H2O Machine Learning and Kalman Filters for Machine Prognostics

Hank Roark

@hankroark

hank@h2o.ai

WHO AMI I

Lead, Customer Data Science @ H2O.ai

John Deere: Research, Software Product Development, High Tech Ventures

Lots of time dealing with data off of machines, equipment, satellites, weather, radar, hand sampled, and on.

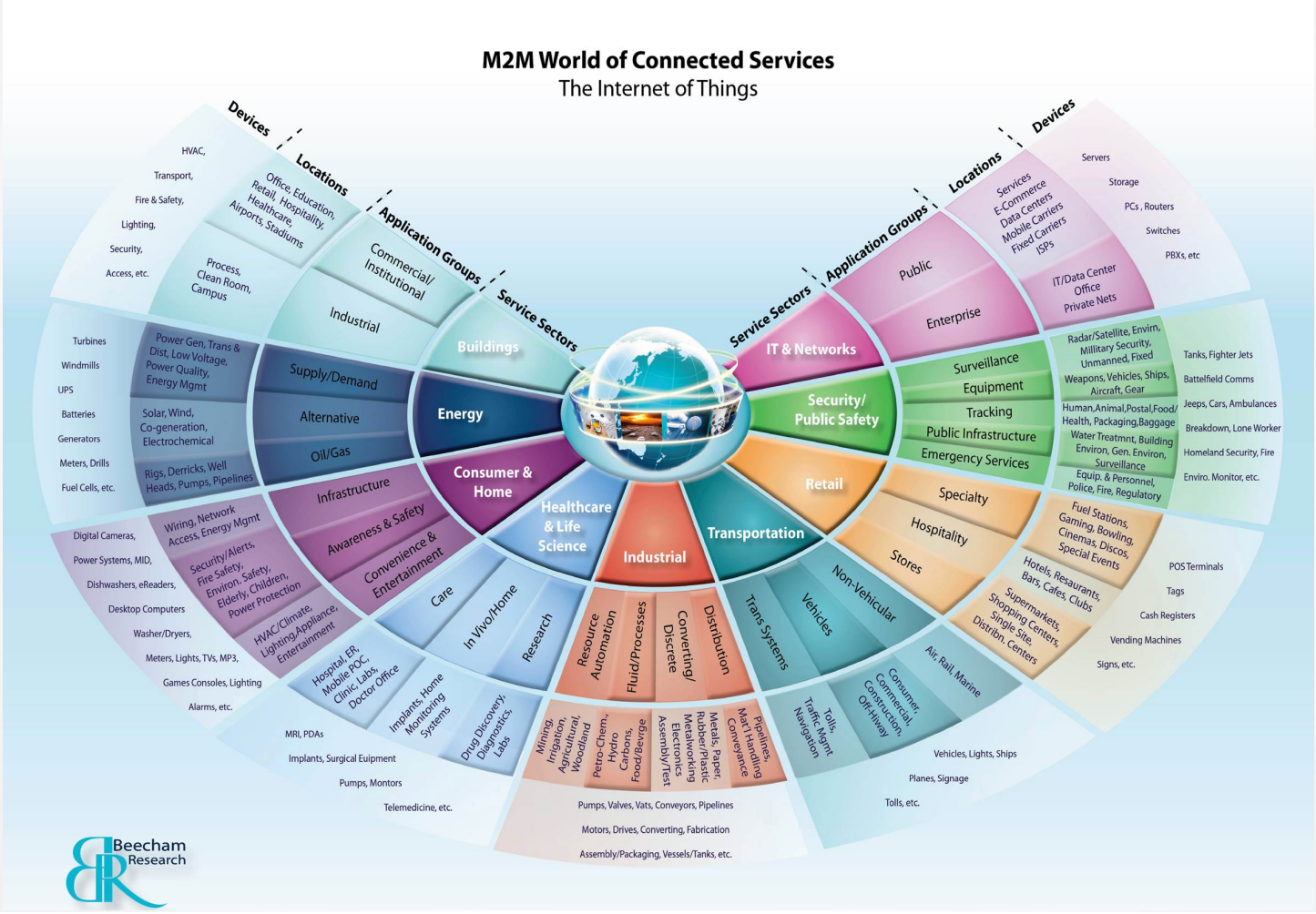
Geospatial, temporal / time series data almost all from sensors.

Previously at startups and consulting (Red Sky Interactive, Nuforia, NetExplorer, Perot Systems, a few of my own)

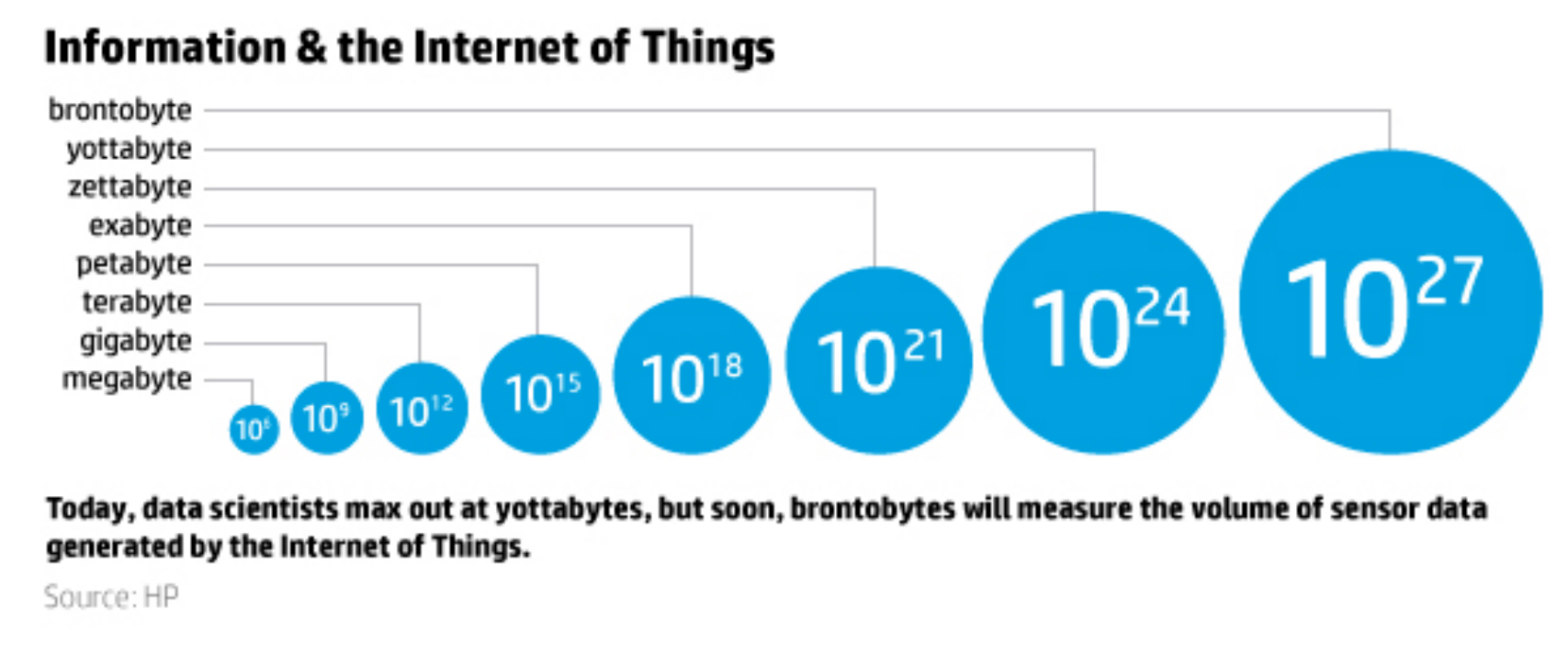
Engineering & Management MIT
Physics Georgia Tech

hank@h2oai.com
@hankroark
https://www.linkedin.com/in/hankroark

IF YOU ARE INTO DATA, THE IOT HAS IT



WHY THIS EXAMPLE?



GET READY FOR BRONTOBYTES!!

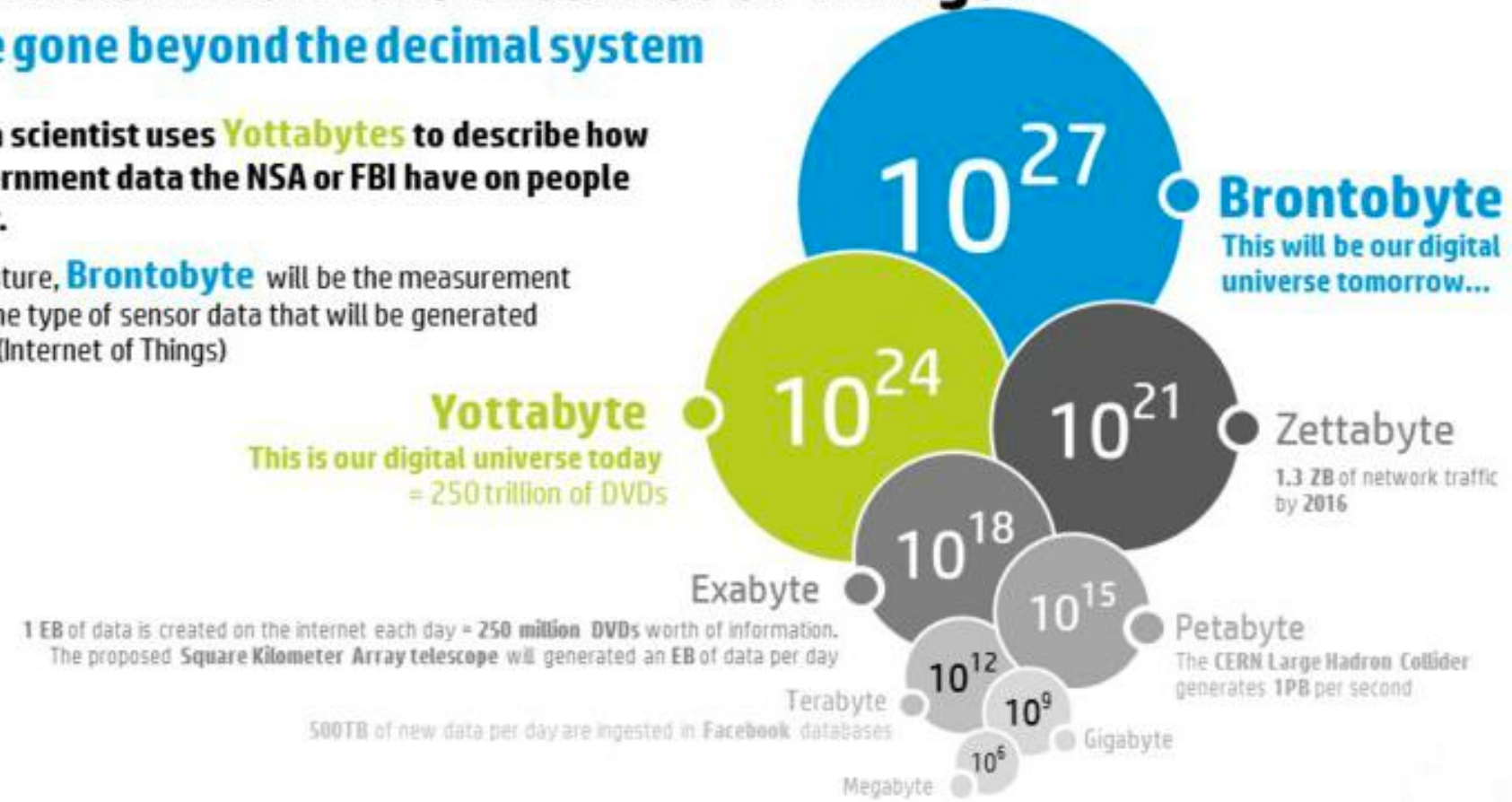
WOW, HOW BIG IS A BRONTOBYTE?

Information from the Internet of Things:

We have gone beyond the decimal system

Today data scientist uses Yottabytes to describe how much government data the NSA or FBI have on people altogether.

In the near future, **Brontobyte** will be the measurement to describe the type of sensor data that will be generated from the IoT (Internet of Things)



This much data will require a fast OODA loop

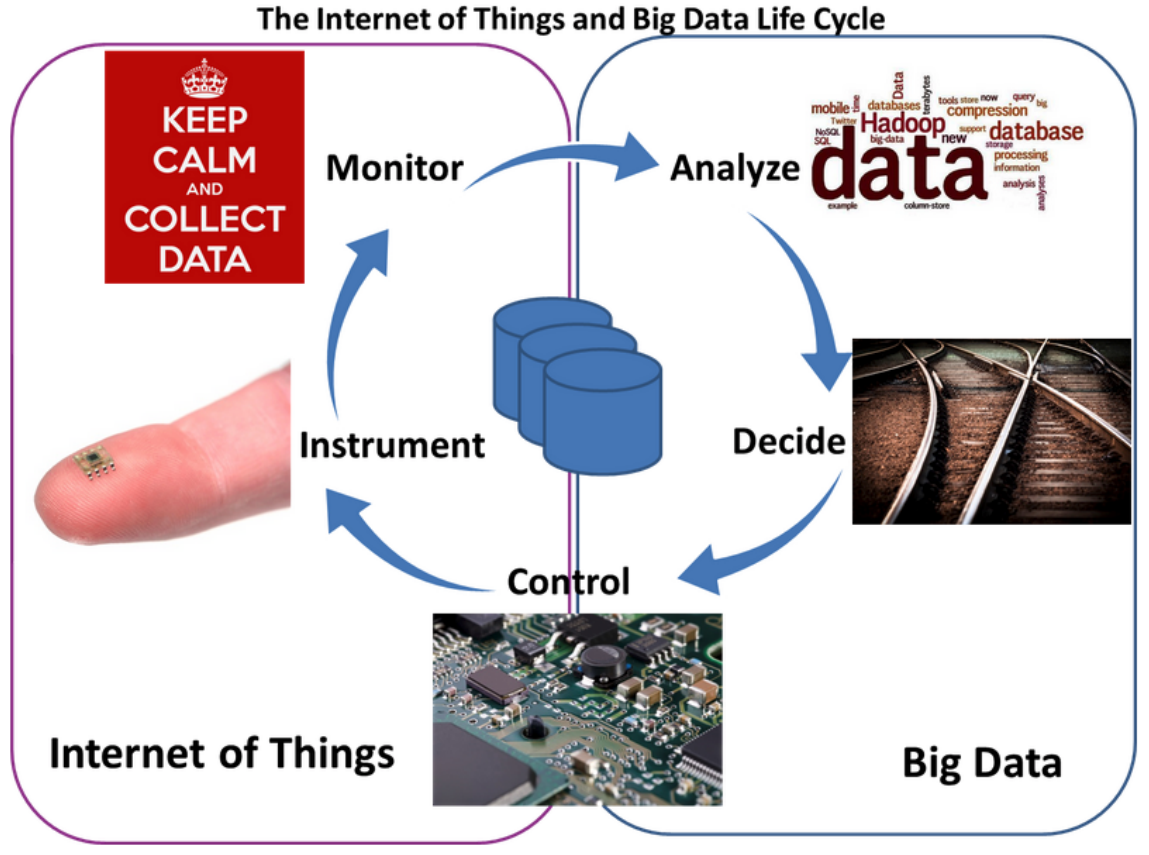


Image courtesy http://www.telecom-cloud.net/wp-content/uploads/2015/05/Screen-Shot-2015-05-27-at-3.51.47-PM.png

EXAMPLE FROM THE IOT

Domain: Prognostics and Health Management

Machine: Turbofan Jet Engines

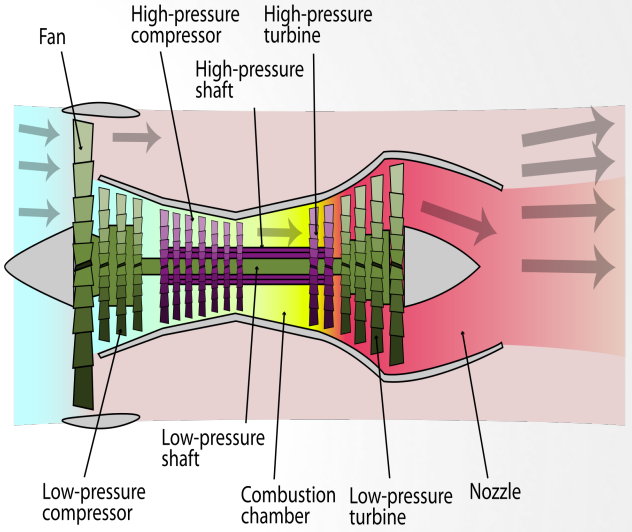
Data Set: A. Saxena and K. Goebel (2008). "Turbofan Engine Degradation Simulation Data Set", NASA Ames Prognostics Data Repository

Predict Remaining Useful Life from Partial Life Runs

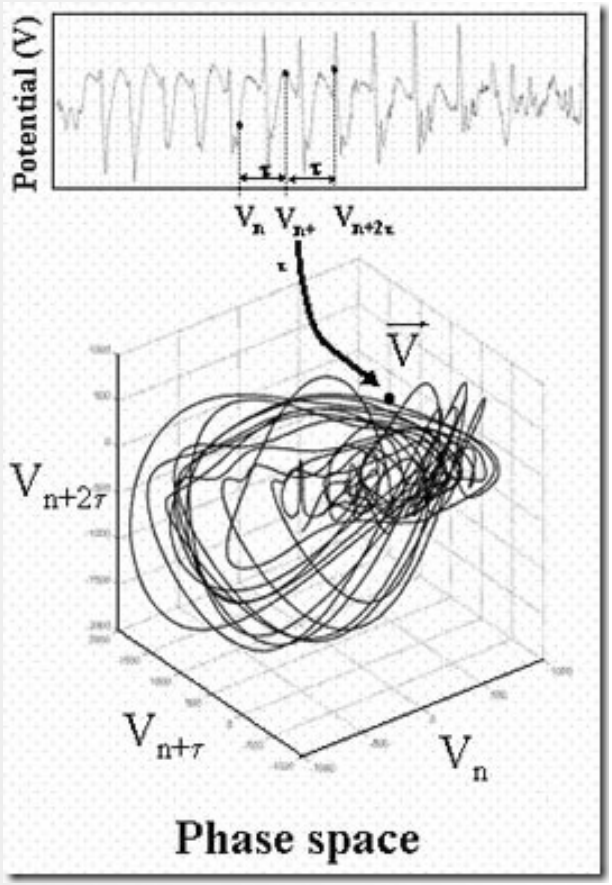
Six operating modes, two failure modes, manufacturing variability

Training: 249 jet engines run to failure

Test: 248 jet engines



INCORPORATING PRIOR STATE



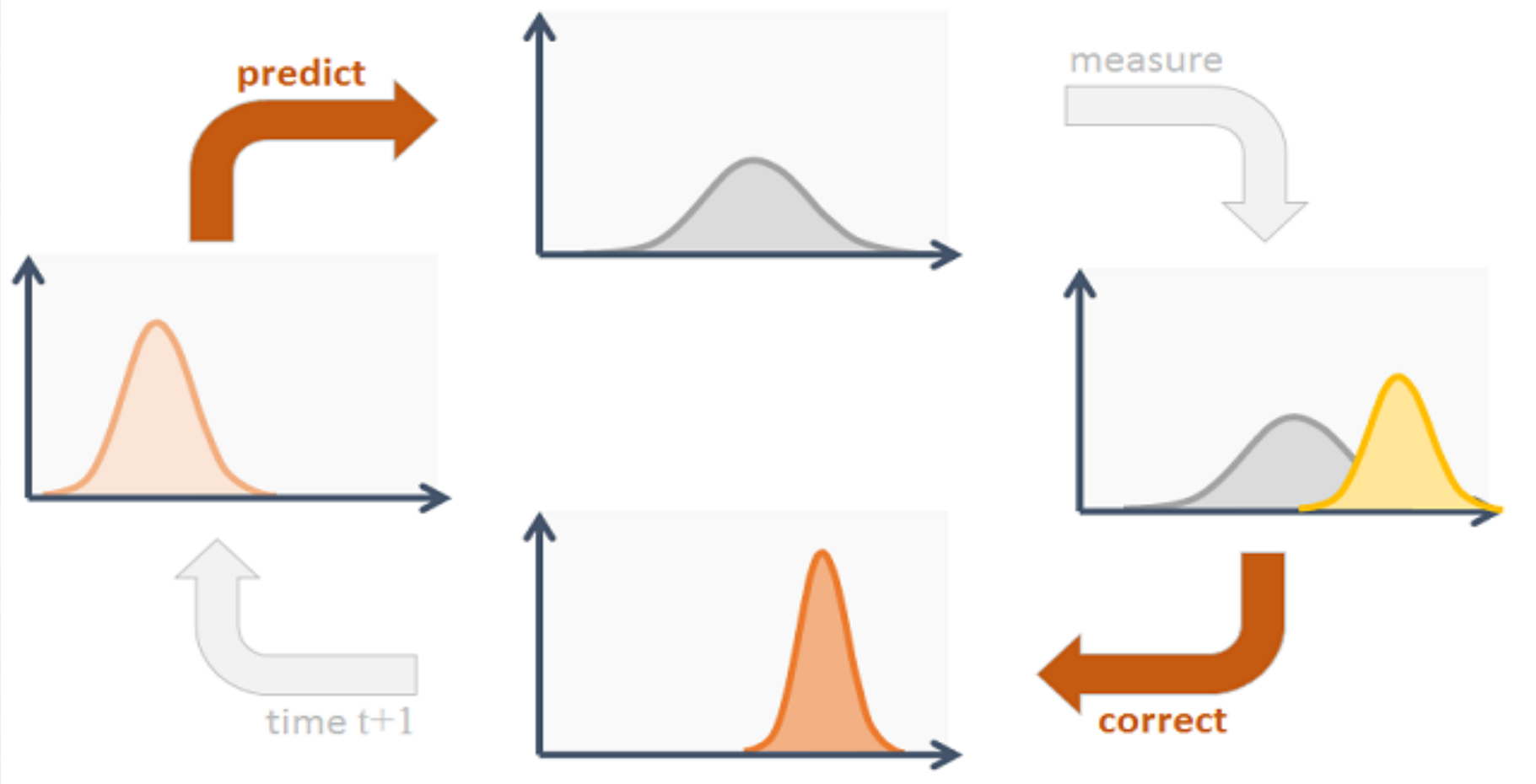
One option:
Phase Space Embedding

Drawbacks:
Incorporates knowledge from small number of prior states

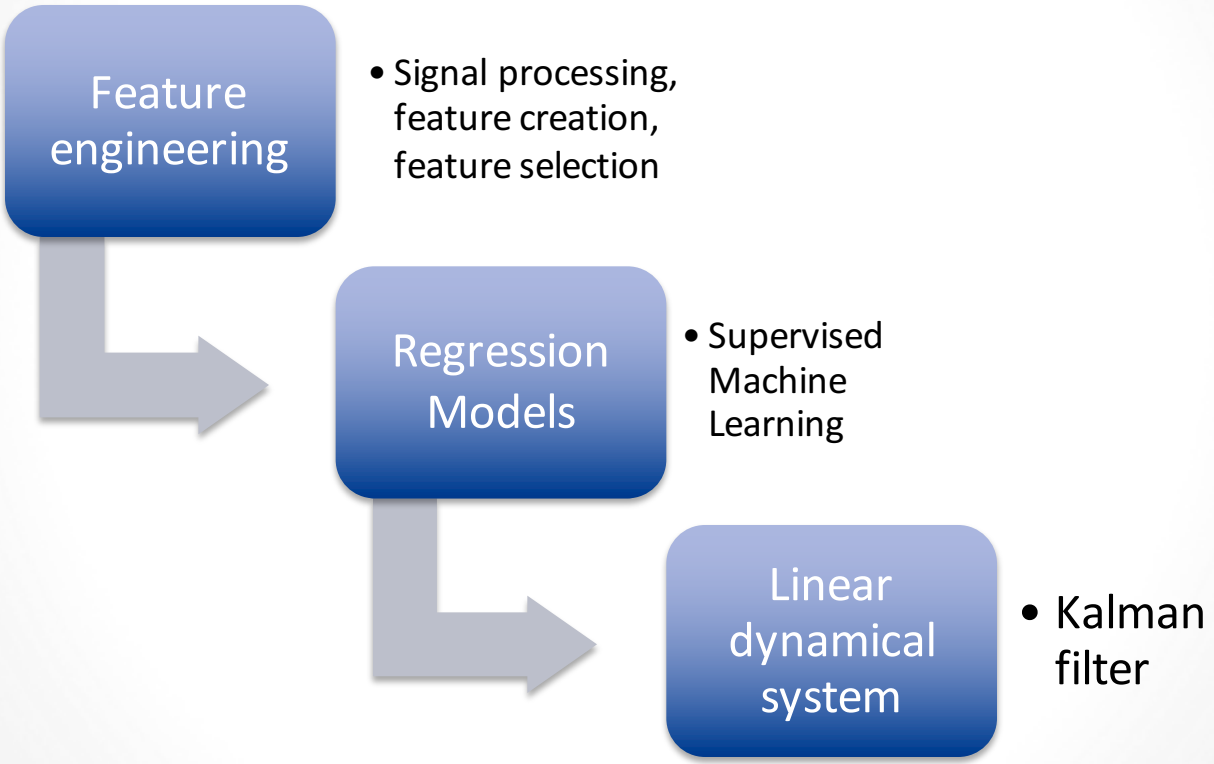
Curse of dimensionality

Disentangling the dynamic core: a research program for a neurodynamics at the large-scale
MICHEL LE VAN QUYEN
Biol. Res. v.36 n.1 Santiago 2003
http://dx.doi.org/10.4067/S0716-97602003000100006

KALMAN FILTER



OVERALL PIPELINE

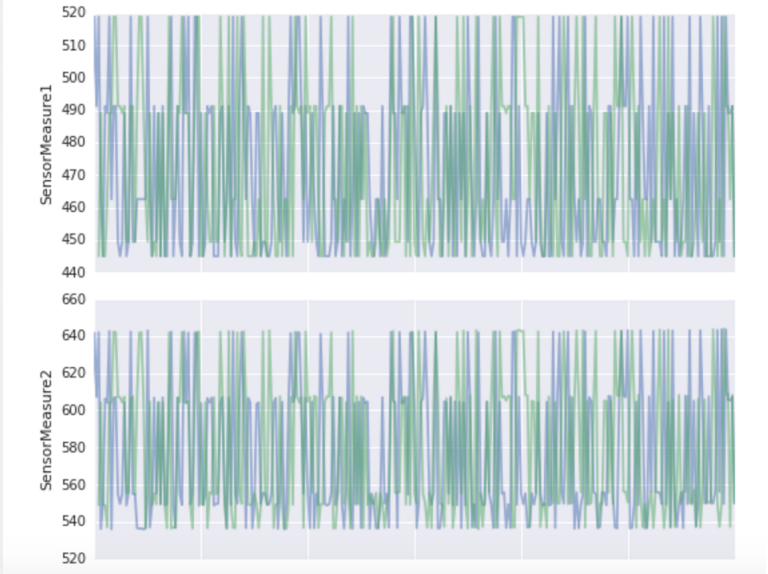


EXPLORATORY DATA ANALYSIS

```
sample_units = train_with_predictor["UnitNumber"] < 3
```

Boolean Indexing

```
g = sns.PairGrid(data=train_pd,
                 x_vars=dependent_var,
                 y_vars=sensor_measure_columns_names + \
                        operational_settings_columns_names,
                 hue="UnitNumber", size=3, aspect=2.5)
g = g.map(plt.plot, alpha=0.5)
g = g.set(xlim=(300,0))
g = g.add_legend()
```

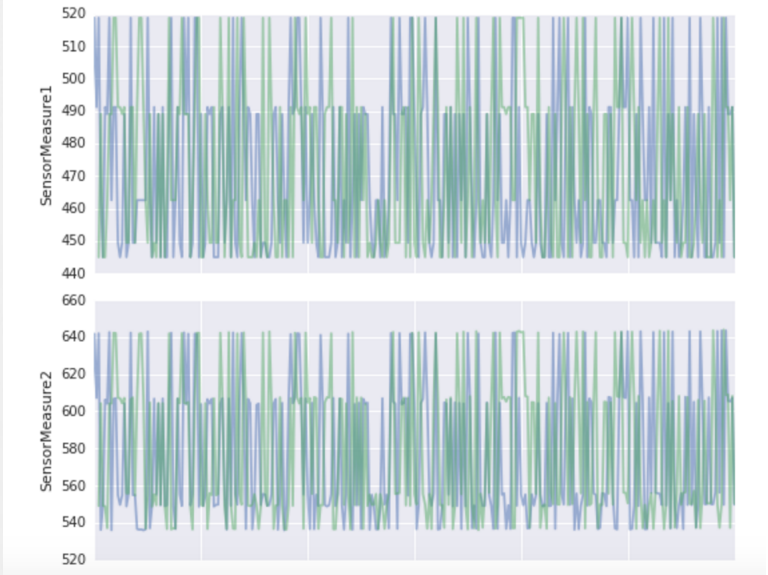


EXPLORATORY DATA ANALYSIS

```
train_pd = train_with_predictor[sample_units].as_data_frame(
```

Sample the data to local memory

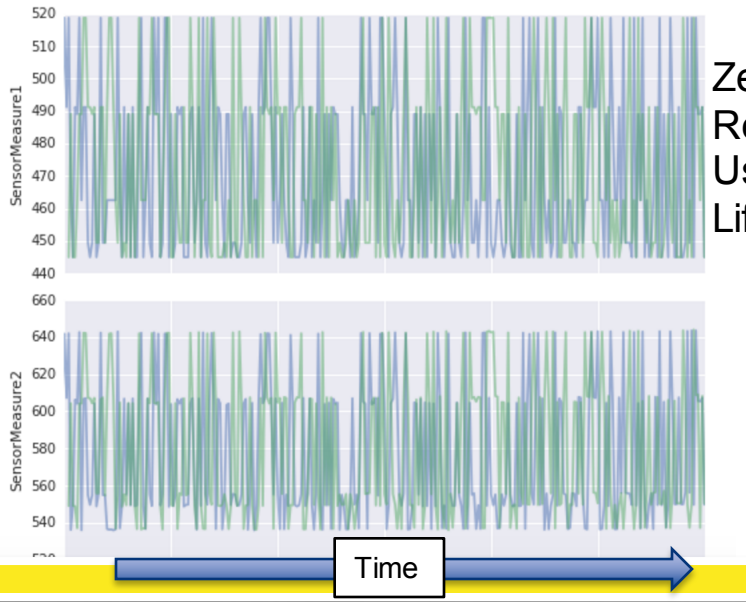
```
g = sns.PairGrid(data=train_pd,
                 x_vars=dependent_var,
                 y_vars=sensor_measure_columns_names + \
                        operational_settings_columns_names,
                 hue="UnitNumber", size=3, aspect=2.5)
g = g.map(plt.plot, alpha=0.5)
g = g.set(xlim=(300,0))
g = g.add_legend()
```



EXPLORATORY DATA ANALYSIS

```
sample_units = train_with_predictor["UnitNumber"] < 3
train_pd = train_with_predictor[sample_units].as_data_fr
```

```
g = sns.PairGrid(data=train_pd,
                 x_vars=dependent_var,
                 y_vars=sensor_measure_columns_names + \
                        operational_settings_columns_names,
                 hue="UnitNumber", size=3, aspect=2.5)
g = g.map(plt.plot, alpha=0.5)
g = g.set(xlim=(300,0))
g = g.add_legend()
```



Zero Remaining Useful Life

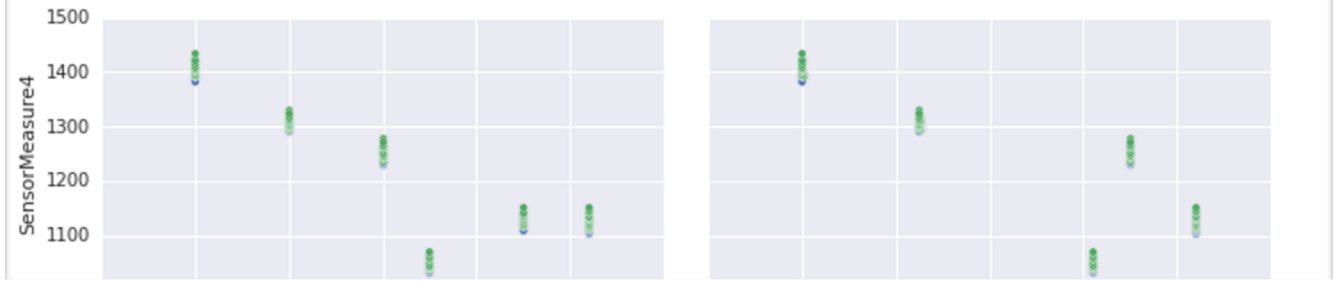
Use your favorite visualization tools

(Seaborn!)

Ugh, where are trends over time

MODEL BASED DATA ENRICHMENT

```
g = sns.pairplot(data=train_pd,
                 x_vars=["OpSet1","OpSet2"],
                 y_vars=["SensorMeasure4", "SensorMeasure3",
                         "SensorMeasure9", "SensorMeasure8",
                         "SensorMeasure13", "SensorMeasure6"],
                 hue="UnitNumber", aspect=2)
```



Sensor measurements appear in clusters

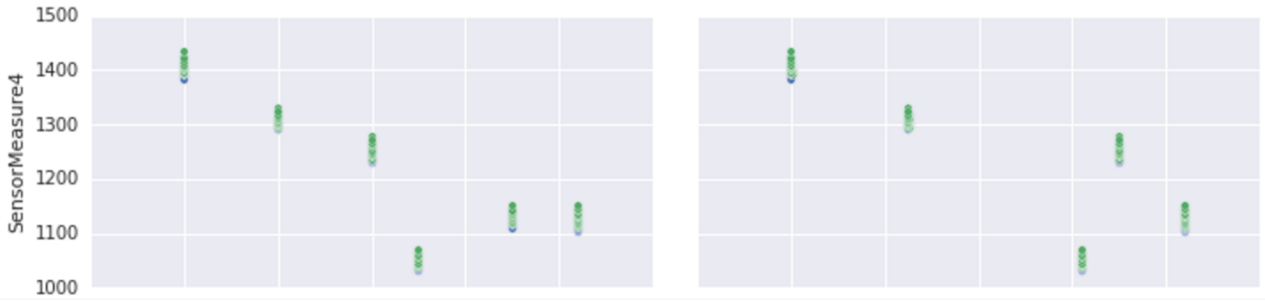
Corresponding to operating mode!

```
from h2o.estimators.kmeans import H2OKMeansEstimator
```

```
operating_mode_estimator = H2OKMeansEstimator(k=operating_modes)
operating_mode_estimator.train(x=operational_settings_columns_names,
                               training_frame=train_with_predictor)
```

FEATURE ENGINEERING

```
g = sns.pairplot(data=train_pd,
                 x_vars=["OpSet1","OpSet2"],
                 y_vars=["SensorMeasure4", "SensorMeasure3",
                         "SensorMeasure9", "SensorMeasure8",
                         "SensorMeasure13", "SensorMeasure6"],
                 hue="UnitNumber", aspect=2)
```



Use H2O k-means to find cluster centers

```
from h2o.estimators.kmeans import H2OKMeansEstimator
```

```
operating_mode_estimator = H2OKMeansEstimator(k=operating_modes)
operating_mode_estimator.train(x=operational_settings_columns_names,
                               training_frame=train_with_predictor)
```

FEATURE ENGINEERING

```
: from h2o.estimators.kmeans import H2OKMeansEstimator

: operating_mode_estimator = H2OKMeansEstimator(k=operating_
  operating_mode_estimator.train(x=operational_settings_colu
                                 training_frame=train_with_pr
```

Enrich existing data with operating mode membership

```
def append_operating_mode(h2o_frame, estimator):
    operating_mode_labels = estimator.predict(h2o_frame)
    operating_mode_labels.set_names(operating_mode_column_name);
    operating_mode_labels = operating_mode_labels.asfactor()
    h2o_frame_augmented = h2o_frame.cbind(operating_mode_labels)
    return h2o_frame_augmented

train_augmented = append_operating_mode(train_with_predictor,operating_mode_estimator)
test_augmented = append_operating_mode(test,operating_mode_estimator)
```

MORE FEATURE ENGINEERING

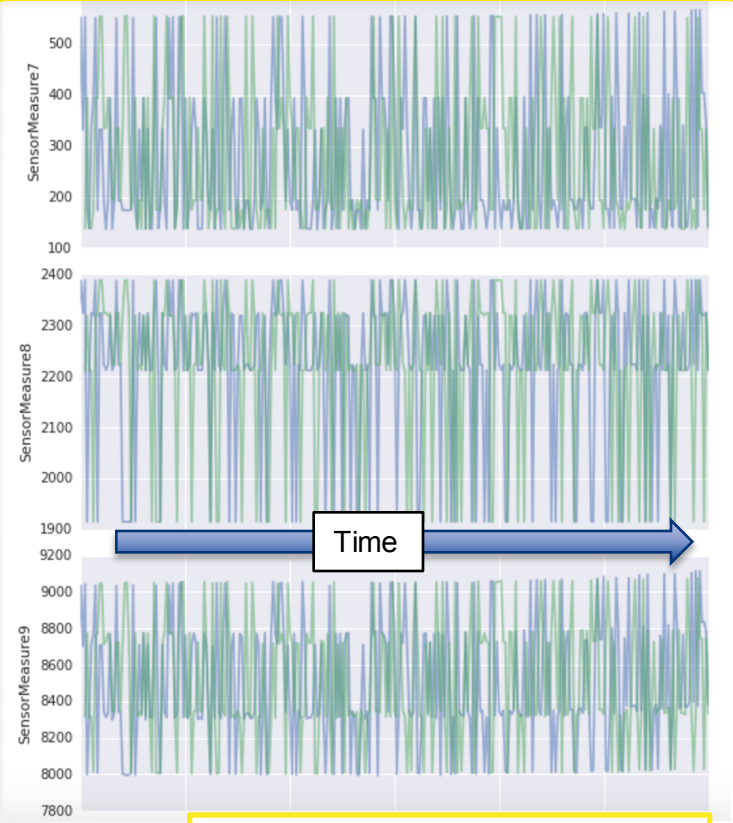
```
def standardize_by_operating_mode(train, test):
    t = train.group_by(operating_mode_column_name).\
        mean(sensor_measure_columns_names).\
        sd(sensor_measure_columns_names).frame

    s = train.merge(t)
    r = test.merge(t)
    standardize_measures_columns_names = []
    for sensor_measure_column_name in sensor_measure_columns_names:
        include_this_measure = True
        # if any of the operating modes shows 0 or NaN standard deviation,
        # do not standardize that sensor measure,
        # nor use it in the model building
        for i in range(0,operating_modes):
            stdev = t[t["OperatingMode"] == str(i),"sdev_"+sensor_measure_column_name][0,0]
            if stdev == 0.0:
                include_this_measure = False
                break
        if include_this_measure:
            new_column_name = "stdized_"+sensor_measure_column_name
            standardize_measures_columns_names.append(new_column_name)
            s[new_column_name] = ((s[sensor_measure_column_name]-
                                   s["mean_"+sensor_measure_column_name])/
                                   s["sdev_"+sensor_measure_column_name])
            r[new_column_name] = ((r[sensor_measure_column_name]-
                                   r["mean_"+sensor_measure_column_name])/
                                   r["sdev_"+sensor_measure_column_name])
    return (s,r,standardize_measures_columns_names)

train_stdized, test_stdized, standardized_measures_columns_names = \
    standardize_by_operating_mode(train_augmented, test_augmented)
```

For non-constant sensor measurements within an operating mode,

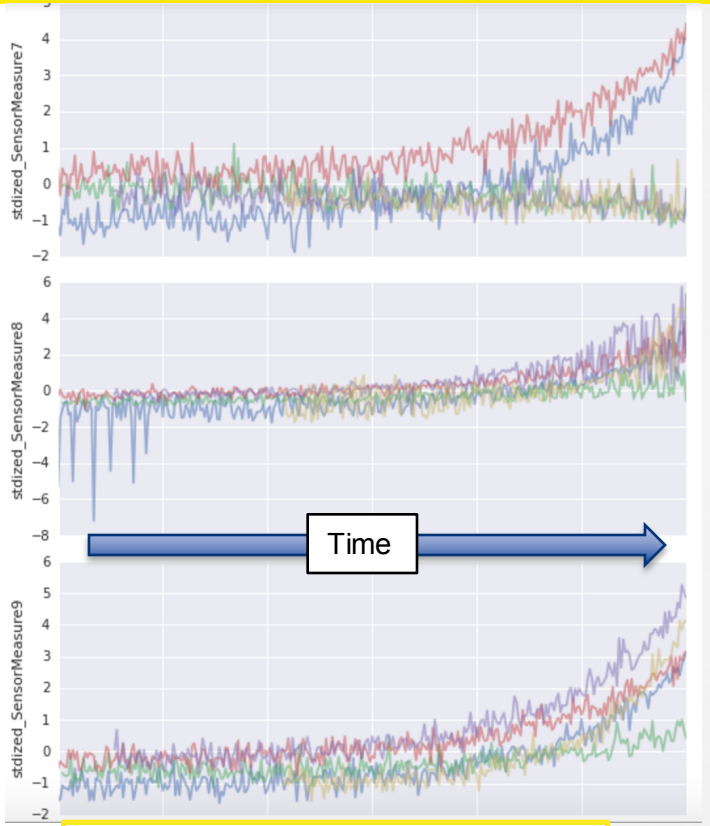
Standardize each sensor measurement by operating mode

Based on the training data

TRENDS OVER TIME!



Before
H2O Data Preparation



Ready for
H2O Learning

MODELING - SIMPLE

```
from h2o.estimators.gbm import H2OGradientBoostingEstimator
```

```
gbm_regressor = H2OGradientBoostingEstimator(distribution="gaussian",
                                             score_each_iteration=True,
                                             stopping_metric="MSE",
                                             stopping_tolerance=0.001,
                                             stopping_rounds=5)
```

```
                      training_frame=train_final,
                      fold_column=fold_column_name)
```

Configure an Estimator

MODELING - SIMPLE

```
from h2o.estimators.gbm import H2OGradientBoostingEstimator
```

```
gbm_regressor = H2OGradientBoostingEstimator(distribution="gaussian",
                                             score_each_iteration=True,
                                             stopping_metric="MSE",
                                             stopping_tolerance=0.001,
                                             )
```

```
gbm_regressor.train(x=independent_vars, y=dependent_var,
                    training_frame=train_final,
                    fold_column=fold_column_name)
```

Train an Estimator

KALMAN FILTER INPUTS

State: [Cycles Remaining, Rate of Change of Cycles Remaining]

State Transition: [[1,1],[0,1]] (takes [RUL,-1] -> [RUL-1, -1])

Observations: regression output of each model in ensemble

Observation Covariance: mean square error of each model on training data (diagonal matrix)

Initial State: [Mean of models, -1]

KALMAN - POST PROCESSING

```
import pykalman as pyk

final_ensembled_preds = {}
pred_cols = [ name for name in predictions_df.columns if "predict" in name]
for unit in predictions_df.UnitNumber.unique():
    preds_for_unit = predictions_df[ predictions_df.UnitNumber == unit ]
    observations = preds_for_unit.as_matrix(pred_cols)
    initial_state_mean = np.array( [np.mean(observations[0]),-1] )
    kf = pyk.KalmanFilter(transition_matrices=a_transition_matrix,\
                          initial_state_mean=initial_state_mean,\
                          observation_covariance=r_observation_covariance,\
                          observation_matrices=h_observation_matrices,\
                          n_dim_state=n_dim_state, n_dim_obs=n_dim_obs)
    mean,_ = kf.filter(observations)
    final_ensembled_preds[unit] = mean
```

SIGNAL PROCESSING + MACHINE LEARNING

- Filtering, Convolution (integrals, differences, etc)
- Time domain to frequency domain (Fourier) or other domain (wavelet)
- Dynamic time warping for sequence similarity
- Spatial-temporal analytics
- Convolution Neural Nets and LSTM-RNN

RESOURCES

- Download and go: http://www.h2o.ai/download
- Documentation: http://docs.h2o.ai/
- Booklets, Datasheet: http://www.h2o.ai/resources/
- Github: http://github.com/h2oai/
- Training: http://learn.h2o.ai/
- This presentation and associated Jupyter notebook (look in 2016_02_23_MachineLearningAndKalmanFiltersForMachinePrognostics): https://github.com/h2oai/h2o-meetups/

THANK YOU