Decision Trees, OHE, Hashing, CTR Prediction

James G. Shanahan [2]

Assistants: Liang Dai[1, 3]

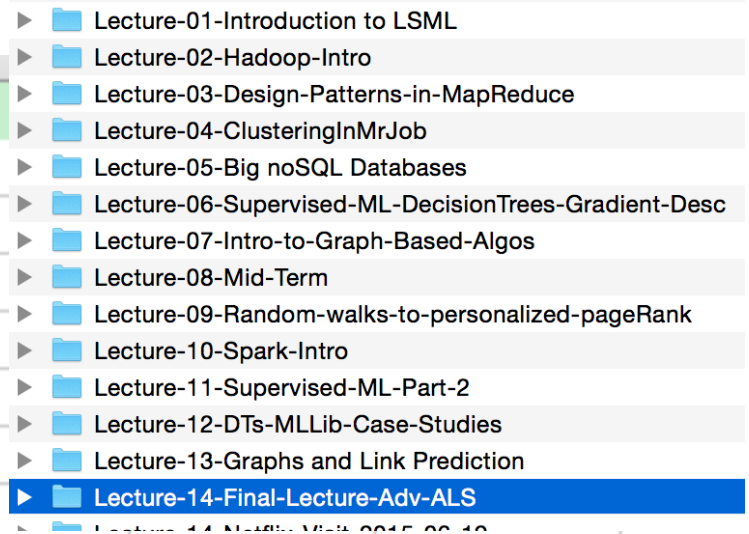
[1]NativeX, [2]iSchool UC Berkeley, CA, [3]UC Santa Cruz

EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com

Live Session #12

April 5, 2016

13 Lectures, 1 Midterm, End of term exam, plus weekly homework and projects



Semester Week	Week in 2016	Monday		
1	Week 3	1/11/16		
2	Week 4	1/18/16		
3	Week 5	1/25/16		
4	Week 6	2/1/16		
5	Week 7	2/8/16		
6	Week 8	2/15/16		
7	Week 9	2/22/16		
8	Week 10	2/29/16	Mid Term Exam	
9	Week 11	3/7/16		
10	Week 12	3/14/16		
Spring Break	Week 13	3/21/16	Spring Break	
11	Week 14	3/28/16		
12	Week 15	4/4/16		
13	Week 16	4/11/16		
14	Week 17	4/18/16		
15	Week 18	4/25/16	End of term	4/29/15
	Week 19	5/2/16		
		5/18/2016 (Wednesday)		Grades Due

Live Session Outline

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
  - Homework HW10, HW11
- **Lecture 11**
  - Spark re-Review
  - Lecture 11 Recap
  - Loss functions
    - Loss term: Understanding loss functions (graphically)
    - Regularization term:
  - SVMs
  - Distributed Gradient descent
  - Logistic regression, diagnostics, priors as a form of regularization
  - Zeppelin notebooks backed by a Spark cluster
- **Wrapup**

HW Schedule and MidTerm

- **HW11 due next Thursday, April 7 at 8AM**
- **HW12 + CTR prediction Project Thursday, April 14 at 8AM**
- **HW13 + CTR prediction Project**
- **HW14?**

Important Links for Week 11

- **Live session Slides**
  - https://www.dropbox.com/s/rq8jgledccu52cj/Lecture-11-Live-Session-Supervised-ML-Part2-2016-03-29.pdf?dl=0
- **Instructions for Peer grading of homework HW**
  - https://www.dropbox.com/s/97m31frthj4ac28/HOMEWORK%20GRADING%20INSTRUCTIONS%20for%20MIDS%20MLS?dl=0
- **Homework HW Folder Questions + Data**
  - HW is a group oriented homework (Data is referenced via dropbox links)
  - https://www.dropbox.com/s/rkxw455jj8ntqxy/MIDS-MLS-HW-11.txt?dl=0
- **Team assignments**
  - https://docs.google.com/spreadsheets/d/1ncFQl5Tovn-16slD8mYjP_nzMTPSfiGeLLzW8v_sMjg/edit?usp=sharing
- **Please submit your homeworks (one per team) going forward via this form (and not thru the ISVC):**
  - https://docs.google.com/forms/d/1ZOr9RnIe_A06AcZDB6K1mJN4vrLeSmS2PD6Xm3eOiis/viewform?usp=send_form

HW11 Questions

- **Please complete HW11 in Zeppelin Notebooks**
- **Problems**
  - HW11 Broadcast versus Caching in Spark
  - HW11 Loss Functions
  - SVMs
  - Optional
    - Distributed Perceptron
    - Distributed Adatron

Live Session Outline: Week 12

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
  - Housekeeping:
    - homework schedule
- **HW11**
- **Lecture 12**
  - Decision Trees
- **Categorical feature encoding**
  - (OHE, Hashing)
- **Digital advertining**
  - Project: CTR Prediction
- **Submitting a Spark Job (Locally and to a cluster on AWS)**
- **Finish RECORDING (bonus points for reminding me!)**

HW Schedule and MidTerm

- **HW9 revision due, Tuesday, November 10, at 6PM as normal)**
  - 9.4 and 9.5 are optional
- **HW 10 will be due Tuesday, November 17 at 6PM**
- **NOTE:**
  - No class on week of Wednesday, November 11 (Immersion week)
  - 3 HWs left Week 11, 12, 13 HW
- **HW11 due Tuesday, November 24 at 6PM**
- **HW12 + CTR prediction Project**
- **HW13 + CTR prediction Project**
- **HW14?**

Live Session Outline: Week 12

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
  - Housekeeping:
    - homework schedule
- **HW11**
- **Lecture 12**
  - Decision Trees
- **Categorical feature encoding**
  - (OHE, Hashing)
- **Digital advertining**
  - Project: CTR Prediction
- **Submitting a Spark Job (Locally and to a cluster on AWS)**
- **Finish RECORDING (bonus points for reminding me!)**

Unit 12 | Decision Trees

Hide Contents

Decision Trees

- Decision tree learning learns a decision tree as a predictive model
- DT maps observations about an item to conclusions about the item's target value.
- Decision tree learning is a supervised machine learning approach whose goal is to recursively partition the world into homogenous zones, i.e., zones where the target value is homogenous (constant)

Decision Tree Approach

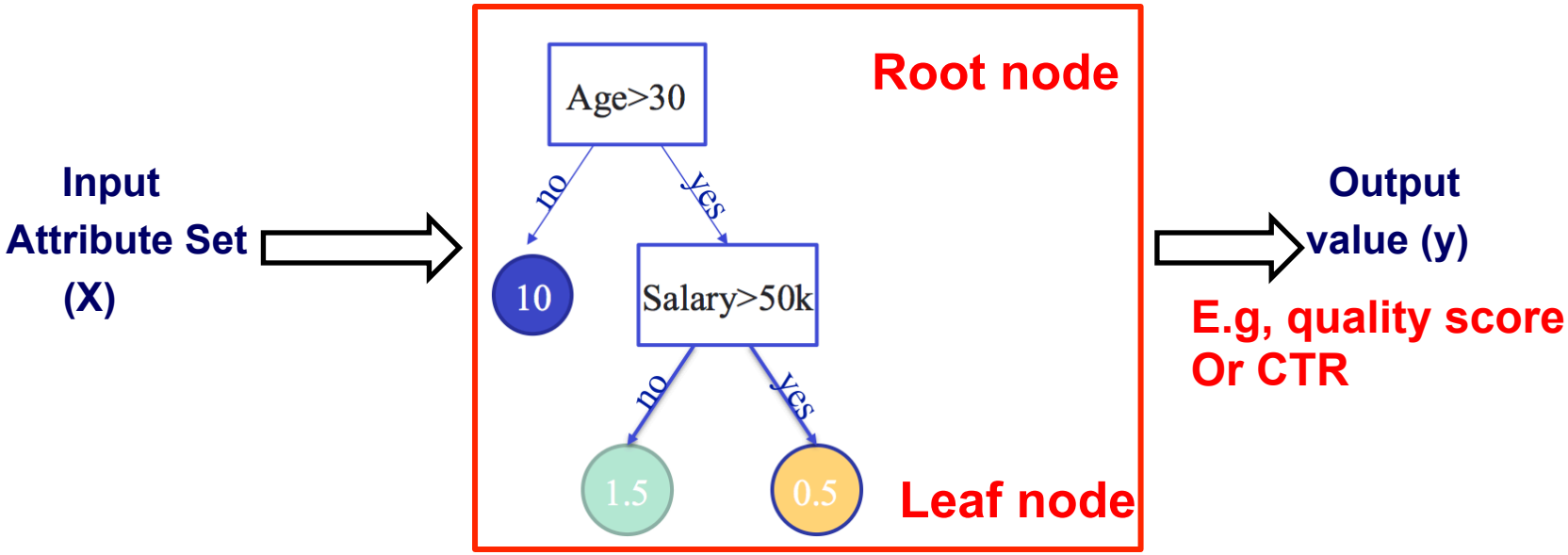
- **A decision tree represents a hierarchical segmentation of the data**
- **The original segment is called the *root node* and is the entire data set**
- **The root node is partitioned into two or more segments by applying a series of simple rules over an input variables**
  - For example, risk = low, risk = not low
  - Each rule assigns the observations to a segment based on its input value

Decision Tree Approach

- **Each resulting segment can be further partitioned into sub-segments, and so on**
  - For example risk = low can be partitioned into income = low and income = not low
- **The segments are also called *nodes*, and the final segments are called *leaf nodes* or *leaves***

Decision Trees

- A decision tree represents a hierarchical segmentation of the world/data
- DT maps observations about an item to conclusions about the item's target value.



The diagram show the classification as task of mapping an input attribute set x into its class label y

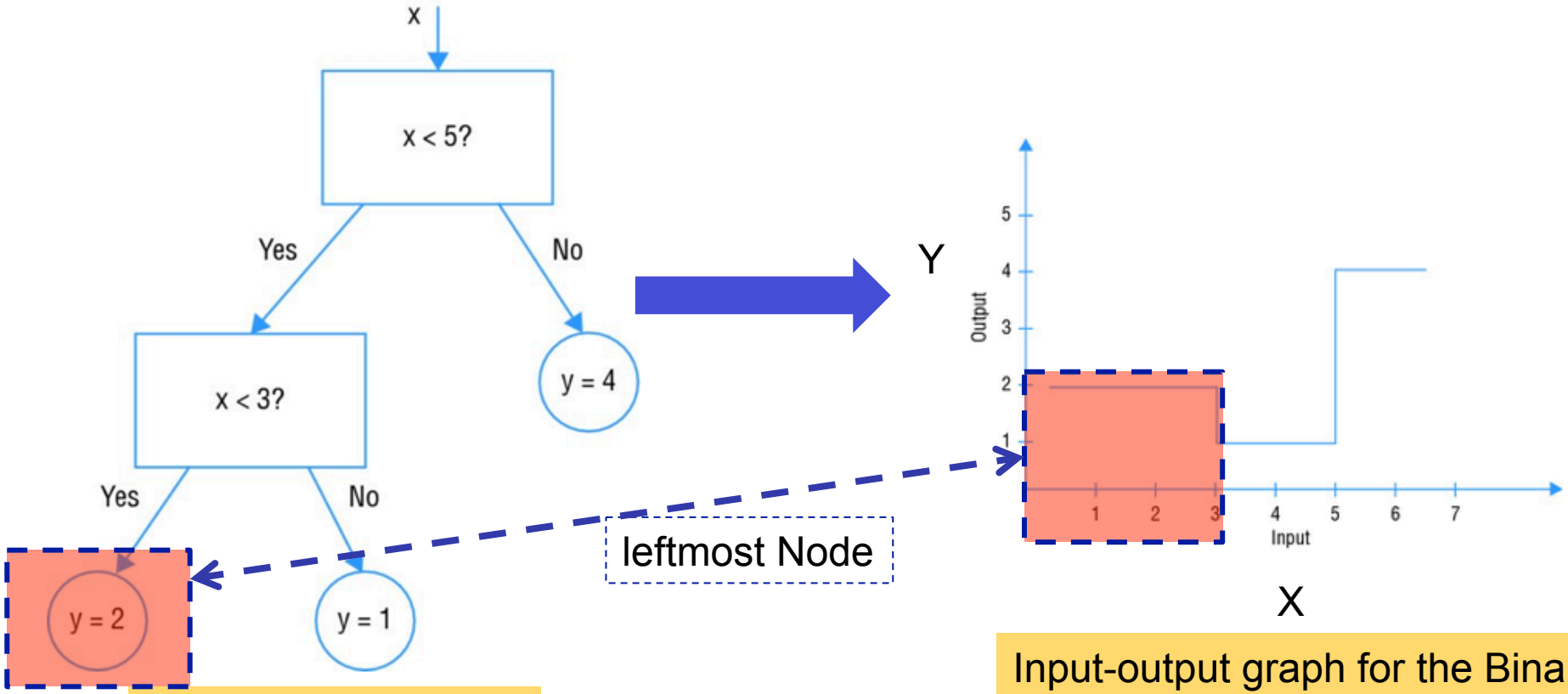
Question on DT

- DTs for regression? For classification?
- Better at regression than classification?

Binary Decision Tree

Slide improvisation

- Classification versus regression



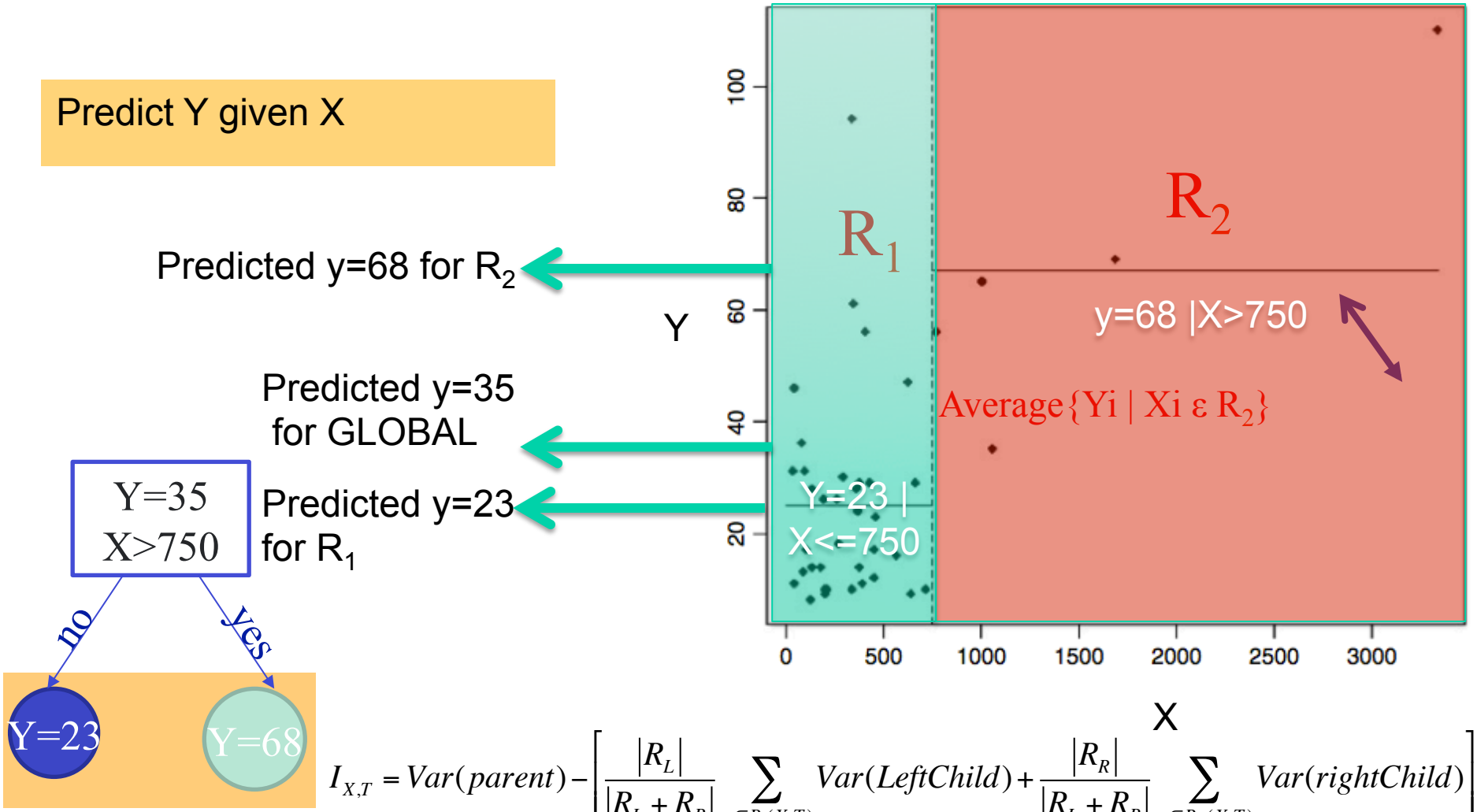
Binary Decision Tree

Input-output graph for the Binary Decision Tree example

Mean value prediction for regression trees with squared error loss

- **Partition the space into *M* regions: R_1, R_2**

Predict Y given X

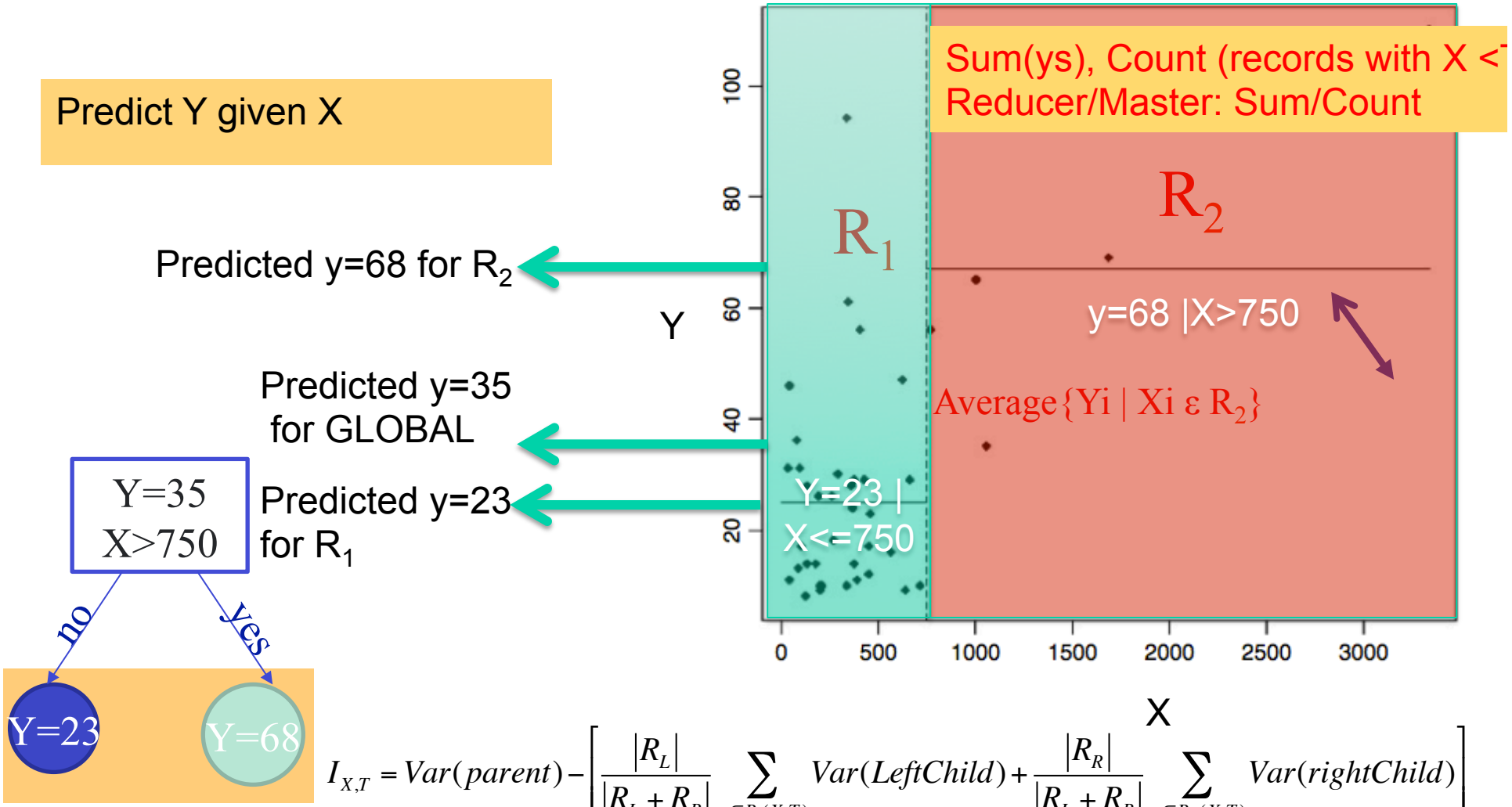


$$I_{X,T} = Var(parent) - \left[\frac{|R_L|}{|R_L + R_R|} \sum_{x_i \in R_l(X,T)} Var(LeftChild) + \frac{|R_R|}{|R_L + R_R|} \sum_{x_i \in R_R(X,T)} Var(rightChild) \right]$$

- **How can we do DT learning at scale?**
- **Commutative, associative?**

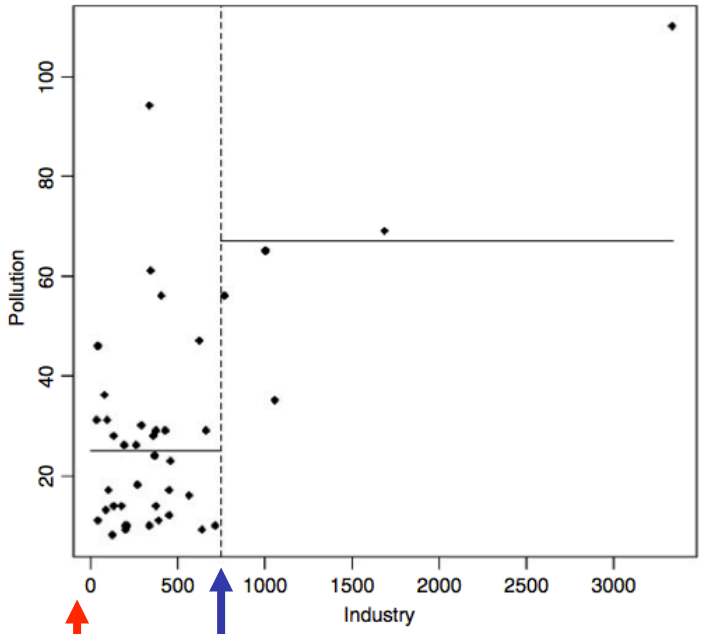
Mean value prediction for regression trees with squared error loss

- **Partition the space into *M* regions: R_1, R_2**



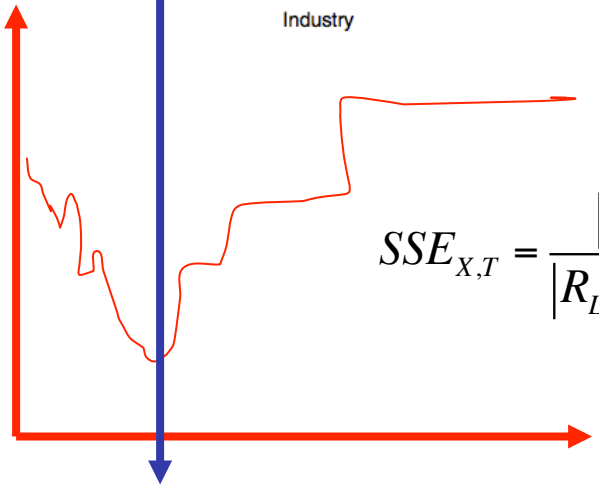
$$I_{X,T} = Var(parent) - \left[\frac{|R_L|}{|R_L + R_R|} \sum_{x_i \in R_l(X,T)} Var(LeftChild) + \frac{|R_R|}{|R_L + R_R|} \sum_{x_i \in R_R(X,T)} Var(rightChild) \right]$$

Regression Tree: Find optimal first split point



Find optimal first split point to split global region into left and right regions

$SSError_T$ As the split point of Industry is varied



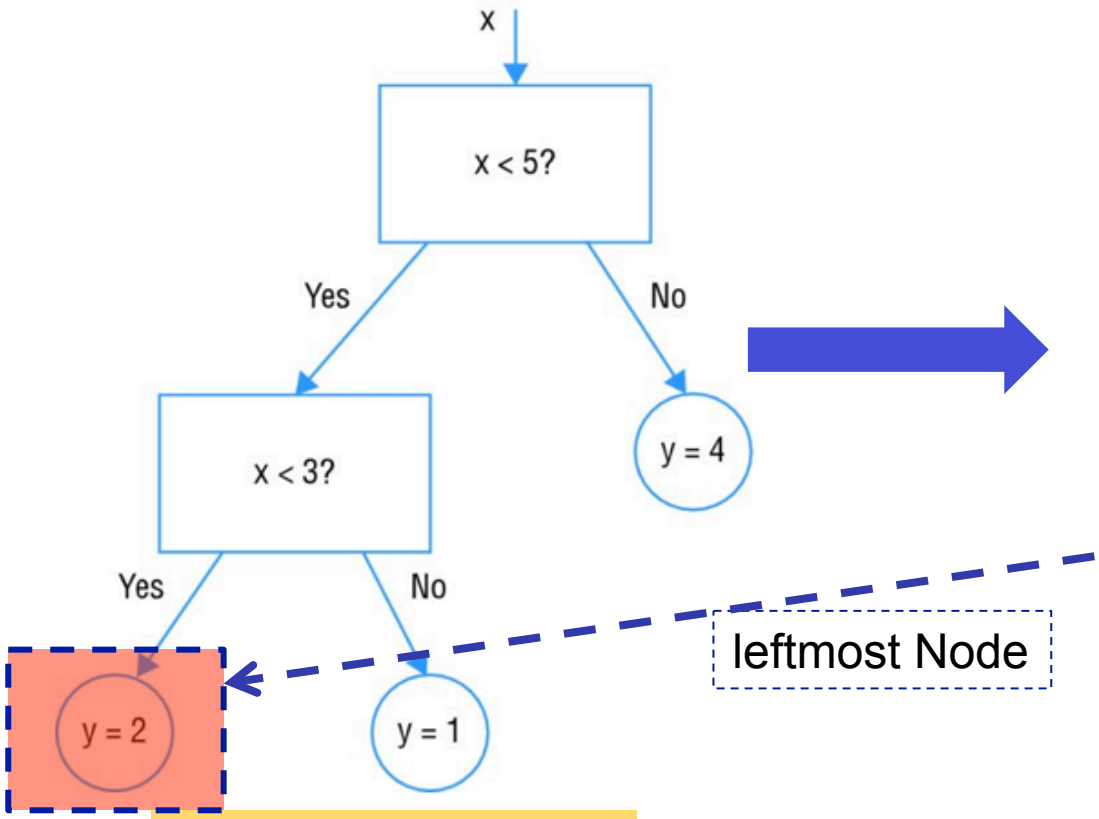
$$SSE_{X,T} = \frac{|R_L|}{|R_L + R_R|} \sum_{x_i \in R_l(X,T)} (y_i - \mu_{R_l})^2 + \frac{|R_R|}{|R_L + R_R|} \sum_{x_i \in R_R(X,T)} (y_i - \mu_{R_R})^2$$

Optimal Split point

Binary Decision Tree: Missing Data

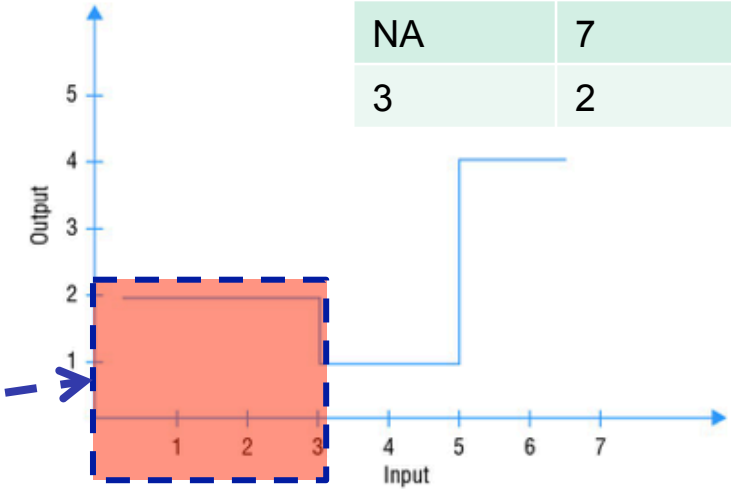
Replace by average
Delete records with missing values

X	Y
1	2
3	2
5	4
6	4
3	1
NA	7
NA	7
3	2



Binary Decision Tree

leftmost Node

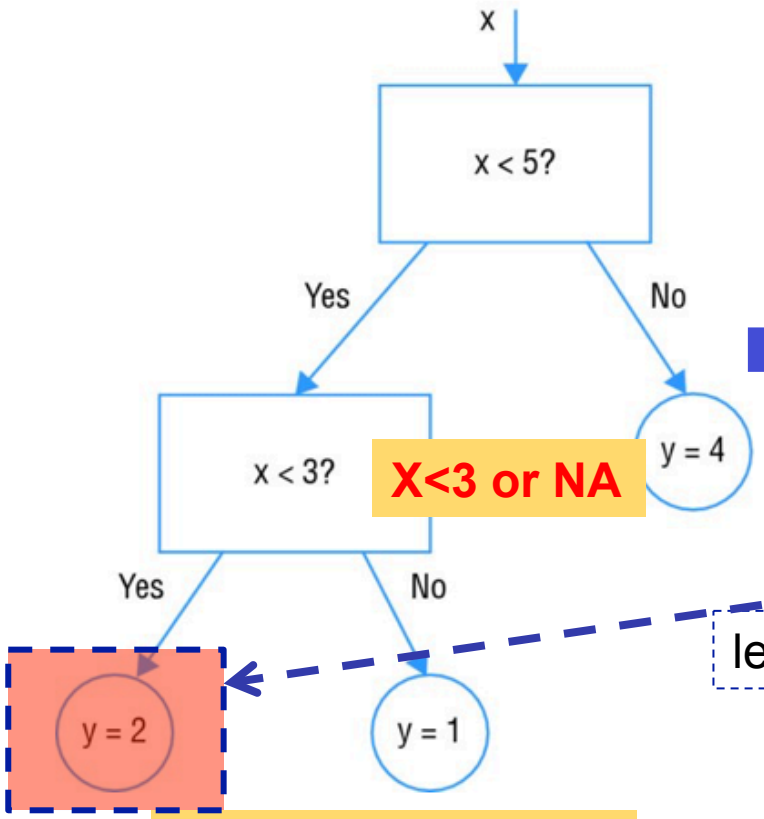


Input-output graph for the Binary Decision Tree example

Binary Decision Tree: Missing Data

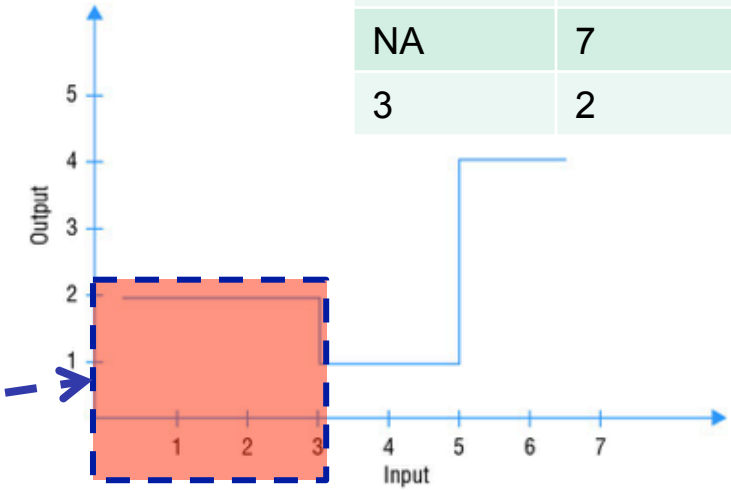
X	Y
1	2
3	2
5	4
6	4
3	1
NA	1.7
NA	7
3	2

X==NA



Binary Decision Tree

leftmost Node



Input-output graph for the Binary Decision Tree example

Classification versus regression Trees

- Tree models where the target variable can take a finite set of values are called classification trees.
- In these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels.

- Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.

Dtrees are desirable: plug and play

- One of the most popular approaches to ML in practice
- Can handle numeric, categorical, nominal features
- No preprocessing required, no standarization
- Handles Missing values naturally and Nas does not affect performance metrics
- Interaction features
- Highly Scaleable
- Variable selection
- Excellent performance on a variety of problems
- Off the shelf with very few hyperparameters

Dtrees are desirable: plug and play

1. *Ability to deal with irrelevant inputs.* Since at every node, we scan all the variables and pick the best, trees naturally do variable selection. And, thus, anything you can measure, you can allow as a candidate without worrying that they will unduly skew your results.

   Trees also provide a variable importance score based on the contribution to error (risk) reduction across all the splits in the tree (see Chapter 5).

2. *No data preprocessing needed.* Trees naturally handle numeric, binary, and categorical variables.

   Numeric attributes have splits of the form $x_j < cut\_value$; categorical attributes have splits of the form $x_j \in \{value1,\ value2, \ldots\}$.

   Monotonic transformations won't affect the splits, so you don't have problems with input outliers. If $cut\_value = 3$ and a value x_j is 3.14 or 3,100, it's greater than 3, so it goes to the same side. Output outliers can still be influential, especially with squared-error as the loss.

3. *Scalable computation.* Trees are very fast to build and run compared to other iterative techniques. Building a tree has approximate time complexity of $O\ (nN \log N)$.

4. *Missing value tolerant.* Trees do not suffer much loss of accuracy due to missing values.

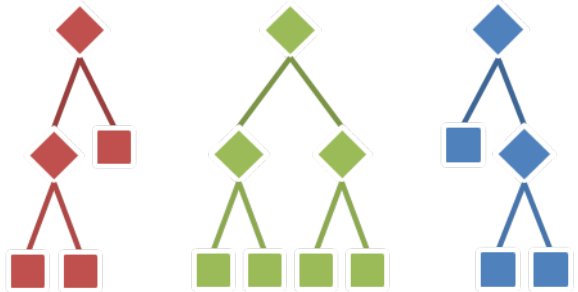
   Some tree algorithms treat missing values as a separate categorical value. CART handles them via a clever mechanism termed "surrogate" splits (Breiman et al., 1993); these are substitute splits in case the first variable is unknown, which are selected based on their ability to approximate the splitting of the originally intended variable.

   One may alternatively create a new binary variable $x_j\_is\_NA$ (not available) when one believes that there may be information in x_j's being missing – i.e., that it may not be "missing at random."

5. *"Off-the-shelf" procedure: there are only few tunable parameters.* One can typically use them within minutes of learning about them.

- **In practice we generally use ensembles of decision trees**

Scaling Up Decision Tree Ensembles

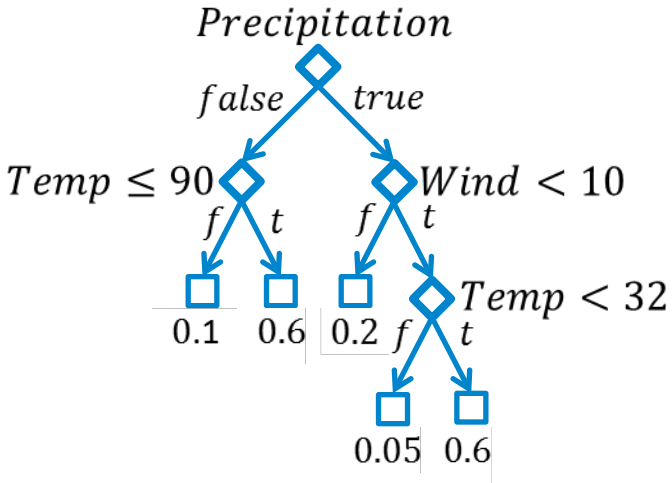


Misha Bilenko (Microsoft)
with Ron Bekkerman (LinkedIn) and John Langford (Yahoo!)

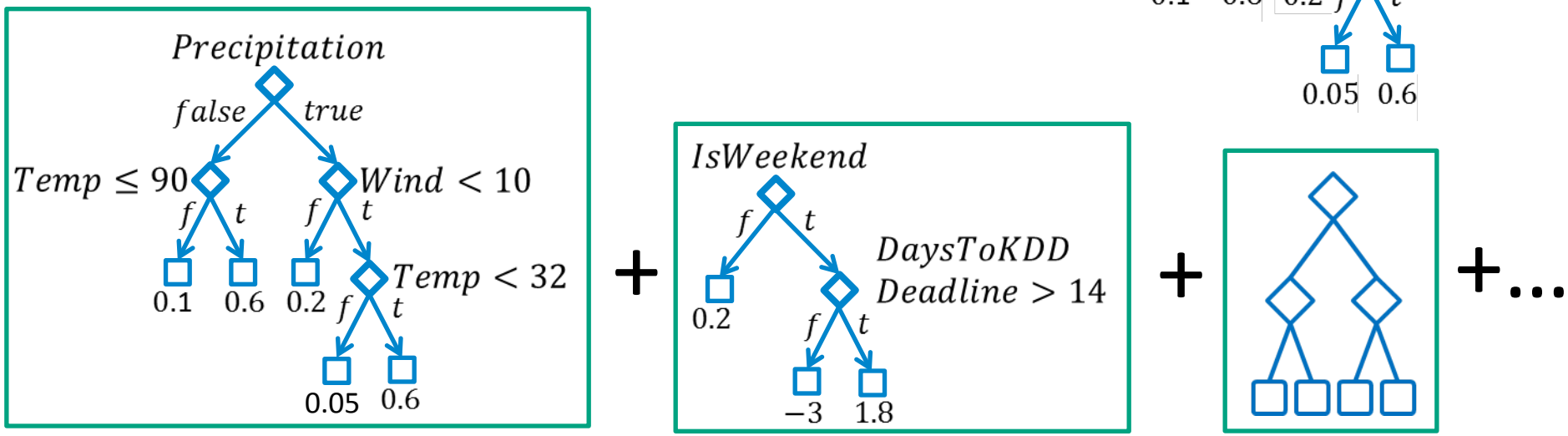
http://hunch.net/~large_scale_survey

Tree Ensembles: Basics

- **Rule-based prediction is natural and powerful (non-linear)**
  - *Play outside: if no rain and not hot, or if snowing but not windy.*
- **Trees hierarchically encode rule-based prediction**
  - Nodes test features to branch
  - Leaves produce predictions
  - Regression trees: numeric outputs



- **Ensembles combine tree predictions**



Tree Ensemble Zoo

- **Different models can define different types of:**
  - Combiner function: voting vs. weighting
  - Leaf prediction models: constant vs. regression
  - Split conditions: single vs. multiple features

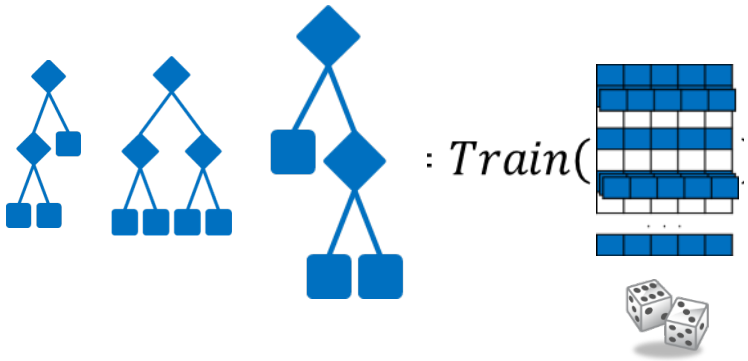
- **Examples (small biased sample, some are not tree-specific)**
  - Boosting: AdaBoost [FS97], LogitBoost [F+00], GBM/MART [F01b], BrownBoost [F01a], Transform Regression [SUML11-Ch9]
  - Random Forests [B01]: Random Subspaces [H98], Bagging [B98], Additive Groves [S+07], BagBoo [P+10]
  - Beyond regression and binary classification: RankBoost [F+03], abc-mart [L09], GBRank [Z+08], LambdaMART [SUML11-Ch8]

Tree Ensembles Are Rightfully Popular

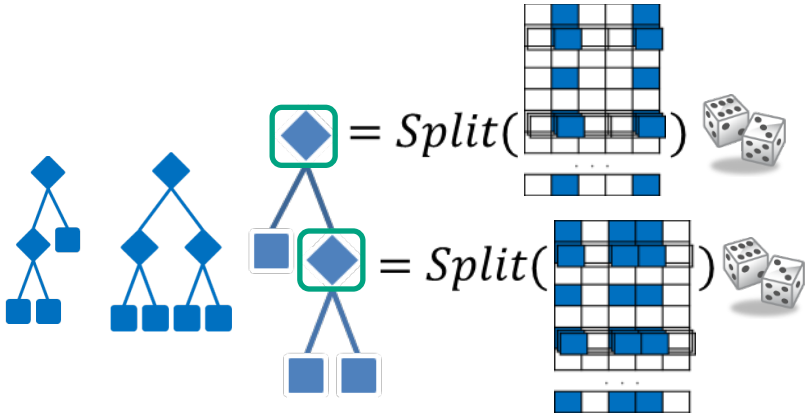
- **State-of-the-art accuracy: web, vision, CRM, bio, …**
- **Efficient at prediction time**
  - Multithread evaluation of individual trees; optimize/short-circuit
- **Principled: extensively studied in statistics and learning theory**
- **Practical**
  - Naturally handle mixed, missing, (un)transformed data
  - Feature selection embedded in algorithm
  - Well-understood parameter sweeps
  - Scalable to extremely large datasets: rest of this section

Naturally Parallel Tree Ensembles

- **No interaction when learning individual trees**
  - **Bagging**: each tree trained on a bootstrap sample of data
  - **Random forests**: bootstrap plus subsample features at each split
  - For large datasets, local data replaces bootstrap -> **embarrassingly parallel**



Bagging tree construction

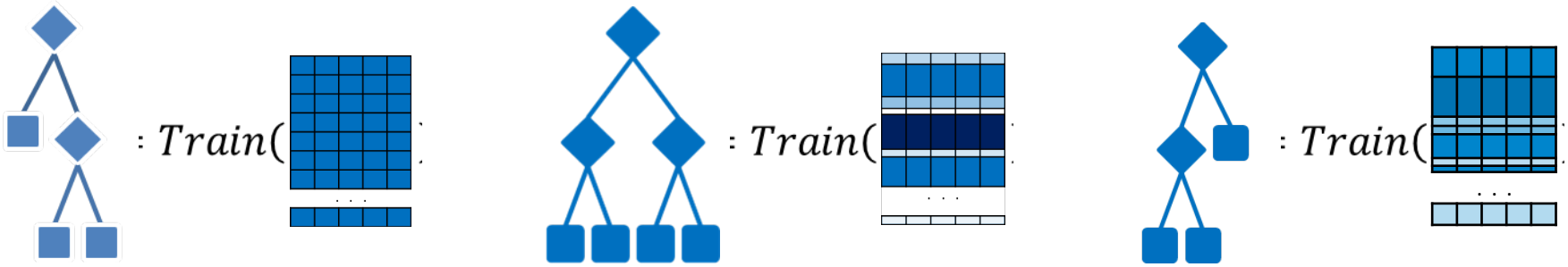


Random forest tree construction

Boosting: Iterative Tree Construction

"Best off-the-shelf classifier in the world" – Breiman

- Reweight examples for each subsequent tree to focus on **errors**



- Numerically: gradient descent in function space
  - Each subsequent tree approximates a step in $-\frac{\partial L}{\partial f}$ direction

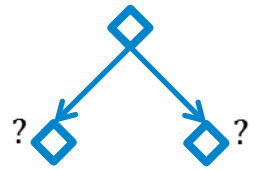
Target for mth model

  - Recompute *target labels* $y^{(m)} = -\left[\frac{\partial L(y,f(x))}{\partial f(x)}\right]_{f(x)=f^{(m-1)}(x)}$
  - Logistic loss: $L(y, f(x)) = \log(1 + \exp(-yf(x))$ $y^{(m)} = \frac{y}{1+\exp(yf(x))}$
  - Squared loss: $L(y, f(x)) = \frac{1}{2}(y - f(x))^2$ $y^{(m)} = y - f(x)$

Efficient Tree Construction

- **Boosting is iterative: scaling up = parallelizing tree construction**

- **For every node: pick best feature to split**
  - For every feature: pick best split-point
    - For every potential split-point: compute gain
      - For every example in current node, add its gain contribution for given split



- **Key efficiency: limiting+ordering the set of considered split points**
  - Continuous features: discretize into bins, splits = bin boundaries
  - Allows computing split values in a single pass over data

Boosting Fits an Additive Model

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)$$

where $b(x; \gamma_m) = G_m(x) \in \{-1, 1\}$ (for Adaboost) is like a set of elementary "basis functions"

This model is fit by minimizing a loss function L averaged over the training data set:

$$\min_{\{\beta_m, \gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i; \gamma_m)\right)$$

Source http://www.stat.ucl.ac.be/

Forward Stagewise Additive Modeling

- **An approximate solution to the minimization problem is obtained via forward stagewise additive modeling (greedy algorithm)**

1. Initialize $f_0(x) = 0$
2. For $m = 1$ to M
   - Compute
     $$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$$
   - Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

Source http://www.stat.ucl.ac.be/

Gradient Boosted Decision Trees

The next seminal work in the theory of Boosting came with the generalization of AdaBoost to any differential loss function. Friedman's Gradient Boosting algorithm (Friedman, J., 2001), with the default values for the algorithm parameters, is summarized in Table 4.8.

Table 4.8: Gradient Boosting as an ISLE-based algorithm.

- General $L(y,\hat{y})$ and y
- $\upsilon = 0.1$
- $\eta = N/2$
- $T_m(\mathbf{x})$: any "weak" learner
- $c_o = \arg\min_c \sum_{i=1}^{N} L(y_i, c)$
- $\{c_m\}_1^M$: "shrunk" sequential partial regression coefficients

$$F_0(\mathbf{x}) = c_0$$

For $m = 1$ to M {

$$(c_m, \mathbf{p}_m) = \arg\min_{c,\mathbf{p}} \sum_{i \in S_m(\eta)} L\left(y_i, F_{m-1}(\mathbf{x}_i) + c \cdot T(\mathbf{x}_i;\mathbf{p})\right)$$

$$T_m(\mathbf{x}) = T(\mathbf{x};\mathbf{p}_m)$$

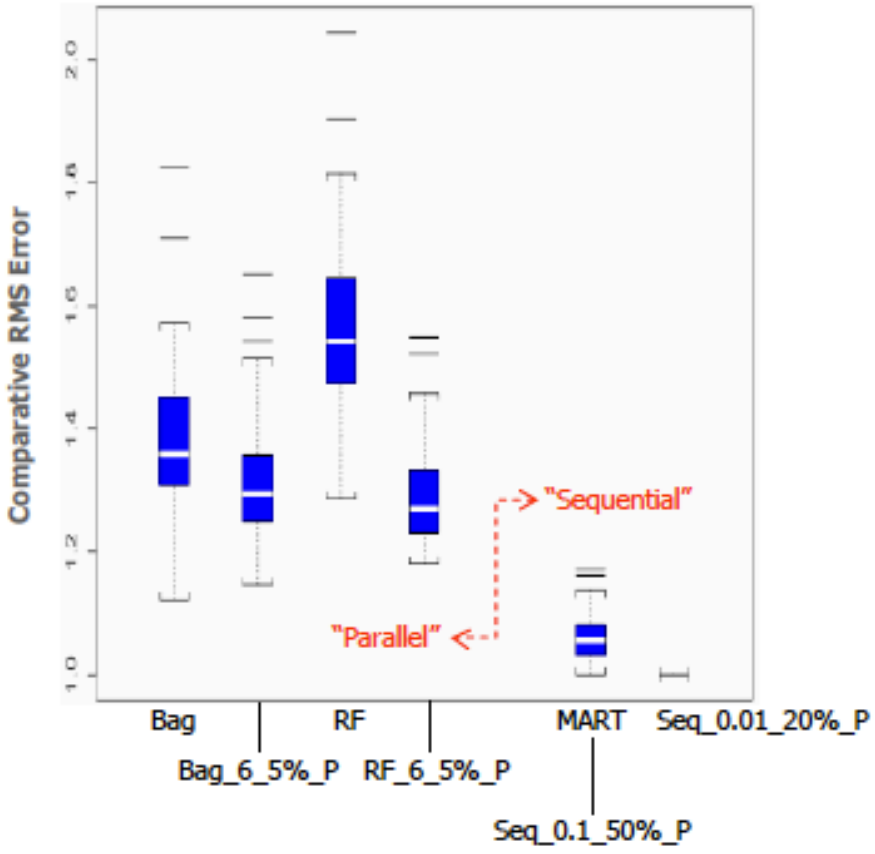
$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \upsilon \cdot c_m \cdot T_m(\mathbf{x})$$

}

write $\{(\upsilon \cdot c_m), T_m(\mathbf{x})\}_1^M$

Ensemble Methods

Parallel vs. Sequential Ensembles

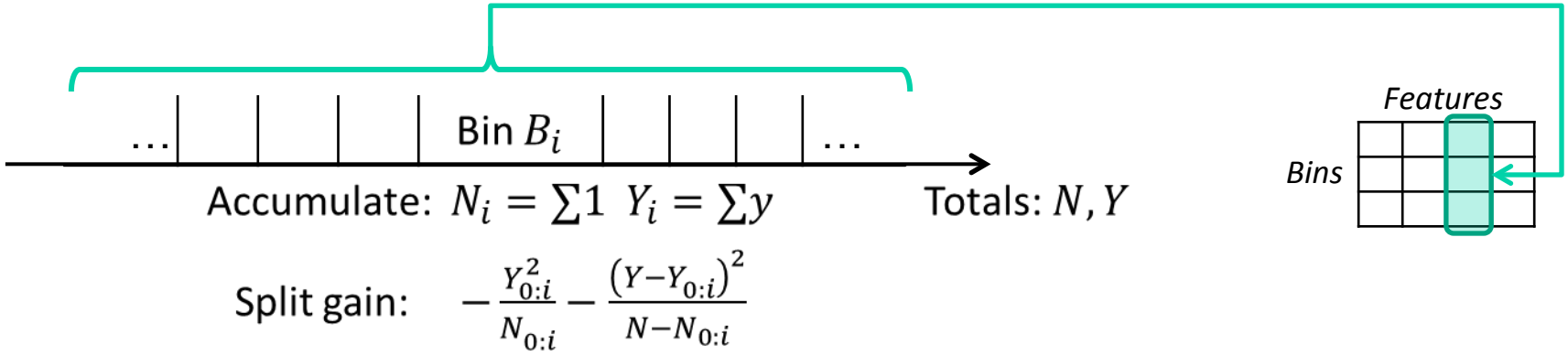


- Simulation study with 100 different target functions (Popescu, 2005)
- xxx_6_5%_P : 6 terminal nodes trees
  5% samples without replacement
  Post-processing – i.e., using estimated "optimal" quadrature coefficients
- Seq_η_ν%_P : "Sequential" ensemble
  6 terminal nodes trees
  η: "memory" factor
  ν% samples without replacement
  Post-processing

- Sequential ISLE tend to perform better than parallel ones
  - Consistent with results observed in classical Monte Carlo integration

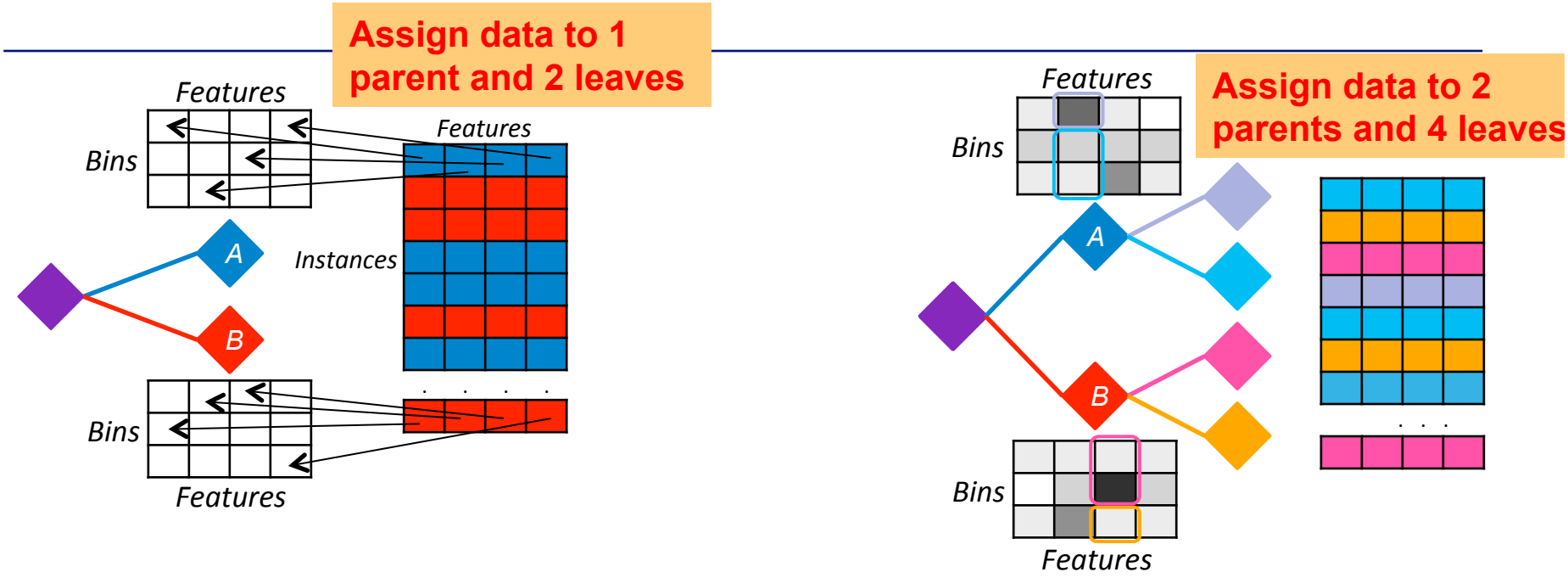
Binned Split Evaluation

- Each feature's range is split into k bins. Per-bin statistics are aggregated in a single pass

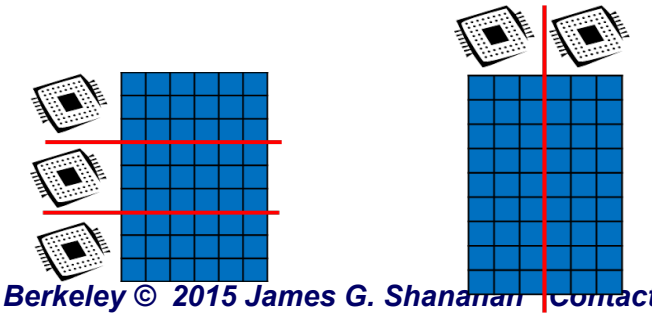


- For each tree node, a two-stage procedure
  - (1) Pass through dataset aggregating node-feature-bin statistics
  - (2) Select split among all (feature,bin) options

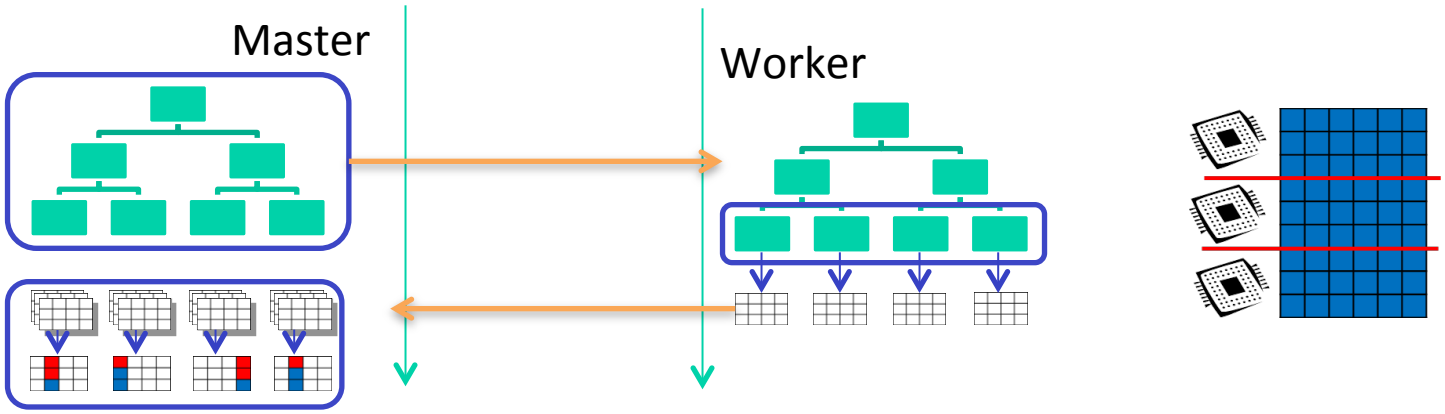
Tree Construction Visualized



- **Observation 1: a single pass is sufficient per tree level**
- **Observation 2: data pass can iterate by-instance or by-feature**
  - Supports horizontally or vertically partitioned data



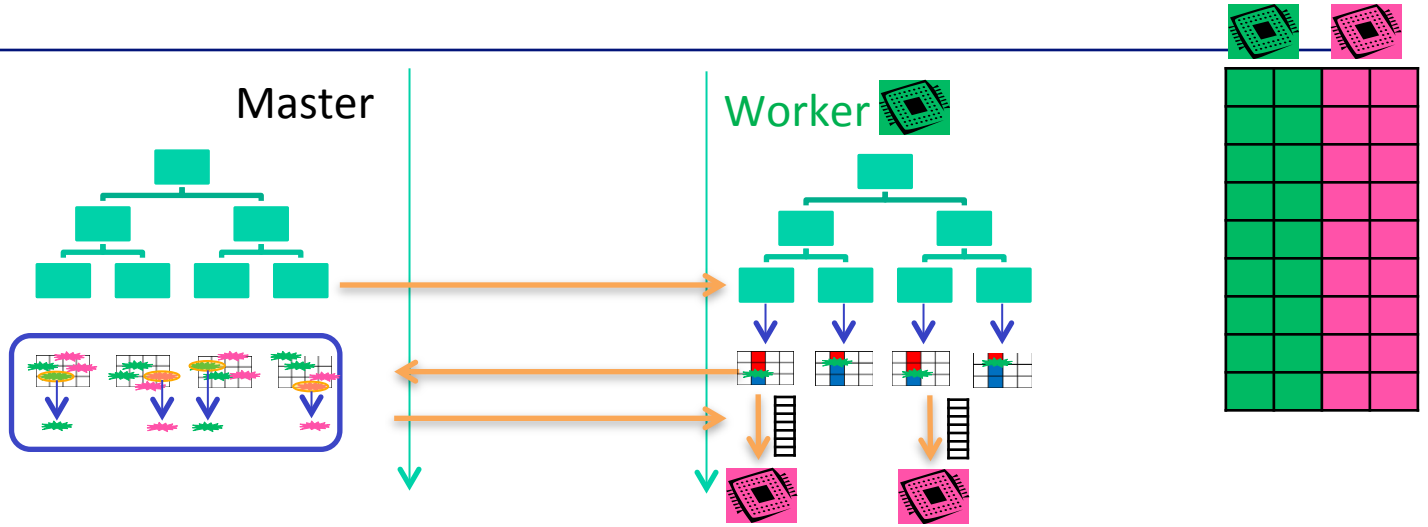
Data-Distributed Tree Construction



- **Master**
  1. Send workers current model and set of nodes to expand
  2. Wait to receive local split histograms from workers
  3. Aggregate local split histograms, select best split for every node

- **Worker**
  2a. Pass through local data, aggregating split histograms
  2b. Send completed local histograms to master

Feature-Distributed Tree Construction



- **Workers maintain per-instance index of current residuals and previous splits**
- **Master**
  1. Request workers to expand a set of nodes
  2. Wait to receive best per-feature splits from workers
  3. Select best feature-split for every node
  4. Request best splits' workers to broadcast per-instance assignments and residuals

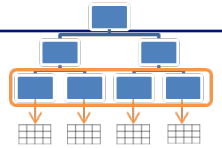
- **Worker**
  2a. Pass through all instances for local features, aggregating split histograms for each node
  2b. Select local features' best splits for each node, send to master

PLANET [SUML11-Ch2]

(Parallel Learner for Assembling Numerous Ensemble Trees)

- **Platform: MapReduce (Google), RPC (!)**
- **Data Partitioning: Horizontal (by-instance)**
- **Binning: during initialization**
  - Single-pass approximate quantiles via [Manku et al. '98]
- **Modulate between distributed and single-machine tree construction**
  - *MapReduceQueue*: distributed tree construction
  - *InMemoryQueue*: single-worker tree construction (when node small enough)

PLANET: ExpandNodes Mapper



- Go through examples, aggregating summaries $(\sum y, \sum 1)$ for every:
  - Node: total statistics for the node (applies for all features)
  - Split: for every feature-bin

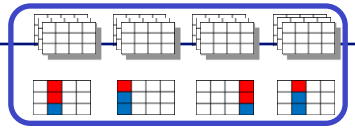
$Mapper.Map(\langle x, y\rangle, model\ M, \text{nodes } N)$

1: $n = \text{TraverseTree(M, } \mathbf{x})$
2: **If** $n \in N$ **then**
3: $\quad \text{agg\_tup}_n \leftarrow y$
4: $\quad$ **For each** $X \in \mathcal{X}$ **do**
5: $\quad\quad v = \text{Value on } X \text{ in } \mathbf{x}$
6: $\quad$ **For each** Split point s of X s.t. $s < v$ **do**
7: $\quad\quad T_{n,X}[s] \leftarrow y$

$Mapper.Finalize()$

1: **For each** $n \in N$ **do**
2: $\quad \text{Output to all reducers}(n, \text{agg\_tup}_n)$
3: $\quad$ **For each** $X \in \mathcal{X}$ **do**
4: $\quad$ **For each** Split point s of X **do**
5: $\quad\quad \text{Output}((n, X, s), T_{n,X}[s])$

PLANET: ExpandNodes Reducer



- **Receives (presorted)**
  - (a) Totals for every node
  - (b) Feature-split histogram for every node
- **Adds up histograms, finds best split among all possible ones.**
- **Emits best split found for given node**

Input: Key k,Value Set V

1: **If** $k == n$ **then**
2: *// Aggregate agg_tup$_n$'s from mappers by pre-sorting.*
3: $\text{agg\_tup}_n = \text{Aggregate}(V)$
4: **Else** // $k == n, X, s$
5: *// Split on ordered feature*
6: $\text{agg\_tup}_{left} = \text{Aggregate}(V)$
7: $\text{agg\_tup}_{right} = \text{agg\_tup}_n$ - agg_tup_{left}
8: $\text{UpdateBestSplit}(S[n], X, s, \text{agg\_tup}_{left}, \text{agg\_tup}_{right})$

PLANET: Master (Controller)

- **Master (Controller)**
  - Creates and sends jobs to *MRQueue* or *InMemoryQueue*
  - Aggregates best splits for each node, updates model
- **Engineering is non-trivial, impactful**
  - MapReduce per-worker setup and tear-down costs are high
    - Forward-schedule: pre-allocate Mappers and Reducers, standby
    - Controller uses RPC to send jobs
  - Categorical attributes
    - Specially handled (different MR keys; still linear using the "Breiman trick")
    - Evaluation speedup: fingerprint, store signature with each node

Ensemble Learning in MapReduce
Simple modification

predictions (z). For a given training record $(\mathbf{x}, y)$, the residual prediction for tree k is $z = y - F_{k-1}(\mathbf{x})$ for a regression problem, and $z = y - \frac{1}{1+exp(-F_{k-1}(\mathbf{x}))}$ for a classification problem. The boosting process is initialized by setting F_0 as some aggregate defined over the Y values in the training dataset. Abstracting out the details, we need three main features in our framework to build boosted models.

- Building multiple trees: Extending the Controller to build multiple trees is straightforward. Since the Controller manages tree induction by reducing the process to repeated node expansion, the only change necessary for constructing a boosted model is to push the root node for tree k onto the MR after tree $k-1$ is completed.

- Residual computation: Training trees on residuals is simple since the current model is sent to every Map Reduce job in full. If the mapper decides to use a training record as input to a node, it can compute the current model's prediction, and hence the residual.

- Sampling: Each tree is built on a sample of D^*. Dappers compute a hash of a training record's id and the tree id. Records hashing into a particular range are used for constructing the tree. This hash-based sampling guarantees that the same sample will be used for all nodes in a tree, but different samples of D^* will be used for different trees.

Build multiple trees
Broadcast ensemble to Task nodes for the spliting map-reduce job

Compute residual on the fly in the map-reduce job for spliting

Hash record id and tree id: records hashing in a particular range are used for building the tree

Live Session Outline: Week 12

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
  - Housekeeping:
    - homework schedule
- **HW11**
- **Lecture 12**
  - Decision Trees
- **Categorical feature encoding**
  - (OHE, Hashing)
- **Digital advertining**
  - Project: CTR Prediction
- **Submitting a Spark Job (Locally and to a cluster on AWS)**
- **Finish RECORDING (bonus points for reminding me!)**

Reference Material

- **OHE and Feature Hashing**
  - https://en.wikipedia.org/wiki/Feature_hashing

Styles of (supervised) Machine Learning

- **Machine Learning via Optimization**
  - Gradient descent
  - E.g., Linear regression, SVMs, logistic regression , Neural networks
  - Requires features to be numeric in nature
- **Other approaches**
  - Decision trees
  - Inductive logic programming
  - Can handle numeric and categorical features directly

Optimization in Spark

- http://spark.apache.org/docs/latest/mllib-optimization.html#Choosing-an-Optimization-Method
  - Stochastic Gradient Descent (Minibatch size = minibatchFraction)
  - L-BFGS

The SGD implementation in GradientDescent uses a simple (distributed) sampling of the data examples. We recall that the loss part of the optimization problem (1) is $\frac{1}{n}\sum_{i=1}^{n} L(\mathbf{w};\mathbf{x}_i,y_i)$, and therefore $\frac{1}{n}\sum_{i=1}^{n} L'_{\mathbf{w},i}$ would be the true (sub)gradient. Since this would require access to the full data set, the parameter `miniBatchFraction` specifies which fraction of the full data to use instead. The average of the gradients over this subset, i.e.

$$\frac{1}{|S|}\sum_{i\in S} L'_{\mathbf{w},i},$$

is a stochastic gradient. Here S is the sampled subset of size $|S| =$ `miniBatchFraction` $\cdot n$.

In each iteration, the sampling over the distributed dataset (RDD), as well as the computation of the sum of the partial results from each worker machine is performed by the standard spark routines.

If the fraction of points `miniBatchFraction` is set to 1 (default), then the resulting step in each iteration is exact (sub)gradient descent. In this case there is no randomness and no variance in the used step directions. On the other extreme, if `miniBatchFraction` is chosen very small, such that only a single point is sampled, i.e. $|S| =$ `miniBatchFraction` $\cdot n = 1$, then the algorithm is equivalent to standard SGD. In that case, the step direction depends from the uniformly random sampling of the point.

Machine Learning and Optimization

Logistic Regression Optimization

Regularized

Logistic Regression: Learn mapping ($\mathbf{w}$) that minimizes logistic loss on training data with a regularization term

$$\min_{\mathbf{w}} \sum_{i=1}^{n} \overbrace{\ell_{0/1}\left(y^{(i)} \cdot \mathbf{w}^{\top}\mathbf{x}^{(i)}\right)}^{\text{Training LogLoss}} + \overbrace{\lambda||\mathbf{w}||_2^2}^{\text{Model Complexity}}$$

Data is assumed to be **numerical**!

Similar story for SVMs, Linear regression etc.

How to deal with categorical features?
- **Animal feature: {cat, dog, tiger}**

Versus, say, Decision Tree Learning:
deals with numerical and categorical features naturally (no transformations required)

For optimization-based ML approaches

- **Need to Convert categorical features to Numeric features**
- **Categorical features are common place**

Example: Click-through Rate Prediction

- User features: Gender, Nationality, Occupation, …
- Advertiser / Publisher: Industry, Location, …
- Ad / Publisher Site: Language, Text, Target Audience, …

How to handle non-numeric features?

Option 1: Use methods that support these features

- Some methods, e.g., Decision Trees, Naive Bayes, naturally support non-numerical features
- However, this limits our options

Gaussian NB, multinomial NB

Option 2: Convert these features to numeric features

- Allows us to use a wider range of learning methods
- How do we do this?

Types of non-numeric features

- ..

Categorical Feature

- Has two or more categories
- No intrinsic ordering to the categories
- E.g., Gender, Country, Occupation, Language

Ordinal Feature

- Has two or more categories
- Intrinsic ordering, but no consistent spacing between categories, i.e., all we have is a relative ordering
- Often seen in survey questions, e.g., "Is your health poor, reasonable, good, excellent"

Non-numeric ➔Numeric: Simple Approach

One idea: Create single numerical feature to represent non-numeric one

Ordinal Features:

- Health categories = {'poor', 'reasonable', 'good', 'excellent'}
- 'poor' = 1, 'reasonable' = 2, 'good' = 3, 'excellent' = 4

We can use a single numerical feature that preserves this ordering … but ordinal features only have an ordering and we introduce a degree of closeness that didn't previously exist

Non-numeric ➔Numeric: Simple Approach

One idea: Create single numerical feature to represent nor numeric one

Categorical Features:

- Country categories = {'ARG', 'FRA', 'USA'}
- 'ARG' = 1, 'FRA' = 2, 'USA' = 3
- Mapping implies FRA is between ARG and USA

Creating single numerical feature introduces relationships between categories that don't otherwise exist

Non-numeric ➔Numeric: OHE one-hot-encoding

Another idea (One-Hot-Encoding): Create a 'dummy' feature for each category

Categorical Features:

- Country categories = {'ARG', 'FRA', 'USA'}
- We introduce one new dummy feature for each category
- 'ARG' ⇒ [1 0 0], 'FRA' ⇒ [0 1 0], 'USA' ⇒ [0 0 1]

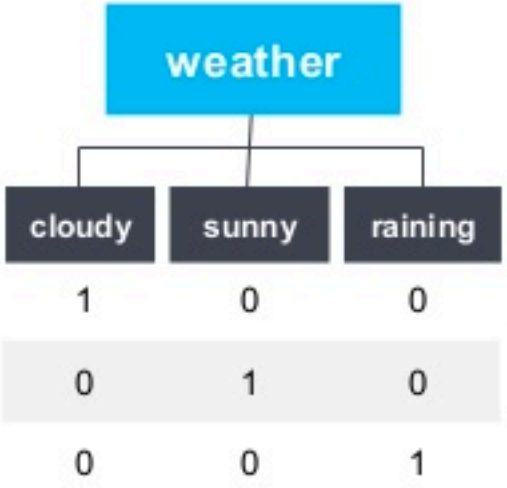
Creating dummy features doesn't introduce spurious relationships

OHE Example:

- **Expand number of variables by cardinality of categorical variable**

day	temperature [F]	weather	bike rentals
3	76	Cloudy	543
4	72	Raining	173
5	78	Sunny	674
6	68	Raining	124

day	temperature [F]	cloudy	sunny	raining	bike rentals
3	76	1	0	0	543
4	72	0	0	1	173
5	78	0	1	0	674
6	68	0	0	1	124



In Natural language processing

NLP: Vectorization

id	message
123	"be happy"
321	"To be or not to be"

id	"be"	"happy"	"be happy"	"to"	"or"	"not"	"to be"	"be or"	"or not"	"not to"
123	1	1	1	0	0	0	0	0	0	0
321	1	0	0	1	1	1	1	1	1	1

OHE: Step 1 of a 2 step process

Step 1: Create OHE Dictionary

Features:

- Animal = {'bear', 'cat', 'mouse'}
- Color = {'black', 'tabby'}
- Diet = {'mouse', 'salmon'}

7 dummy features in total

- 'mouse' category distinct for Animal and Diet features

OHE Dictionary: Maps category to dummy featu

OHE: Step 2 of a 2 step process

Step 2: Create Features with Dictionary

Datapoints:

- A1 = [‘mouse’, ‘black’, -]
- A2 = [‘cat’, ‘tabby’, ‘mouse’]
- A3 = [‘bear’, ‘black’, ’salmon’]

Step 1

OHE Dictionary: Maps each category to dummy feature

- (Animal, ‘bear’) ⇒ 0
- (Animal, ‘cat’) ⇒ 1
- (Animal, ‘mouse’) ⇒ 2
- (Color, ‘black’) ⇒ 3
- …

Step 2

OHE Features:

- Map non-numeric feature to it’s binary dummy feature
- E.g., A1 = [0, 0, 1, 1, 0, 0, 0]

OHE Example

Step 2: Create Features with Dictionary

Datapoints:

- A1 = [‘mouse’, ‘black’, -]
- A2 = [‘cat’, ‘tabby’, ‘mouse’]
- A3 = [‘bear’, ‘black’, ’salmon’]

OHE Features:

- Map non-numeric feature to it’s binary dummy feature
- E.g., A1 = [0, 0, 1, 1, 0, 0, 0]

OHE Dictionary: Maps each category to dummy feature

- (Animal, ‘bear’) ⇒ 0
- (Animal, ‘cat’) ⇒ 1
- (Animal, ‘mouse’) ⇒ 2
- (Color, ‘black’) ⇒ 3
- …

OHE features are sparse

For a given categorical feature only a single OHE feature is non-zero — can we take advantage of this fact?

Dense representation: Store all numbers

- E.g., A1 = [0, 0, 1, 1, 0, 0, 0]

Sparse representation: Store indices / values for non-zeros

- Assume all other entries are zero
- E.g., A1 = [(2,1), (3,1)]

OHE feature vectors are Sparse, and can be long

OHE features are sparse

Sparse Representation

Example: Matrix with 10M observation and 1K features

- Assume 1% non-zeros

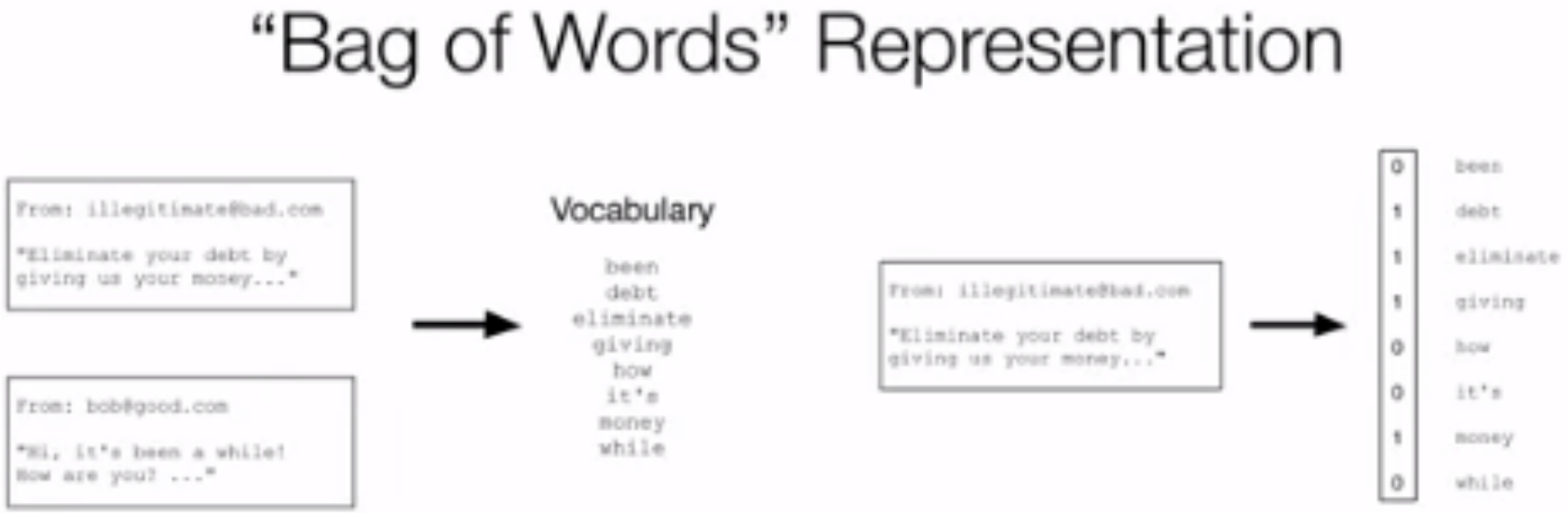
Dense representation: Store all numbers

- Store 10M × 1K entries as doubles ⇒ 80GB storage

Sparse representation: Store indices / values for non-zeros

- Store value and location for non-zeros (2 doubles per entry)
- 50× savings in storage!
- We will also see computational saving for matrix operations

OHE features are sparse: BOW



Represent each document with a vocabulary of words

Over 1M words in English [Global Language Monitor, 2014]

We sometimes consider bigrams or adjacent words (similar idea to quadratic features)

OHE Challenges

- **Feature Hashing**

How to reduce OHE features?

One Option: Discard rare features

- Might throw out useful information (rare ≠ uninformative)
- Must first compute OHE features, which is expensive

Another Option: Feature hashing

- Use hashing principles to reduce feature dimension
- Obviates need to compute expensive OHE dictionary
- Preserves sparsity
- Theoretical underpinnings

Feature Hashing: High-level idea

Hash tables are an efficient data structure for data lookup, and hash functions also useful in cryptography

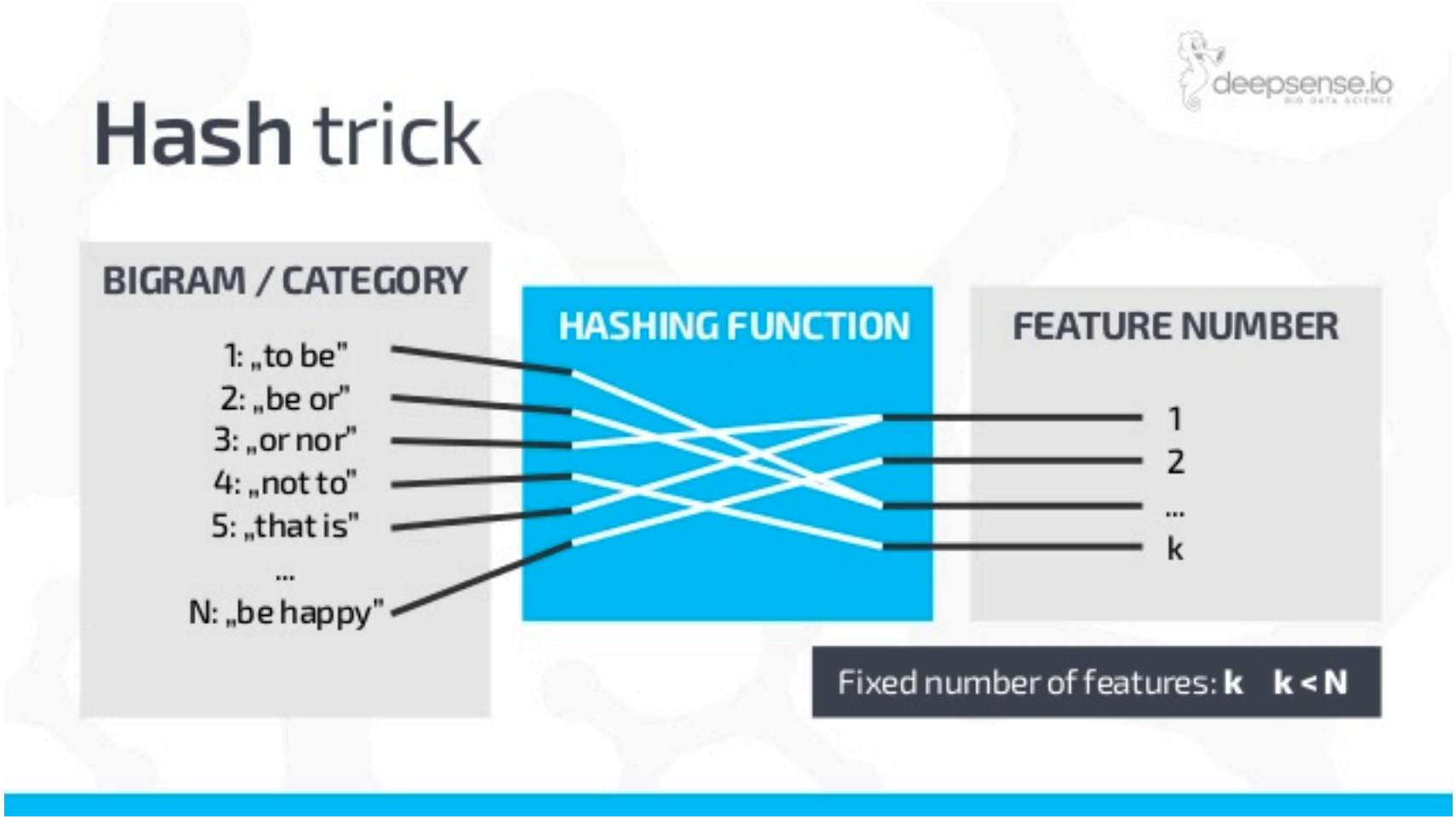
Hash Function: Maps an object to one of m buckets

- Should be efficient and distribute objects across buckets

In our setting, objects are feature categories

- We have fewer buckets than feature categories
- Different categories will 'collide', i.e., map to same bucket
- Bucket indices are hashed features

Dictionary: 1:1 mapping
But with hashing we can have a many to 1 mapping (hash collision)


deepsense.io
Hash trick
BIGRAM / CATEGORY
1: „to be"
2: „be or"
3: „or nor"
4: „not to"
5: „that is"
...
N: „be happy"
HASHING FUNCTION
FEATURE NUMBER
1
2
...
k
Fixed number of features: k k < N

OHE versus feature hashing

Datapoints:

- A1 = [‘mouse’, ‘black’, -]
- A2 = [‘cat’, ‘tabby’, ‘mouse’]
- A3 = [‘bear’, ‘black’, ’salmon’]

OHE Features:

- Map non-numeric feature to it’s binary dummy feature
- E.g., A1 = [0, 0, 1, 1, 0, 0, 0]

OHE Dictionary: Maps each category to dummy feature

- (Animal, ‘bear’) ⇒ 0
- (Animal, ‘cat’) ⇒ 1
- (Animal, ‘mouse’) ⇒ 2
- (Color, ‘black’) ⇒ 3
- …

For each feature and value: MD5 Hashfunction % **4**

Feature Hashing Example

Datapoints: 7 feature categories

- A1 = [‘mouse’, ‘black’, -]
- A2 = [‘cat’, ‘tabby’, ‘mouse’]
- A3 = [‘bear’, ‘black’, ’salmon’]

Hashed Features:

- A1 = [0 0 1 1]
- A2 = [2 0 1 0]

Hash Function: $m = 4$

- H(Animal, ‘mouse’) = 3
- H(Color, ’black’) = 2
- H(Animal, ’cat’) = 0
- H(Color, ’tabby’) = 0
- H(Diet, ’mouse’) = 2

OHE versus feature hashing

Datapoints:

- A1 = [‘mouse’, ‘black’, -]
- A2 = [‘cat’, ‘tabby’, ‘mouse’]
- A3 = [‘bear’, ‘black’, ’salmon’]

OHE Features:

- Map non-numeric feature to it’s binary dummy feature
- E.g., A1 = [0, 0, 1, 1, 0, 0. 0]

OHE Dictionary: Maps each category to dummy feature

- (Animal, ‘bear’) ⇒ 0
- (Animal, ‘cat’) ⇒ 1
- (Animal, ‘mouse’) ⇒ 2
- (Color, ‘black’) ⇒ 3
- …

Four buckets (and not seven) so collisions
H(Animal, mouse) = 3rd bucket

Feature Hashing Example

Datapoints: 7 feature categories

- A1 = [‘mouse’, ‘black’, -]
- A2 = [‘cat’, ‘tabby’, ‘mouse’]
- A3 = [‘bear’, ‘black’, ’salmon’]

Hash Function: $m = 4$

- H(Animal, ‘mouse’) = 3
- H(Color, ’black’) = 2
- H(Animal, ’cat’) = 0
- H(Color, ’tabby’) = 0
- H(Diet, ’mouse’) = 2

Collisions for A2

Hashed Features:

- A1 = [0 0 1 1]
- A2 = [2 0 1 0]

Next: Build Sparse vector?

Note here that we have a value of 2 in Feature 0, which is due to the fact that two of the categorical features for A2 both map to bucket 0.

Collisions are ok: preserve similarity

- As we saw in this toy example, when we have more feature categories than we have buckets, we're bound to have collisions.
- And thus, arbitrary feature categories will be grouped together.
- It turns out that in spite of the slightly strange behavior, feature hashing has nice theoretical properties.

- In particular, many learning methods (e.g., SVMs) can be interpreted as relying on training data only in terms of pairwise inner products between the data

Optimization-based ML uses inner products
Hashed features preserve similarity in a dot product sense

- In the context of OHE features, if we're going to approximate these OHE features with a more compact feature representation, then we want this compact feature representation to do a good job of approximating inner products computed on OHE features.
- And it turns out that under certain conditions, hashed features lead to good approximations of these inner products, theoretically.
- Moreover, hashed features have been used in practice and have resulted in good practical performance on various text classification tasks.
- And so in summary, hash features are a reasonable alternative to OHE features.

Why are hashed features reasonable?

Hash features have nice theoretical properties

- Good approximations of inner products of OHE features under certain conditions
- Many learning methods (including linear / logistic regression) can be viewed solely in terms of inner products

Good empirical performance

- Spam filtering and various other text classification tasks

Hashed features are a reasonable alternative for OHE features

Hashed features and parallelization

- Moreover, hashed features have been used in practice and have resulted in good practical performance on various text classification tasks.
- And so in summary, hash features are a reasonable alternative to OHE features.
- Hash features can also be computed in a fairly straightforward fashion in a distributed setting.
- Once we've decided on a hash function, we first need to apply this function to the raw data.

Hashed features and parallelization

Distributed Computation

```
trainHash = train.map(applyHashFunction)
                 .map(createSparseVector)
```

Step 1: Apply hash function on raw data

- Local computation and hash functions are usually fast
- No need to compute OHE features or communication

Step 2: Store hashed features in sparse representation

- Local computation
- Saves storage and speeds up computation

Hashed features and parallelization

Distributed Computation

```
trainHash = train.map(applyHashFunction)
                 .map(createSparseVector)
```

Step 1: Apply hash function on raw data

- Local computation and hash functions are usually fast
- No need to compute OHE features or communication

Once we've decided on a hash function, we first need to apply this function to the raw data.
This is a local computation requiring no communication, and hash functions are generally fast and computable.
As shown in the Spark code snippet, this step can be written by a single map expression.
In the second step, we can store hashed features in a sparse representation, in order to save storage space and reduce the computational burden in subsequent steps of our pipeline.

Live Session Outline: Week 12

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
  - Housekeeping:
    - homework schedule
- **HW11**
- **Lecture 12**
  - Decision Trees
- **Categorical feature encoding**
  - (OHE, Hashing)
- **Digital advertining**
  - Project: CTR Prediction
- **Submitting a Spark Job (Locally and to a cluster on AWS)**
- **Finish RECORDING (bonus points for reminding me!)**

Online click prediction

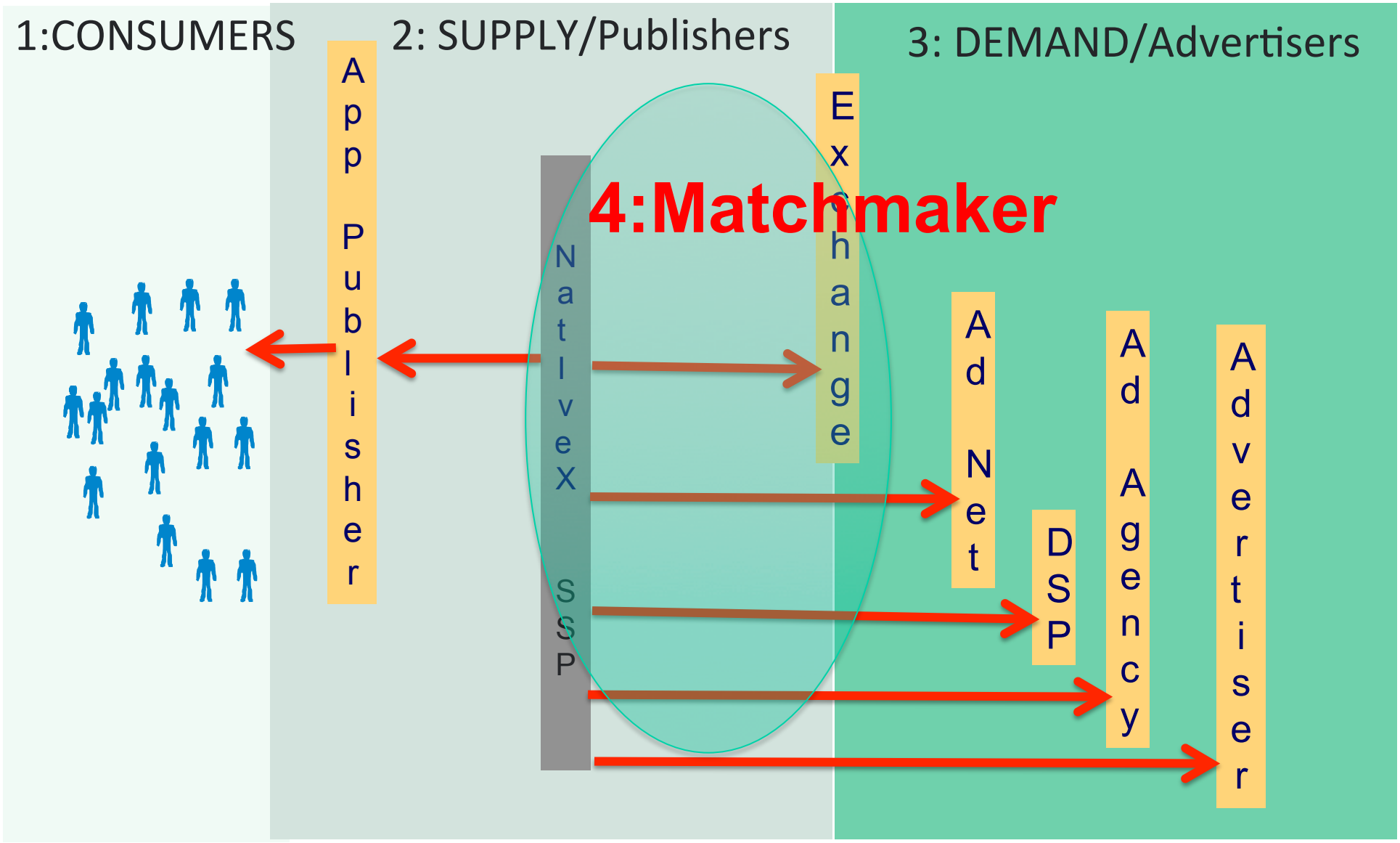
- Online click prediction is a specific problem in the world of advertising and is an excellent example of a high-velocity, high-throughput problem where decisions must be made with a very low latency.
- In general, this class of problems has a large number of applications.
  - Online trading, website optimization, social media, the Internet of Things, sensor arrays, and online gaming all generate high-velocity data and can benefit from fast processes that make decisions in light of the most recent information possible.

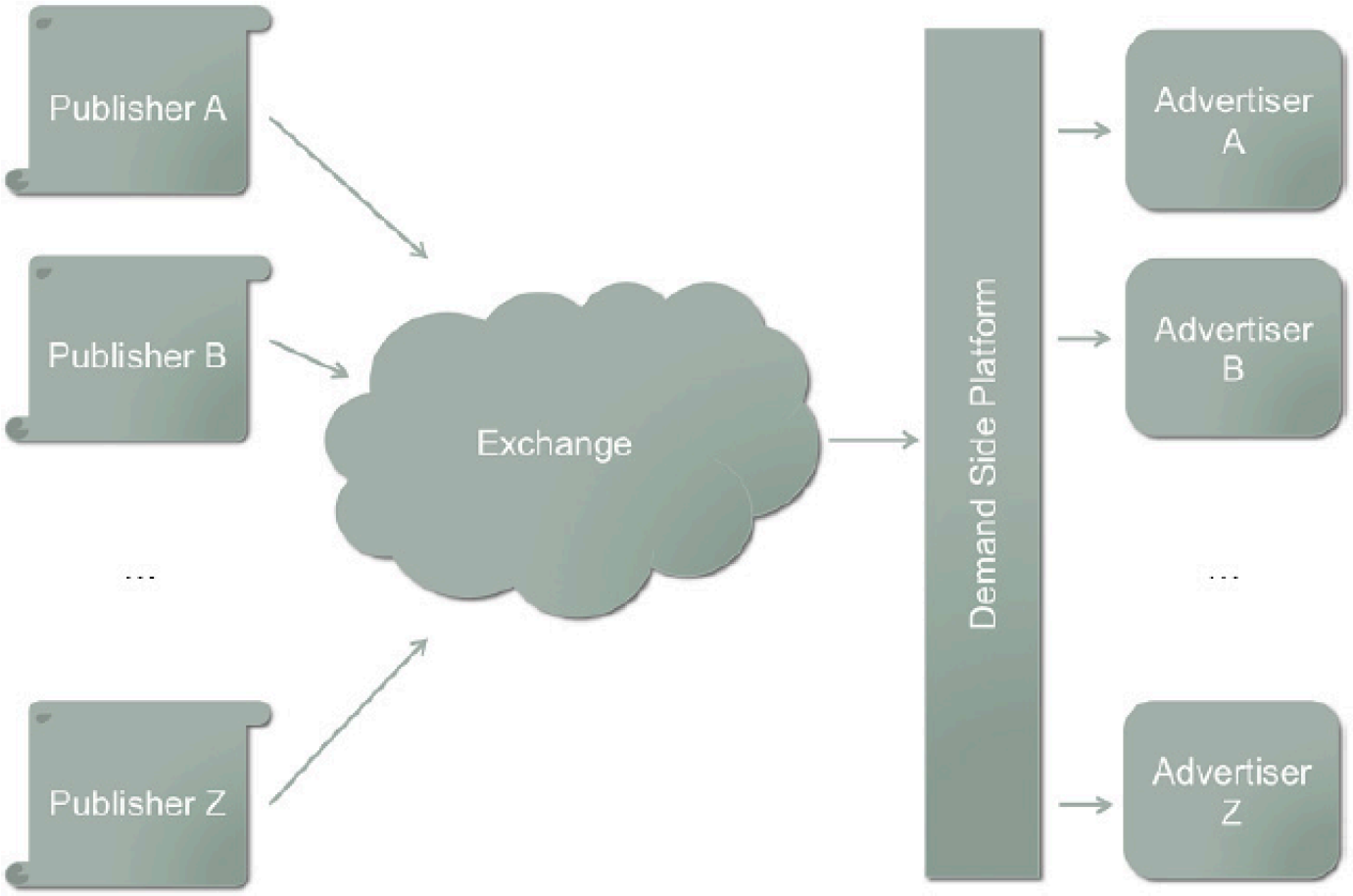
- From an advertiser's standpoint, we can only proxy a positive impact through interaction with an advert.
- So, we try to predict the likelihood of interaction with an advert on a web page, based on all of the information surrounding a user and the context.
- Due to the volume and velocity of data generated on the web and the requirement for a rapid answer, this is not a simple problem.

SMART City

- The solutions presented here should be equally applicable to different applications.
- For example, if we consider a smart city, where all cars are enabled with sensors to determine the speed, velocity, and chosen route, low-latency decisions could be used in order to open/close contraflow lanes or reroute traffic to keep it moving.
- Clearly, this application has a widerreaching impact on society than advertising does, but it still follows a similar problem pattern.
- Can we take a huge amount of data that's arriving with high velocity and build models around this to perform decisioning quickly?

Digital Advertising Ecosystem





The online advertising ecosystem. Publishers make their content (advertising slots) available through an exchange that allows access programmatically, that is, through an intelligent algorithm. A demand-side platform aggregates demand for content from advertisers and buys on their behalf.

Exchanges and DSPs

You can see that the exchange is the medium through which publishers (that's websites such as theguardian.com, huffingtonpost.com, and more) can sell advertising space (slots and spaces of a predefined

size) to individual advertisers (think companies like Nike, Adidas, O2, and Vodafone).

Advertisers often buy through what is known in the industry as a demand-side platform (DSP).

A DSP acts on behalf of multiple advertisers and combines buying power to reach the very best publishers on the web.

There are many examples of DSPs on the web, such as Turn,66

MediaMath,67 and DataXu.68 These platforms are a conduit through which it's possible to buy advertising. We can easily draw parallels with the world of equity trading. Not everybody is allowed on the trading floor, so representatives act on behalf of many individuals who wish to buy.

In this parallel, the DSP is equivalent to the trader, where the advertising exchange is equivalent to the stock exchange. In the first case, the DSP aggregates demand, just as an individual trader does. In the second case, both exchanges act as a platform for equities to be

bought and sold.

- In both the advertising and equity markets, an intelligent algorithm now more often performs trading than a human. In the world of finance, this is often referred to as high-frequency trading. In the advertising world, this is known as programmatic buying, because advertising space is bought through the use of a computer program (an intelligent algorithm)

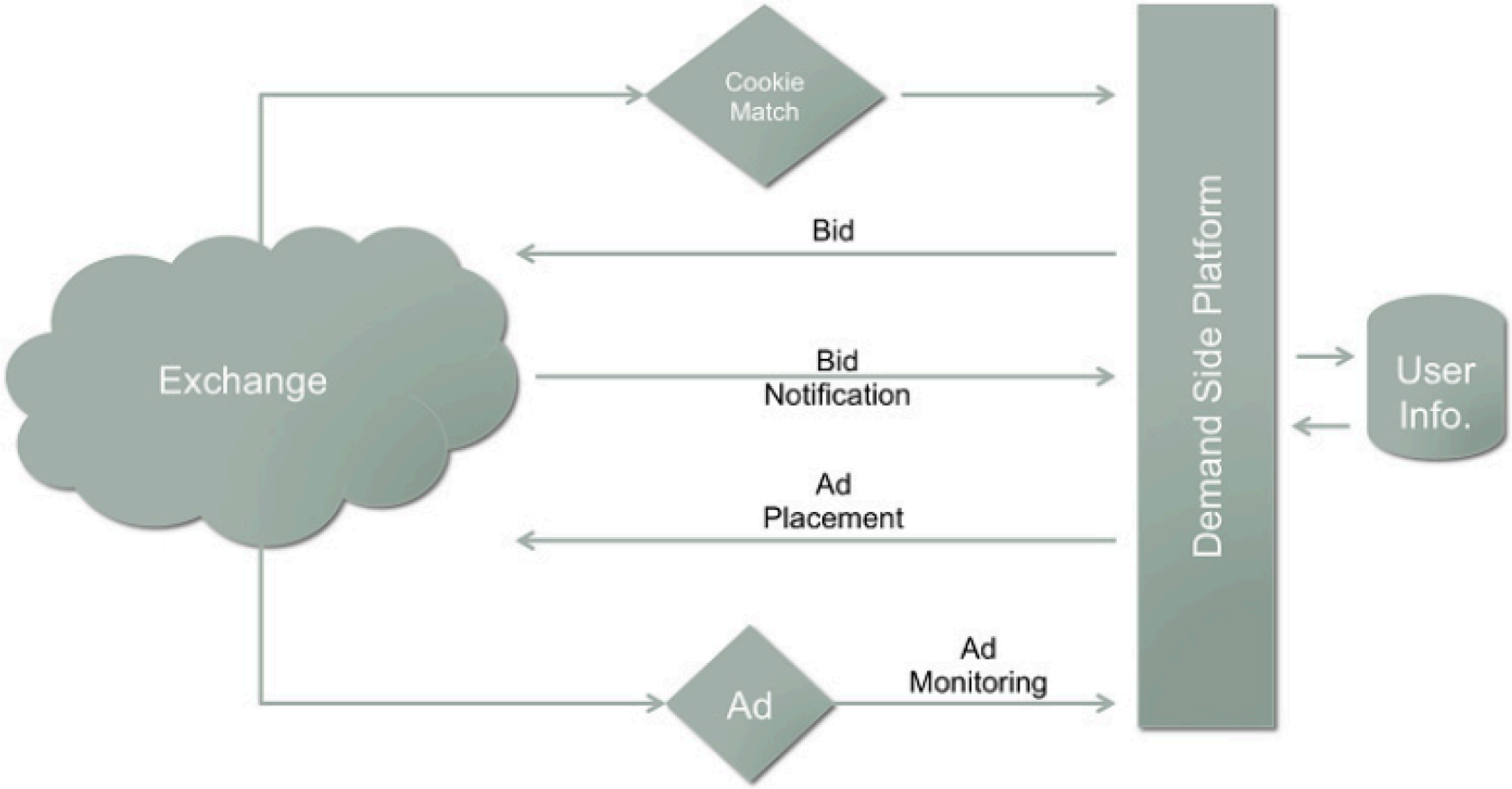


Figure 5.2 A graphical overview of the steps involved to buy ad placements on an exchange. Events should be read from top to bottom. First, a cookie match occurs. This can be performed either exchange side or DSP side. Once the DSP has a unique identifier that it can use, it will use this to look up additional information about the user and construct a bid price (the subject of section 5.4.6). If the bid is successful, bid notification is returned for logging by the DSP. The next stage is for the DSP to specify the advert to be served. Physically, this can be served by the exchange itself or by the DSP. Events within the advert are fired directly back to the DSP regardless of where the ad is hosted.

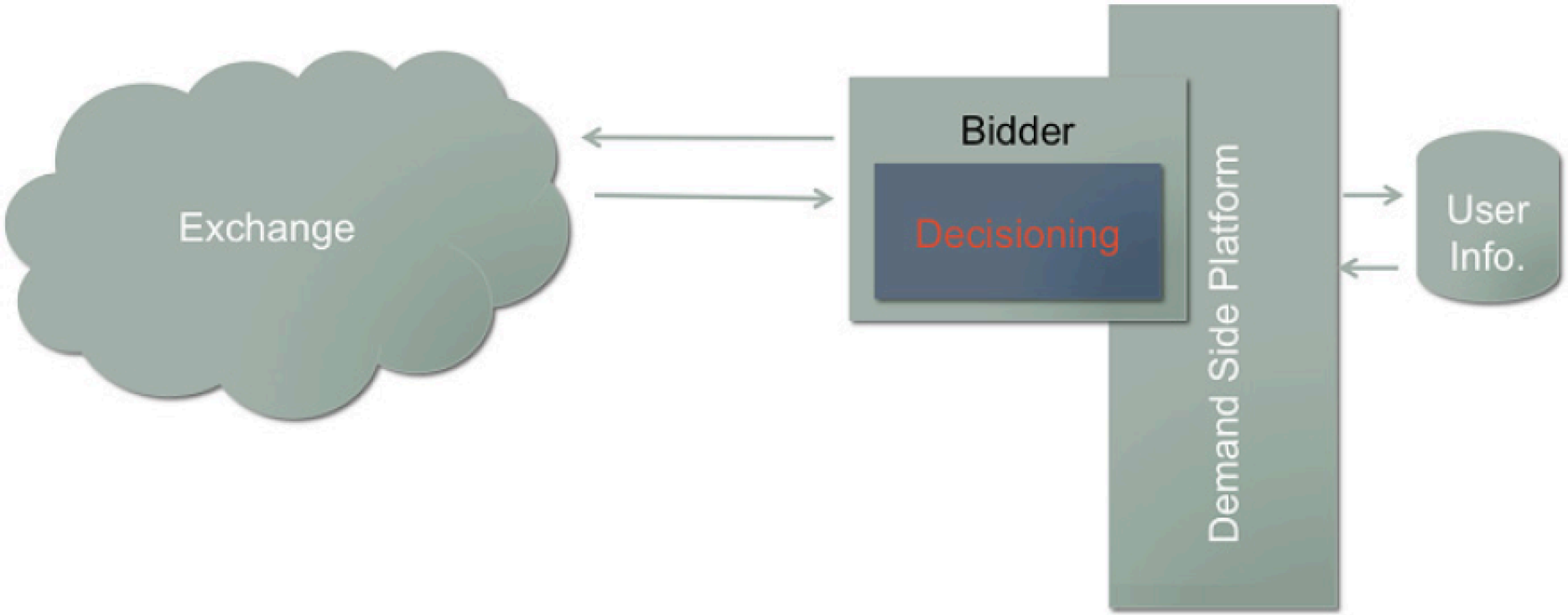


Figure 5.3 The bidder consists of an outer shell that's responsible for performing the crucial tasks associated with bidding and the decision engine that's responsible for taking into account all data associated with the bid..

Mobile advertising example

- **Data pipelines both realtime/ops, batch/BI, hybrid pipelines such as predictive analytics**

- **Use mobile advertising as an example Application**
- **Data flow at high level**
- **Product needs**
  - Operational data
  - Offline modeling data

- **As a data scientist you will wear many hats**

Advertising ~2% of US GDP; $140B WW

Opinions regarding the practice of online advertising are polarized among web users

"***Half the money I spend on advertising is wasted; the trouble is, I don't know which half.***" **- John Wanamaker, father of modern advertising.**

- Less than 1% of all impressions lead to measureable ROI

Despite its problems (Attribution, etc.)

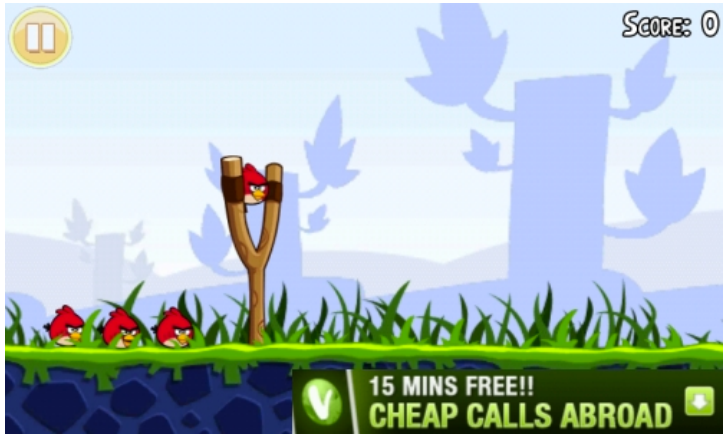
- **US GDP = $14.1 Trillion (Global $56 Trillion, $56x10^{12}$)**
- **US Advertising Spend**
  - ~$275 Billion across all media
    - (2% of GDP since the early 1900s)
- **In 2015, Worldwide online advertising was $180B**
  - I.e., about 20% of all ad spending across all media
  - $42 billion (out of $140) global mobile-advertising market in 2014
  - *$100 billion global mobile-advertising market in 2016*

Mobile Publisher: How do I make money?

Consumer:
App user

- Paid app download
- In app purchases
- In app Advertising

Publisher:
App Developer



Making Money from Apps

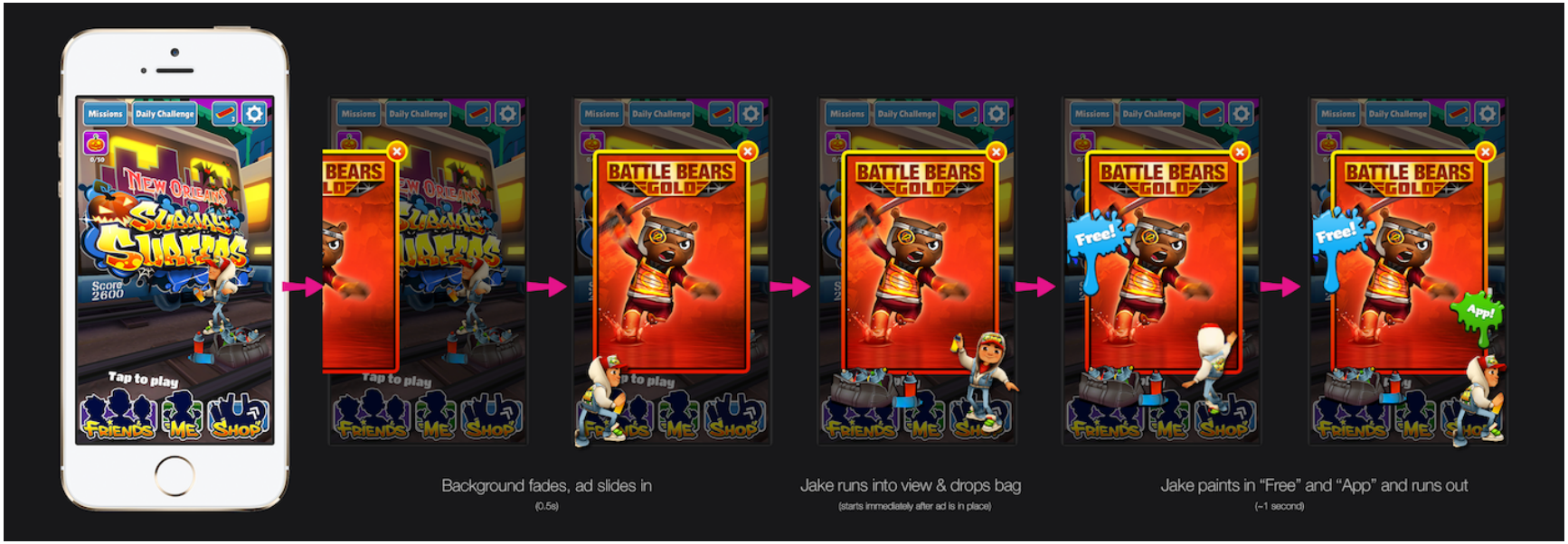
- **93% of downloaded apps in 2013 (globally) are free apps !**

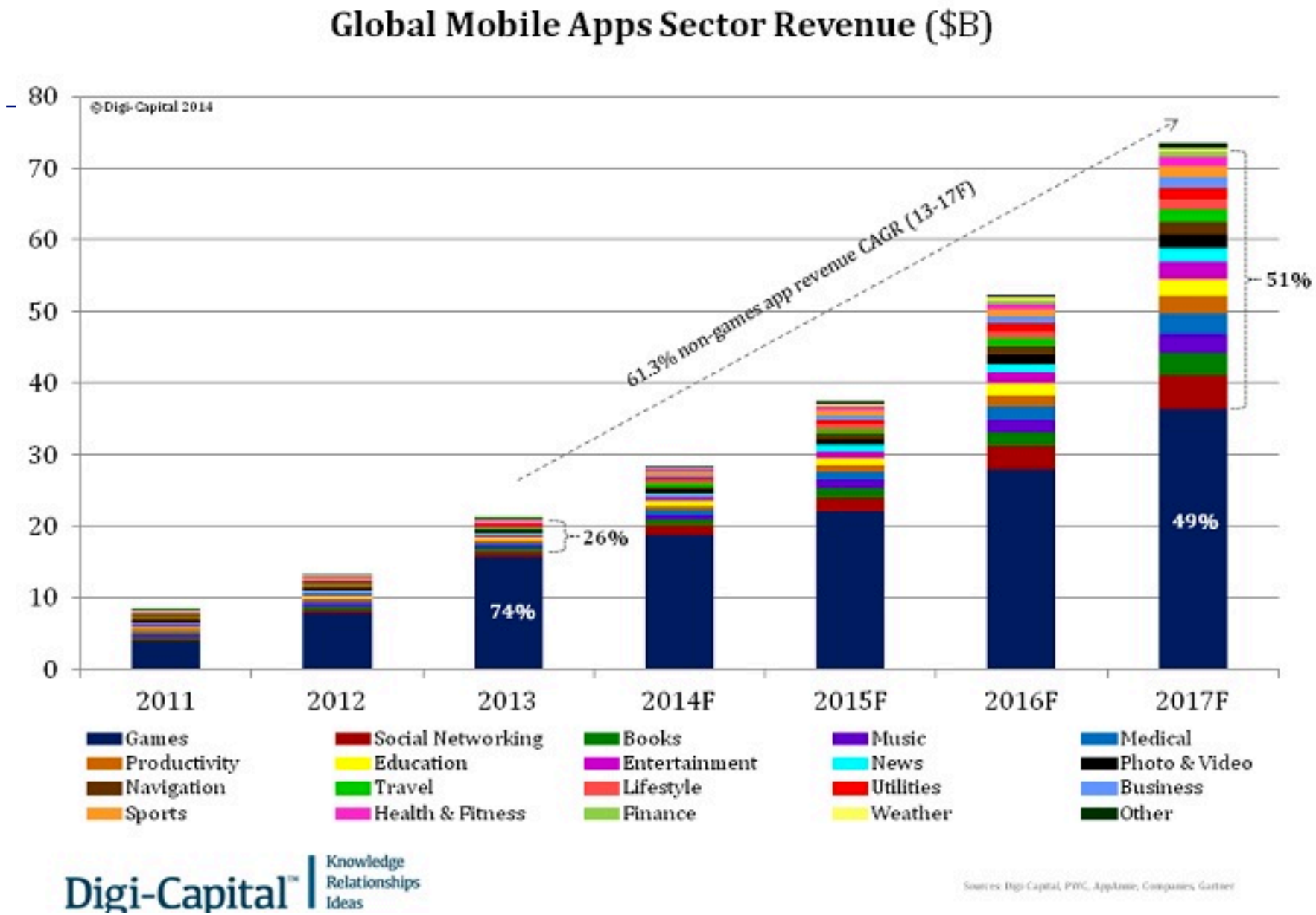
- **76% of revenue generated from apps (globally) in 2013 is from in-app purchases**
  - [http://www.forbes.com/sites/chuckjones/2013/03/31/apps-with-in-app-purchase-generate-the-highest-revenue/]
- **In the Freeium economy**
  - To make money from apps, publishers must maintain customer satisfaction through superior app performance and design,
  - then monetize though advertising and in-app purchases

[http://venturebeat.com/2014/03/27/mobile-app-monetization-freemium-is-king-but-in-app-ads-are-growing-fast/, IDC, AppAnnie]


(2) Facebook
https://www.facebook.com
Getting Started
Outlook Web App
Home – nativeX Intra
Daily eCPM Reports
Discount Golf Cours
About Chandoo.org
Photos – Dropbox
Search for people, places and things
Home
James
Dan Larsen shared Surfer Magazine's photo.
You've seen photos of Jaws from the channel, now watch it from a drone that was hovering above the lineup: http://bit.ly/1fabFIM
Like · Comment · Share · 6 hours ago ·
Johann Schleier-Smith and 6 others like this.
Dan Larsen Forecast is big for Maui's north shore. Be safe my friends, and live to ride again and again. And take lots of video! Mahalo
6 hours ago · Like · 1
Marzena Kmiecik That is incredible to watch!
5 hours ago · Like
Write a comment...
Nitro · Suggested Post
Like Page
"Paperless" is a term we've heard for years.
Isn't it about time you made it a reality? Nitro is ditching paper this year, and you can too! Get Nitro Pro today and save 15%!
Save Money. Go Paperless!
www.nitropdf.com
Industrial Design Sale
dotandbo.com
Up to 64% Off on Industrial Modern Home Decor. Discover Now!
Ronald Mannak likes this.
North Social
NORTH SOCIAL
Create a custom Facebook page in minutes... without writing a single line of code!
Like · 222,947 people like North Social.
Coin: Less is More
onlycoin.com
Pre-order now & save 50%. Declutter your wallet and never leave your card behind again.
Brynn Evans likes this.
The Retrofit Source Inc.
Maximum Output Headlights
Stock Sucks! And so do your headlights. We offer high-performance components to fix that!
Like · Andy Wong likes The Retrofit Source Inc.
Facebook © 2014
English (US) · Privacy · Terms · Cookies · More

Native Advertising





http://venturebeat.com/2014/04/29/mobile-apps-could-hit-70b-in-revenues-by-2017-as-non-game-categories-take-off/

Mobile Ad Spend to Top $100 Billion Worldwide in 2016, 51% of Digital Market

- **US and China will account for nearly 62% of global mobile ad spending next year**

Mobile Internet Ad Spending Worldwide, 2013-2019

	2013	2014	2015	2016	2017	2018	2019
Mobile internet ad spending (billions)	**$19.20**	**$42.63**	**$68.69**	**$101.37**	**$133.74**	**$166.63**	**$195.55**
—% change	117.9%	122.1%	61.1%	47.6%	31.9%	24.6%	17.4%
—% of digital ad spending	16.0%	29.4%	40.2%	51.1%	59.4%	65.9%	70.1%
—% of total media ad spending	3.7%	7.8%	11.9%	16.5%	20.5%	24.1%	26.8%

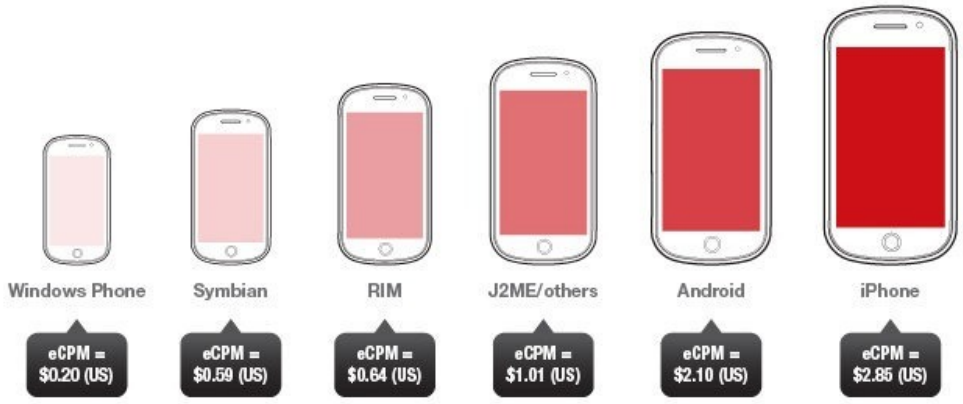
Note: includes display (banners, video and rich media) and search; excludes SMS, MMS and P2P messaging-based advertising; ad spending on tablets is included
Source: eMarketer, March 2015

186887 www.eMarketer.com

- http://www.emarketer.com/Article/Mobile-Ad-Spend-Top-100-Billion-Worldwide-2016-51-of-Digital-Market/1012299#sthash.FBfZAlaC.dpuf

CPMs on Mobile are catching up

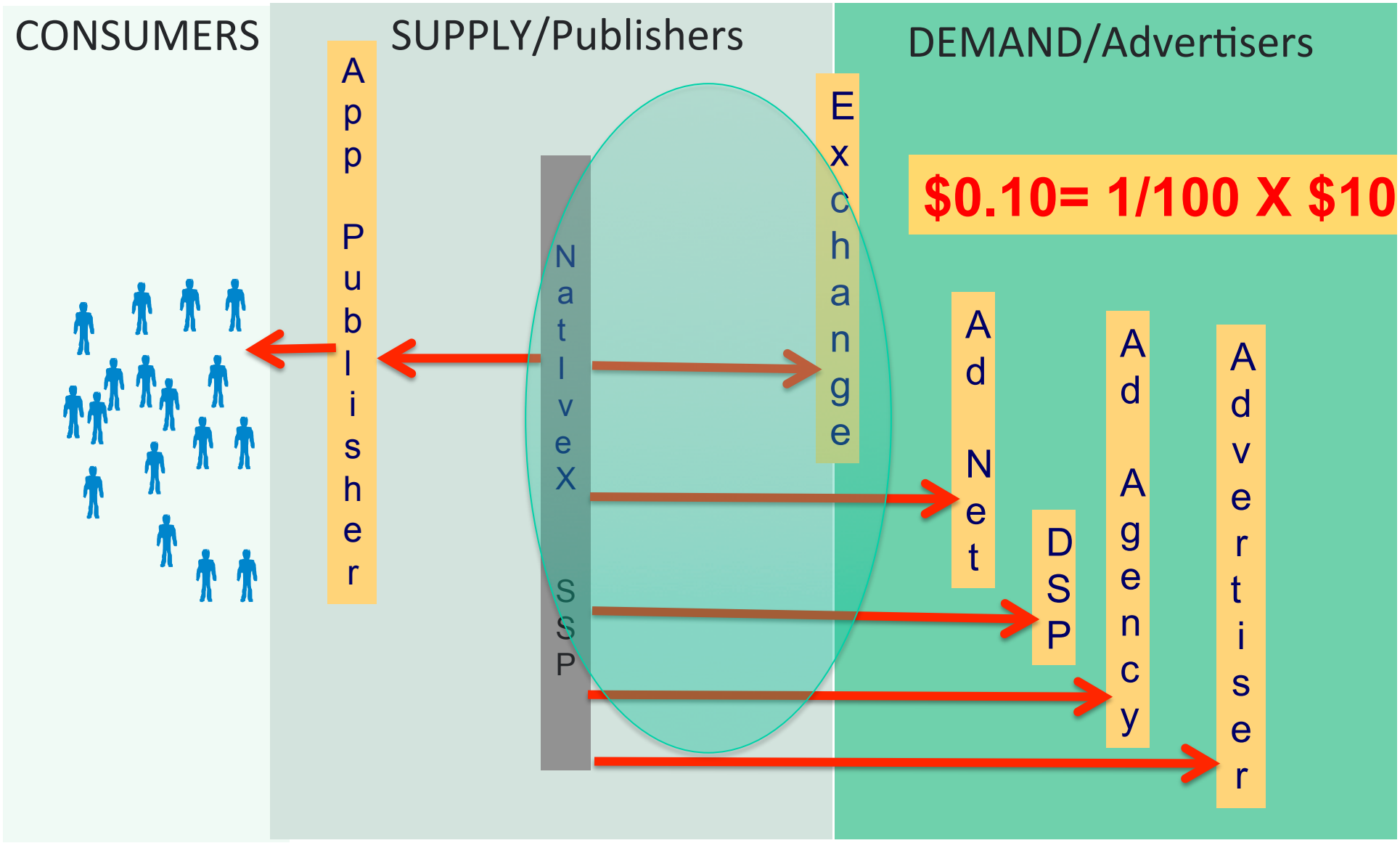
- **Unit of counting is 1,000 displays/impressions/views**
- **CPM: cost of displaying the ad 1,000 times**
- **Mobile Advertising: What is the average CPM on mobile?**
  - The effective cost per thousand impressions (CPM) for desktop web ads is about $3.50, while the CPM for mobile ads is just $0.75.
  - Video-based CPMs typically > $15-$20



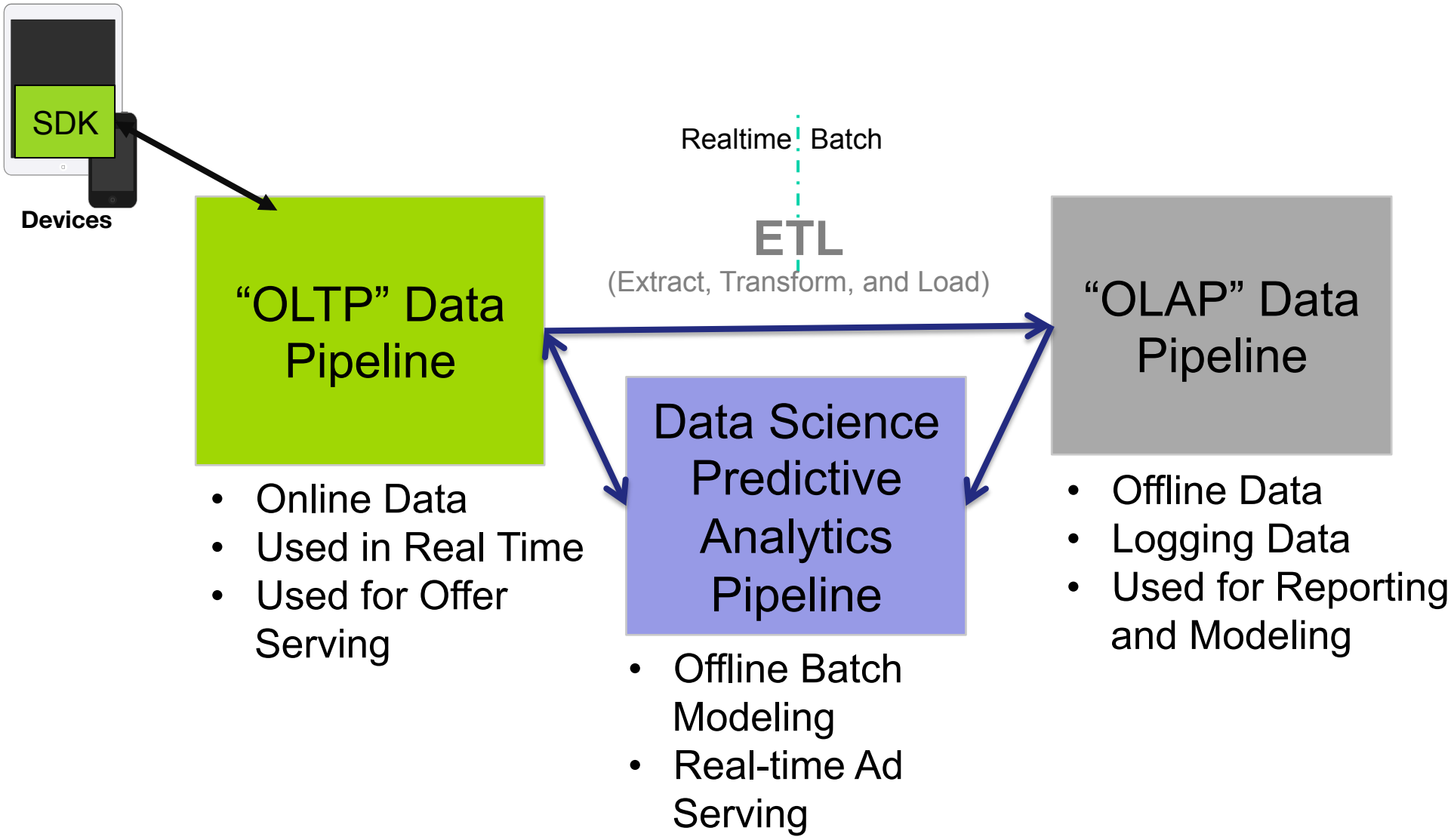
http://www.quora.com/Mobile-Advertising/What-is-the-average-CPM-on-mobile

http://mashable.com/2012/10/23/mobile-ad-prices/

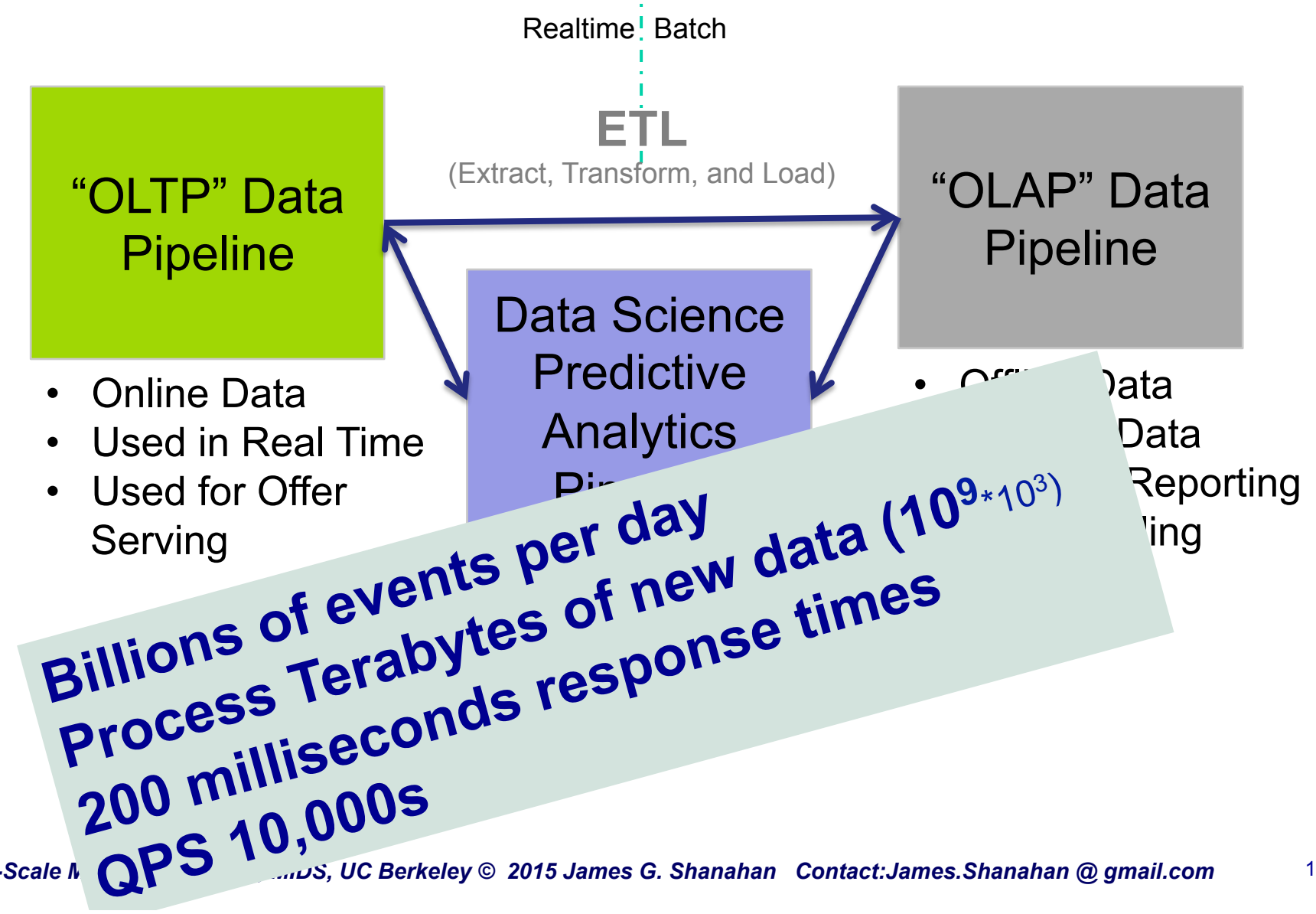
NativeX: Art and Science of Native Mobile Advertising



Ad serving data pipelines



Ad serving data pipelines



Ranking Ads (more) at Turn Inc.

Online Advertising is Big Business

Multiple billion dollar industry

$43B in 2013 in USA, 17% increase over 2012
[PWC, Internet Advertising Bureau, April 2013]

Higher revenue in USA than cable TV and nearly the same as broadcast TV
[PWC, Internet Advertising Bureau, Oct 2013]

Large source of revenue for Google and other search engines

Digital Advertising: players

- --

Publishers: NYTimes, Google, ESPN
- Make money displaying ads on their sites

Advertisers: Marc Jacobs, Fossil, Macy's, Dr. Pepper
- Pay for their ads to be displayed on publisher sites
- They want to attract business

Matchmakers: Google, Microsoft, Yahoo
- Match publishers with advertisers
- In real-time (i.e., as a specific user visits a website)

Consumers

Why advertisers pay

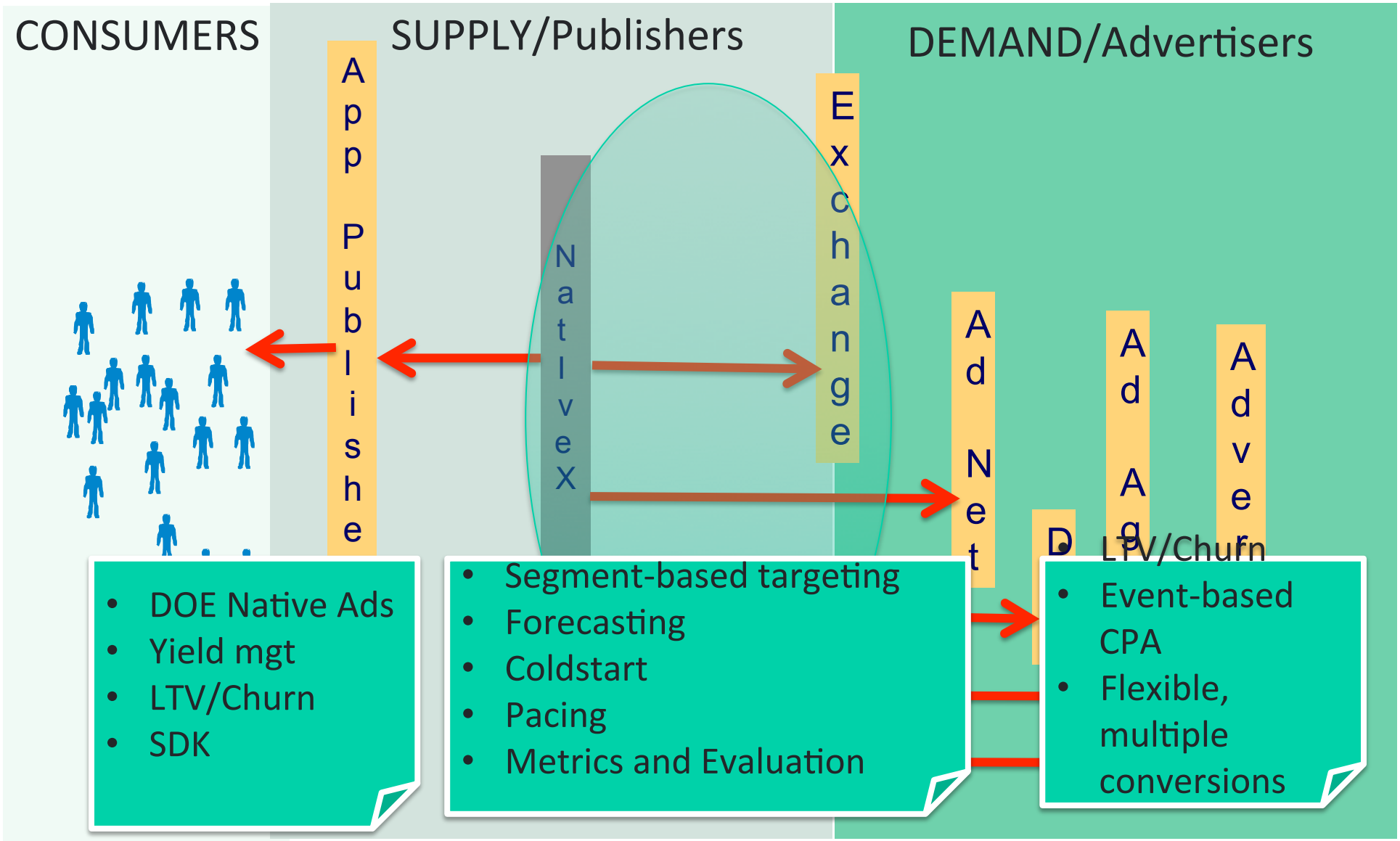
Impressions

- Get message to target audience
- e.g., brand awareness campaign

Performance

- Get users to do something
- e.g., click on ad (pay-per-click) ← Most common
- e.g., buy something or join a mailing list

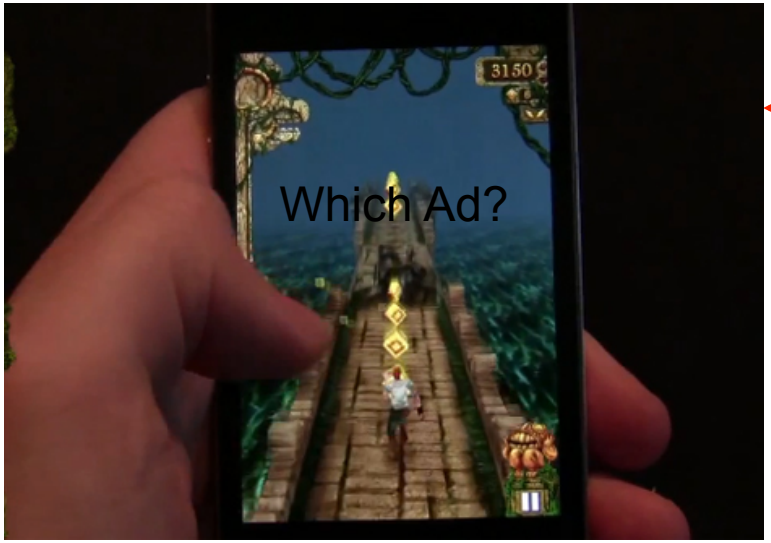
Digital Advertising



Publisher: Which ad to show? And how much?

Publisher or their surrogate the matchmaker (aka ad network) need to decide:

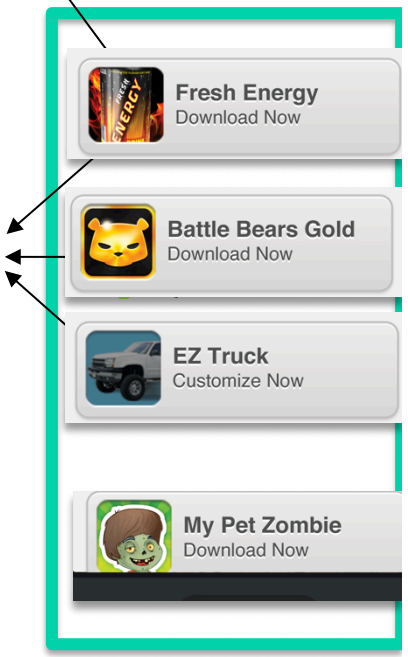
1. **Which ad to show**
2. **How much to charge for showing the ad (ad impression)?**



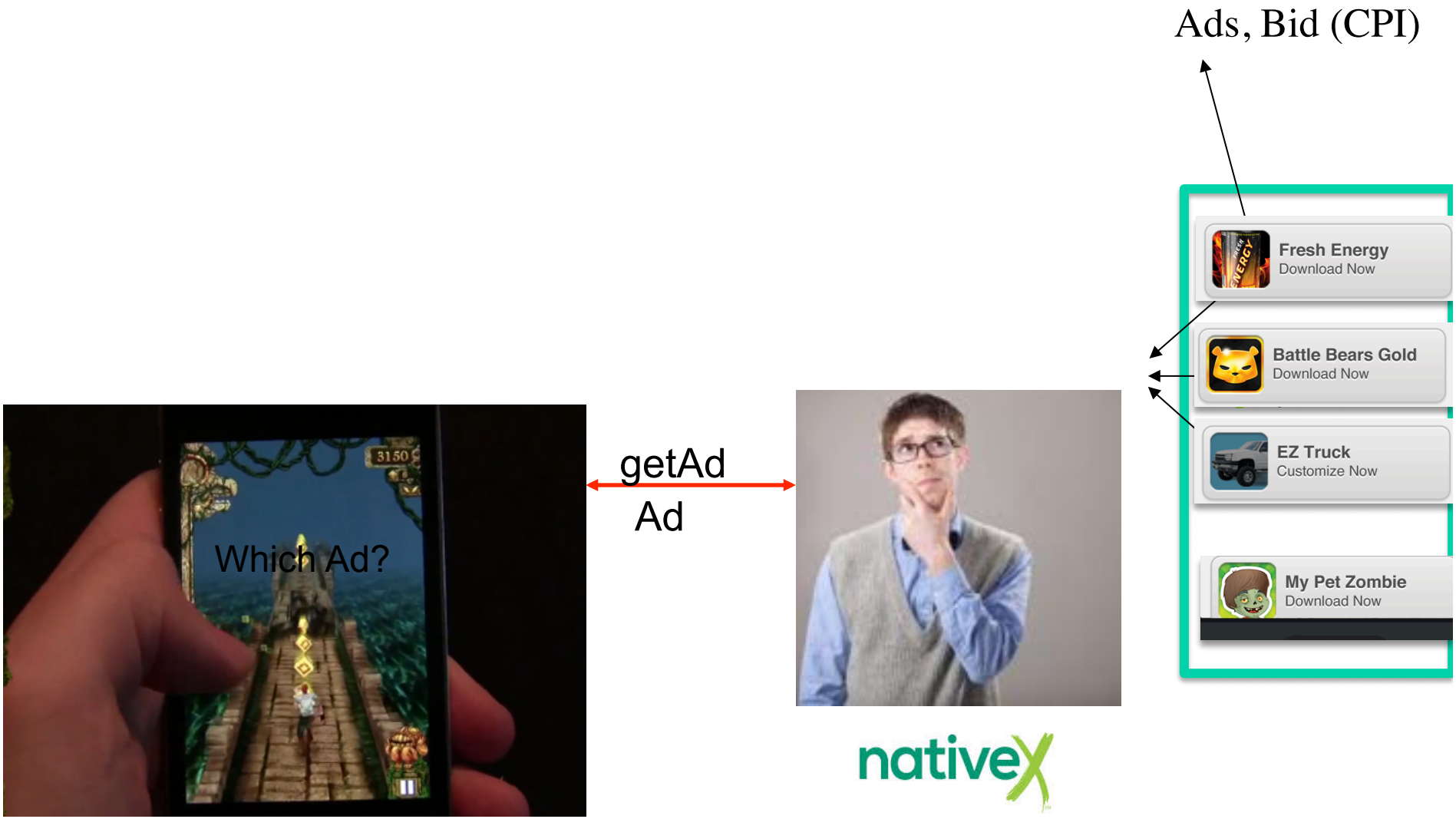
getAd

Ad

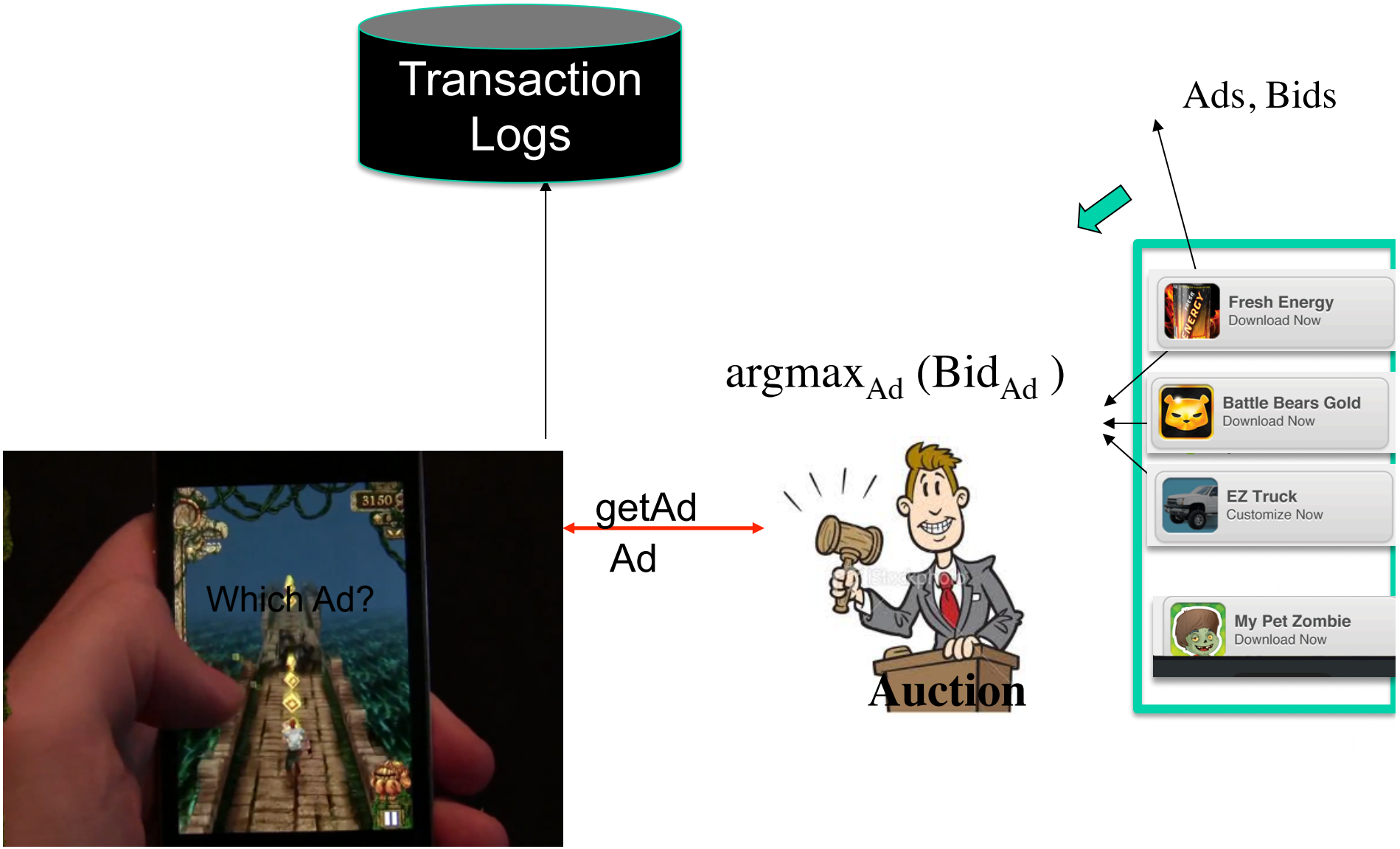
Bids



Publisher: Which ad to show? And how much?



Publisher: Which ad to show?



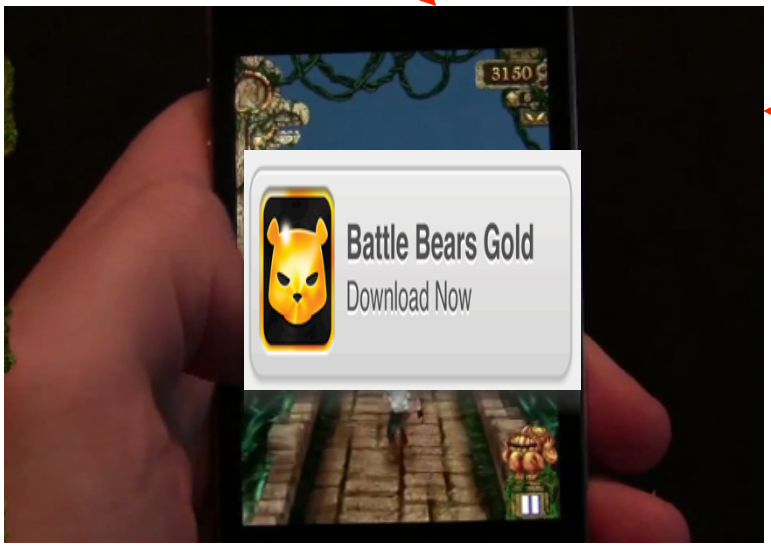
Publisher: Cost per install

Transaction Logs

Ads, Bids

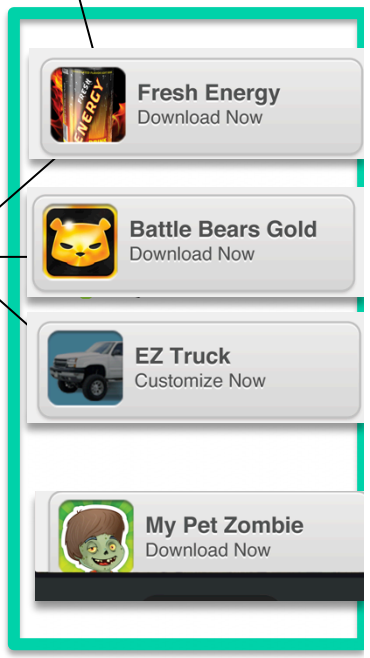
Install

$\text{argmax}_{Ad}\ (\text{Bid}_{Ad})$

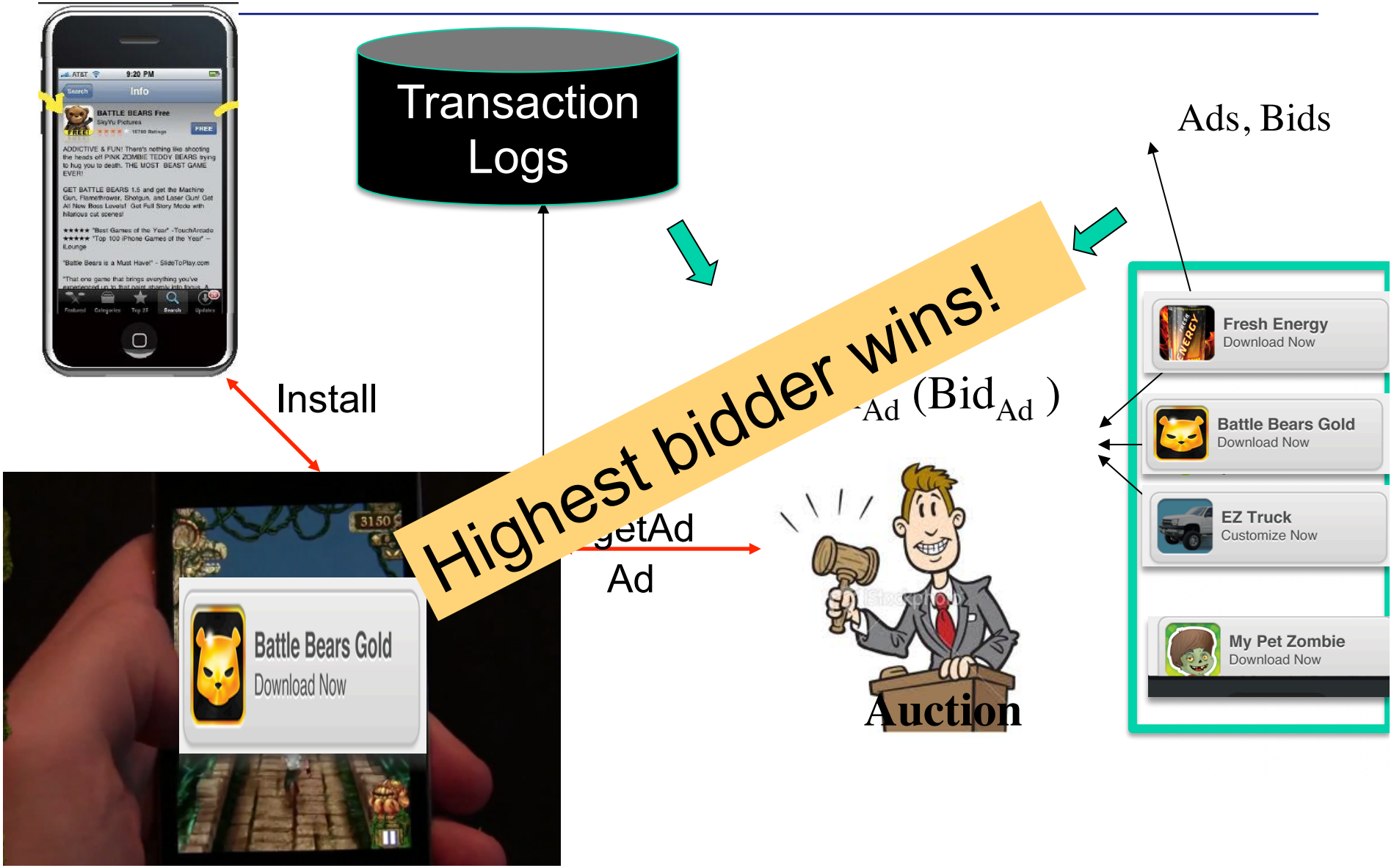


getAd

Ad

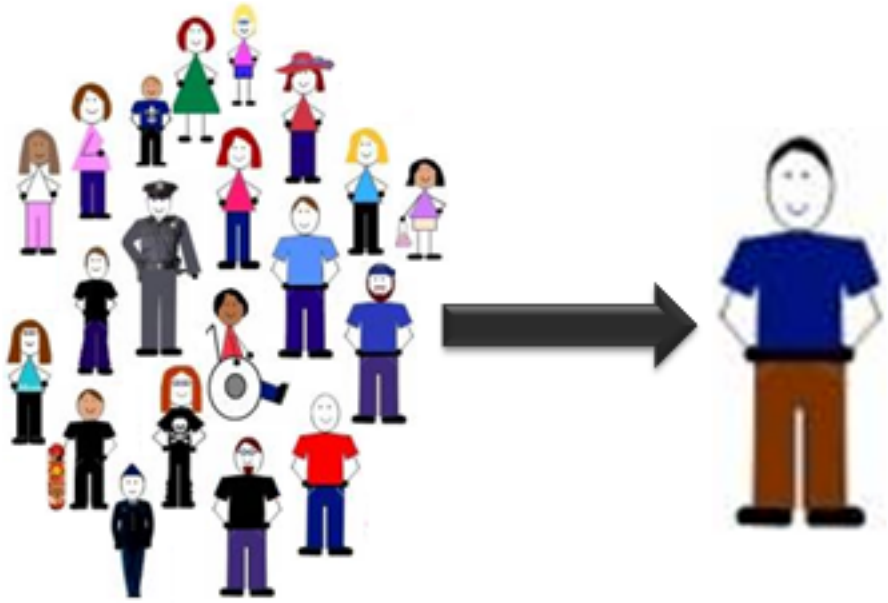


Publisher: Cost per install



CONVERSION RATE=5%

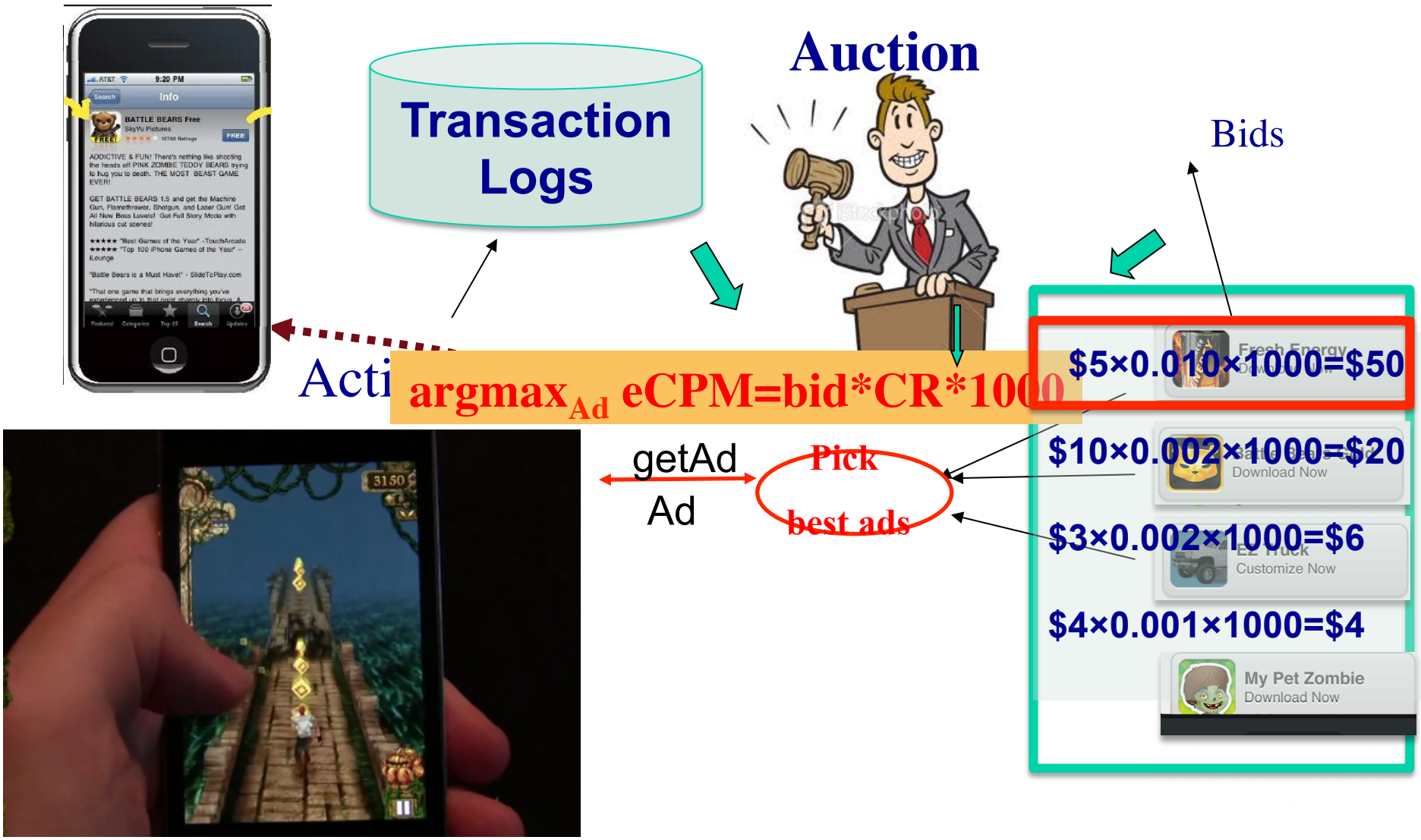
Calculating conversion rate: 1 sale ÷ 20 visitors × 100% = 5% CR



20 Viewers 1 Install

Expected revenue (AKA eCPM) is 5% × $10 × 1,000 = $500.00

NativeX conducts an eCPM-based Auction



NativeX conducts an eCPM-based Auction

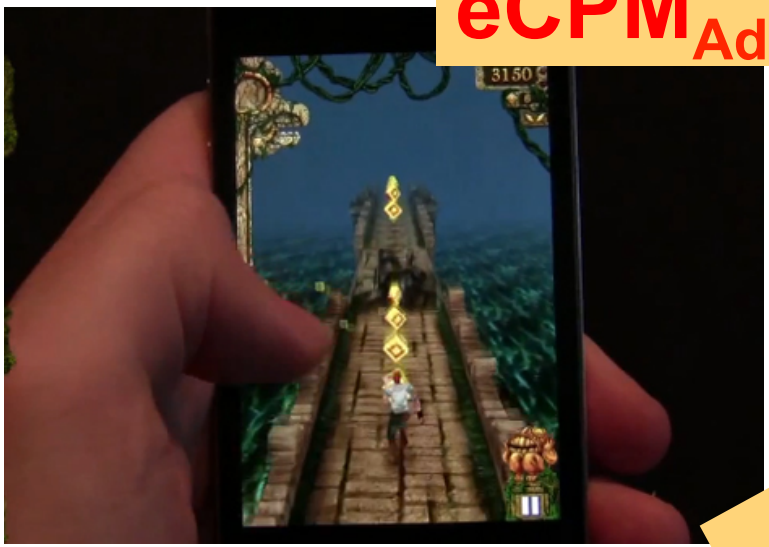
Bids

Action

$$eCPM_{Ad} = CR_{Ad} \times Bid_{Ad} \times 1000$$

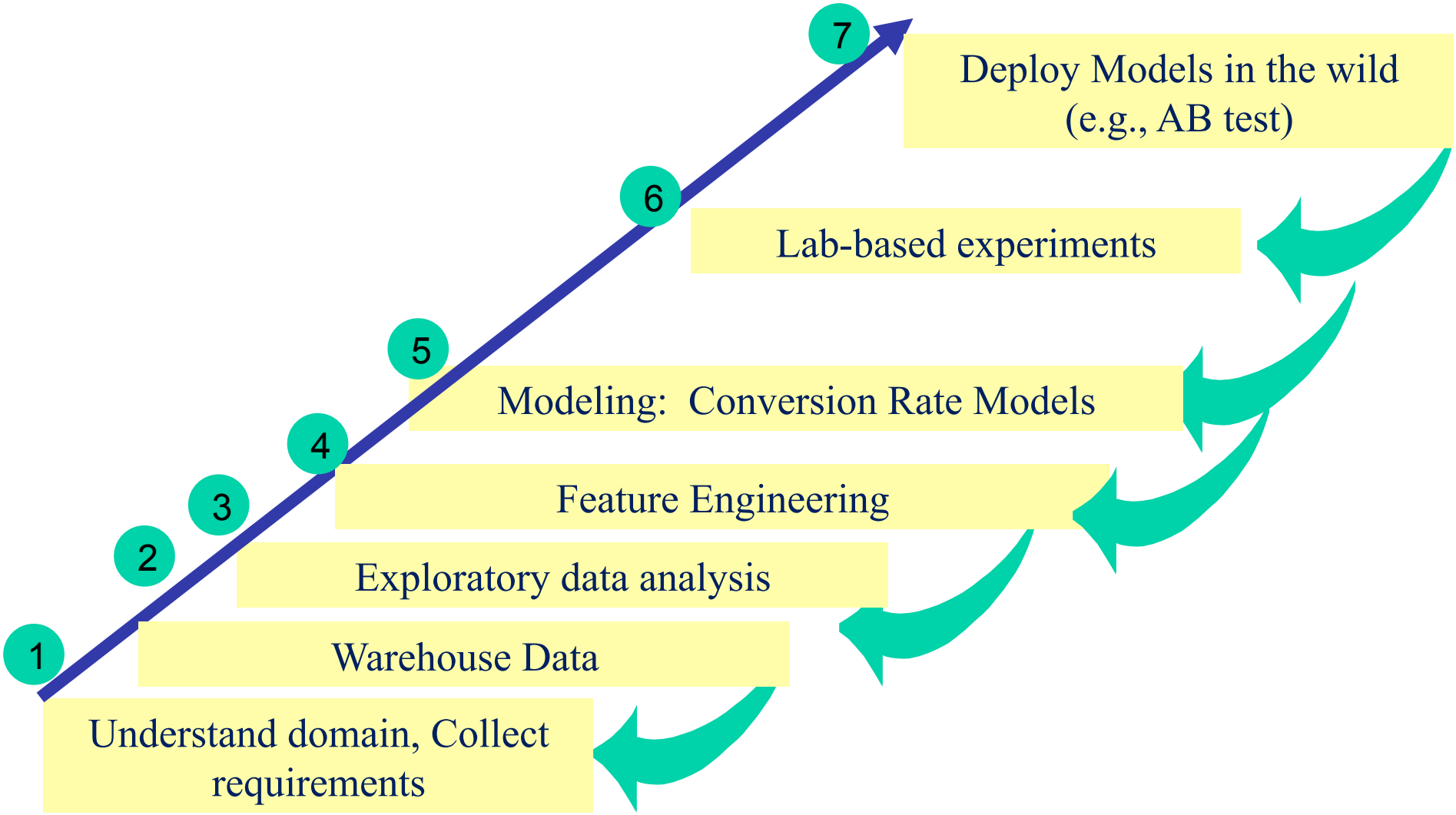
Ad

best ads



$100B Equation

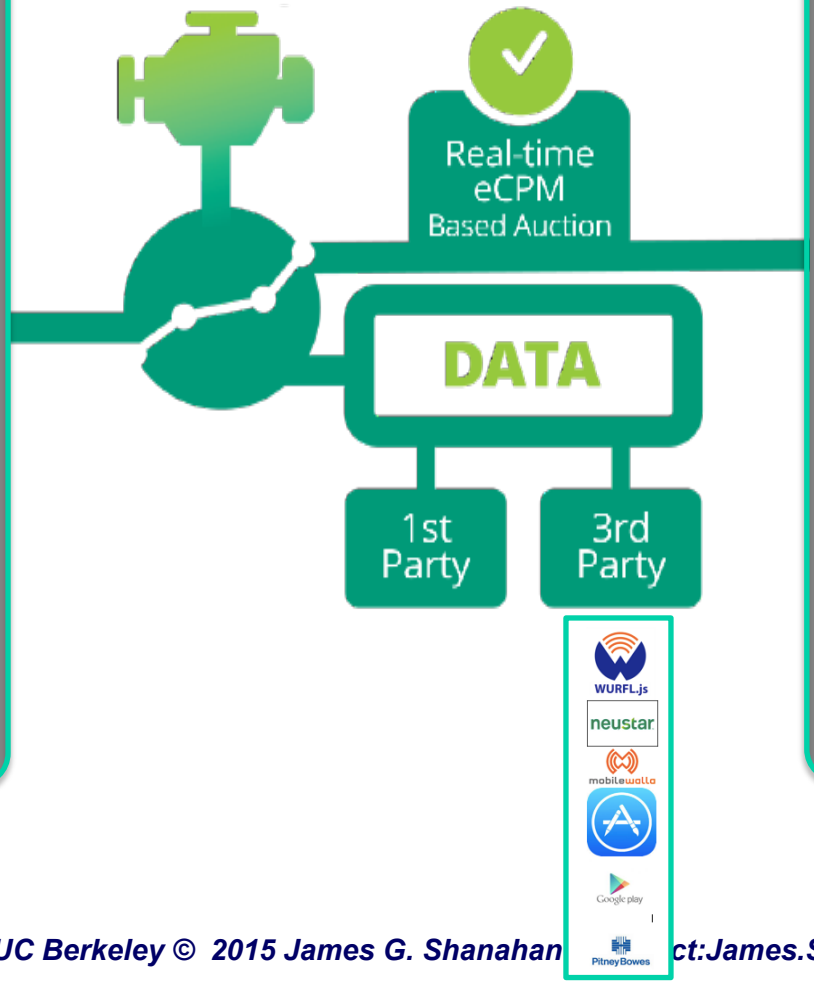
7 Steps in Modeling: E.g., Conversion Rate Modeling



Campaign-specific models for CTR/CR

Modeling Features:

- Geo location
- Device
- Reviews (star rating, review text; Geo location of reviews)
- Social media Tweets/ FB posts
- Categories on Android and iOS
- Creative Message
- User profiles (RFM based on network behavior)
- Device Behavioral (based on installed apps on device RFM, recommendations, categories)
- Graph-based features
- Others….



Multiple Ad Sources:

- DSPs, Exchanges
- Ad Networks
- Internal/Self-service

Multiple conversion types:

- CPM, CPC, CPI, CPCV, CPA, CPE

De-duplication

Optimization by geo

How to price an impression?

Efficient Matchmaking

Idea: Predict probability that user will click each ad and choose ads to maximize probability

- Estimate $\mathbb{P}$(click | predictive features)
- Conditional probability: probability **given** predictive features

Predictive features

- Ad's historical performance
- Advertiser and ad content info
- Publisher info
- User info (e.g., search / click history)

Publishers Get Billions of Impressions Per Day

But, data is high-dimensional, sparse, and skewed

- Hundreds of millions of online users
- Millions of unique publisher pages to display ads
- Millions of unique ads to display
- Very few ads get clicked by users

Massive datasets are crucial to tease out signal

Goal: Estimate $\mathbb{P}(\text{click} \mid \text{user, ad, publisher info})$
Given: Massive amounts of labeled data

CRITEO CTR Prediction from 39 features

Criteo is a matchmaker

Display advertising problem (decide which ads to show on webpages such as New York Times

Click-Through Rate Prediction Lab

This lab covers the steps for creating a click-through rate (CTR) prediction pipeline. You will work with the Criteo Labs dataset that was used for a recent Kaggle competition.

This lab will cover:

- ####*Part 1:* Featurize categorical data using one-hot-encoding (OHE)
- ####*Part 2:* Construct an OHE dictionary
- ####*Part 3:* Parse CTR data and generate OHE features
  - #### *Visualization 1:* Feature frequency
- ####*Part 4:* CTR prediction and logloss evaluation
  - #### *Visualization 2:* ROC curve
- ####*Part 5:* Reduce feature dimension via feature hashing
  - #### *Visualization 3:* Hyperparameter heat map

Note that, for reference, you can look up the details of the relevant Spark methods in Spark's Python API and the relevant NumPy methods in the NumPy Reference

Criteo CTR Data set and phase

- CTR Prediction from 39 features
- 4.3 Gig of data compressed

- Phase 1: work on a sample of the data (week 12)
- Phase 2: work on the full dataset (week 13)

Project: Phase 1

- Work on a click-through rate prediction pipeline in Spark.
- The goal of the project is to implement a click-through rate prediction pipeline using various techniques that we've discussed

Dataset

- This data includes 39 features describing users, ads, and publishers.
- Many of these features contain a large number of categories.
- Additionally, for privacy purposes the features have been anonymized.
- We'll be using a small fraction of the Kaggle data.
- And the Kaggle data itself is only a small fraction
- of Criteo's actual data.

CRITEO (DSP): A DSP dataset

- use some real-world data generated from Criteo, a leading DSP.
- Criteo has made this data available in order to spur research in this area, but it comes with a catch. The data is absent of any semantics, so we don't know what each field refers to or the meaning of the values contained within.

Criteo provides us with 13 features, each taking an integer value and 26 categorical features. The data is presented such that for a single log line, all integer features occur first, followed by the categorical features.

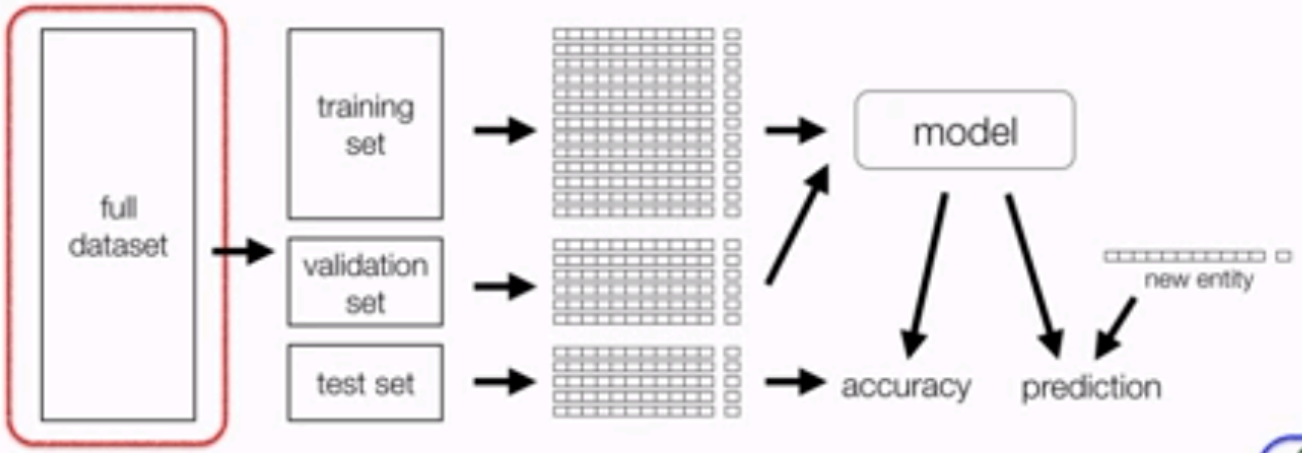
According to the readme provided with the data, the integer features consist of mostly count information (for example, the number of times a user has performed something), whereas the values of the categorical features have been hashed into 32 bits. This means that although integer values have integer semantics (a higher number means an event has been performed more than a lower number), we can't prescribe any such semantics to the categorical values. We might imagine that each categorical value represents some property, for example, which site the user is on at the time of the bid request. Note that hashing ensures consistency across bid requests, that is, the same hashed value implies the same non-hashed value. So although we don't know the meaning of a particular hash, the meaning is consistent every time we see that hashed value.

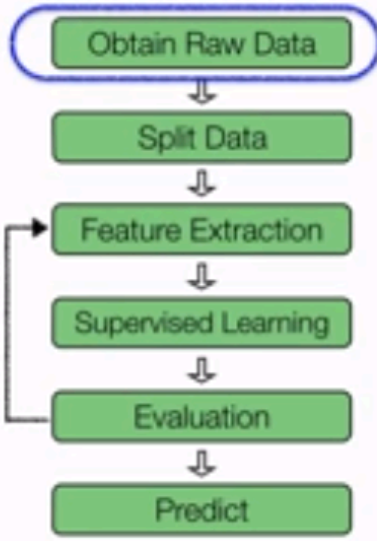
To be compatible with the VW file format, we first need to change the class labels. VW refers to positive instances with a 1 and negative instances with a -1. More important, we

Main Tasks

- **You'll split this data set into training, validation, and test data sets.**
- **You'll next need to extract features to feed into a supervised learning model.**
- **And feature extraction is the main focus of this lab.**
  - You'll create OHE features, as well as hashed features, and store these features using a sparse representation.
  - You will also visualize feature frequency to get a better sense of the sparsity pattern of this data.
  - Given a set of features, either OHE or hashed features, you will use MLlib to train logistic regression models.
- **You will then perform Hyperparameter tuning to search for a good regularization parameter, evaluating the results via log loss, and visualizing the results of your grid search.**
- **You'll also look at a ROC plot to understand the trade-off between the false positive and the true positive rates.**
- **You will then evaluate the final model, again, using log loss and compare its accuracy to that of a simple baseline that always predicts the fraction of training points that correspond to click-through events.**
- **You will also compare the accuracy of models train using OHE features and hashed features.**
- **The final model you train could be used for feature click-through rate prediction.**

Display advertising





Raw Data: Criteo Dataset from Kaggle competition

- We'll work with subsample of larger CTR dataset
- 39 masked user, ad and publisher features
- Full Kaggle dataset has 33M distinct categories (and this dataset is a small subset of Criteo's actual data)

Book Link

https://www.dropbox.com/s/6759pjjj9eulo6y/Algorithms_of_the_In_v9_MEAP.pdf?dl=0

5.5.1 Vowpal Wabbit data format

Vowpal Wabbit uses a specific file format that is very flexible but can be difficult to understand at first. Before we get started, we need to convert the file provided to us by Criteo into one that Vowpal Wabbit can understand. A sample of the raw data provided by Criteo is given in figure 5.4.

Class	Integer 1	Integer 2		Integer 12	Integer 13	Categorical 1	Categorical 2		Categorical 25	Categorical 26
0	1	7		5	0	68fd1e64	80e26c9b		7b4723c4	25c83c98
0	2	35	...	44	1	68fd1e64	f0cf0024	...	41274cd7	25c83c98
1	2	3		1	14	287e684f	0a519c5c		c18be181	25c83c98

Did the user interact?

13 properties expressed as integers. Likely to be event counts relating to user.

26 categorical properties associated with the bid request.

The method we use here is taken from the paper by Olivier Chapelle et al.[85] and is based on some simple observations. First, recall that, in our model, the log odds of the positive event occurring are linearly related to the input data, namely,

$$ln\left(Pr(y=1|\bar{x},\bar{\beta})\Big/Pr(y=-1|\bar{x},\bar{\beta})\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n$$

calibrate the model.

https://www.dropbox.com/s/s4x7wp8gjsh021d/TISTRespPredAds-Chappelle-CTR-Prediction-2014Paper.pdf?dl=0

where we now use the model probability in the numerator and denominator of this equation. If we apply Bayes' rule to the top and bottom of the odds and simplify, we obtain the following relationship:

$$\Pr(y=1|\bar{x})\Big/\Pr(y=-1|\bar{x}) = \Pr(\bar{x}|y=1)\Pr(y=1)/\Pr(\bar{x}|y=-1)\Pr(y=-1)$$

Using the fact that only negative instances of a click are down sampled and, as in Chapelle's paper, using Pr' to denote the resultant probability distribution after sampling the data down, then this may be equivalently written as follows.

$$\Pr(y=1|\bar{x})\Big/\Pr(y=-1|\bar{x}) = (\Pr'(\bar{x}|y=1)\Pr'(y=1))/(\frac{\Pr'(\bar{x}|y=-1)\Pr'(y=-1)}{r})$$

Where r is rate by which the negative samples have been down sampled. This leads us to the relationship between the odds in the original dataset, vs. that in the down sampled dataset:

$$\frac{\Pr(y=1|\bar{x})}{\Pr(y=-1|\bar{x})} = r\frac{Pr'(y=1|\bar{x})}{Pr'(y=-1|\bar{x})}$$

Thus, returning to our original model equation,

$$ln\left(r^{Pr'(y=1|\bar{x},\bar{\beta})}/_{Pr'(y=-1|\bar{x},\bar{\beta})}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n$$

$$\ln(r) + ln\left(Pr'(y=1|\bar{x},\bar{\beta})/_{Pr'(y=-1|\bar{x},\bar{\beta})}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n$$

$$ln\left(Pr'(y=1|\bar{x},\bar{\beta})/_{Pr'(y=-1|\bar{x},\bar{\beta})}\right) = (\beta_0 - \ln(r)) + \beta_1 x_1 + \cdots + \beta_n x_n$$

Thus, you can train on a down-sampled data set, removing only negative training instances, but if you wish to obtain output from the model that reflects the probability of the positive event accurately, you must complete a further step.

As you can see, your trained model will have an intercept that's ln(r) lower than that which would be obtained if you didn't used a down-sampled set. So if you wish to correct for this, you must add ln(r) to the coefficient b_0 after training. Don't forget, though, that although we've down-sampled the data, it was also down-sampled by an unknown amount by Criteo! Consequently, we're unable to calibrate our model effectively without additional information.

jupyter MIDS-MLS-Project-Criteo-CTR Last Checkpoint: 4 hours ago (autosaved)

File Edit View Insert Cell Kernel Help Python 2

Markdown Cell Toolbar: None

Click-Through Rate Prediction Lab

This lab covers the steps for creating a click-through rate (CTR) prediction pipeline. You will work with the Criteo Labs dataset that was used for a recent Kaggle competition.

This lab will cover:

- ***Part 1:*** **Featurize categorical data using one-hot-encoding (OHE)**
- ***Part 2:*** **Construct an OHE dictionary**
- ***Part 3:*** **Parse CTR data and generate OHE features**
  - ***Visualization 1:*** **Feature frequency**
- ***Part 4:*** **CTR prediction and logloss evaluation**
  - ***Visualization 2:*** **ROC curve**
- ***Part 5:*** **Reduce feature dimension via feature hashing**

Notebook

https://www.dropbox.com/s/if47ua2frf5qt4o/MIDS-MLS-Project-Criteo-CTR.ipynb?dl=0

Live Session Outline: Week 12

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
  - Housekeeping:
    - homework schedule
- **HW11**
- **Lecture 12**
  - Decision Trees
- **Categorical feature encoding**
  - (OHE, Hashing)
- **Digital advertining**
  - Project: CTR Prediction
- **Submitting a Spark Job (Locally and on a cluster on AWS)**
- **Finish RECORDING (bonus points for reminding me!)**

Cover next time

- **End of Lecture**