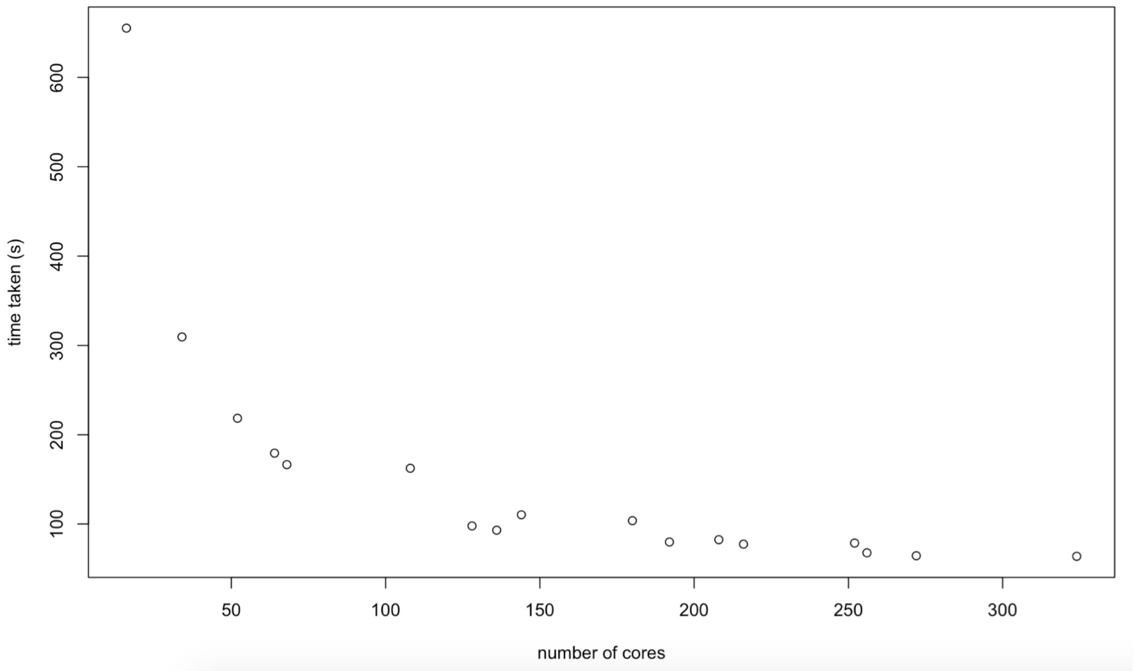
- In general $\frac{\min(coresmax,324)}{coresper} memper \leq 1800$

Example benchmark: Linear regression

- Bench marks were conducted under two different settings, large n small p, and small p large n. Tests were run under max cores (324).

n	p	Time(s)
10^8	100	222.6732
10^9	100	1990.064
10^10	100	32018.55
100	10^5	2434.37
100	10^6	3717.93

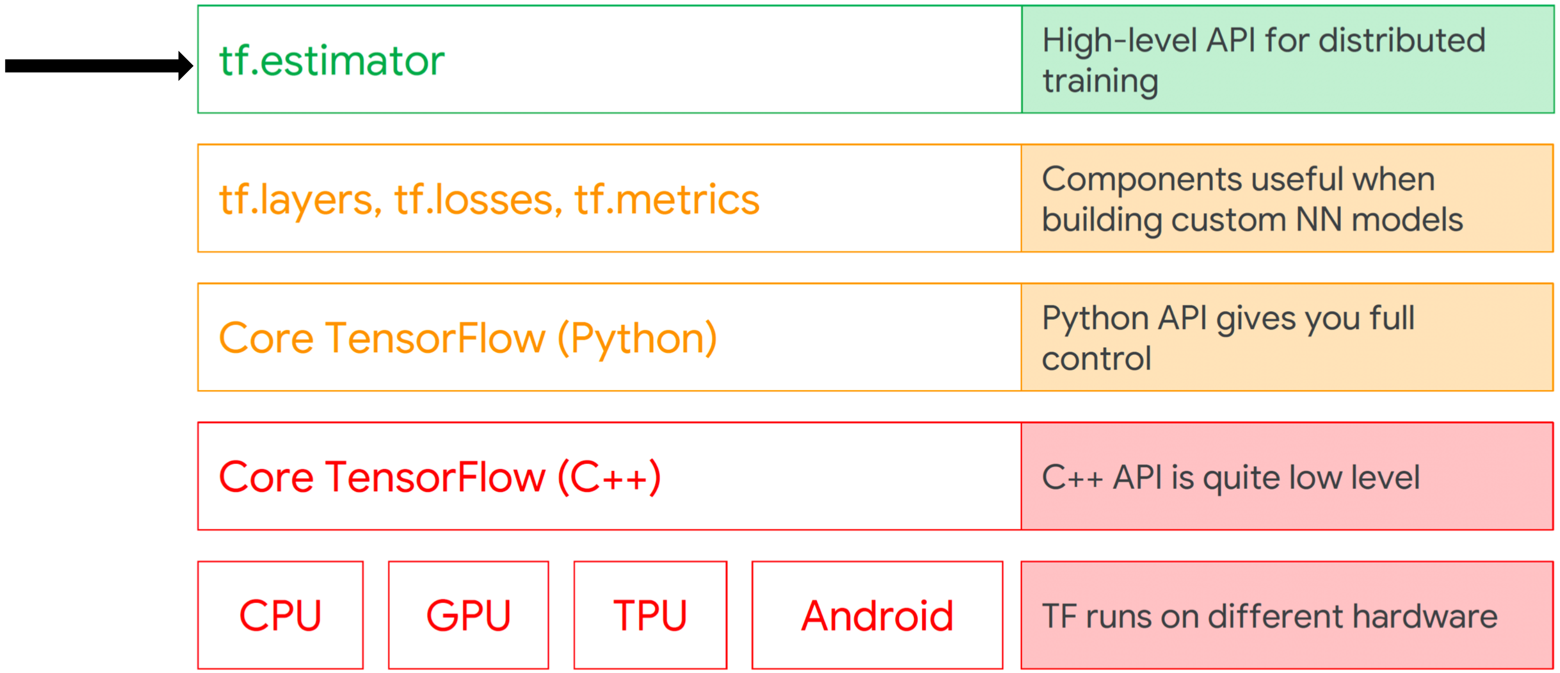
Computation time vs number of cores



Data used here has 10^8 rows and 10 columns.

100gb memory per executor

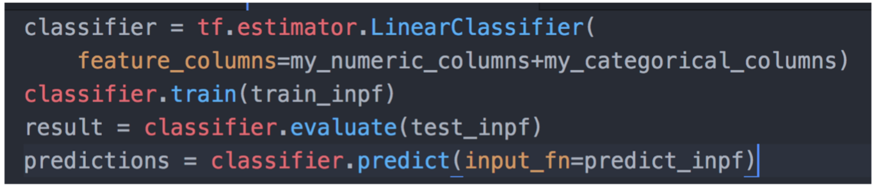
TensorFlow



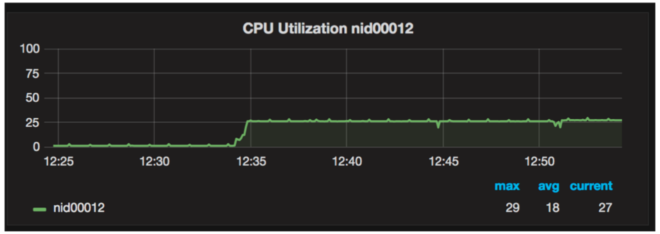
tf.estimator	High-level API for distributed training
tf.layers, tf.losses, tf.metrics	Components useful when building custom NN models
Core TensorFlow (Python)	Python API gives you full control
Core TensorFlow (C++)	C++ API is quite low level
CPU, GPU, TPU, Android	TF runs on different hardware

TensorFlow

- Linear and logistic regression already implemented as Estimators.
- Main functions such as training, testing, etc. ready to use.
- Many variants of SGD available.
- No need to load data in memory:
  Easy to load batches of data from file.
- Same sequential code can be used on any set of devices including CPUs, GPUs, TPUs.

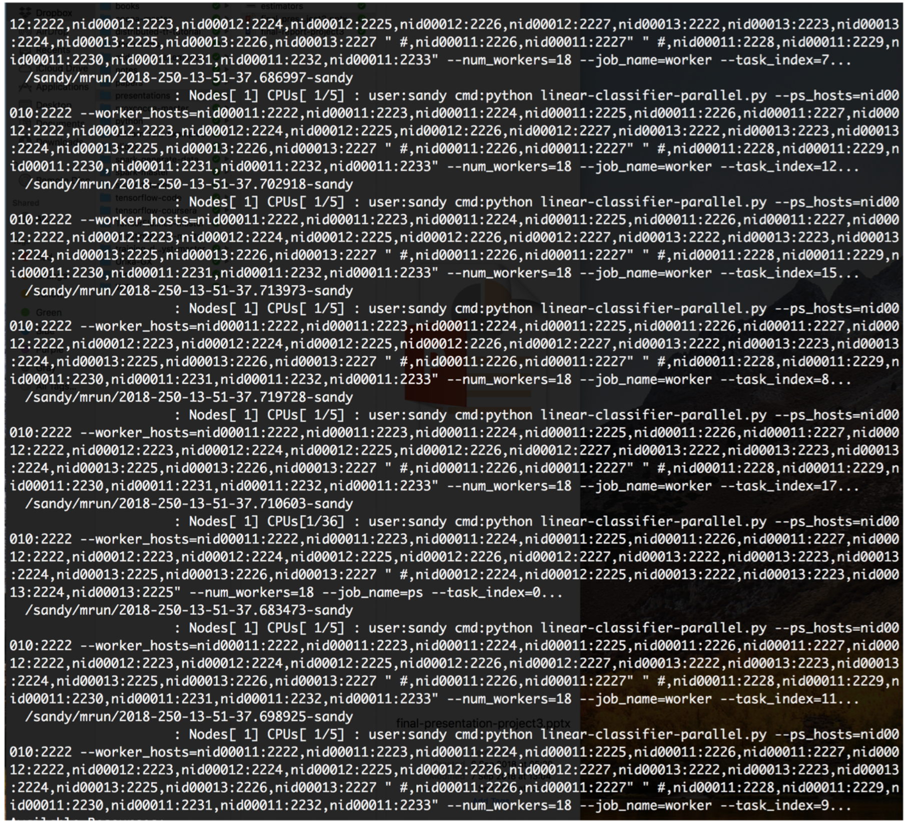


```
classifier = tf.estimator.LinearClassifier(
    feature_columns=my_numeric_columns+my_categorical_columns)
classifier.train(train_inpf)
result = classifier.evaluate(test_inpf)
predictions = classifier.predict(input_fn=predict_inpf)
```

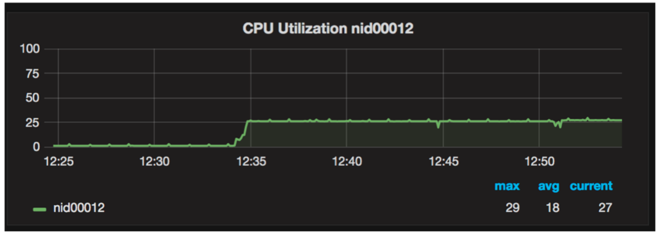


- Lacks some basic functionalities, such as retrieving regression coefficients and normalising data.
- Computation time explodes for increasing values of p.
- Not very performant on CPUs:
  Need to start multiple jobs on the same node to exploit computational power.
- Each parameter server and cluster must be started separately and lengthy cluster settings must be specified each time.

TensorFlow



```
12:2222,nid00012:2223,nid00012:2224,nid00012:2225,nid00012:2226,nid00012:2227,nid00013:2222,nid00013:2223,nid00013
:2224,nid00013:2225,nid00013:2226,nid00013:2227 " #,nid00011:2226,nid00011:2227" " #,nid00011:2228,nid00011:2229,n
id00011:2230,nid00011:2231,nid00011:2232,nid00011:2233" --num_workers=18 --job_name=worker --task_index=7...
  /sandy/mrun/2018-250-13-51-37.686997-sandy
                    : Nodes[ 1] CPUs[ 1/5] : user:sandy cmd:python linear-classifier-parallel.py --ps_hosts=nid00
010:2222 --worker_hosts=nid00011:2222,nid00011:2223,nid00011:2224,nid00011:2225,nid00011:2226,nid00011:2227,nid000
12:2222,nid00012:2223,nid00012:2224,nid00012:2225,nid00012:2226,nid00012:2227,nid00013:2222,nid00013:2223,nid00013
:2224,nid00013:2225,nid00013:2226,nid00013:2227 " #,nid00011:2226,nid00011:2227" " #,nid00011:2228,nid00011:2229,n
id00011:2230,nid00011:2231,nid00011:2232,nid00011:2233" --num_workers=18 --job_name=worker --task_index=12...
  /sandy/mrun/2018-250-13-51-37.702918-sandy
                    : Nodes[ 1] CPUs[ 1/5] : user:sandy cmd:python linear-classifier-parallel.py --ps_hosts=nid00
010:2222 --worker_hosts=nid00011:2222,nid00011:2223,nid00011:2224,nid00011:2225,nid00011:2226,nid00011:2227,nid000
12:2222,nid00012:2223,nid00012:2224,nid00012:2225,nid00012:2226,nid00012:2227,nid00013:2222,nid00013:2223,nid00013
:2224,nid00013:2225,nid00013:2226,nid00013:2227 " #,nid00011:2226,nid00011:2227" " #,nid00011:2228,nid00011:2229,n
id00011:2230,nid00011:2231,nid00011:2232,nid00011:2233" --num_workers=18 --job_name=worker --task_index=15...
  /sandy/mrun/2018-250-13-51-37.713973-sandy
                    : Nodes[ 1] CPUs[ 1/5] : user:sandy cmd:python linear-classifier-parallel.py --ps_hosts=nid00
010:2222 --worker_hosts=nid00011:2222,nid00011:2223,nid00011:2224,nid00011:2225,nid00011:2226,nid00011:2227,nid000
12:2222,nid00012:2223,nid00012:2224,nid00012:2225,nid00012:2226,nid00012:2227,nid00013:2222,nid00013:2223,nid00013
:2224,nid00013:2225,nid00013:2226,nid00013:2227 " #,nid00011:2226,nid00011:2227" " #,nid00011:2228,nid00011:2229,n
id00011:2230,nid00011:2231,nid00011:2232,nid00011:2233" --num_workers=18 --job_name=worker --task_index=8...
  /sandy/mrun/2018-250-13-51-37.719728-sandy
                    : Nodes[ 1] CPUs[ 1/5] : user:sandy cmd:python linear-classifier-parallel.py --ps_hosts=nid00
010:2222 --worker_hosts=nid00011:2222,nid00011:2223,nid00011:2224,nid00011:2225,nid00011:2226,nid00011:2227,nid000
12:2222,nid00012:2223,nid00012:2224,nid00012:2225,nid00012:2226,nid00012:2227,nid00013:2222,nid00013:2223,nid00013
:2224,nid00013:2225,nid00013:2226,nid00013:2227 " #,nid00011:2226,nid00011:2227" " #,nid00011:2228,nid00011:2229,n
id00011:2230,nid00011:2231,nid00011:2232,nid00011:2233" --num_workers=18 --job_name=worker --task_index=17...
  /sandy/mrun/2018-250-13-51-37.710603-sandy
                    : Nodes[ 1] CPUs[1/36] : user:sandy cmd:python linear-classifier-parallel.py --ps_hosts=nid00
010:2222 --worker_hosts=nid00011:2222,nid00011:2223,nid00011:2224,nid00011:2225,nid00011:2226,nid00011:2227,nid000
12:2222,nid00012:2223,nid00012:2224,nid00012:2225,nid00012:2226,nid00012:2227,nid00013:2222,nid00013:2223,nid00013
:2224,nid00013:2225,nid00013:2226,nid00013:2227 " #,nid00012:2224,nid00012:2225,nid00013:2222,nid00013:2223,nid000
13:2224,nid00013:2225" --num_workers=18 --job_name=ps --task_index=0...
  /sandy/mrun/2018-250-13-51-37.683473-sandy
                    : Nodes[ 1] CPUs[ 1/5] : user:sandy cmd:python linear-classifier-parallel.py --ps_hosts=nid00
010:2222 --worker_hosts=nid00011:2222,nid00011:2223,nid00011:2224,nid00011:2225,nid00011:2226,nid00011:2227,nid000
12:2222,nid00012:2223,nid00012:2224,nid00012:2225,nid00012:2226,nid00012:2227,nid00013:2222,nid00013:2223,nid00013
:2224,nid00013:2225,nid00013:2226,nid00013:2227 " #,nid00011:2226,nid00011:2227" " #,nid00011:2228,nid00011:2229,n
id00011:2230,nid00011:2231,nid00011:2232,nid00011:2233" --num_workers=18 --job_name=worker --task_index=11...
  /sandy/mrun/2018-250-13-51-37.698925-sandy
                    : Nodes[ 1] CPUs[ 1/5] : user:sandy cmd:python linear-classifier-parallel.py --ps_hosts=nid00
010:2222 --worker_hosts=nid00011:2222,nid00011:2223,nid00011:2224,nid00011:2225,nid00011:2226,nid00011:2227,nid000
12:2222,nid00012:2223,nid00012:2224,nid00012:2225,nid00012:2226,nid00012:2227,nid00013:2222,nid00013:2223,nid00013
:2224,nid00013:2225,nid00013:2226,nid00013:2227 " #,nid00011:2226,nid00011:2227" " #,nid00011:2228,nid00011:2229,n
id00011:2230,nid00011:2231,nid00011:2232,nid00011:2233" --num_workers=18 --job_name=worker --task_index=9...
```



- Lacks some basic functionalities, such as retrieving regression coefficients and normalising data.
- Computation time explodes for increasing values of p.
- Not very performant on CPUs:
  Need to start multiple jobs on the same node to exploit computational power.
- Each parameter server and cluster must be started separately and lengthy cluster settings must be specified each time.

Airline data

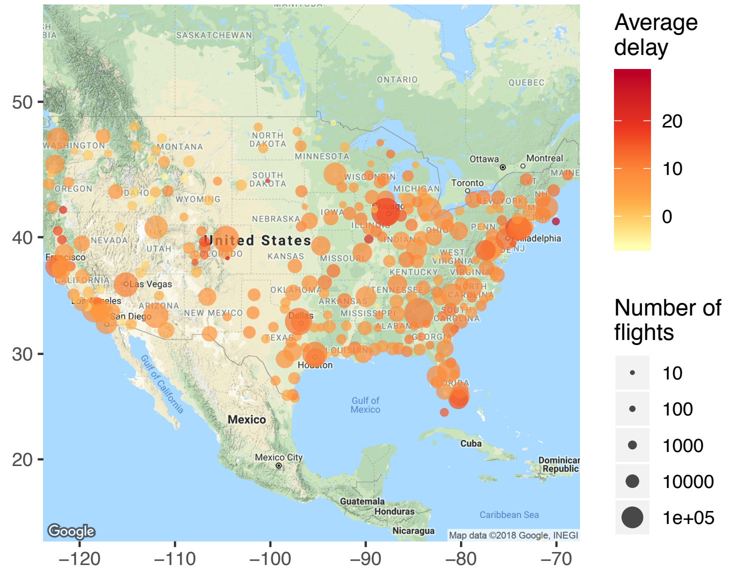
Airline data

- Flight arrival and departure details for all commercial flights within the USA, from October 1987 to April 2008
- 120 million records in total.
- Information available for each flight: date, arrival and departure times, flight times, origin and destination, carrier.

Questions

- When is the best time of day/day of week/time of year to fly to minimise delays?
- Are some routes more affected by delays?

Number of departures in 2008 and average delay on departure



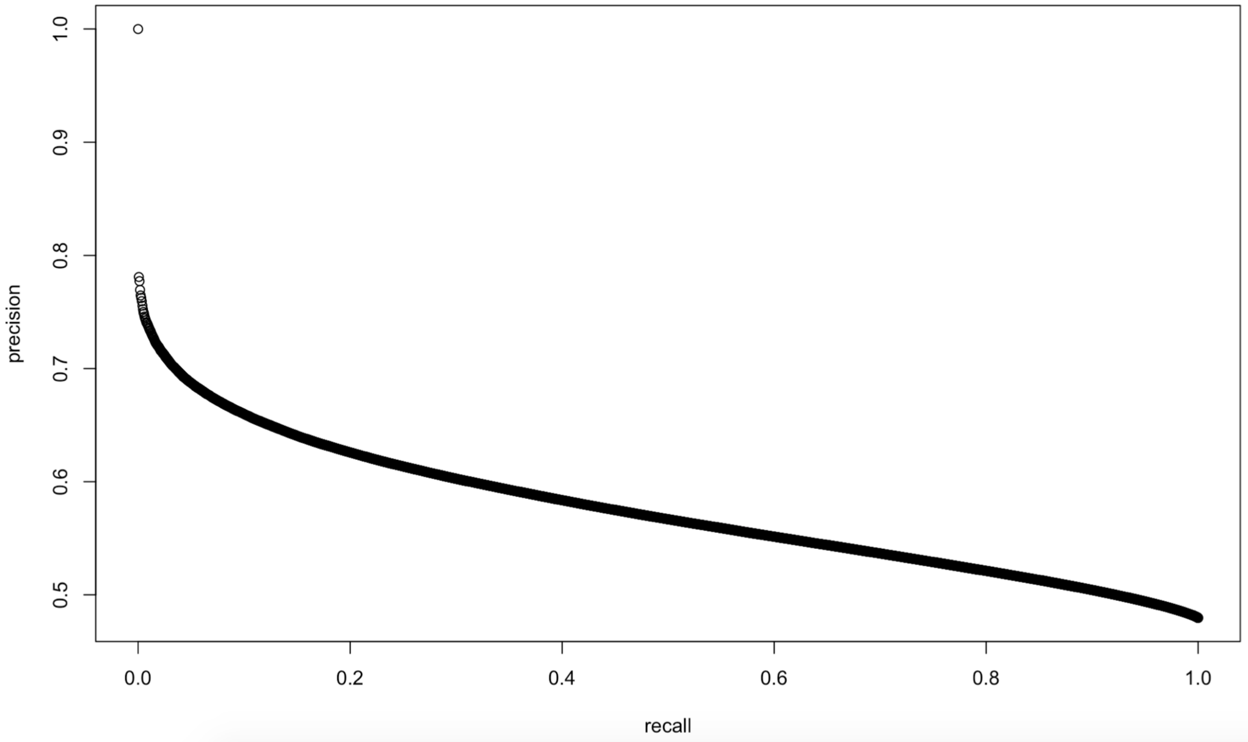
Airline data

- Logistic regression: Predict whether a flight is delayed or not.
- Training set is data recorded between 1987-2007 (~112m rows), test set data recorded in 2008 (~8m rows). Takes about 15 minutes to train (with LASSO penalisation) on Spark.
- After some feature engineering and expanding out factor variables, data contained 779 features or columns.
- Model formula is: $Delay \sim CRSDepTime + CRSArrTime + CRSElapsedTime + Month + DayofMonth + DayofWeek + UniqueCarrierID + Origin + Destination + Cos(CRSDepTime) + Sin(CRSDepTime) + Cos(CRSArrTime) + Sin(CRSArrTime) + Distance + Distance^2 + Distance^3 + Year + Year^2 + Year^3 + Year^4 + Year^5 + Year^6$

Model coefficients and findings

- Months most likely to contain delays are: December, January, June, July. September and May least likely
- Friday, Thursday are more likely to contain delays. Saturday and Sunday least likely.
- Flights originating from Atlanta International Airport, Detroit Metro Airport, Chicago O'Hare International Airport are most likely to cause delays. Palmdale Regional Airport and Lihue Airport least likely.
- Flights arriving at Newark Liberty International Airport, Seattle–Tacoma International Airport, Atlanta International Airport are most likely to cause delays. Guam International Airport and Palmdale Regional Airport.
- Delays most likely between 15:00-20:00, least likely between 4:00-9:00
- Delays are less likely to happen in recent years, and more likely to happen the longer the flight.

Precision and Recall



Precision and Recall curve of model applied to the 2008 data

Area under the curve is 0.58

Wrapping up

Final remarks

Output

1. Blogpost explaining how to do linear and logistic regression on Urika containing:
   - guidelines and example benchmarks;
   - airline data used as practical example.
2. Report of the literature review and code available on GitHub.
3. Set of suggestions/feature requests for TensorFlow.

Challenges

- Some libraries are less mature/robust than is often advertised.
- Frequent updates and changes to the framework
- No backward compatibility.

Thank you!

To all our supervisors;

To Adrian Tate, Nina Mujkanovic, and Alessandro Rigazzi from CERL;

To the CRAY support team;

To Per Nyberg, VP Artificial Intelligence at CRAY;

The ATI student services and helpdesk.

The Alan Turing Institute

turing.ac.uk
@turinginst