Categorical Data and One-Hot-Encoding

Logistic Regression Optimization

Regularized

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

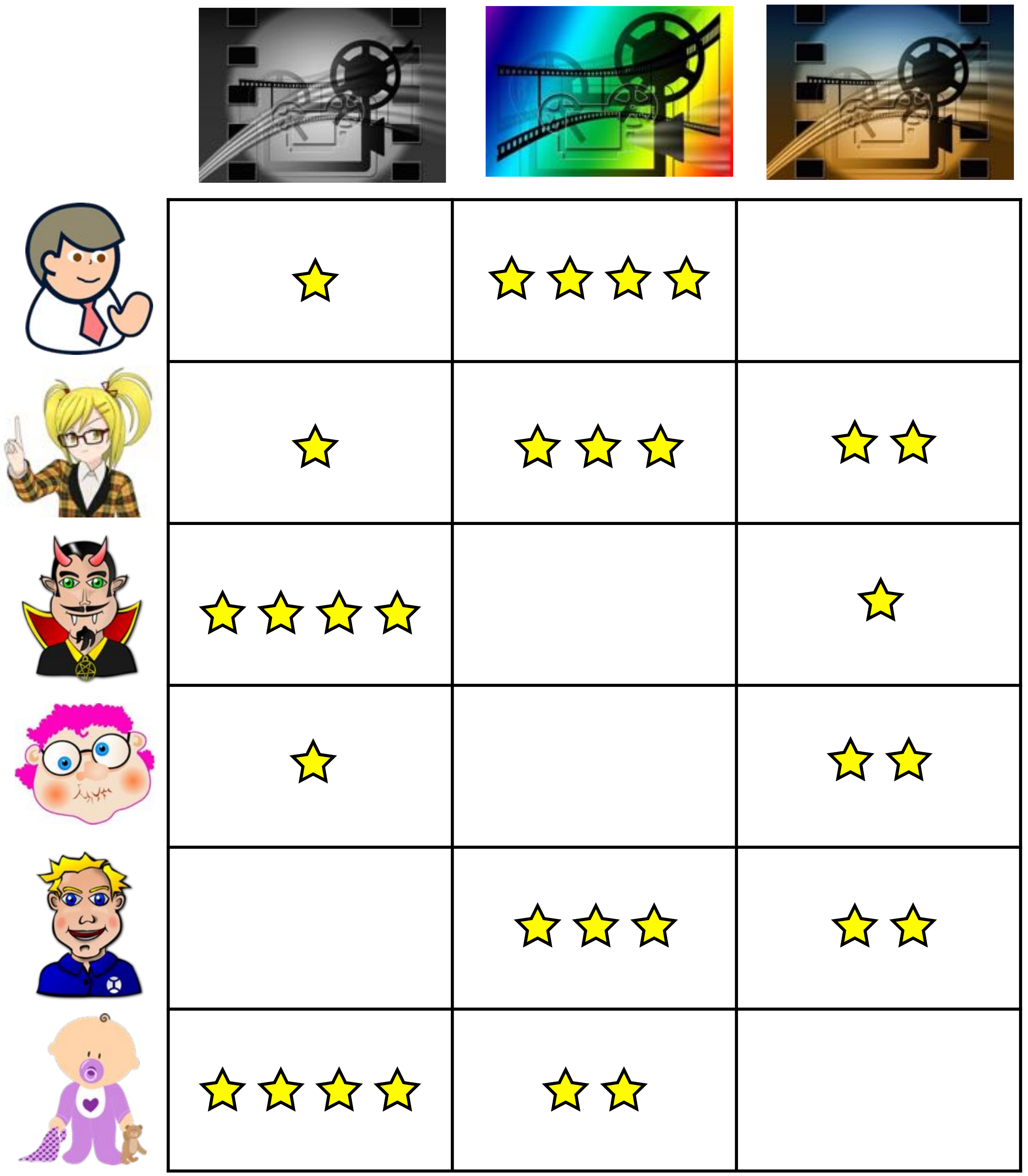
$$\min_{\mathbf{w}} \sum_{i=1}^{n} \overbrace{\ell_{0/1}\left(y^{(i)} \cdot \mathbf{w}^\top \mathbf{x}^{(i)}\right)}^{\text{Training LogLoss}} + \overbrace{\lambda ||\mathbf{w}||_2^2}^{\text{Model Complexity}}$$

Data is assumed to be **numerical**!

Similar story for linear regression and many other methods

Raw Data is Sometimes Numeric

Images



User Ratings

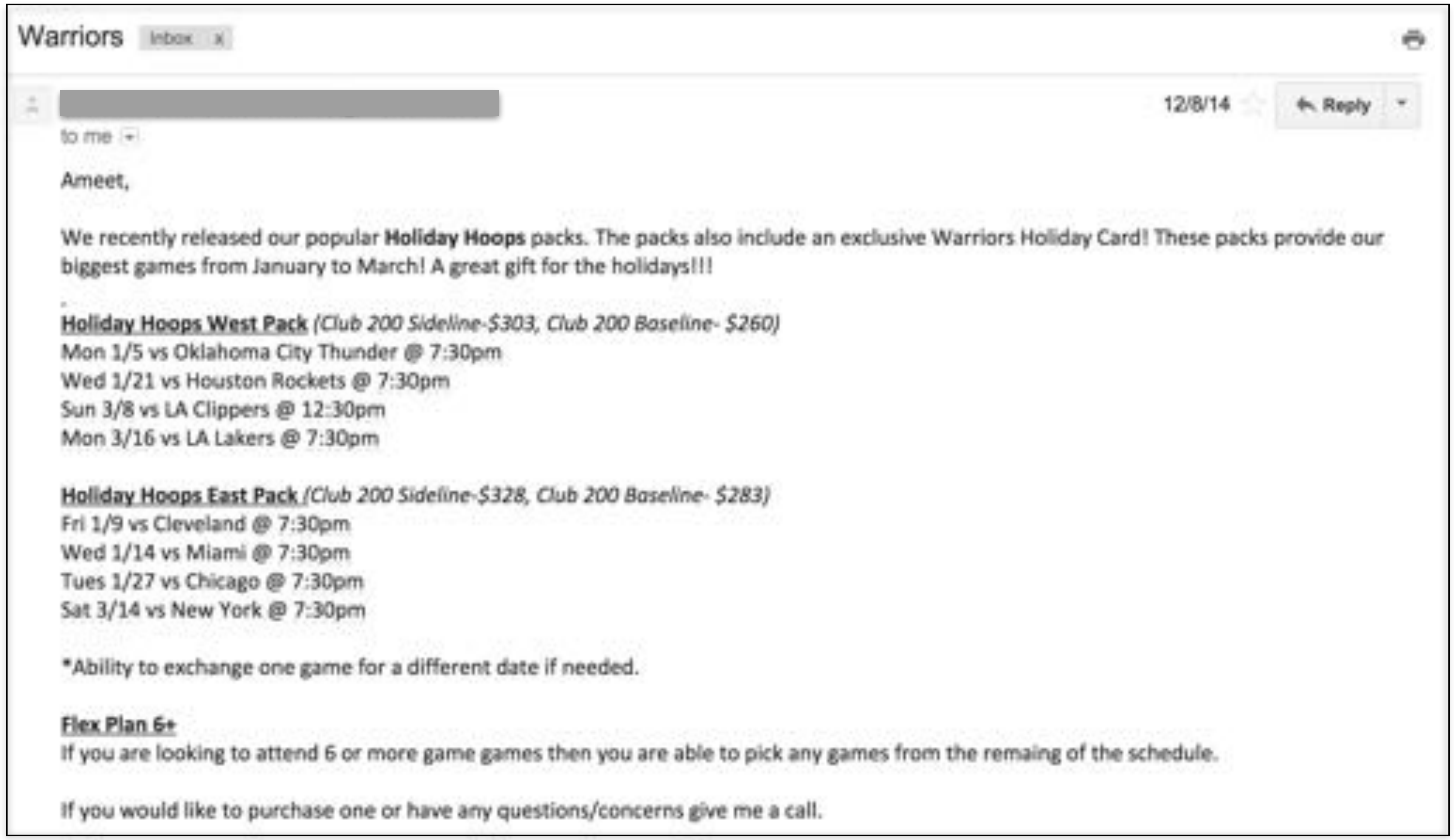
Raw Data is Often Non-Numeric

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content=
        "text/html; charset=us-ascii" />
        <script type="text/javascript">
            function reDo() {top.location.reload();}
            if (navigator.appName == 'Netscape') {top.onresize = reDo;}
            dom=document.getElementById;
        </script>
    </head>
    <body>
    </body>
</html>
```

Web hypertext

Email



Warriors Inbox x

[redacted] 12/8/14 Reply

to me

Ameet,

We recently released our popular **Holiday Hoops** packs. The packs also include an exclusive Warriors Holiday Card! These packs provide our biggest games from January to March! A great gift for the holidays!!!

Holiday Hoops West Pack *(Club 200 Sideline-$303, Club 200 Baseline- $260)*
Mon 1/5 vs Oklahoma City Thunder @ 7:30pm
Wed 1/21 vs Houston Rockets @ 7:30pm
Sun 3/8 vs LA Clippers @ 12:30pm
Mon 3/16 vs LA Lakers @ 7:30pm

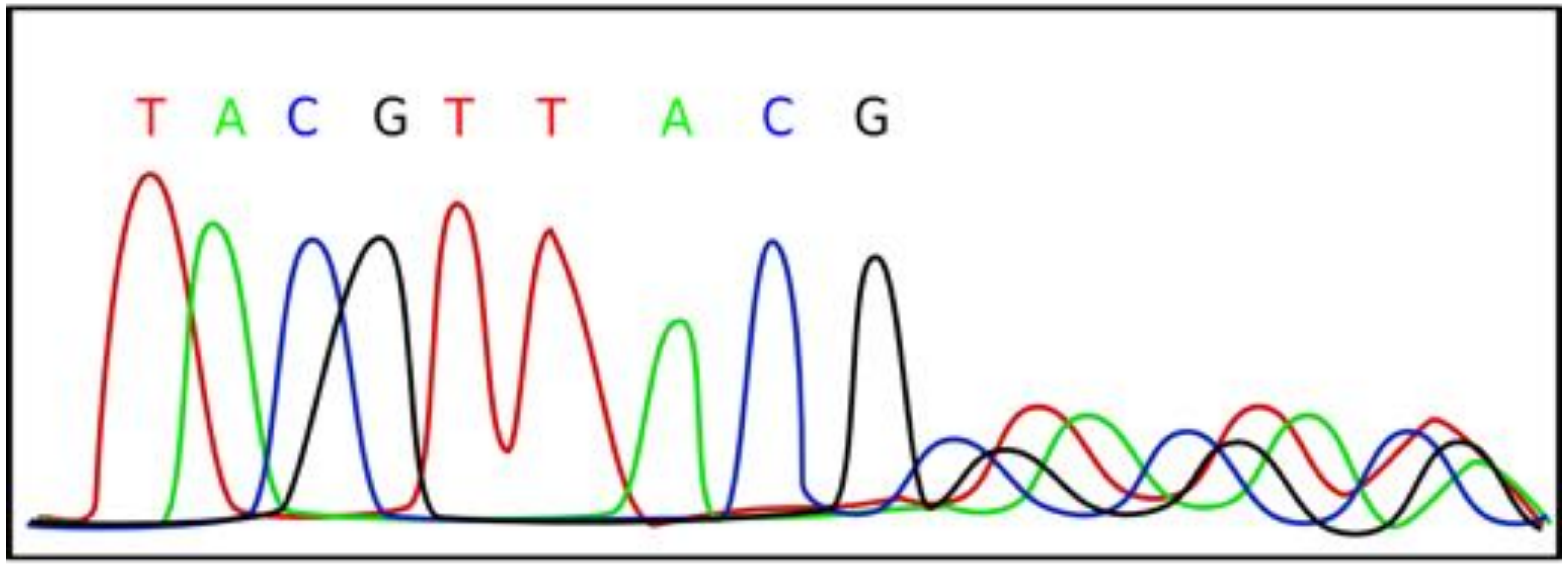
Holiday Hoops East Pack *(Club 200 Sideline-$328, Club 200 Baseline- $283)*
Fri 1/9 vs Cleveland @ 7:30pm
Wed 1/14 vs Miami @ 7:30pm
Tues 1/27 vs Chicago @ 7:30pm
Sat 3/14 vs New York @ 7:30pm

*Ability to exchange one game for a different date if needed.

Flex Plan 6+
If you are looking to attend 6 or more game games then you are able to pick any games from the remaing of the schedule.

If you would like to purchase one or have any questions/concerns give me a call.

Genomic Data



Raw Data is Often Non-Numeric

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

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

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

One idea: Create single numerical feature to represent non-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

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

Computing and Storing OHE Features

Example: Categorical Animal Dataset

Features:

- Animal = {‘bear’, ‘cat’, ’mouse’}
- Color = {‘black’, ‘tabby’}
- Diet (optional) = {‘mouse’, ‘salmon’}

Datapoints:

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

How can we create OHE features?

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 each category to dummy feature

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

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
- …

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)]

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

Feature Hashing

UCLA Cal

databricks

Non-Numeric ⇒ Numeric

One-Hot-Encoding: Create a ‘dummy’ feature for each category

Creating dummy features doesn’t introduce spurious relationships

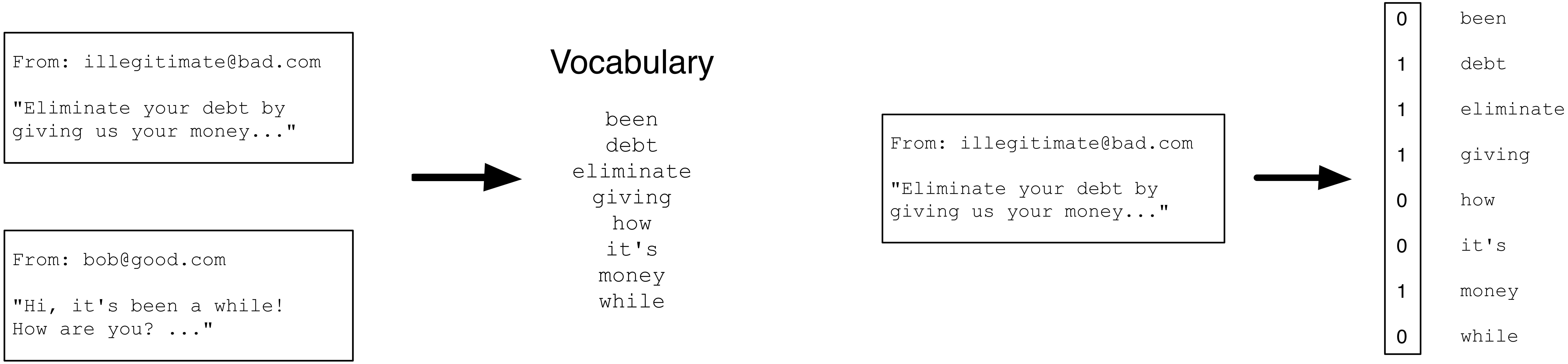
Dummy features can drastically increase dimensionality

- Number of dummy features equals number of categories!

Issue with CTR prediction data

- Includes many names (of products, advertisers, etc.)
- Text from advertisement, publisher site, etc.

"Bag of Words" Representation



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)

High Dimensionality of OHE

Statistically: Inefficient learning

- We generally need bigger n when we have bigger d (though in distributed setting we often have very large n)
- We will have many non-predictive features

Computationally: Increased communication

- Linear models have parameter vectors of dimension d
- Gradient descent communicates the parameter vector to all workers at each iteration

How Can We Reduce Dimension?

One Option: Discard rare features

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

Can view as an unsupervised learning preprocessing step

Another Option: Feature hashing

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

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

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

Hashed Features:

- A1 = [0 0 1 1]

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

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]
- A3 = [1 1 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

- H(Animal, ’bear’) = 0
- H(Color, ’black’) = 2
- H(Diet, ’salmon’) = 1

Why Is This 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

Distributed Computation

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

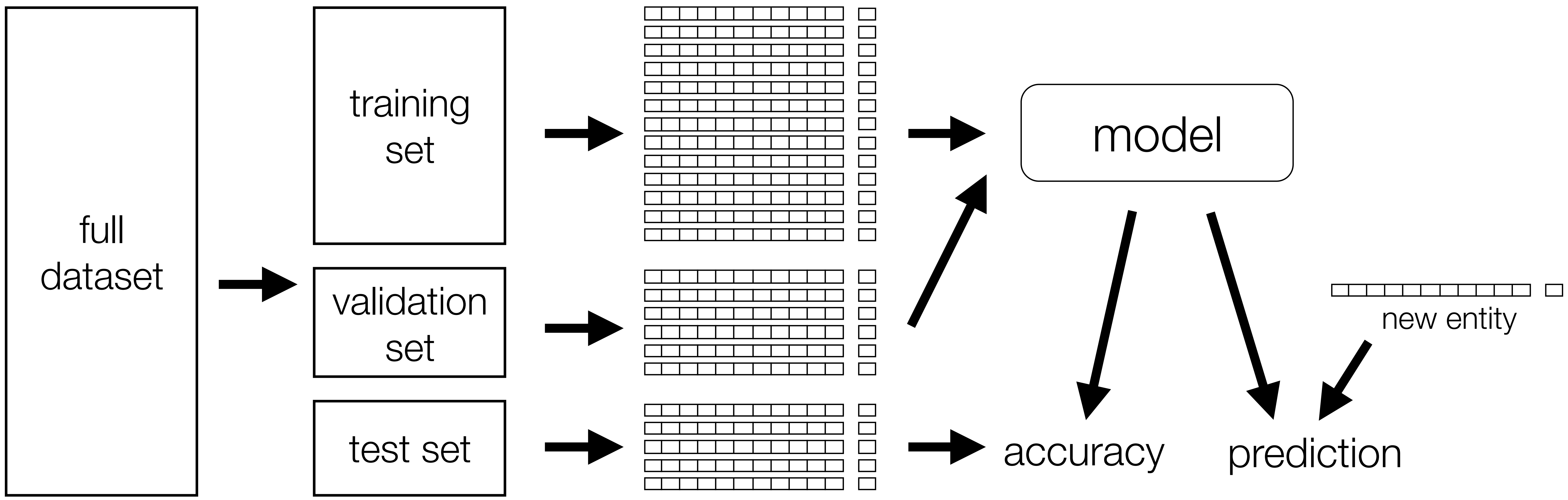
Step 1: Apply hash function on raw data

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

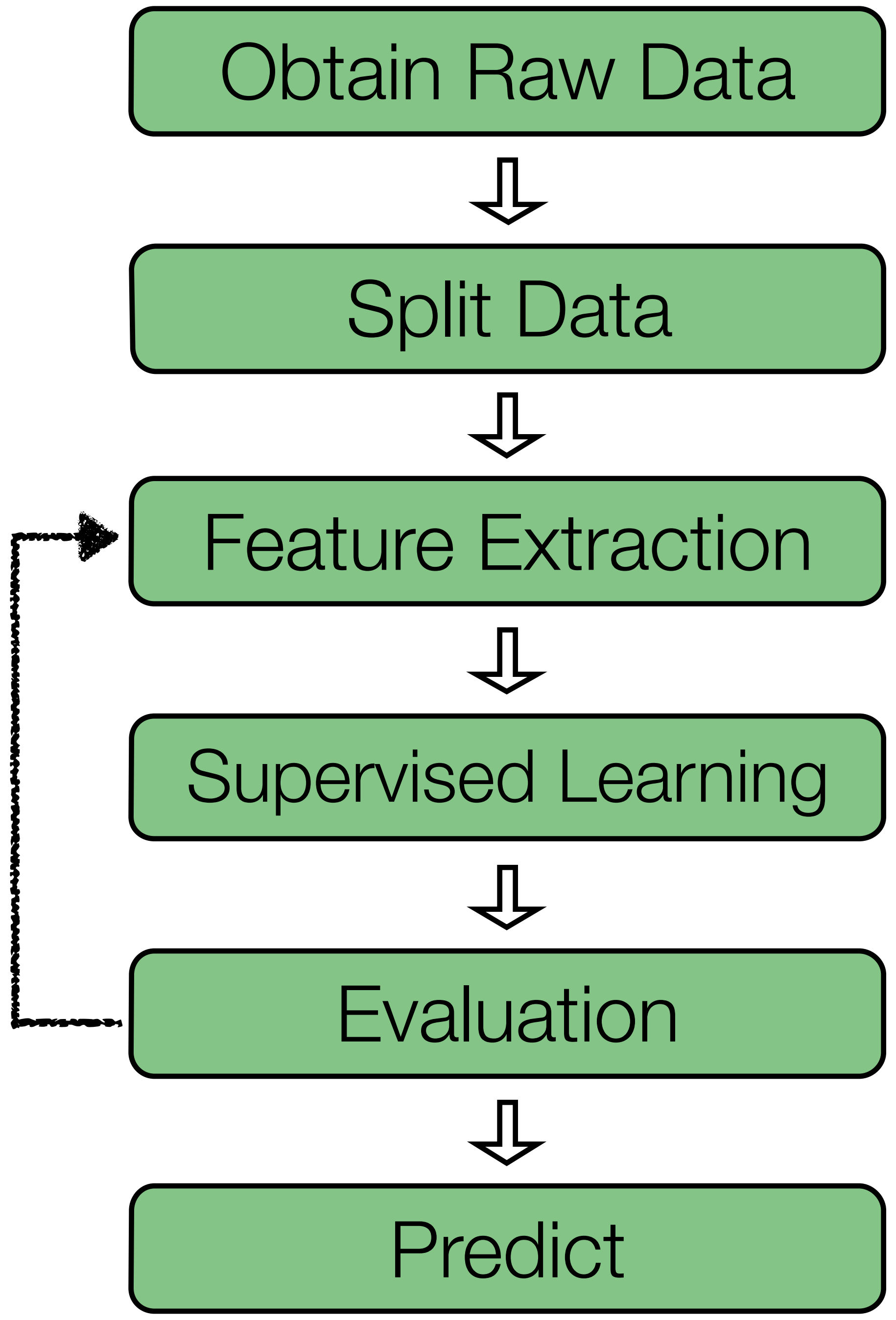
Step 2: Store hashed features in sparse representation

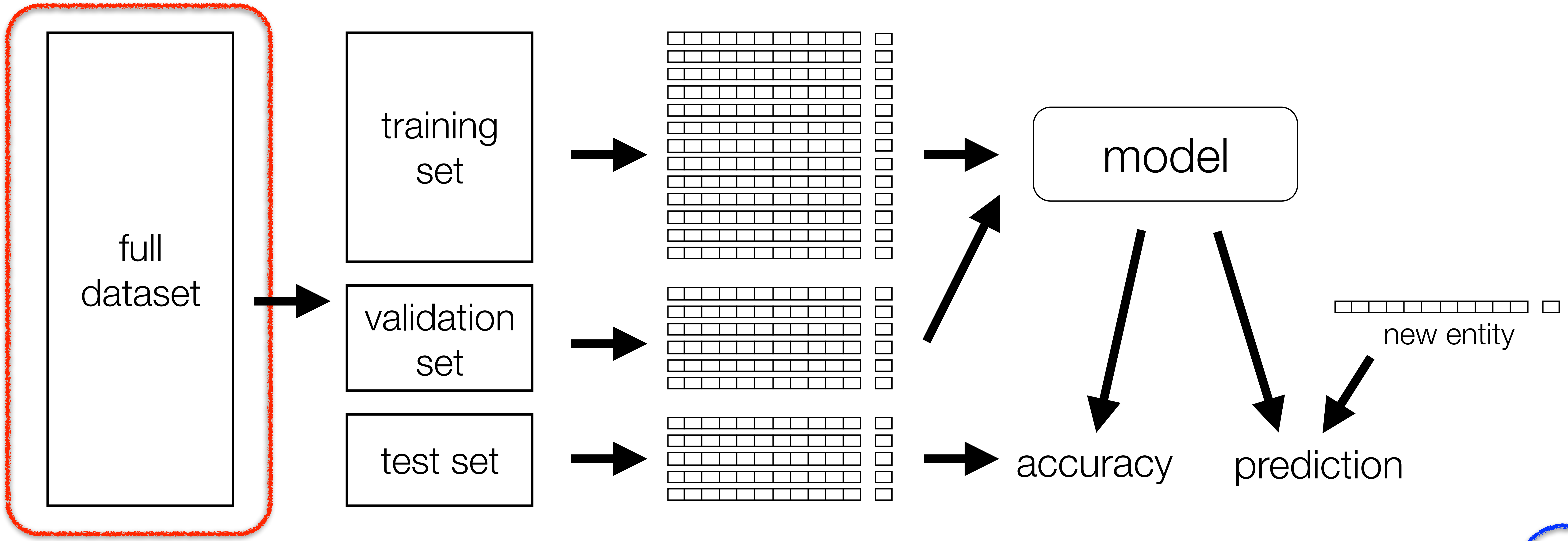
- Local computation
- Saves storage and speeds up computation

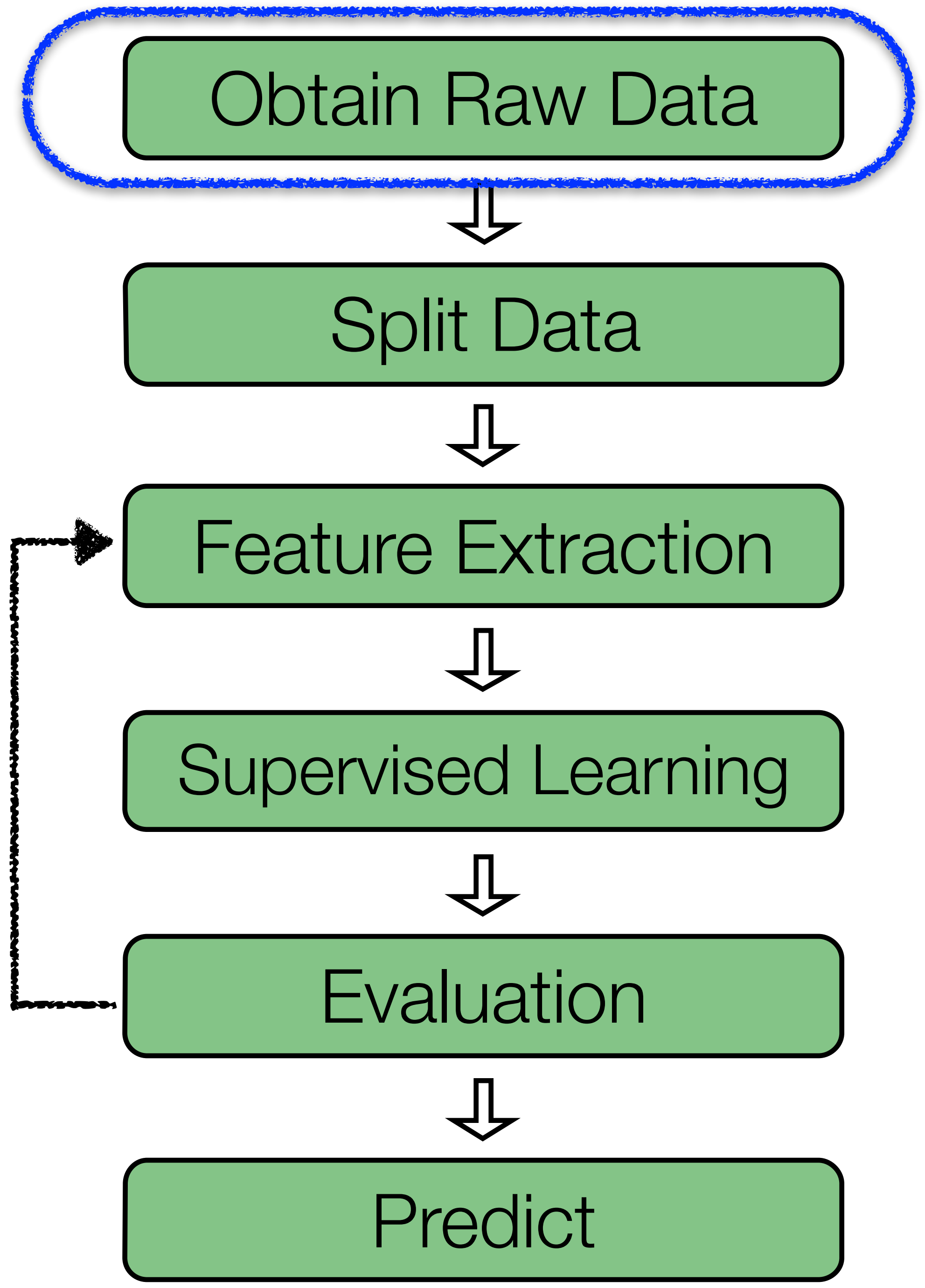
CTR Prediction Pipeline / Lab Preview



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

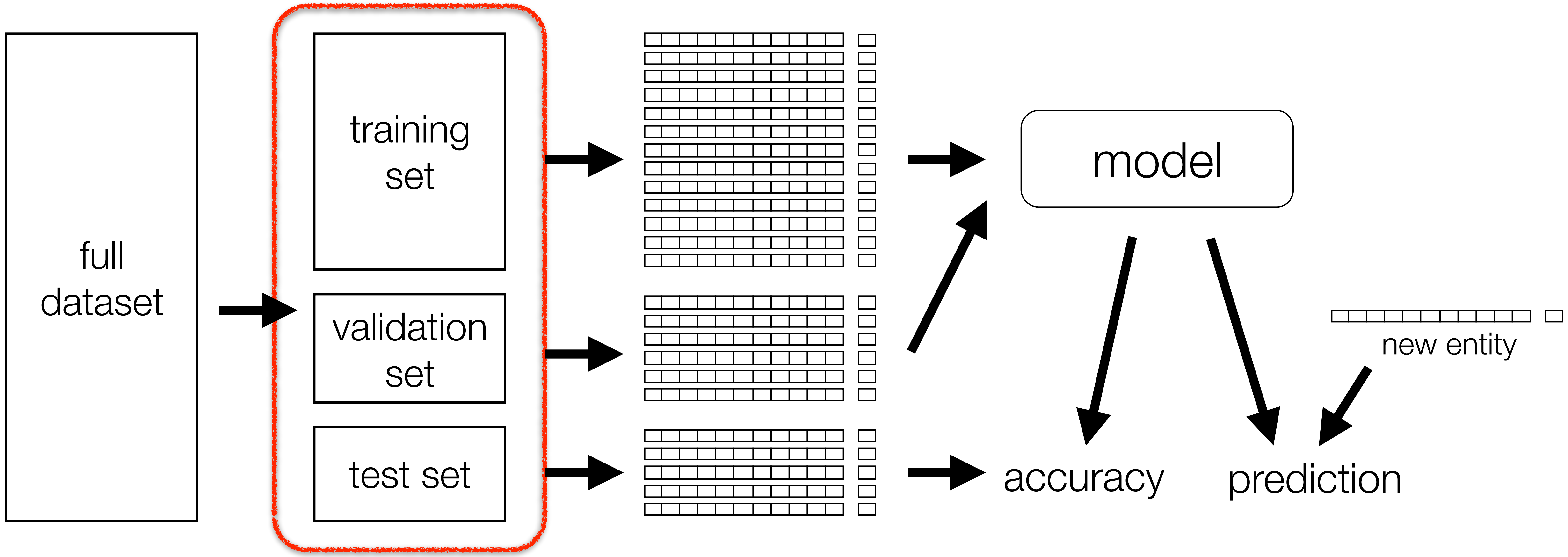




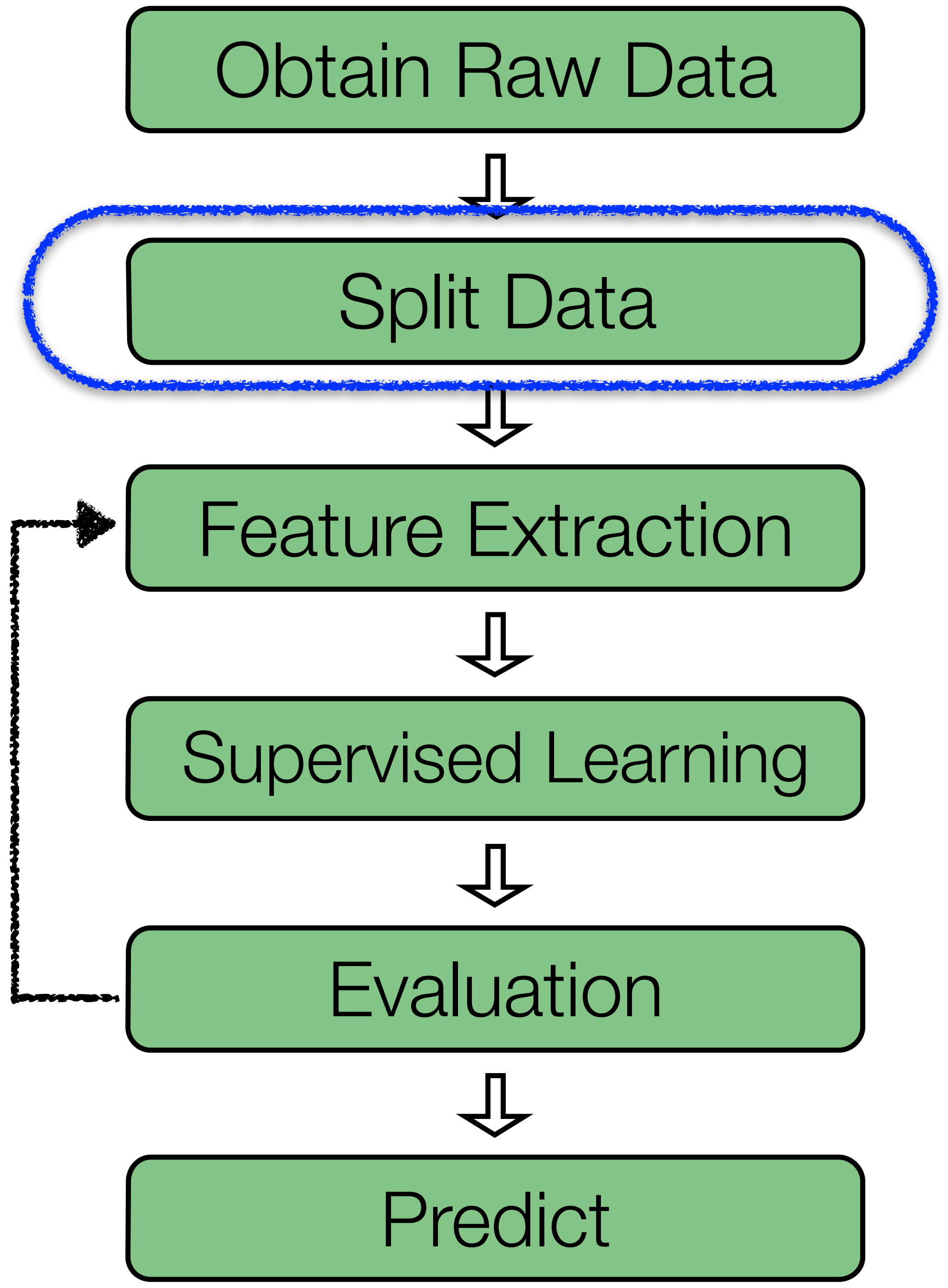


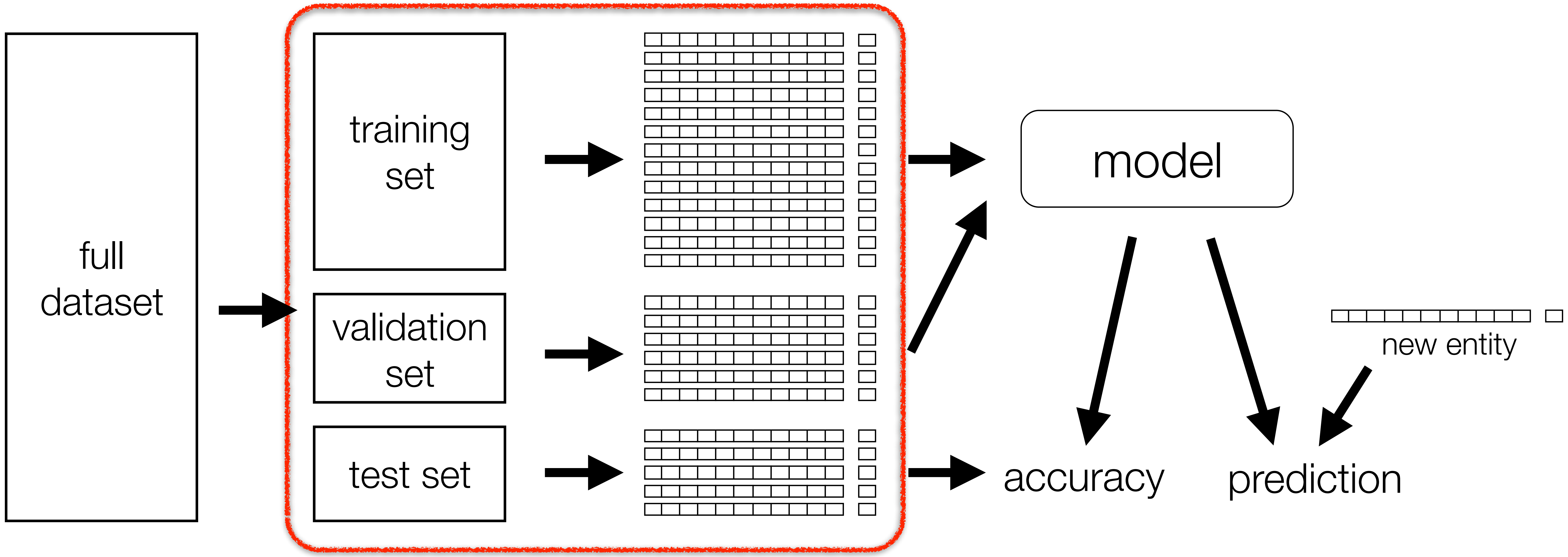
Raw Data: Criteo Dataset from Kaggle competition

- 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)



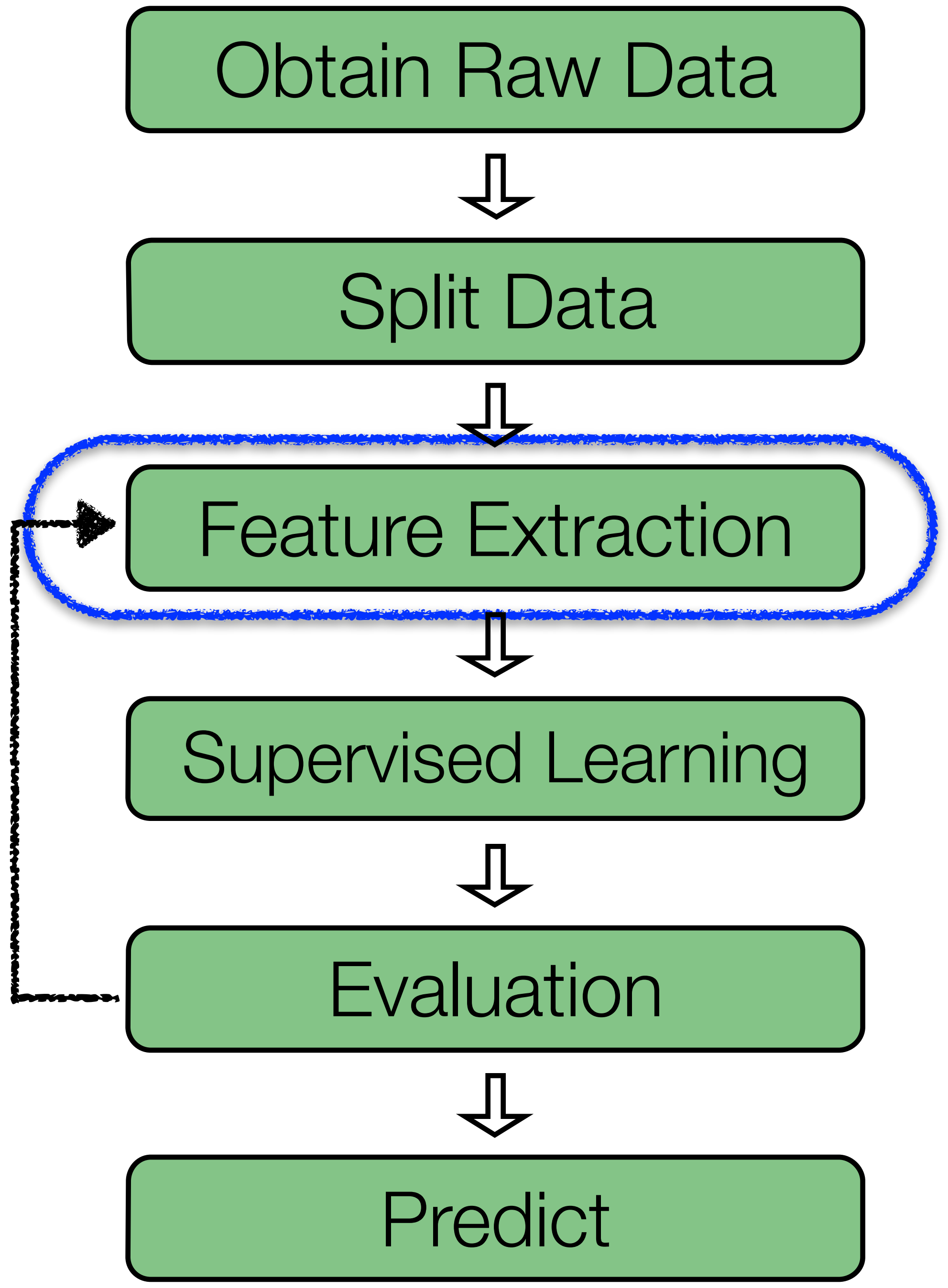
Split Data: Create training, validation, and test sets

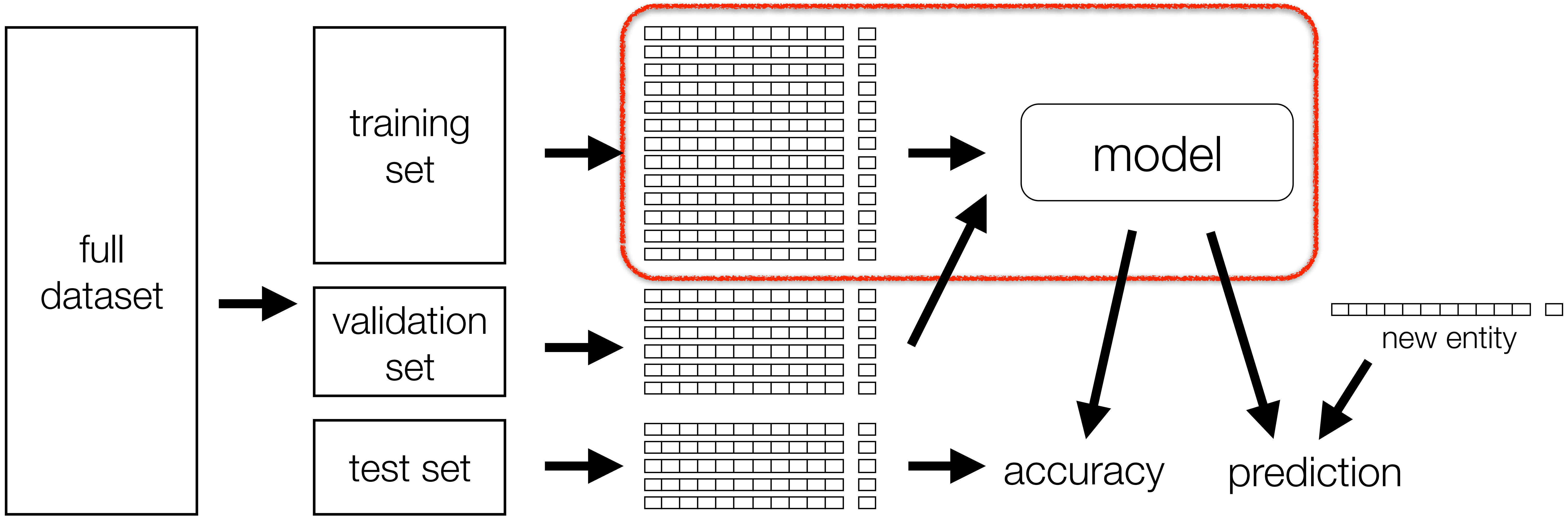


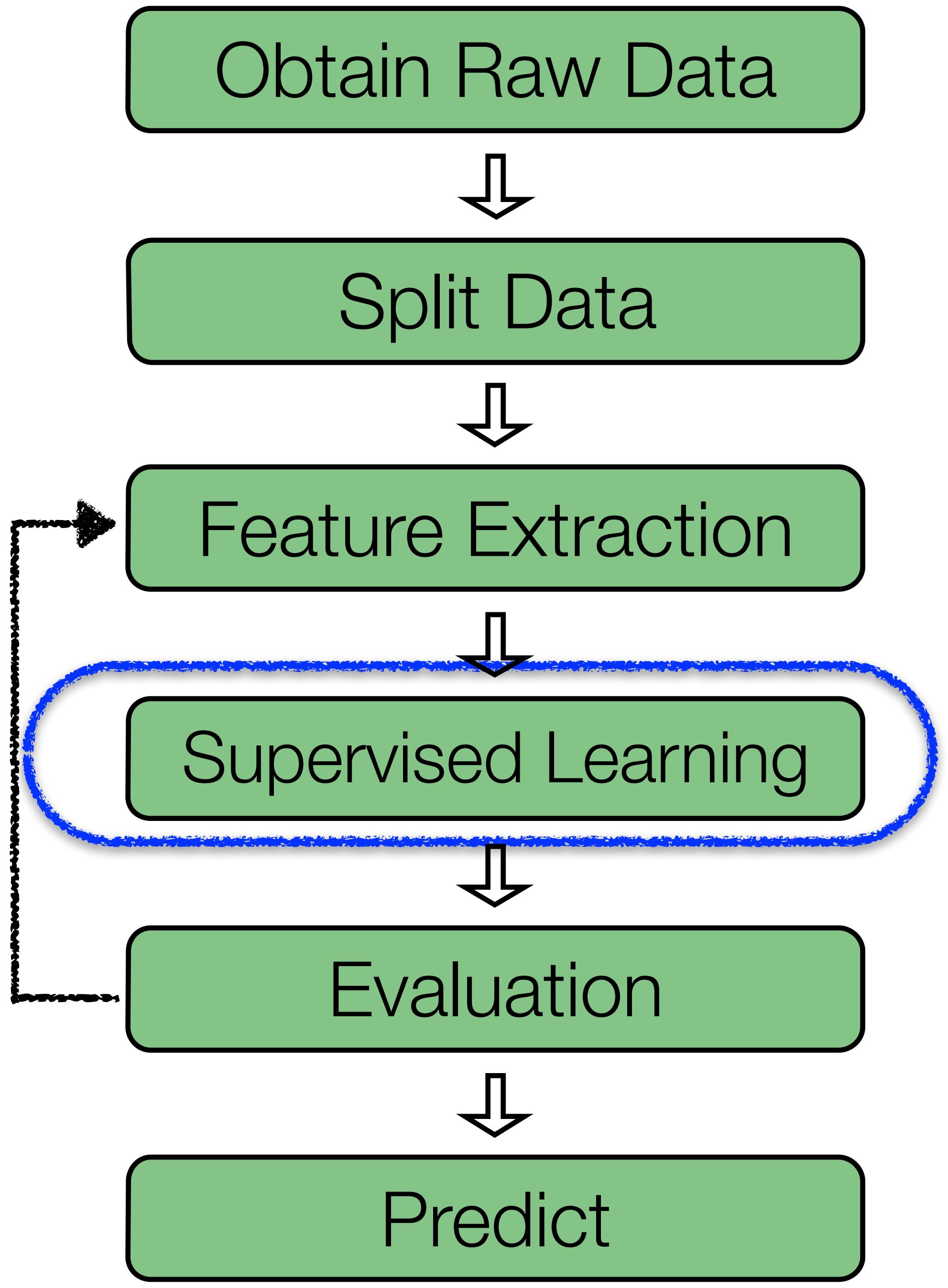


Feature Extraction: One-hot-encoding and feature hashing

- We'll use a sparse data representation
- We'll visualize feature frequency
- Feature extraction is the main focus of this lab

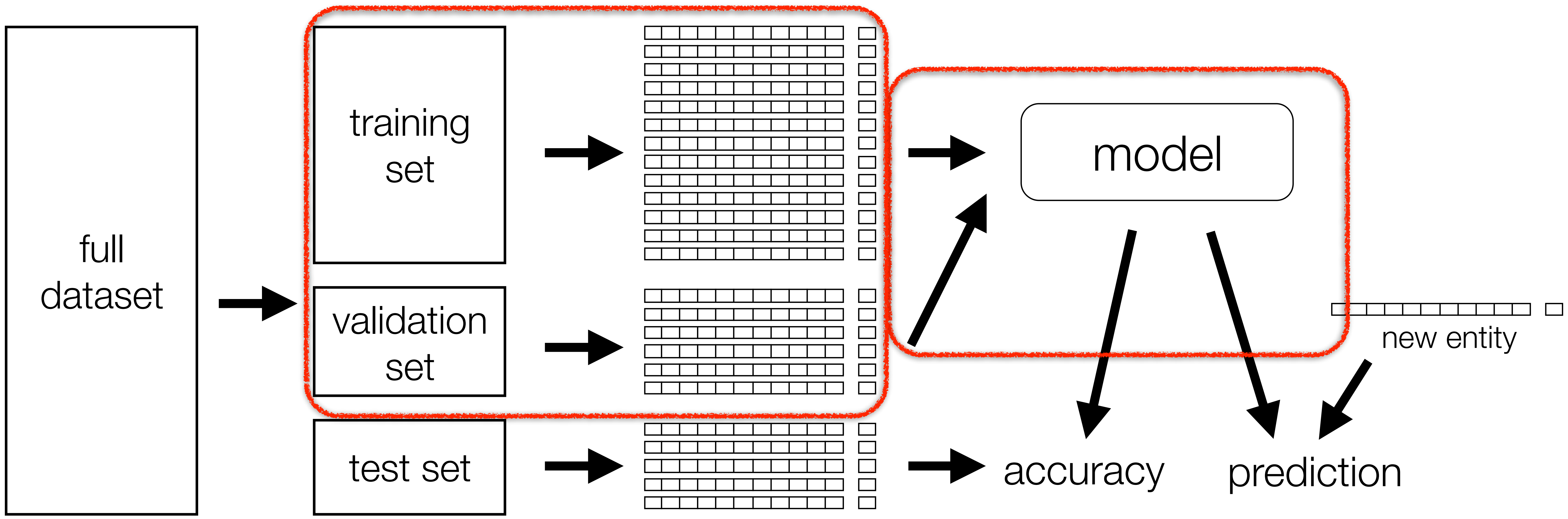






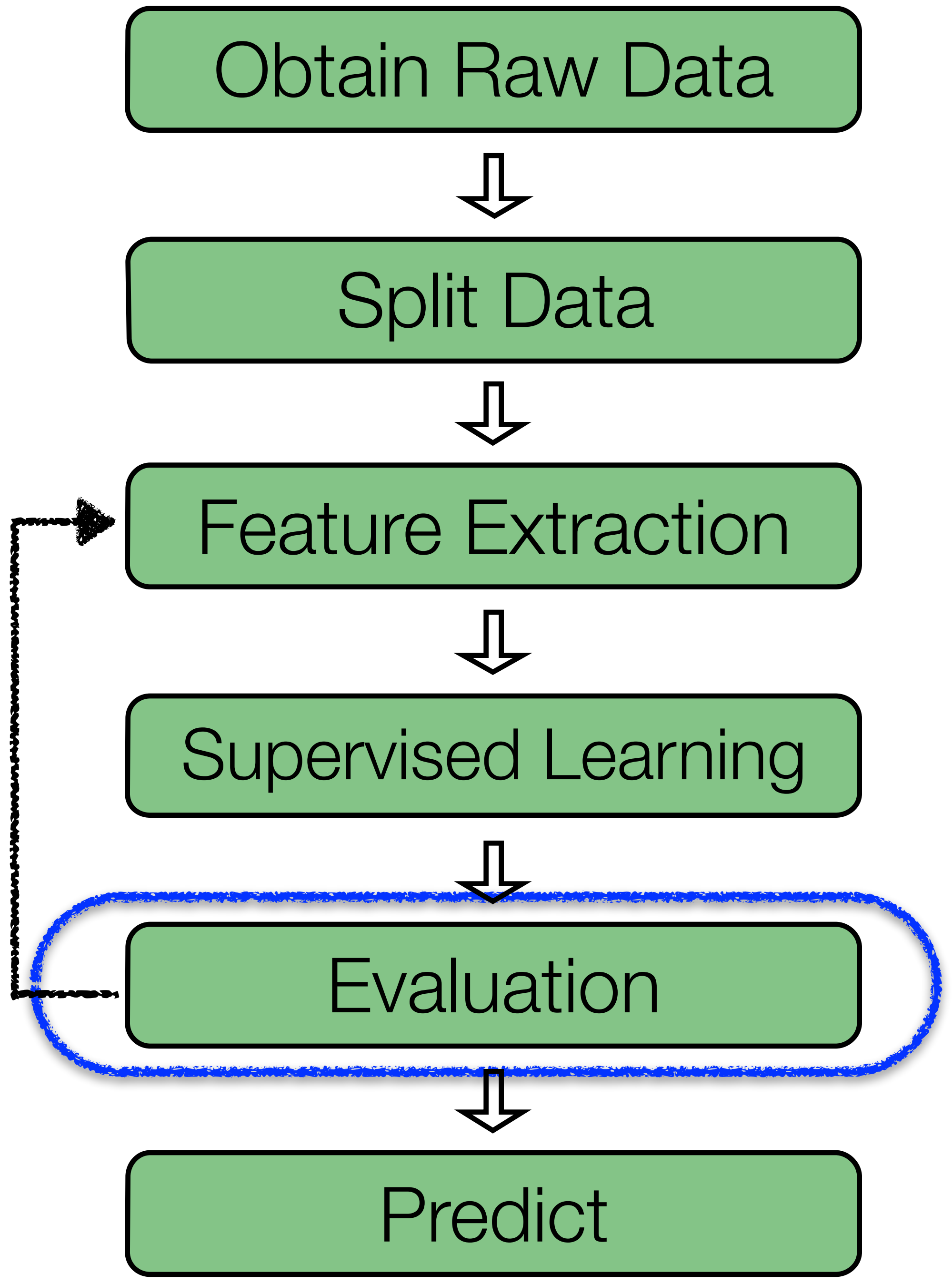
Supervised Learning: Logistic regression

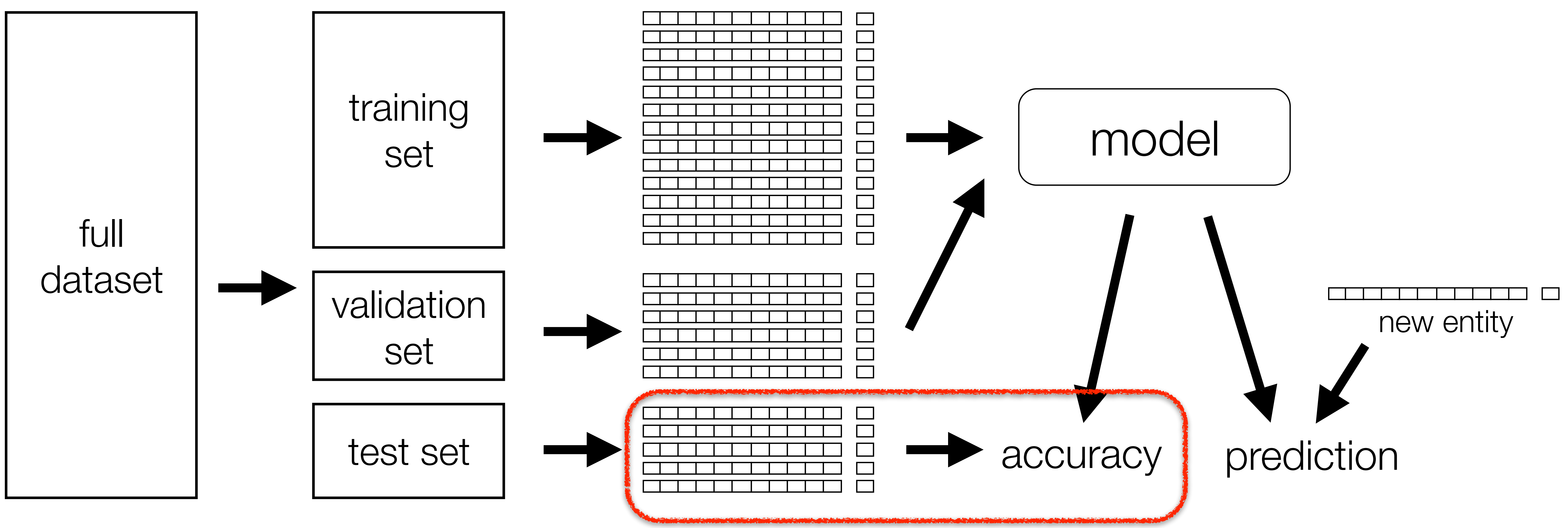
- Use MLlib implementation

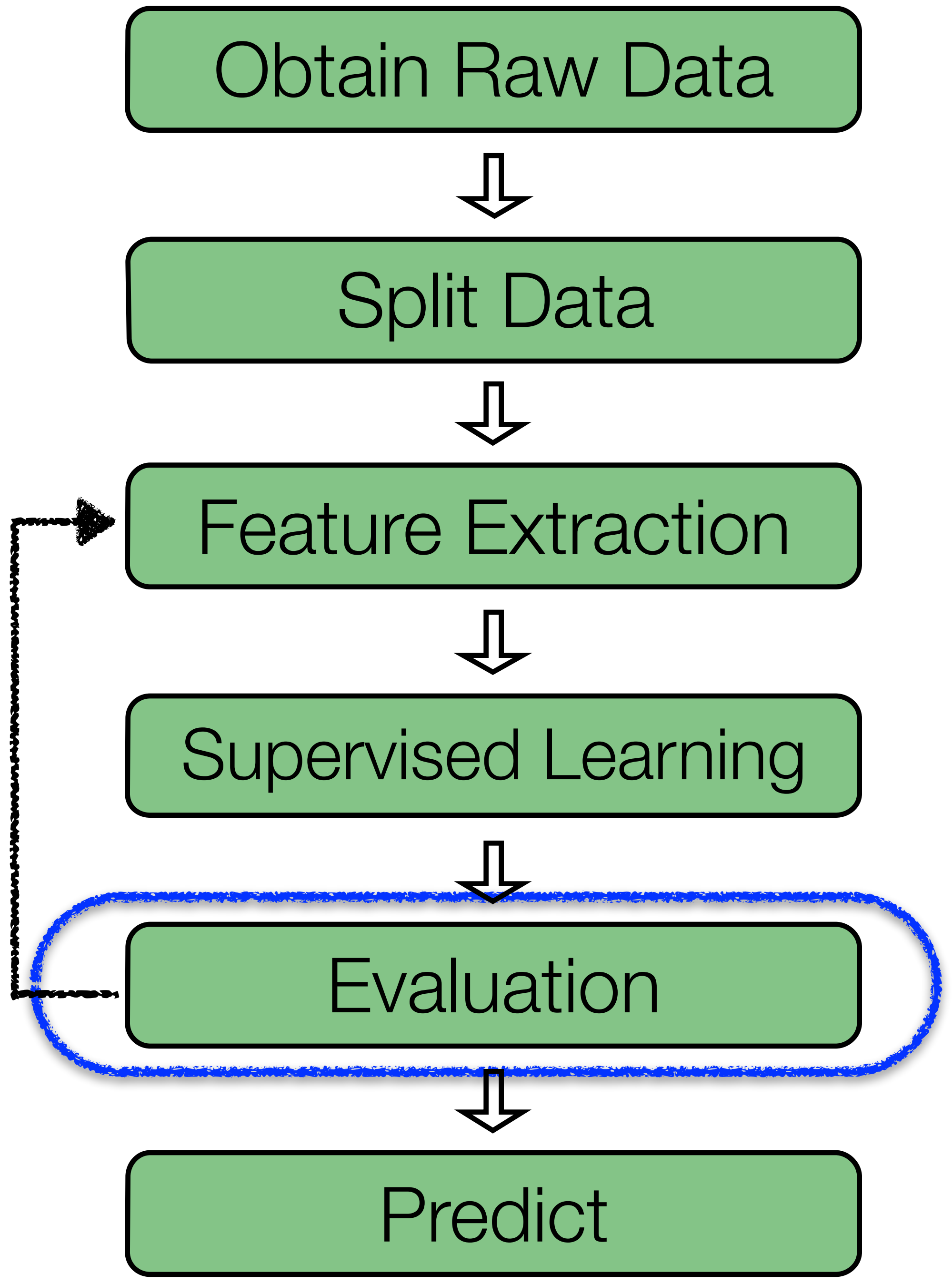


Evaluation (Part 1): Hyperparameter tuning

- Grid search to find good values for regularization
- Evaluate using logistic loss
- Visualize grid search
- Visualize predictions via ROC curve

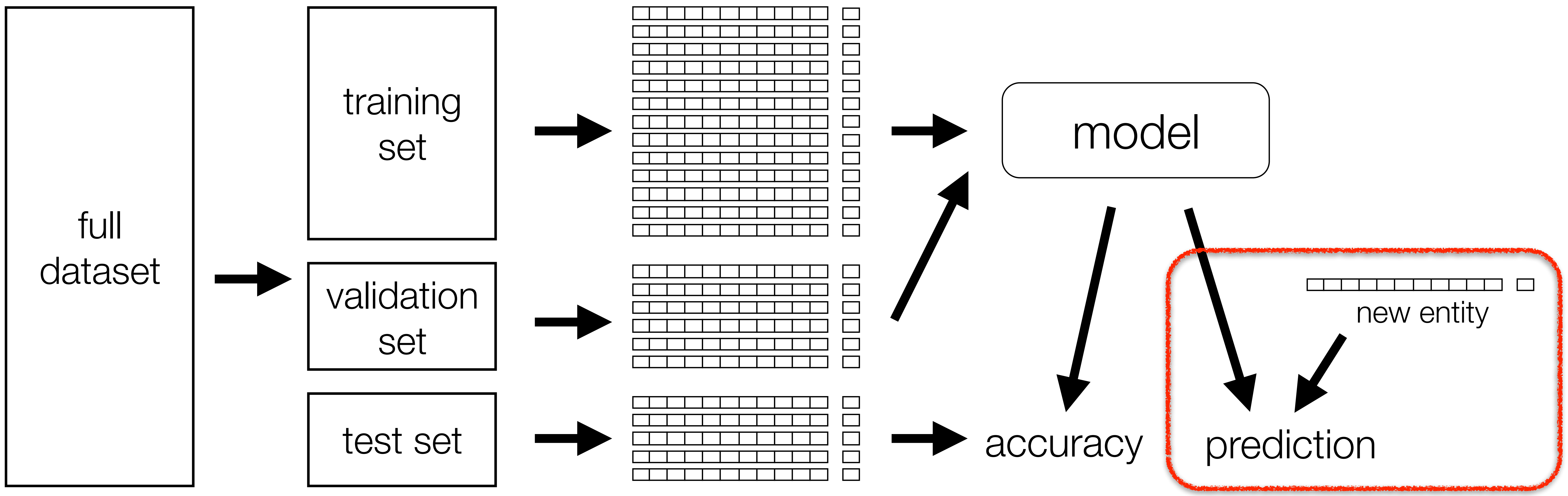






Evaluation (Part 2): Evaluate final model

- Evaluate using logistic loss
- Compare to baseline model (always predict value equal to the fraction of training points that correspond to click-through events)



Predict: Final model could be used to predict click-through rate for new user-ad tuple (we won't do this though)