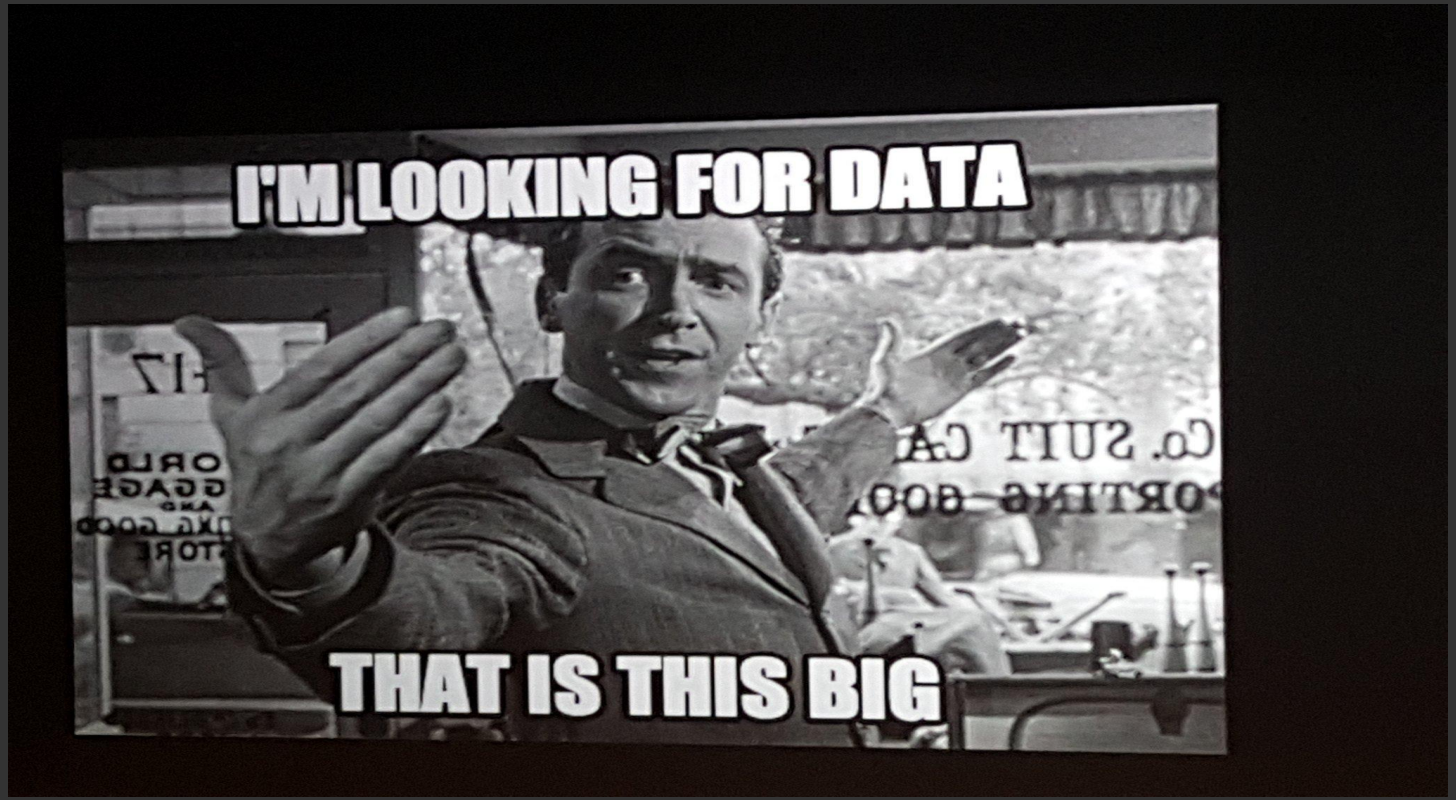
Deploying Large (Spark) ML models and scoring in near-real time @scale

Running the last mile of the Data Science journey

Subhojit Banerjee
DataScientist/DataEngineer
@subbubanerjee
https://medium.com/@subhojit20_27731

Big Data* is not easy



Big Data* is not easy

Gartner found just 14% of companies surveyed had big data projects in production in 2015 and unchanged from the year before and slowly inching towards 15% in year 2016

**Big Data - my apologies for using the term*

**Big Data ~ Large unstructured and structured data that can't fit on a single node*

Big Data is not easy

- Cloudera: $261M in revenue, $187M in losses (down from $205M the year before, the only company to narrow its loss)
- Hortonworks: $184M in revenue, $251M in losses (up from $180M the year before)
- Alteryx: $85M in revenue, $24M in losses (up from $21M)
- Splunk: $950M in revenue, $355M in losses (up from $279M)
- Tableau: $827M in revenue, $144M in losses (up from $84M)

Big Data is not easy

Big Data is not easy

Gartner's top obstacles for big data success were:

- Determining how to get value from Big data
- Most companies are not set up culturally or organizationally to succeed, wishing themselves agile and hoping data silos will disappear "magically"
- Technical expertise is lacking in areas like agile model development and deployment to provide quick turnarounds.

If only people were as malleable as data

Big Data is not easy

This talk is to address the last problem - “Democratising” large scale machine learning model deployment and model scoring (in near real time)

Key Takeaways

- A better than random model has revenue generating potential from day one.Hence try to build a robust data science pipeline where models can be quickly iterated on.
- Pyspark models ***CAN** be deployed in a Scala Pipeline and vice versa.
- Spark Models **CAN** be scored in “near” real time using external tools without paying the spark “distributed tax” i.e. latencies associated with spark execution plan.
- Spark Models **CAN** be dockerized and hence can leverage on best practices refined out of years of software engineering i.e. CI/CD, A/B tests etc
- And all the above can be done in a matter of minutes i.e. model creation to exposing the model @scale as an API.
- GDPR compliant features **CAN** generate models with 0.88 ROC

** vanilla spark has model persistence available since 2.0.0*

01 Business case

02 First solution

03 Problems faced

04 Better solution

05 Demo

06 Conclusion

07 Questions

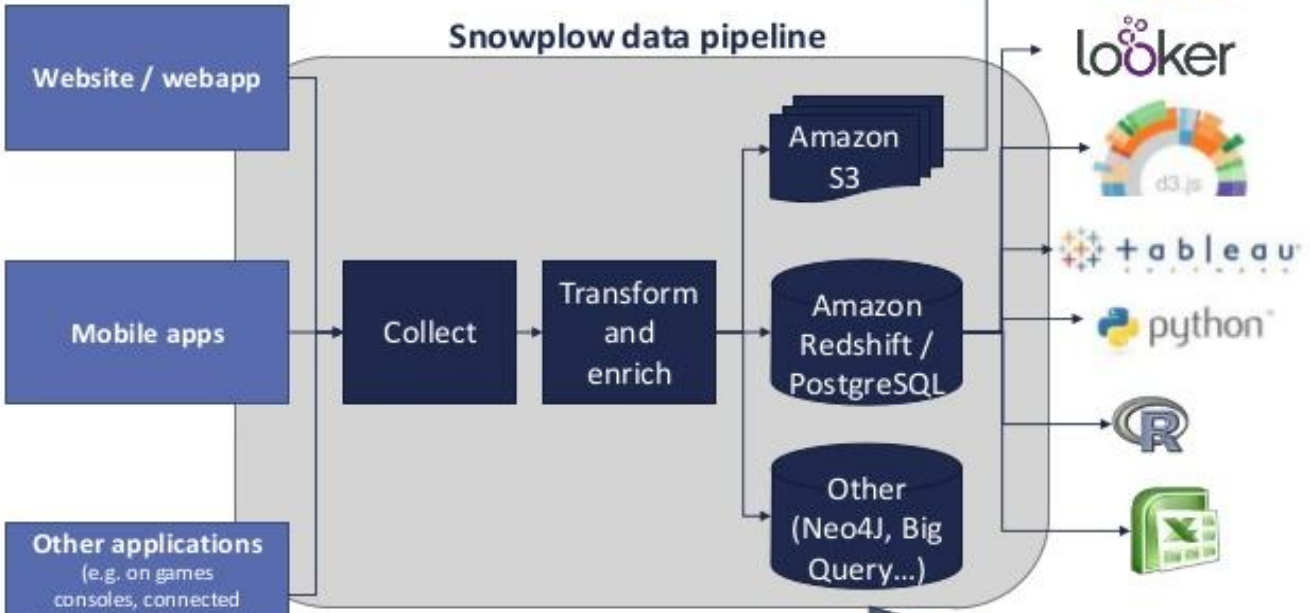
Business use case

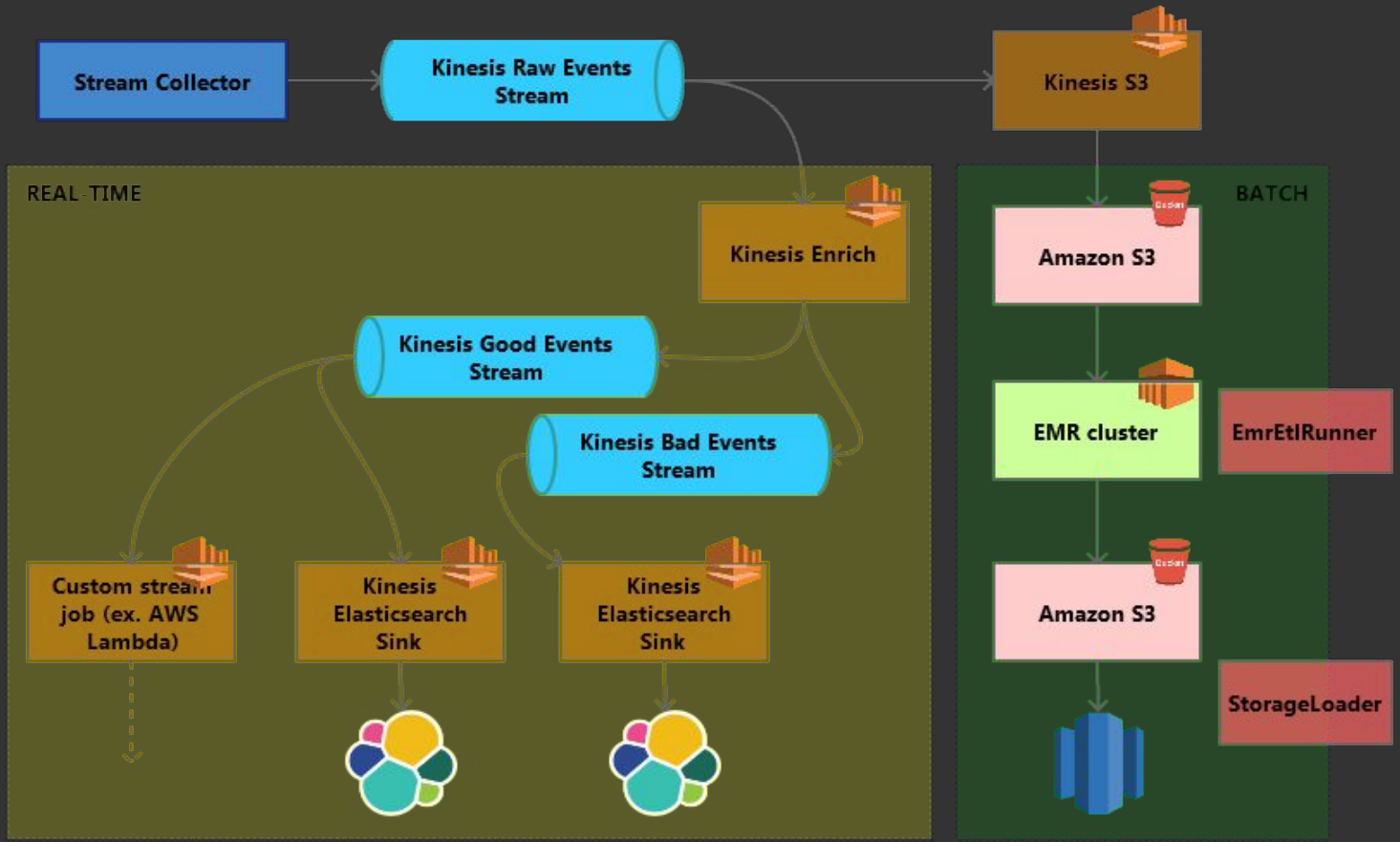
Is real time segmentation of the user into buy vs defer clusters using GDPR compliant features possible on the website?

Business use case

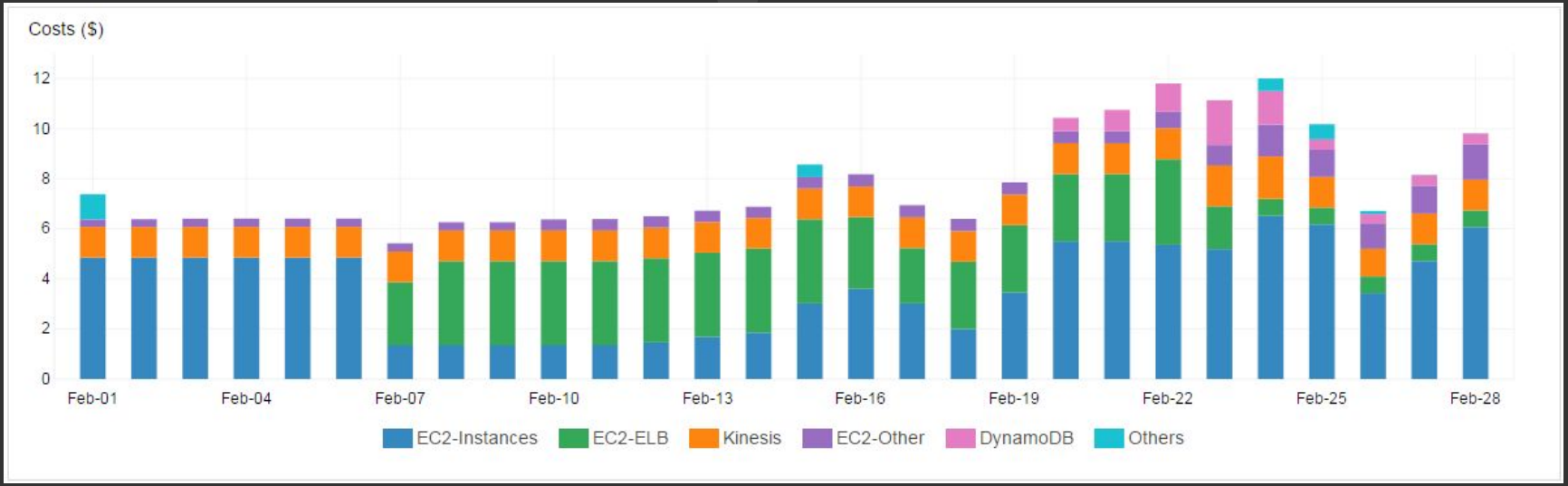
But first, we need to collect data to act on it in real time

Snowplow is an event data collection and warehousing platform




Stream Collector
Kinesis Raw Events Stream
Kinesis S3
REAL-TIME
Kinesis Enrich
BATCH
Amazon S3
Kinesis Good Events Stream
Kinesis Bad Events Stream
EMR cluster
EmrEtlRunner
Custom stream job (ex. AWS Lambda)
Kinesis Elasticsearch Sink
Kinesis Elasticsearch Sink
Amazon S3
StorageLoader

- ~23 million events per month
- ~140G of granular event data collected
- cost of 6 - 12 euro per day

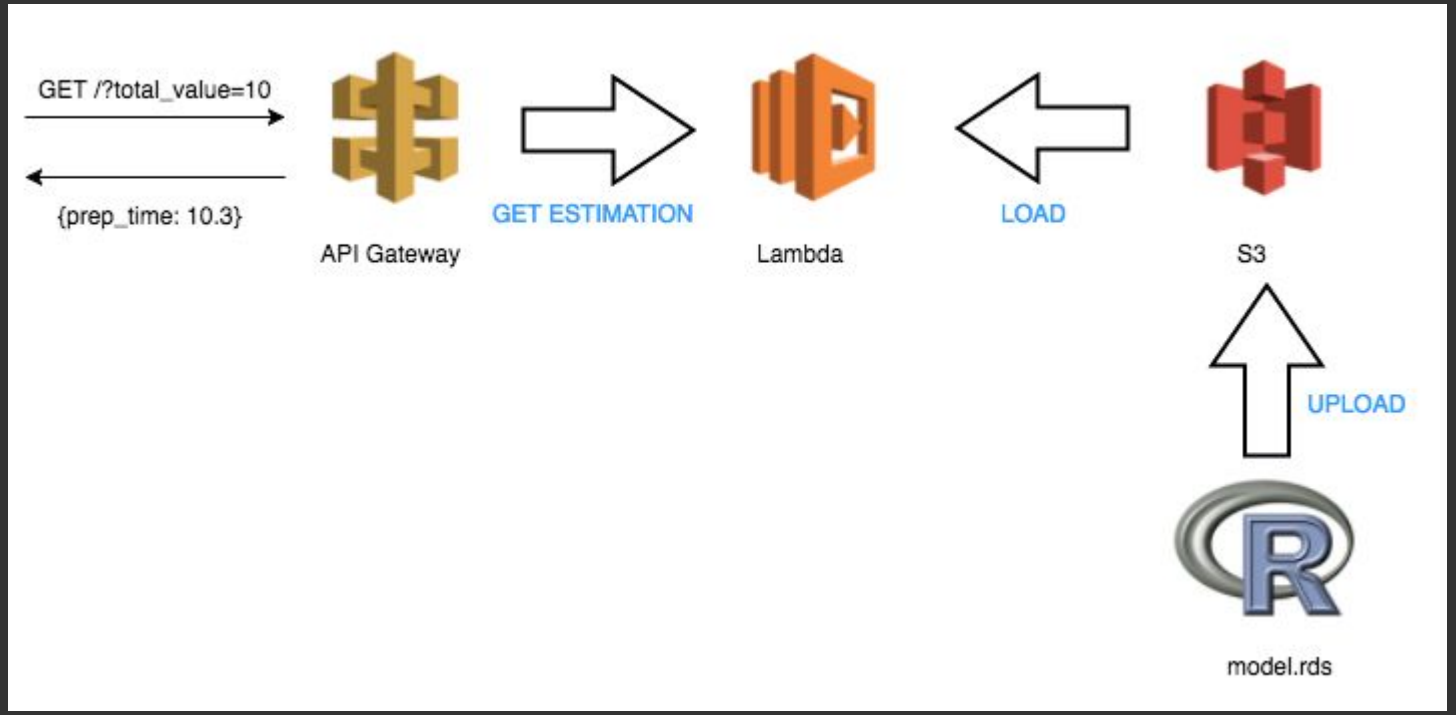


Business use case

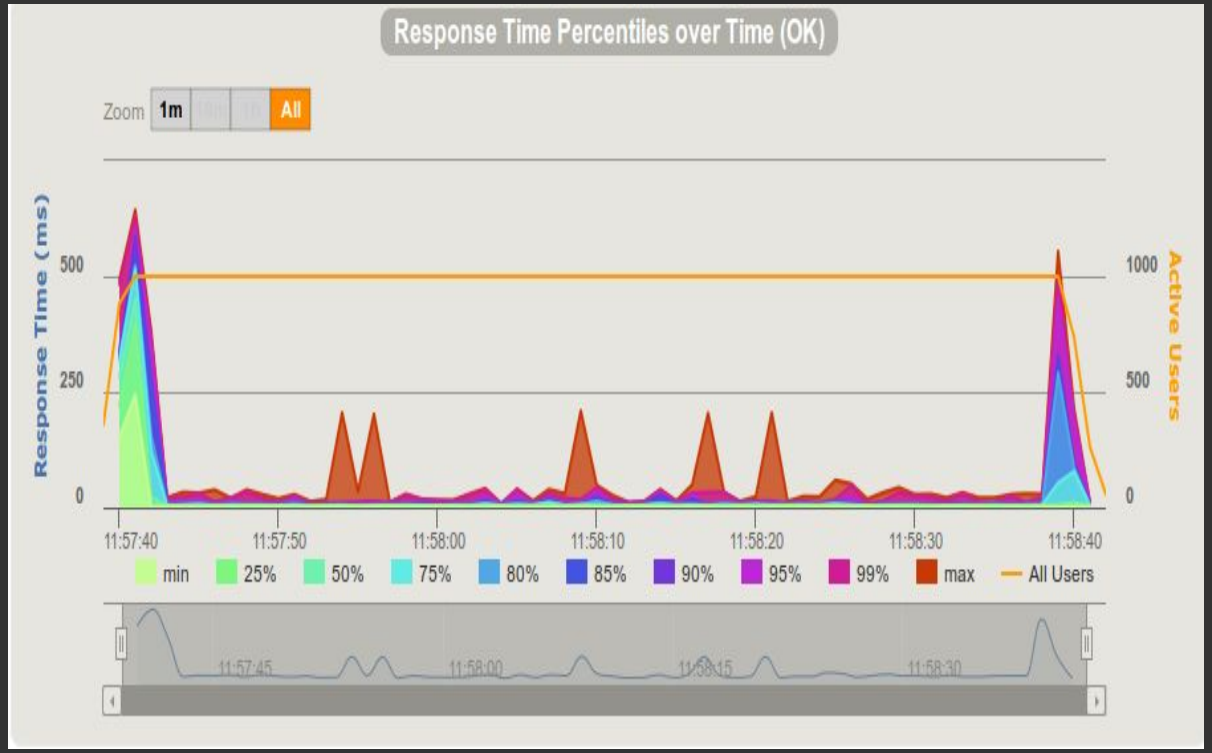
So now that we have the real time pipeline can we train and score our ML model ?

First Solution

Aws lambda based serverless architecture



First solution



STATISTICS

Executions

	Total	OK	KO
	29377	29377	0
Mean req/s	459.016	459.016	-

Response Time (ms)

	Total	OK	KO
Min	0	0	-
50th percentile	3	3	-
75th percentile	5	5	-
95th percentile	16	16	-
99th percentile	406	406	-
Max	644	644	-
Mean	12	12	-
Std Deviation	59	59	-

Problem faced

The first model was a markov chain on the sequence of webpages visited

Problems faced

- Had to hack core R libraries to bring down the 212 MB ML model library to fit the 50MB compressed AWS Lambda restriction
- R is not support by AWS lambda, hence had to hack through the restriction
- Every time front end would change, our models needs to change and old data cannot be used to retrain the model.

Learnings

The effort was totally worth it as first hand one could see the scale and economies of “serverless”

“If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility... The computer utility could become the basis of a new and important industry.” -John McCarthy(1961 @ MIT Centinnial)

Better solution

Requirements:

- Decrease the time for the models to move from notebook to production
- Super Scalable - Handling Terabytes should be a cakewalk
- Has to eliminate recoding of Spark feature pipelines and models from research to production i.e. my pyspark model should be deployable into a Scala pipeline with zero or minimal code changes
- Serving/inference has to be superfast for spark models

Better solution

Requirements technical analysis

- For model serving to be super fast, it was clear that inference needed to be outside the realms of Spark context.
- Has to be completely vendor neutral i.e. create a model in AWS and should be able to deploy the model in a pipeline on GCP and vice-versa.
- True test of serialization/portability: can I zip the model and send it to my coworker in an email

Why is Spark Slow in scoring

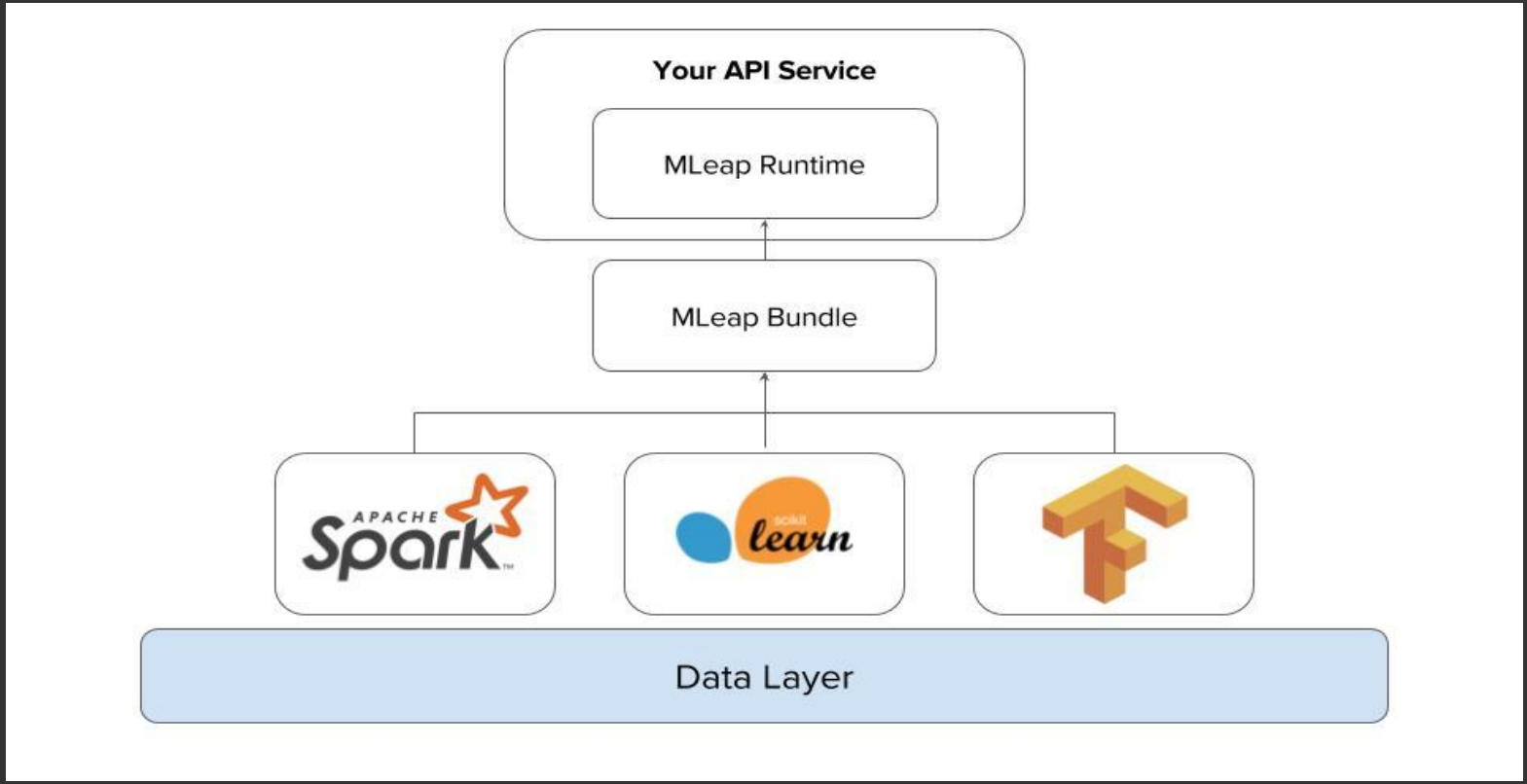
Mleap

First things first: Mleap was possible due to the good work of Hollin Wilkins and Mikhail Semeniuk

What is Mleap

- Is a common serialization format and execution engine for machine learning pipelines.
- Supports Spark,Scikit-learn, tensorflow
- Once you serialize the models you can run it in any platform … AWS, GCP ...
- For most parts you don’t have to modify any internal code except TF
- Is completely open source, so you can open the hood.

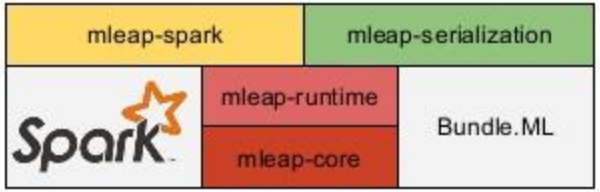
Mleap


Your API Service
MLeap Runtime
MLeap Bundle
APACHE Spark
scikit learn
Data Layer

Mleap

MLeap Components

- core - provides linear algebra system, regression models, and feature builders
- runtime - provides DataFrame-like "LeapFrame" and transformers for it
- spark - provides easy conversion from Spark transformers to MLeap transformers
- serialization - common serialization format for Spark and MLeap (Bundle.ML)



New features: expanded serialization formats to include both json and protobuf for large models (i.e. random forests with thousands of features)

Mleap runtime

- Provides leap frame, which stores data for transformation by mleap transformers
- Mleap transformers use mleap-core building blocks to transform leap frame
- Mleap transformers correspond one to one with spark transformers
- No dependencies on Spark

Mleap Serialization - Bundle.ml

- Provides common serialization for both Spark and Mleap
- 100% protobuf/JSON based for easy reading, compact data and portability
- Can be written to zip files and hence completely portable.

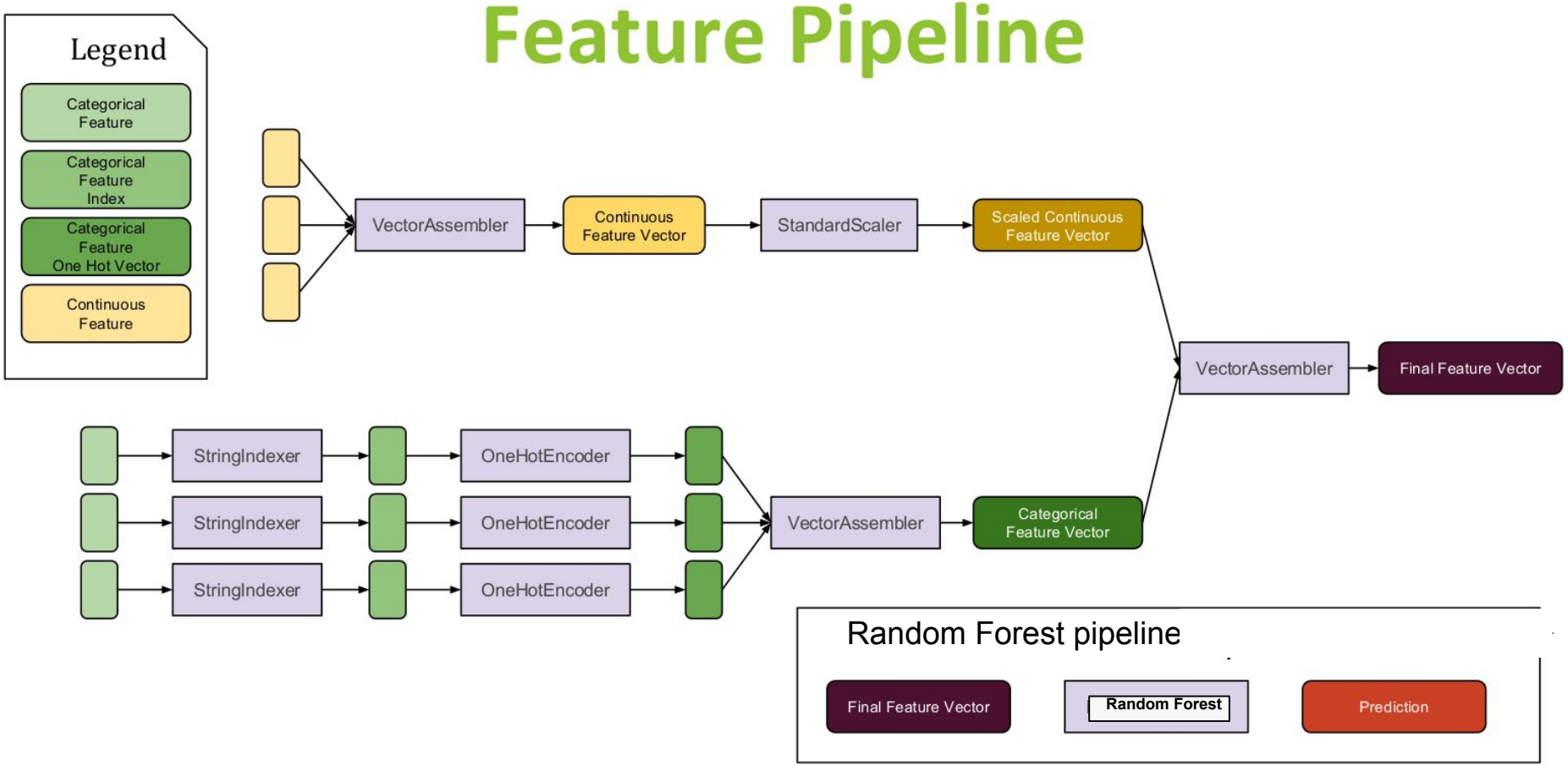
Mleap

MLeap Spark

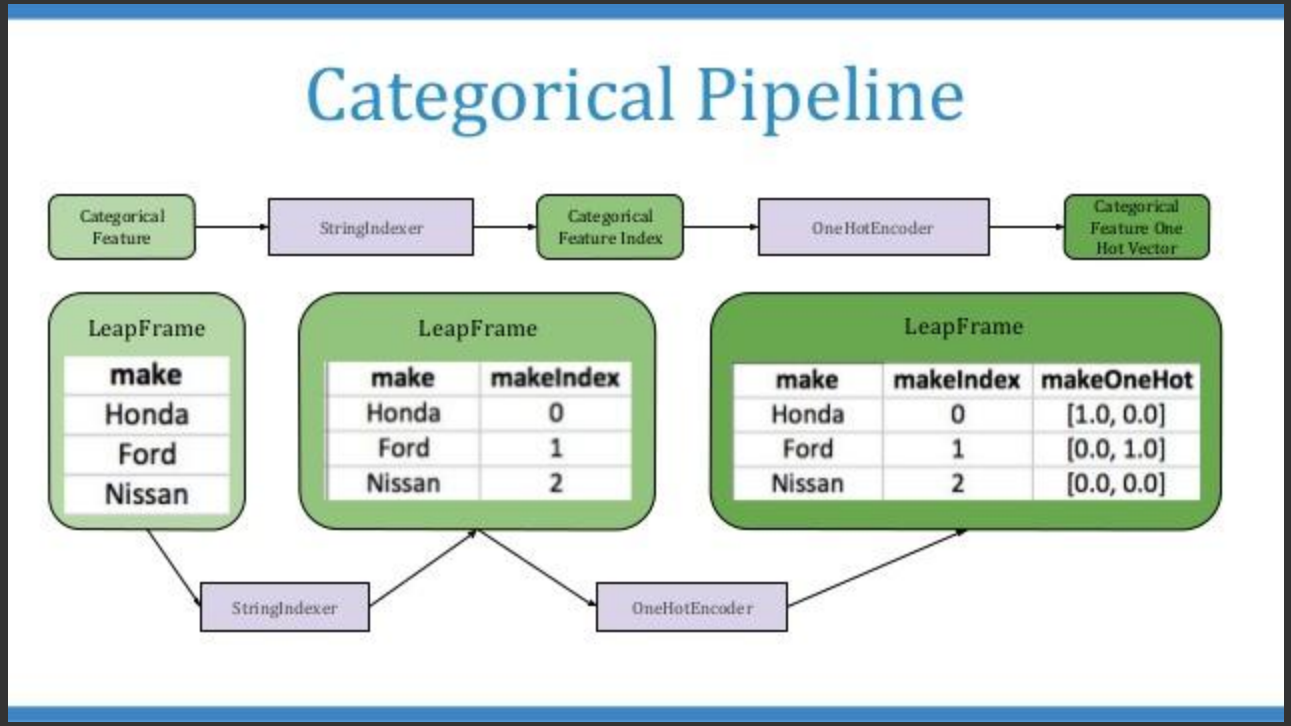
- Train an ML pipeline with Spark then export it to MLeap

- Execute an MLeap pipeline against a Spark DataFrame

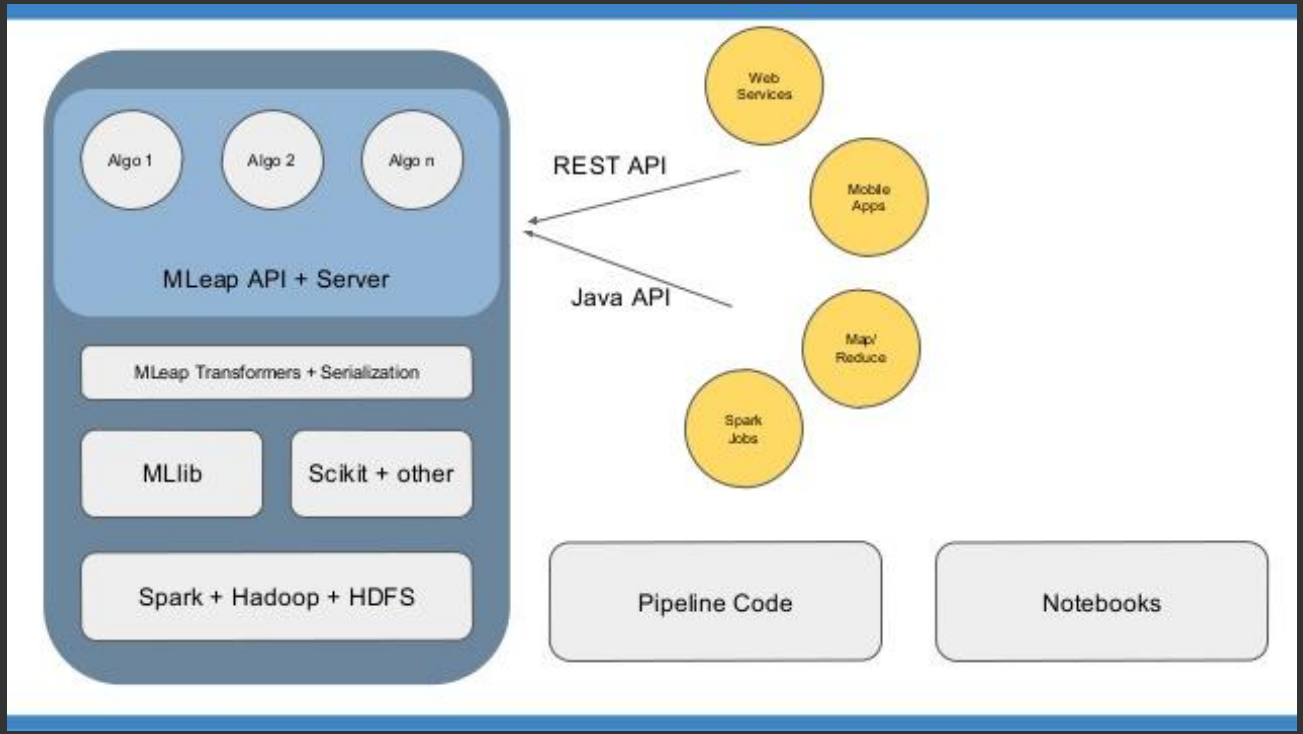
Feature Pipeline



Mleap


Categorical Pipeline
Categorical Feature
StringIndexer
Categorical Feature Index
OneHotEncoder
Categorical Feature One Hot Vector
LeapFrame
make
Honda
Ford
Nissan
LeapFrame
make
makeIndex
Honda
0
Ford
1
Nissan
2
LeapFrame
make
makeIndex
makeOneHot
Honda
0
[1.0, 0.0]
Ford
1
[0.0, 1.0]
Nissan
2
[0.0, 0.0]
StringIndexer
OneHotEncoder

Mleap


Algo 1
Algo 2
Algo n
MLeap API + Server
MLeap Transformers + Serialization
MLlib
Scikit + other
Spark + Hadoop + HDFS
REST API
Java API
Web Services
Mobile Apps
Map/ Reduce
Spark Jobs
Pipeline Code
Notebooks

Mleap available transformers

Support for all available Spark transformers:

- Binarizer,BucketedRandomLSH,Bucketizer,ChisqSelector,
- Vectorizer,DCT,ElementwiseProduct,HashingTermFrequency
- PCA, onehotencoder,Ngram,QuantileDiscretizer,Tokenizer
- etc

Mleap available transformers

Support for all available Spark Classification transformers:

- DecisionTreeClassifier,GradientBoostedTreeClassifier,
- LogisticRegression,LogisticRegressionCv,NaiveBayesClassifier,
- OnevsRest,RandomForestClassifier,SupportVectorMachines,
- MultilayerPerceptron

Mleap available transformers

Support for all available Spark Regression transformers:

- AFTSurvivalRegression,DecisionTreeRegression,
- GeneralizedLinearRegression,RandomForestRegression etc

Mleap available transformers

Support for all available Clustering transformers:

- BisectingKmeans,Kmean,LDA etc

Mleap unavailable transformers

- ALS, but its being worked on currently. Soon to be available

Mleap comparison

Name	Available languages (Creation)	Supports pre and post Transformation?	Supports Custom Transformation?	Supports Spark ML?	Scoring Latency	Reliable Documentation?	Active/larger community	Free?	Scores Multiple Point Simultaneously?
ScienceOPS	Python, R	Yes	Yes	Yes	Real Time	Yes	Yes	No	Yes
PMML	Python, R, Java	No	No	Yes	Depends on the scoring engine	Yes	Yes	Yes	No
PFA	Python, R, Java	Yes	No	Yes	Depends on the scoring engine	Yes	No	Yes	Yes
jPMML	Python, R, Java, Scala	Yes	No	Yes	Real Time	Yes	Yes	No	No
H2O	Python, R, Scala	No	No	Yes	Real Time	Yes	Yes	Yes	No
Aloha	Scala	Yes	Yes	No	Real Time	No	No	Yes	No
Embedded Spark	Python, R, Java, Scala	Yes	Yes	Yes	Not as fast (order of seconds)	Yes	Yes	Yes	Yes
MLeap	Python, Scala	Yes	Yes	Yes	Real Time	No	No	Yes	Yes

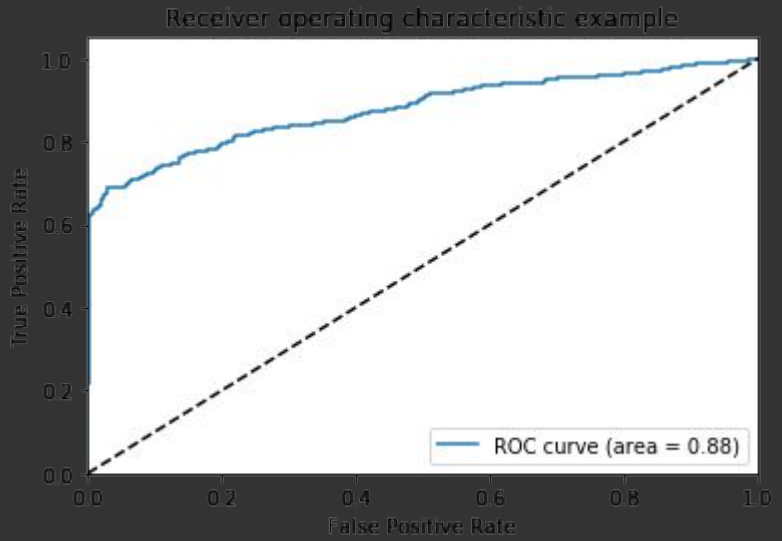
Demo

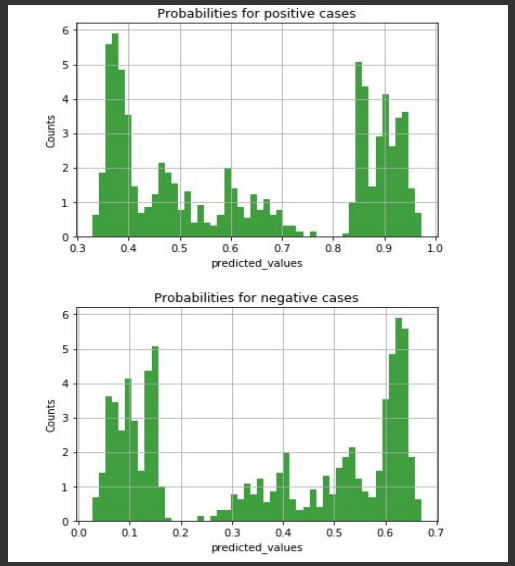
What will you see:

- Build a Pyspark model on Kaggle Data in a Jupyter notebook.
- Export the serialized model into a JSON and protobuf format (with just addition of a few libraries)
- Load the serialized pipeline into a docker container for near real model serving using a REST scala interface. (takes about 50ms for a model that spark serves in 1.5 seconds)
- Serve the Docker container from a Scalable AWS REST API in minutes

Results Achieved

Using GDPR compliant features .88 ROC and 94% on precision recall using a Pyspark model using a Single Random Forest Model (with no fancy model stacking) served in under ~50 ms on 100G of event data





Recsys15 Paper

Table 1: List of features used in models.

Group	Feature Description	Number/Type
Session features	Numerical time features of the start/end of the session (month, day, hour, minute, second, etc.)	2 × 7 Num
	Categorical time features of the start/end of the session (month, day, month-day, month-day-hour, hour, minute, weekday)	2 × 7 Categ
	Length of the session in seconds	1 Num
	Number of clicks, unique items, categories and item-category pairs in the session	4 Num
	Top 10 items and top 5 categories by the number of clicks in the session	15 Categ
	IDs of the first/last item clicked at least $k = 1, 2 \ldots, 6$ times in the session	12 Categ
	Vector of click numbers and total durations for 100 items and 50 categories that were the most popular in the whole training set	150 × 2 Num
Paired session-item features	Item ID	1 Categ
	Total and relative number of clicks in the session for the given item	2 Num
	Numerical time features of the first/last click on the item (month, day, hour, minute, second, etc.)	2 × 7 Num
	Categorical time features of the first/last click on the item (month, day, month-day, month-day-hour, hour, minute, weekday)	2 × 7 Categ
	Number of seconds between the first and the last click on the item	1 Num
	Total duration of the clicks on the item in the session and of all item's categories seen in the session	2 Num
	Number of unique categories seen in the session for a given item	1 Num

Conclusion

Dockerizing ML models can instantly leverage decades of effort spent on productionizing software engineering. You automatically get benefits of scaling, reliability due to CI/CD, fault tolerance, high availability, A/B testing, automation and a rich ecosystem of metrics that the new datascience discipline need not reinvent.

Conclusion

My company recently assembled a team to try to do “deep learning” that consisted of two PhD statisticians, a ML PhD and a PhD engineer.

- The statisticians worked their asses off trying to build clever approximations of distributions involved to better select features.
- The ML PhD built a cutting-edge adversarial approach to the problem and wrote it up in tensorflow.
- The engineer wrote “from sklearn.ensemble import random_forest” because that’s what she knew would probably do fine.

At the end of the day, the statisticians had wrong assumptions and the adversarial model didn’t have enough data to get anywhere, but the random forest kept on trucking.

References

- Mleap: https://github.com/combust/mleap
- Neils talk in Pydata 2017 about productionizing ML: https://www.youtube.com/watch?v=f3I0izerPvc
- https://github.com/romovpa/ydf-recsys2015-challenge