DePaul University College of Computing and Digital Media

Casey Bennett, PhD

Oct.2, 2019

Homework 1

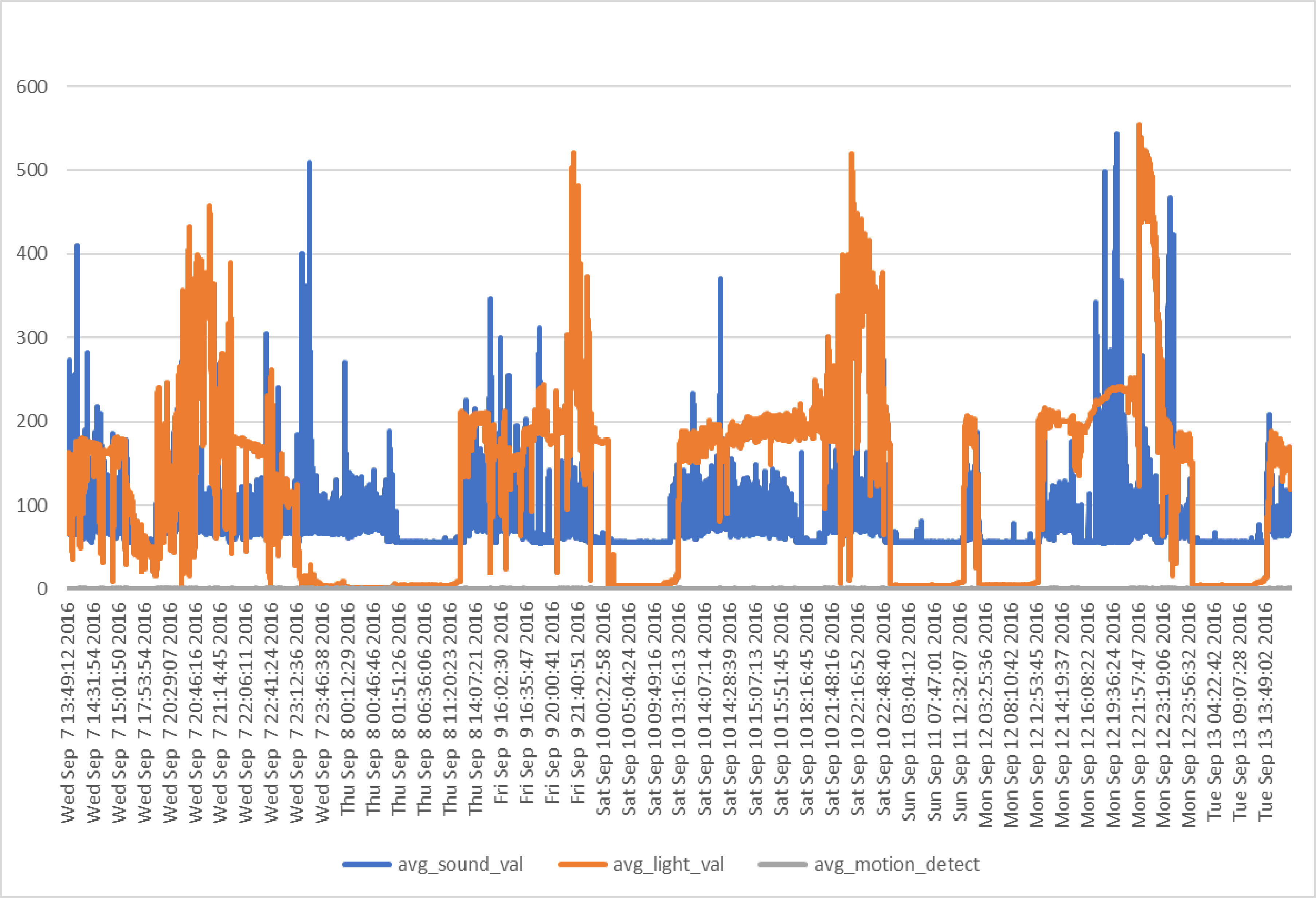
There were a few common issues:

1) Interpreting difference in performance (std dev)
   - 2 “different” numbers are not necessarily different in a statistical sense

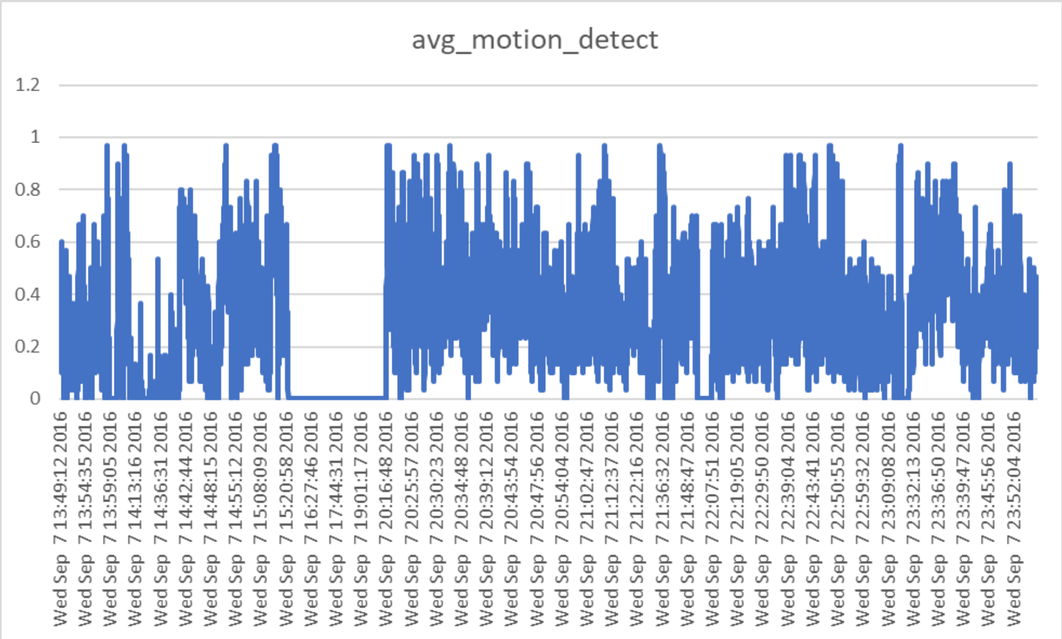
2) The “black box” problem (feature selection)

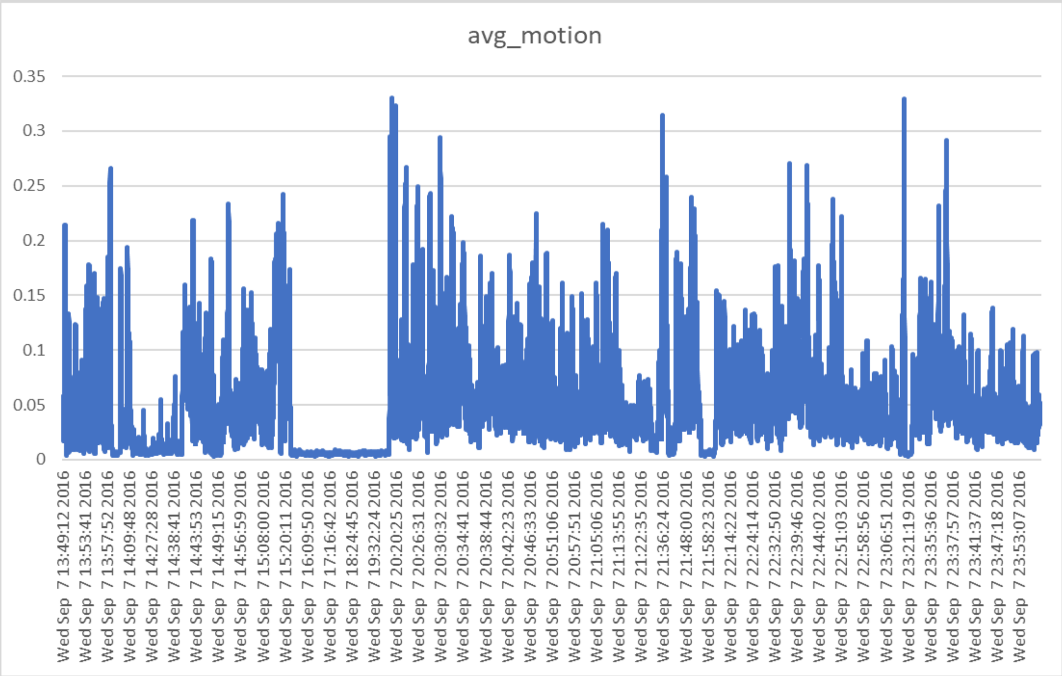
3) Communication as data scientist (explaining the “why”)

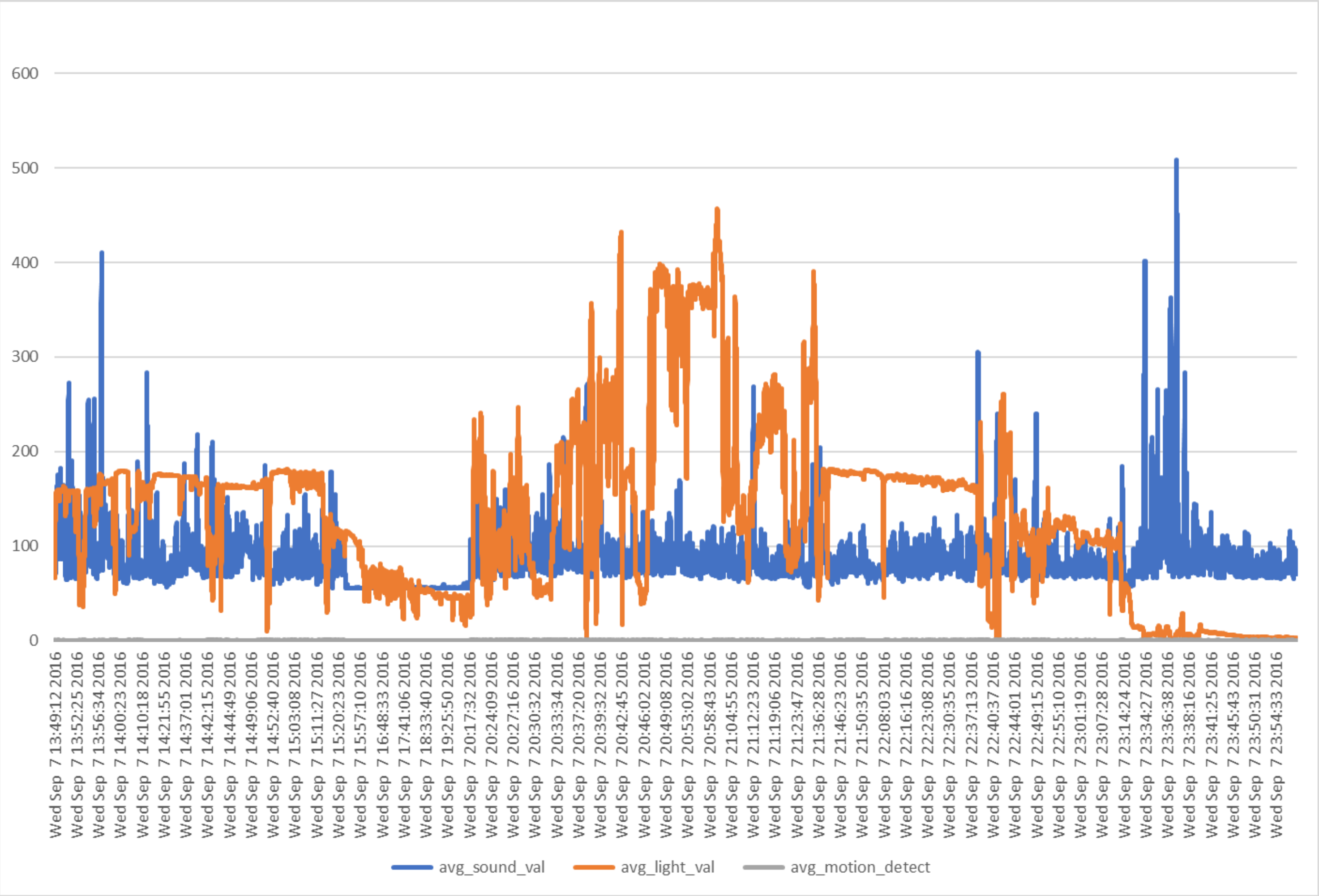
Weekly Circadian data



Wednesday data



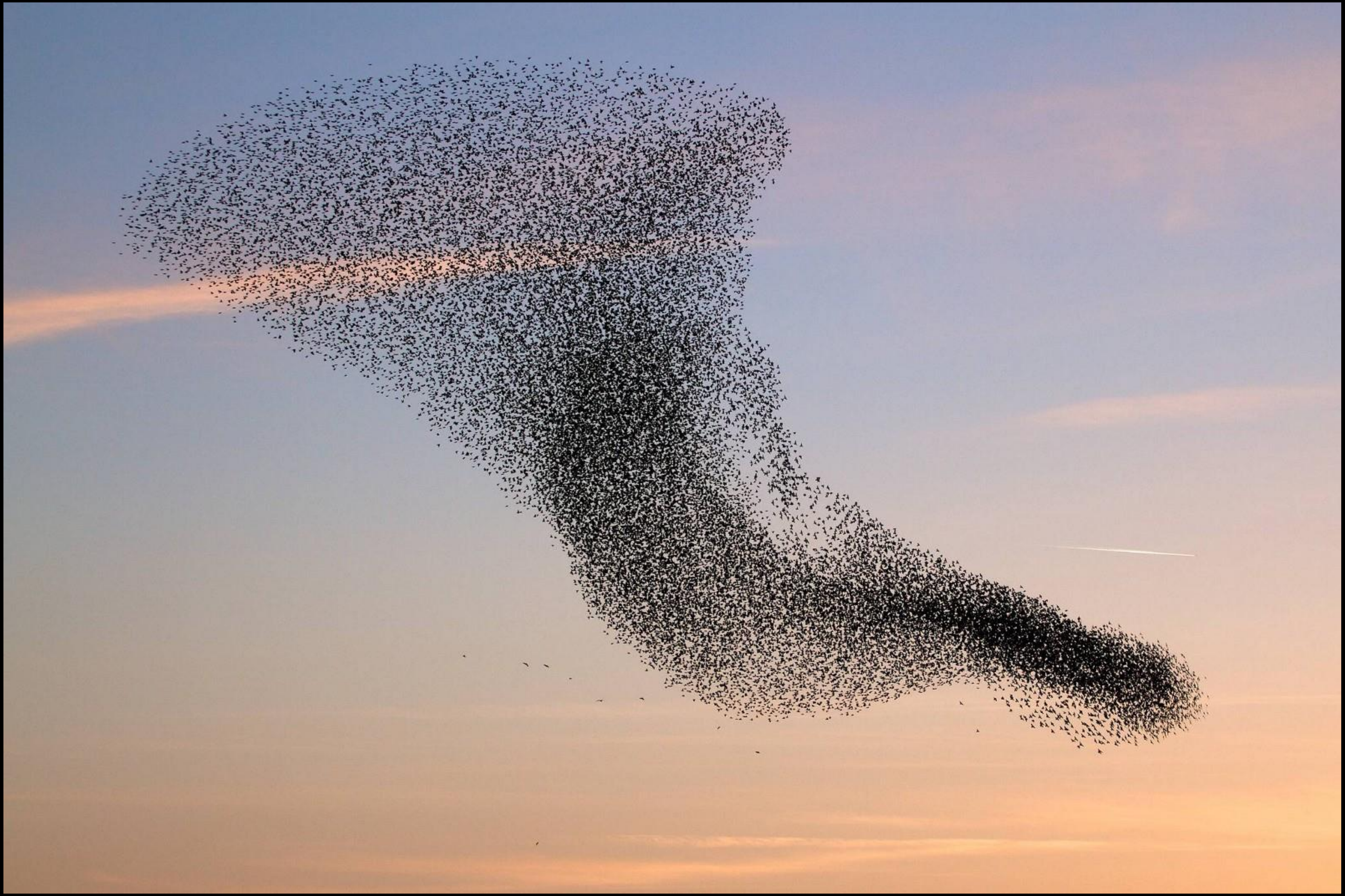




Boosting and Ensemble Methods

https://pollev.com/caseybennett801

or text “caseybennett801” to 37607

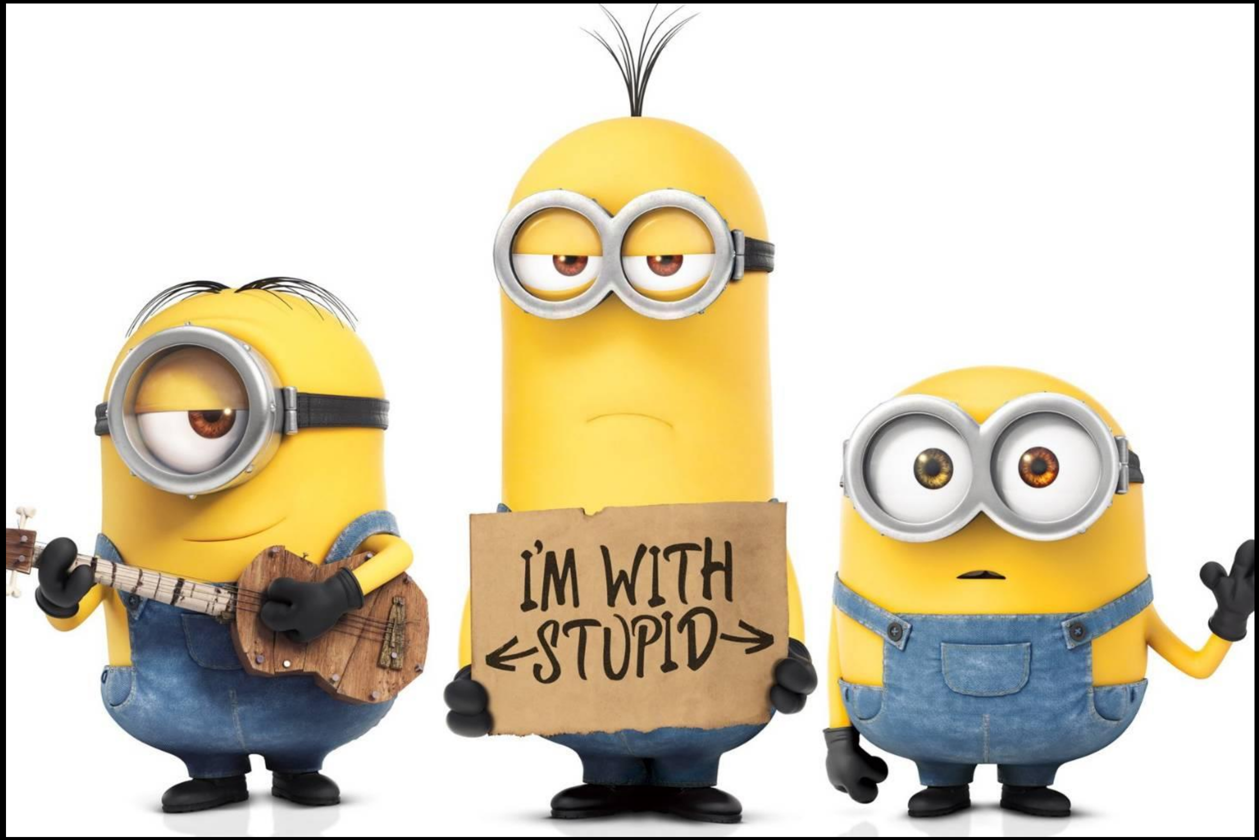


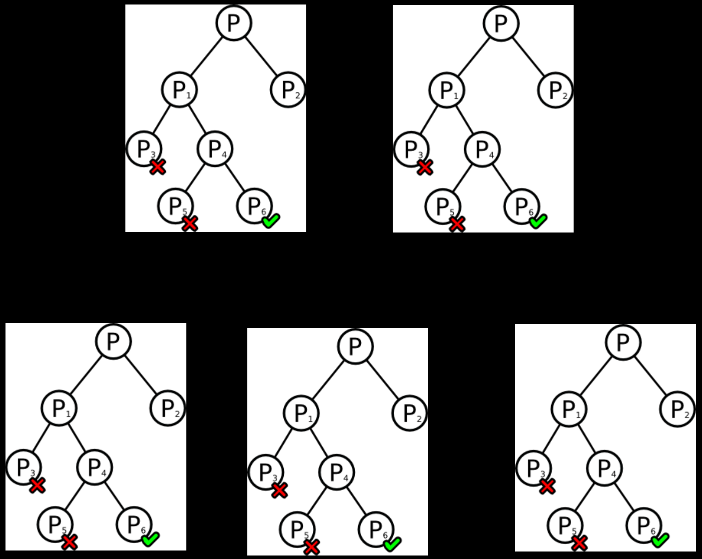
Collective Intelligence

- Markets can make smart decisions, even if the individuals within aren’t so bright, or lack info
- Ensemble Learning (e.g. Voting)

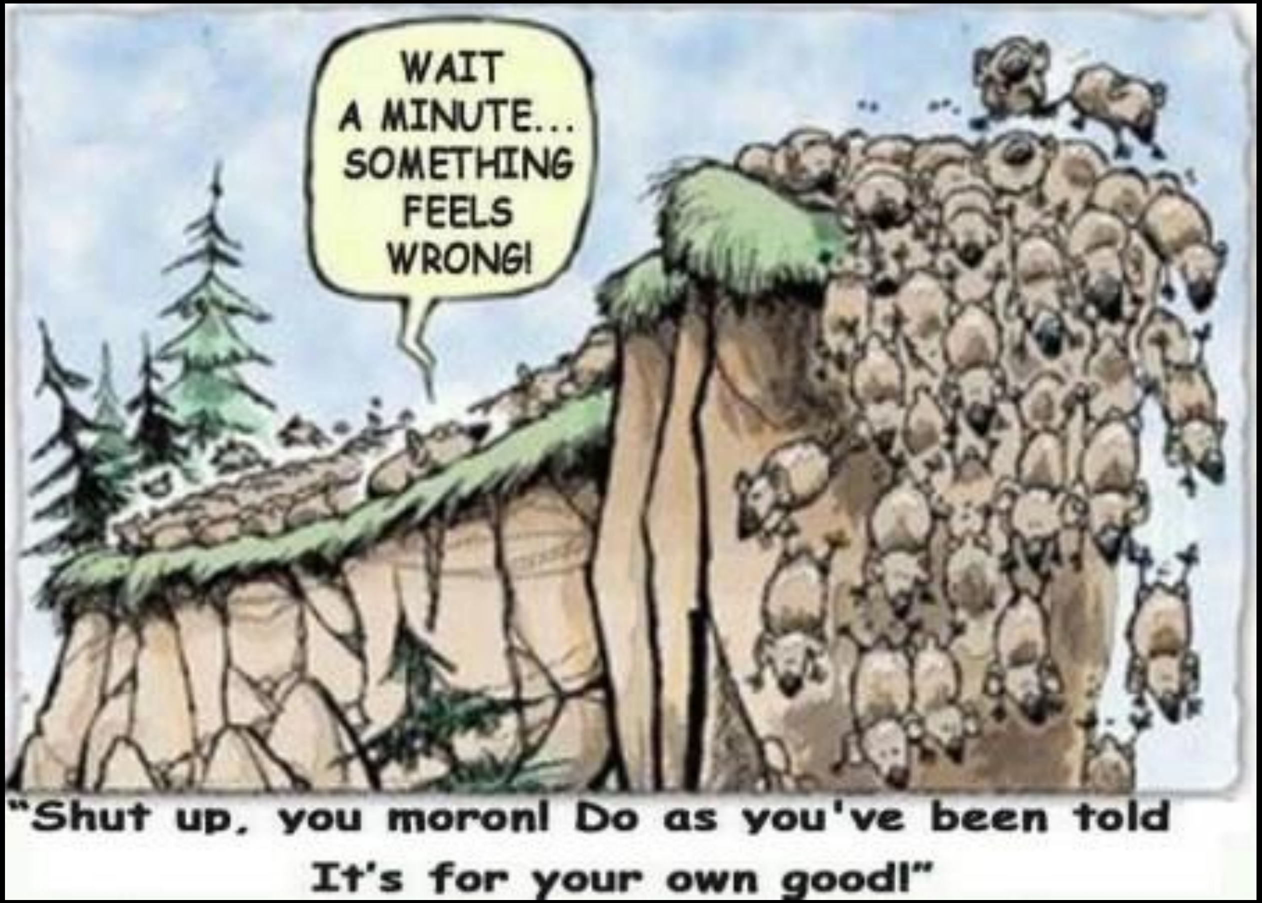
Different kinds of idiots working together





- We can accomplish this by giving different individuals different parts of the information
- e.g. different subsets of data or variables
- Random Forests (bagging)

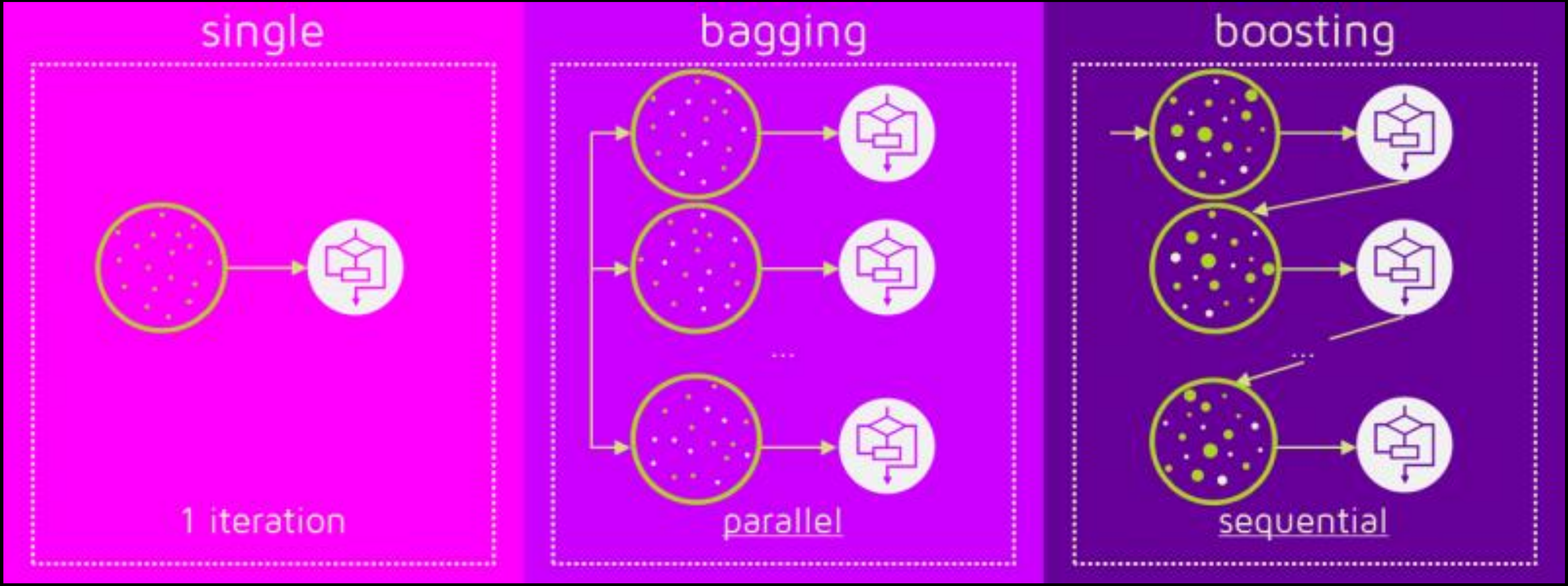
What if there was a smarter way though?



If each idiot just does the same thing as the last, how might we overcome that?

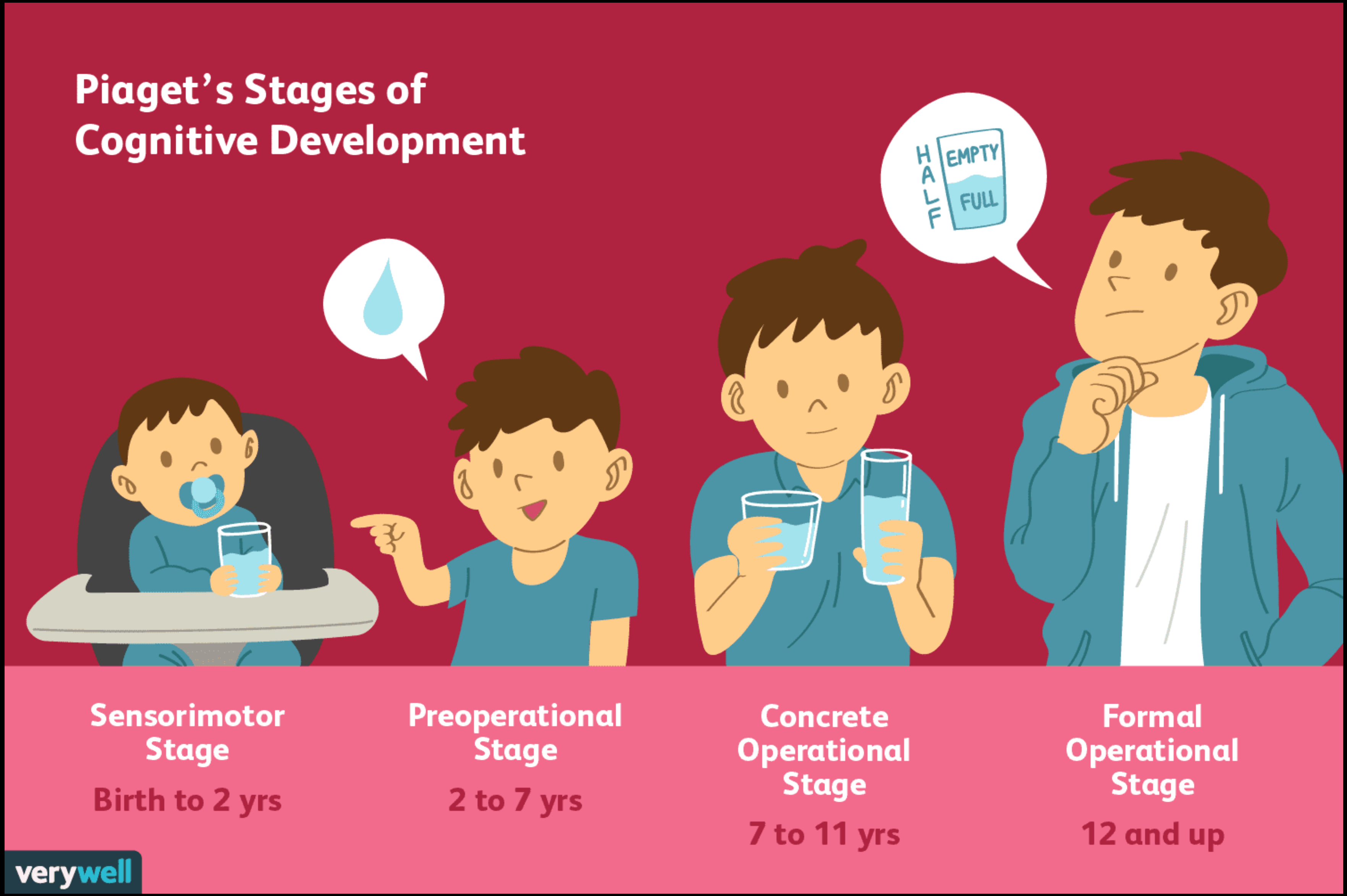
What if those idiots learned from each other?

Learn from mistakes


single
bagging
boosting
1 iteration
parallel
sequential

How do human babies learn?

Work of Jean Piaget



A-Not-B Error Phenomenon

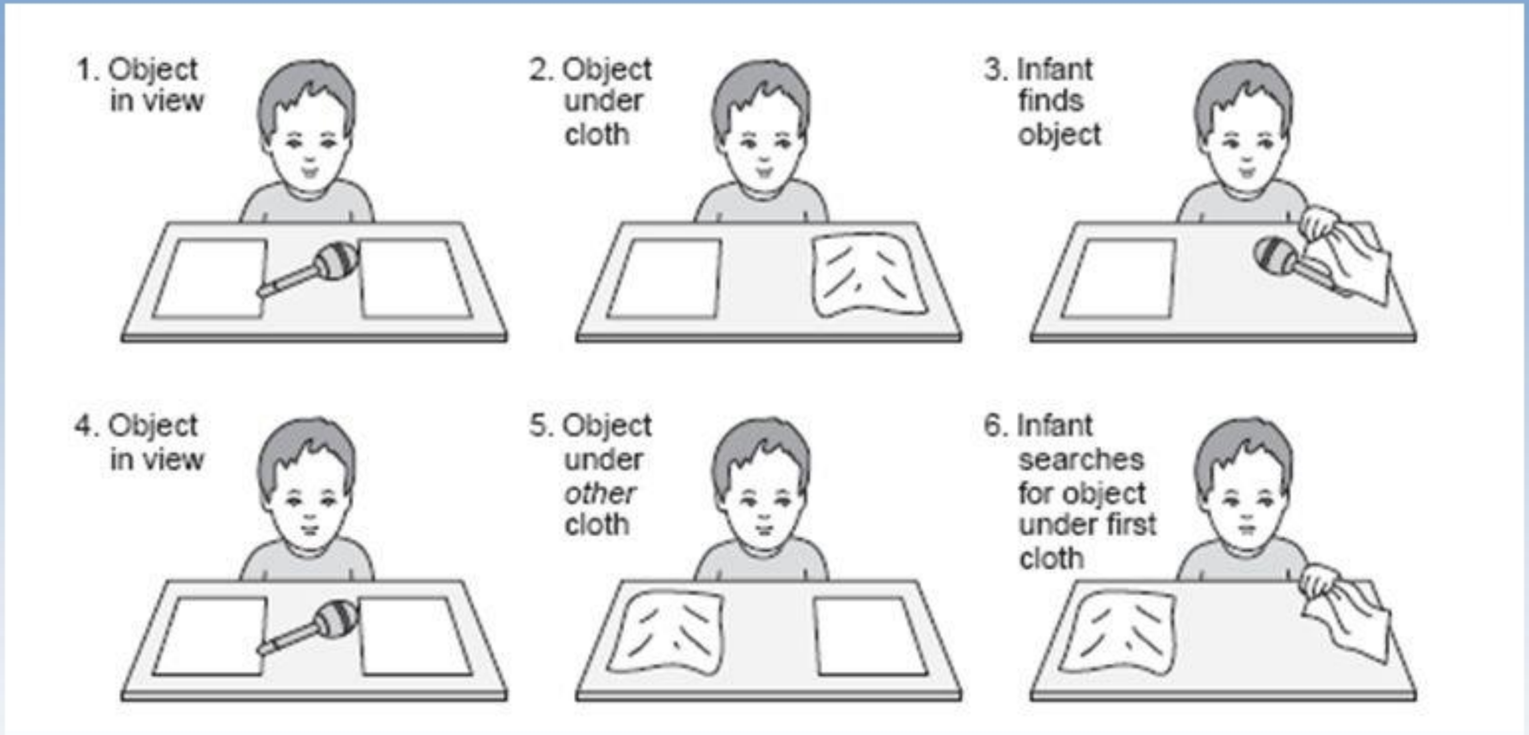
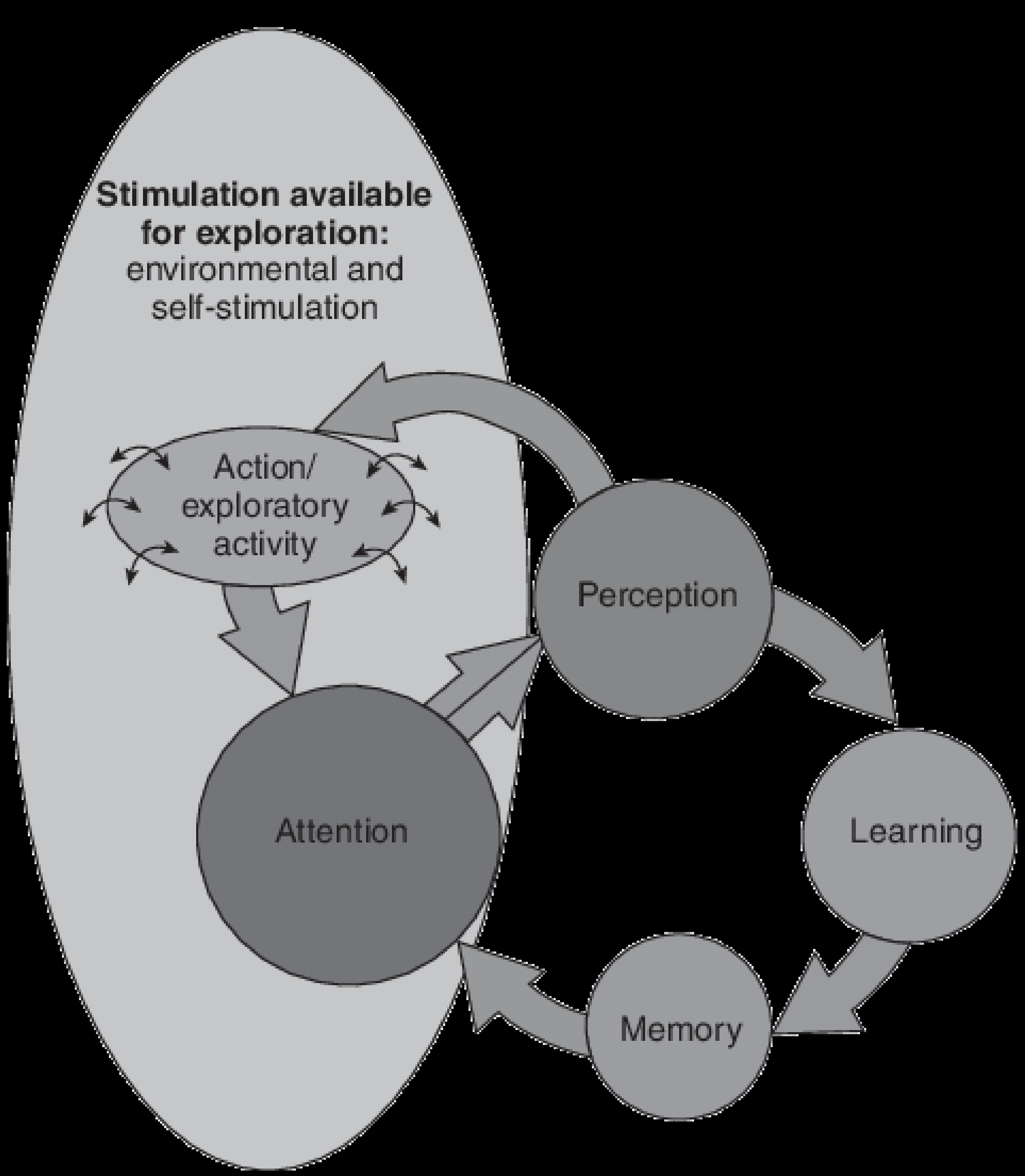


Figure 6.5



Selective Attention

Selective Attention is the ability to *focus* on critical aspects of the task, which allows learning to take place

Boosting

Two Major Types of Boosting

1) Ada Boost
   - iteratively creates a series of “weighted” learners (models)

2) Gradient Boosting
   - builds on Ada boost, by using gradient descent to decide on when and how to add new learners to the ensemble

Ada Boost - Basics

1) Generally use decision stumps as learners (trees of depth 1, having a single split)
2) After each model is added, we re-weight observations, so that misclassified examples are given higher weights
3) Future learners then focus on trying to accurately classify these “more difficult” patterns
4) Predictions are made by taking a vote of all learners for new instances, weighted by their accuracy on the training set

Ada Boost – Key Points

- Both samples and models are weighted
- Each new model added to the ensemble focuses more on previously misclassified examples
- Predictions are made by weighted voting of the whole ensemble (all the models built)

Gradient Boosting

1) Builds on Ada Boost, by trying to minimize some *loss function* (e.g. error rate, cost, etc.)
2) For each new learner that is added, we essentially choose a split point to try to minimize that loss
3) Then we go thru the same re-weighting process as the underlying Ada Boost approach, and do it again
4) We stop adding models either once the performance stops improving, or we reach some preset maximum number of models

Gradient Boosting

- This essentially becomes a *gradient descent* problem, which we will return to in more detail next week when discussing Neural Networks

- Over time, people have started using slightly more complex trees with Gradient Boosting, but generally the depth is still kept no more than 3

Boosting generally uses a “weighted” voting scheme, do you think that is the best? Why or why not?

Voting Schemes

1) Weighted Voting vs. un-Weighted Voting
2) Majority Voting
3) Average of Probabilities
4) Maximum Probability
5) Many many others variations

Boosting Pseudocode

1) Load Data
2) Split data into cross-validation folds (or test/training split)
3) Using all data (weighted) for new tree
4) Evaluate features from subset using metric (e.g. info gain, gini index, sum of squared errors)
5) Pick best feature and create split
6) Recurse down the tree, repeating steps #4-6, until max depth
7) Reweight training samples based on misclassification
8) Go back to step #3 and start with new tree, until maximum tree number

   OR Loss function stops changing below some threshold (for gradient boosting)
9) Combine *weighted* tree predictions (e.g. mean, voting)
10) Calculate performance (Accuracy, AUC, RMSE, etc.)

Code Implementation

- Scikit method
- Spark method

Code Implementation

#SciKit Gradient Boosting

```
GradientBoostingRegressor(loss='ls', learning_rate=0.1, n_estimators=100, subsample=1.0,
                          criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1,
                          min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_decrease=0.0,
                          min_impurity_split=None, init=None, random_state=None,
                          max_features=None, alpha=0.9, max_leaf_nodes=None,
                          presort='auto', validation_fraction=0.1, n_iter_no_change=None, tol=0.0001)
```

#Spark Gradient Boosting

```
GBTRegressor(labelCol="idxLabel", featuresCol="idxFeatures", maxIter=20, maxDepth=5,
             lossType='squared', minInstancesPerNode=1, featureSubsetStrategy="auto",
             subsamplingRate=1.0, stepSize = 0.1)
```

- There are also *Classifier* versions for Gradient Boosting in both Scikit and Spark, for when you have a continuous categorical or binary target variable you are trying to predict

Code Implementation

- Pay careful attention to a few parameters:
  - Number of Trees (aka estimators or iterations)
  - Learning rate (aka step size)
  - Loss function
  - Max depth of each tree

Variations on Boosting

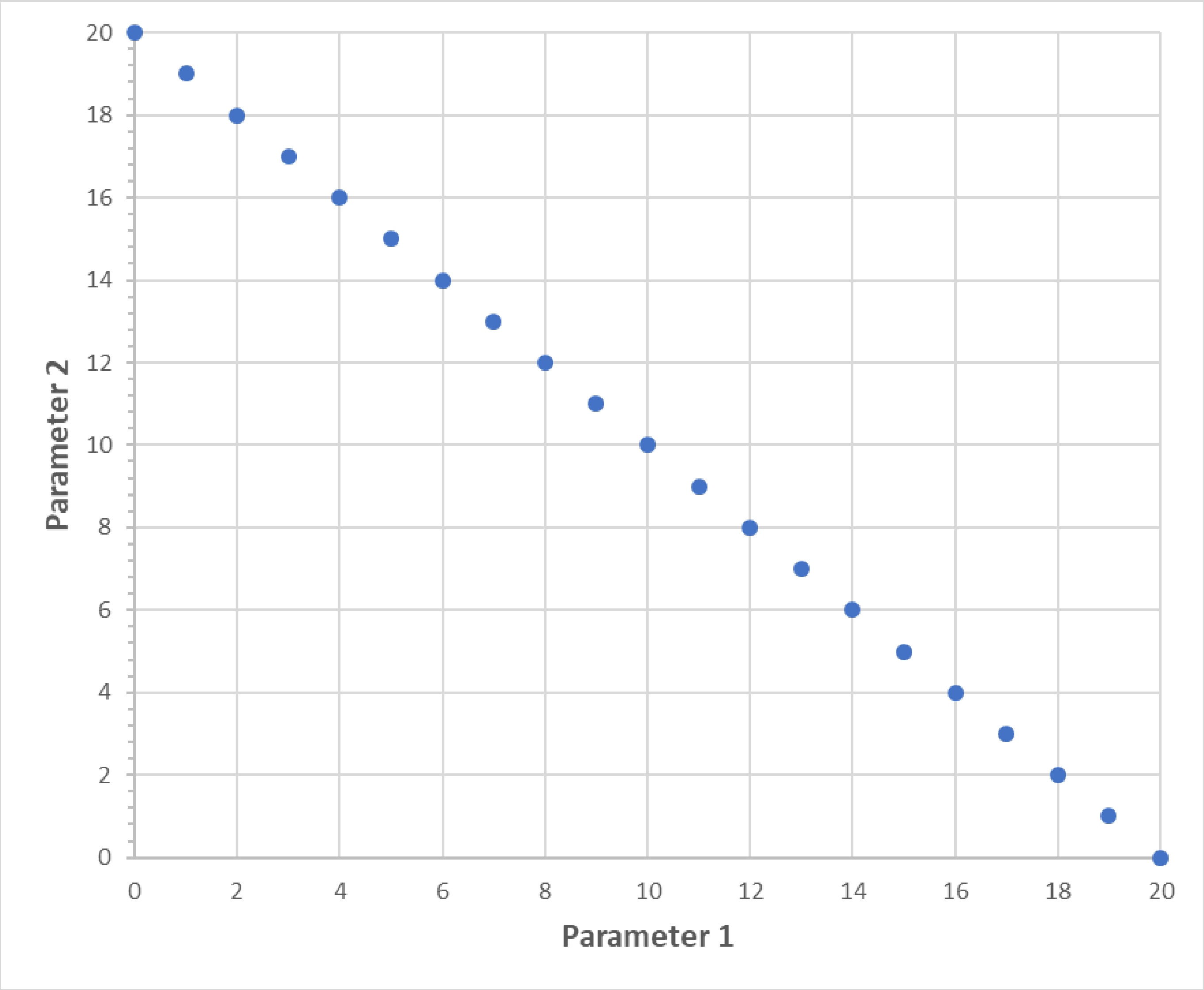
1) Stochastic Gradient Boosting
   - Essentially combining elements of bagging and random forests (subsampling samples or features) to a boosted ensemble

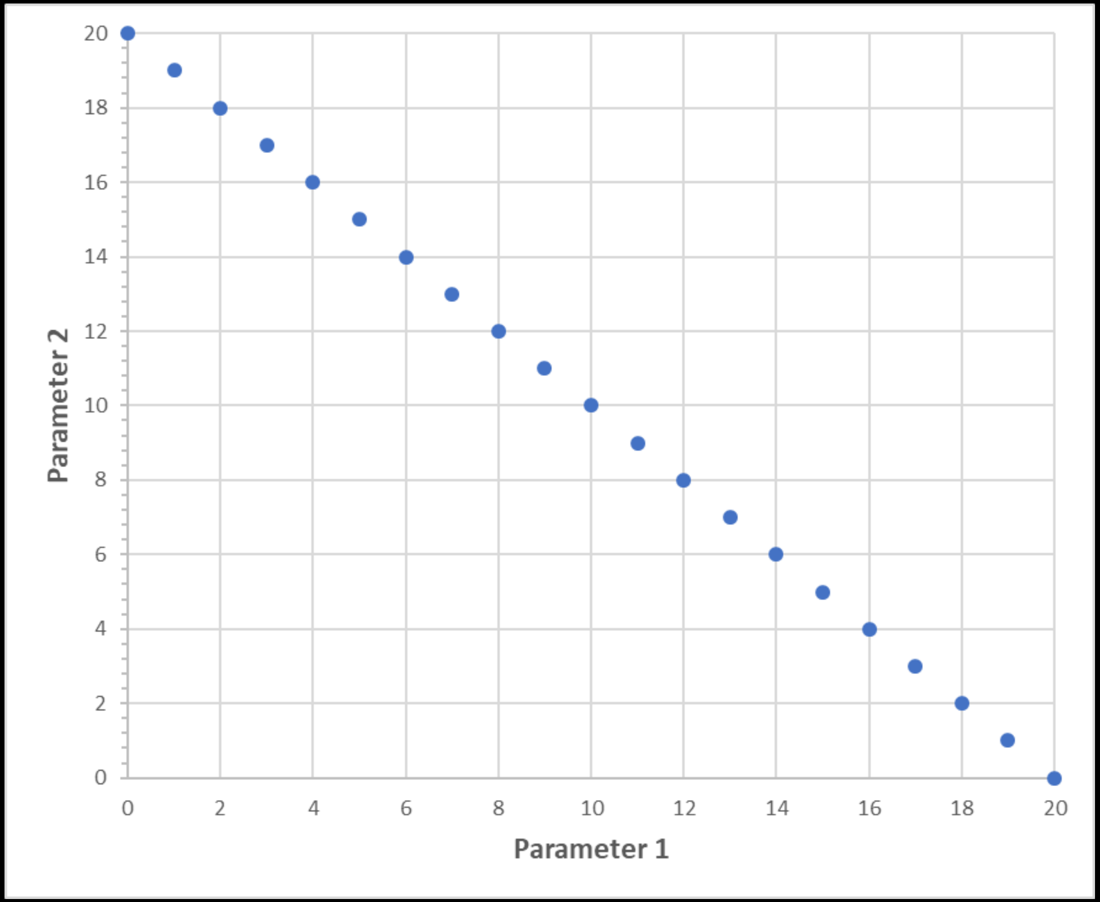
2) XGBoost
   - *Extreme* gradient boosting

XGBoost

- Used to win a number of recent Kaggle competitions
- Builds on gradient boosting, by adding some computational tricks and parallelizing processing, as well as methods for dealing with missing data
- Separate Python library for this, not part of Scikit or Spark

XGBoost – Grid Search





How might we search the parameter space in a "smart" way?

Special Topic: Grid Search

- Figuring out the correct parameters for your model is fundamental problem in data science and machine learning, and often just as important as the data and model choice itself
- Grid Search is available in both Scikit and Spark
- You may also hear this referred to as *hyperparameter optimization*
- In reality, with large datasets or more complex models, you may be limited in what you can do here computationally

Final Project

1) Proposals are due in a couple weeks

2) Final Project is 35% of your total grade

Potential Dataset links:

1. Kaggle datasets - https://www.kaggle.com/datasets
2. UCI dataset repo - https://archive.ics.uci.edu/ml/datasets.html
3. Google dataset search - https://toolbox.google.com/datasetsearch

For next week

1) Paper Review #1 is due sometime this week
2) HW2 is due before next class by 6pm (Chicago local time)
3) Keep the project proposal in the back of your mind
4) Posting the coding templates next week (Scikit, Spark)