Machine Learning (1/2)

Outline

- This Lecture (~~Wes~~Pieter)
  - Intro to Machine Learning
  - Relationship to Programming Languages
  - Taxonomy of ML Approaches
  - Basic Clustering
  - Basic Linear Models
- Next Lecture (Ray)
  - Advanced ML Algorithms (e.g., Baysean Learning, Decision Trees, Support Vector Machines, Neural Networks...)
  - Concerns and Evaluation Techniques

Modeling Bug Report Quality

Pieter Hooimeijer and Westley Weimer
University of Virginia
Charlottesville, VA. 22903
{pieter, weimer}@cs.virginia.edu *

ABSTRACT

Software developers spend a significant portion of their resources handling user-submitted bug reports. For software that is widely deployed, the number of bug reports typically outstrips the resources available to triage them. As a result, some reports may be dealt with too slowly or not at all.

We present a descriptive model of bug report quality based on a statistical analysis of surface features of over 27,000 publicly available bug reports for the Mozilla Firefox project. The model predicts whether a bug report is triaged within a given amount of time. Our analysis of this model has implications for bug reporting systems and suggests features that should be emphasized when composing bug reports.

We evaluate our model empirically based on its hypothetical performance as an automatic filter of incoming bug reports. Our results show that our model performs significantly better than chance in terms of precision and recall. In addition, we show that our model can reduce the overall cost of software maintenance in a setting where the average cost of addressing a bug report is more than 2% of the cost of ignoring an important bug report.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*Debugging aids*; D.2.8 [**Software Engineering**]: Metrics; D.2.9 [**Software Engineering**]: Management—*Life cycle*; D.2.9 [**Software Engineering**]: Management—*Time estimation*

General Terms

Economics, Experimentation, Human Factors, Management, Measurement

*This research was supported in part by National Science Foundation Grants CNS 0627523 and CNS 0716478 and Air Force Office of Scientific Research grant BAA 06-028, as well as gifts from Microsoft Reserach. The information presented here does not necessarily reflect the position or the policy of the government and no official endorsement should be inferred.

ASE'07, November 5–9, 2007, Atlanta, Georgia, USA.

Keywords

bug report triage, issue tracking, statistical model, information retrieval

1. INTRODUCTION

A significant portion of overall software development is spent addressing defects. Boehm and Basili claim that maintenance consumes over 70% of the total lifecycle cost of a software product [5]. Modifying existing code, dealing with defects, and otherwise evolving software are major parts of that maintenance [12]. Large projects often use bug reporting and triage systems to cope with defect reports [2]. However, the number of reports typically exceeds the resources available to address them; rather than having the development resources to deal with every defect, even mature software projects are forced to ship with both known and unknown bugs [10]. A lack of resources often constrains developers to deal with some bug reports too slowly or not at all.

A further complication is that many submitted bug reports may be spurious duplicates or descriptions of non-defects. Previous studies have found that as many as 36% of bug reports were duplicates or otherwise invalid [3]. The triage work in evaluating bug reports consumes developer time and effort [14]. Bug report triage and evaluation are a significant part of modern software engineering for many large projects.

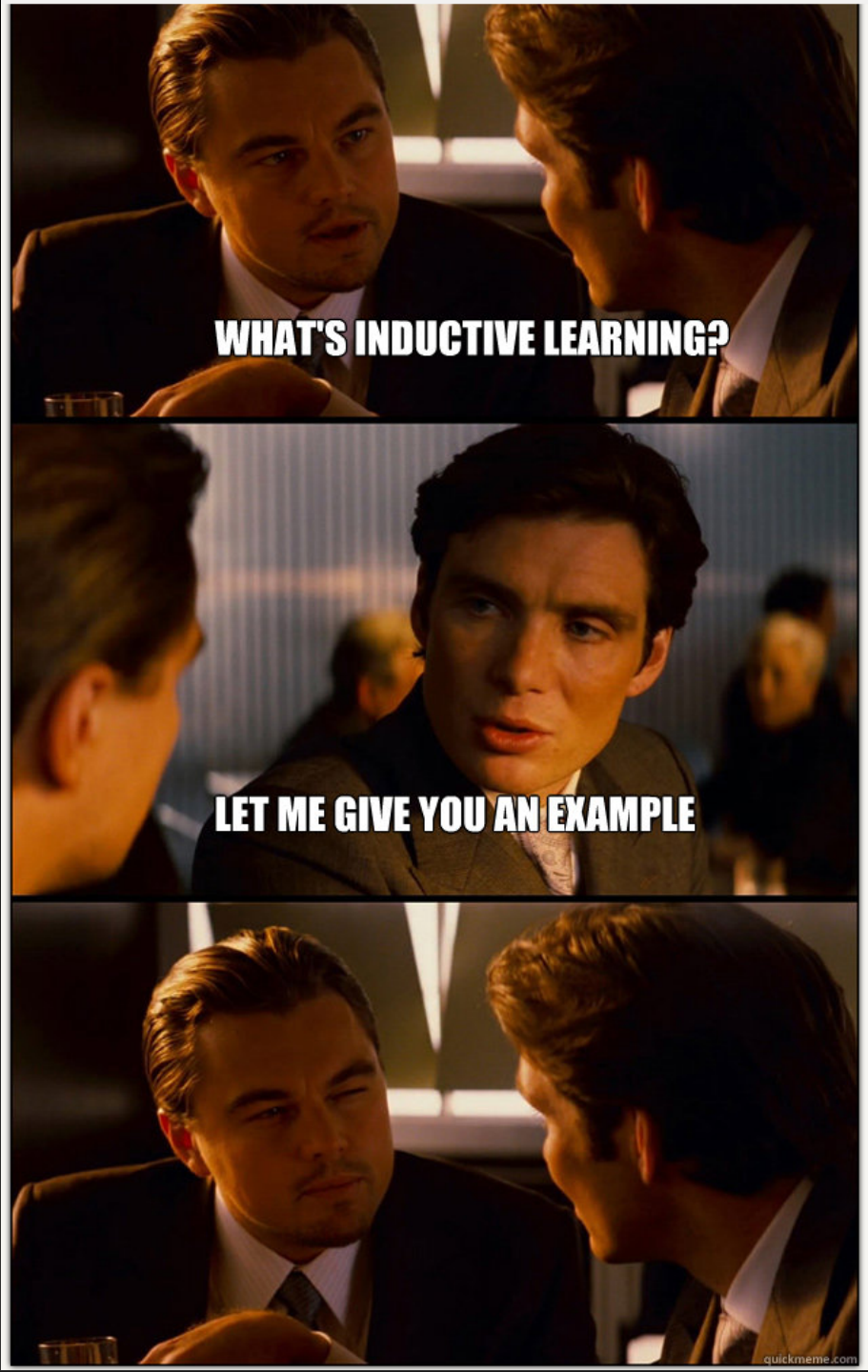
In this paper we attempt to reduce bug report triage costs by separating the wheat from the chaff. We present a model of bug report quality that predicts whether developers will choose to address a bug report, by measuring whether the report is addressed within a given amount of time. We do this by classifying bug reports as either "cheap" or "expensive" to triage. For the purposes of this paper, *triage* is the act of inspecting a bug report, understanding its contents, and making the initial decision regarding how to address the report.

Our model is based on features that can easily be gathered from bug report submissions, without referring to past bug reports. We base our model on a statistical analysis of over 27,000 bug reports from the Mozilla Firefox project, and we experimentally validate its predictive power. Our model can be used by developers to filter incoming bug reports or to aid with bug report prioritization.

In practice, Mozilla and many other open source software projects make use of bug reporting and triage software [2] that is open to the public. The assumption is that allowing users to report and potentially help fix bugs improves

Machine Learning Defined

- **Machine learning** is a subfield of AI concerned with algorithms that allow computers to *learn*. There are two types of learning:
  - **Deductive** learning uses axioms and rules of inference to construct new true judgments. See "Automated Theorem Proving" lecture.
  - **Inductive** learning method extract rules and patterns out of massive datasets. Given many examples, they attempt to generalize. We'll discuss this now.


WHAT'S INDUCTIVE LEARNING?
LET ME GIVE YOU AN EXAMPLE
quickmeme.com

Machine Learning in Context

- Machine Learning is sometimes called the part of AI that works in practice. (cf. “AI complete”)
- ML combines statistics and data mining with algorithms and theory
- Successful applications of ML:
  - detecting credit card fraud; stock market prediction; speech and handwriting recognition; medical diagnosis; market basket analysis; ...

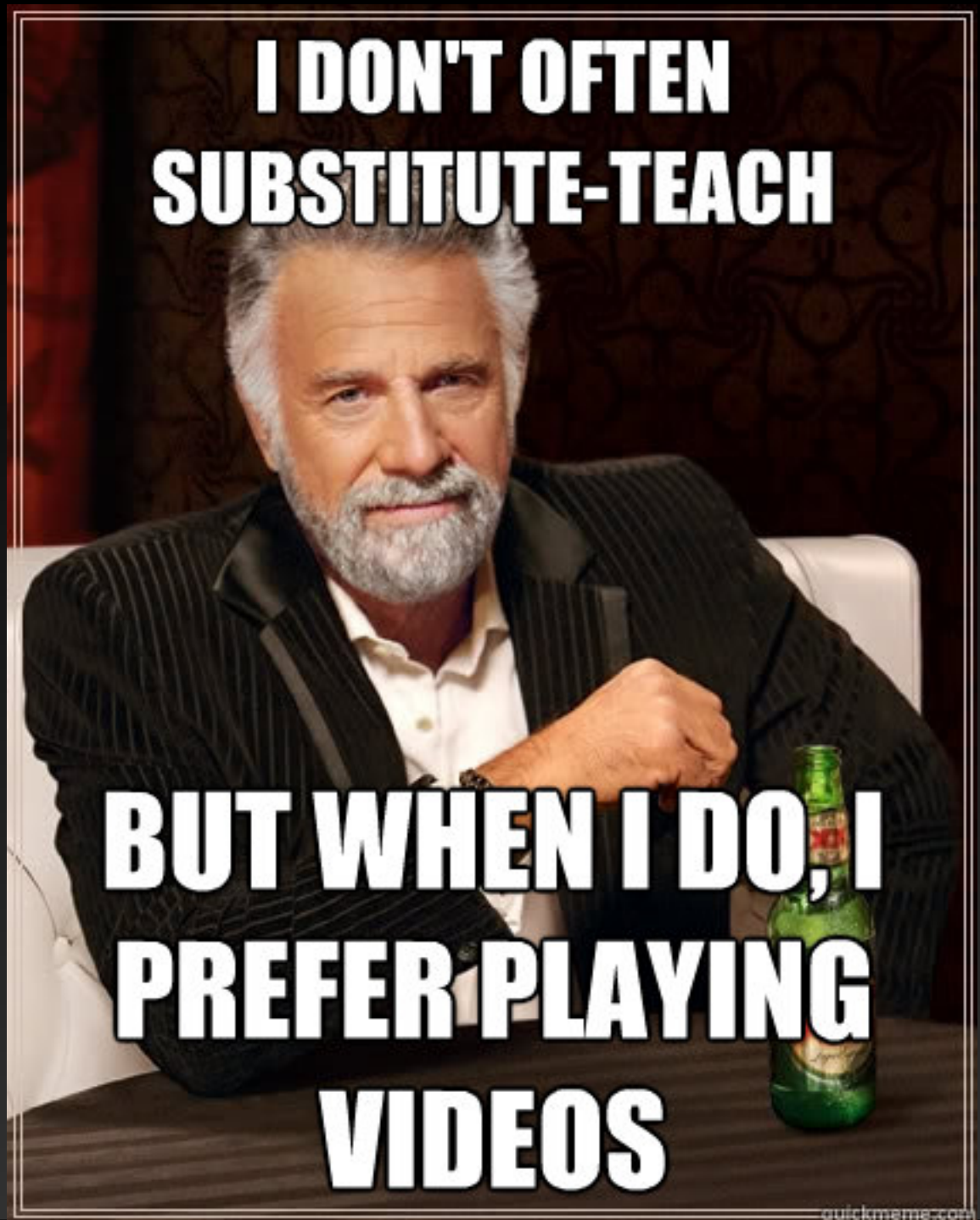
ML in PL?

- Why does ML belong in a PL course?

- Westley Weimer, George C. Necula: ***Mining Temporal Specifications for Error Detection***. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS) 2005: 461-476
- Pieter Hooimeijer, Westley Weimer: ***Modeling bug report quality***. Conference on Automated Software Engineering (ASE) 2007: 34-43
- Westley Weimer, Nina Mishra: ***Privately Finding Specifications***. IEEE Trans. Software Engineering 34(1): 21-32 (2008)
- Nicholas Jalbert, Westley Weimer: ***Automated Duplicate Detection for Bug Tracking Systems***. Conference on Dependable Systems and Networks (DSN) 2008
- Raymond P.L. Buse, Westley Weimer: ***Automatic Documentation Inference for Exceptions***. International Symposium on Software Testing and Analysis (ISSTA) 2008: 273-281
- Raymond P.L. Buse, Westley Weimer: ***A Metric for Software Readability***. International Symposium on Software Testing and Analysis (ISSTA) 2008: 121-130 (best paper award)
- Raymond P.L. Buse, Westley Weimer: ***The Road Not Taken: Estimating Path Execution Frequency Statically***. Submitted to International Conference on Software Engineering (ICSE) 2009 on September 5.
- Elizabeth Soechting, Kinga Dobolyi, Westley Weimer: ***Semantic Regression Testing for Tree-Structured Output***. Submitted to International Conference on Software Engineering (ICSE) 2009 on September 5.
- Claire Le Goues, Westley Weimer: ***Specification Mining With Few False Positives***. Submitted to Tools and Algorithms for the Construction and Analysis of Systems (TACAS) 2009 on October 9.

ML in PL?

- Often in PL we try to form judgments about complex human-related phenomena
- ML can help form the basis of an analysis:
  - e.g., readability, bug reports, path frequency, ...
- or ML can help automate an action:
  - e.g., specification mining, documentation, regression testing ...
- PL is often concerned with scalable analyses, which give rise to huge data sets
  - ML helps us to make sense of them


I DON'T OFTEN
SUBSTITUTE-TEACH
BUT WHEN I DO, I
PREFER PLAYING
VIDEOS

Today's Programming

Sumit Gulwani:

Automating String Processing in Spreadsheets using Input-Output Examples

POPL 2011 (Austin, Texas)

TtKiM

- Is this machine learning?
- How does this approach relate to other AI techniques?
- What are the inputs and outputs for this approach?


1/2

What You'll Learn

- What kinds of problems can & can't it solve?
- What should you know about ML?
  - How to cast a problem in ML terms (e.g., creating a descriptive model)
  - How to pick the right ML algorithm
  - How to evaluate the results
    - Relevant statistics (e.g., precision, recall)
    - Relative feature importance
    - Practical details

No Silver Bullet

- ML can be handy, but using it takes practice
- Researchers often incorrectly apply ML without understanding its principles
  - "They **threw machine learning at it** ..."
- ML rarely gives guarantees about performance
- ML takes creativity
  - Forming the model (e.g., picking features)
  - Interpreting the results

ML Algorithm Types

- Output Types
  - **Numeric**. Examples: How tall will you be, based on your birth weight? How much will you charge to your credit card this month, based on last month?
    - ML example: linear regression
  - **Binary**. Example: Does this image contain a human face or not? Is calling A() after B() a bug or not?
    - ML example: decision tree
  - **Discrete**. Example: Is this office, game or system software? How many sorts of computer intrusions are there, based on attacker behavior?
    - ML example: k-means clustering

ML Algorithm Types

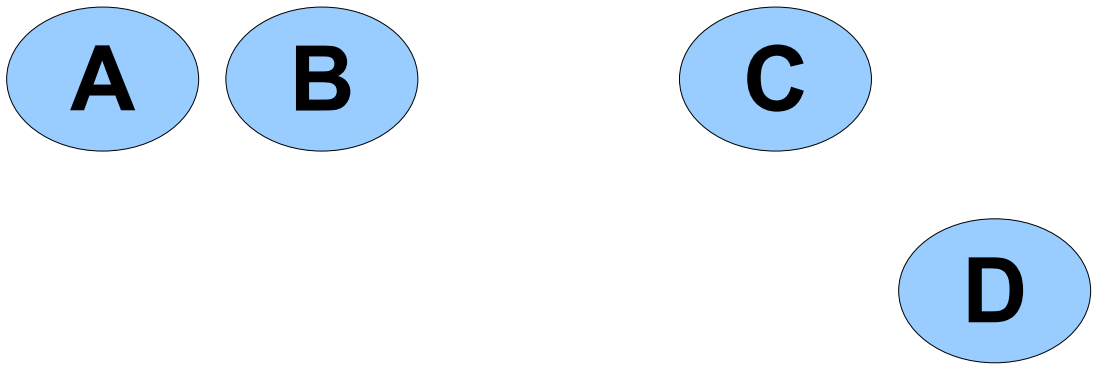
- Input Types
  - **Supervised**. Some provided **training** examples are **labeled** with the **right** answer. Example: here are five images with faces and five without to get you started, now tell me if this next image has a face or not; here are five resolved bug reports and five that were never resolved, now tell me if this next report will get resolved or not.
  - **Unsupervised**. No labeled answers. Example: here are ten network intrusions: how would you organize them? Here's some seismic data: notice anything?

Clustering

- **Clustering** is the classification of objects into different groups
- Clustering partitions a dataset into subsets such that elements of each subset share common traits
  - Most commonly: proximity in some distance metric
- Clustering is an unsupervised learning method
- **Hierarchical clustering** finds successive clusters using previously-established clusters
  - Top-down = divisive. Bottom-up = agglomerative.

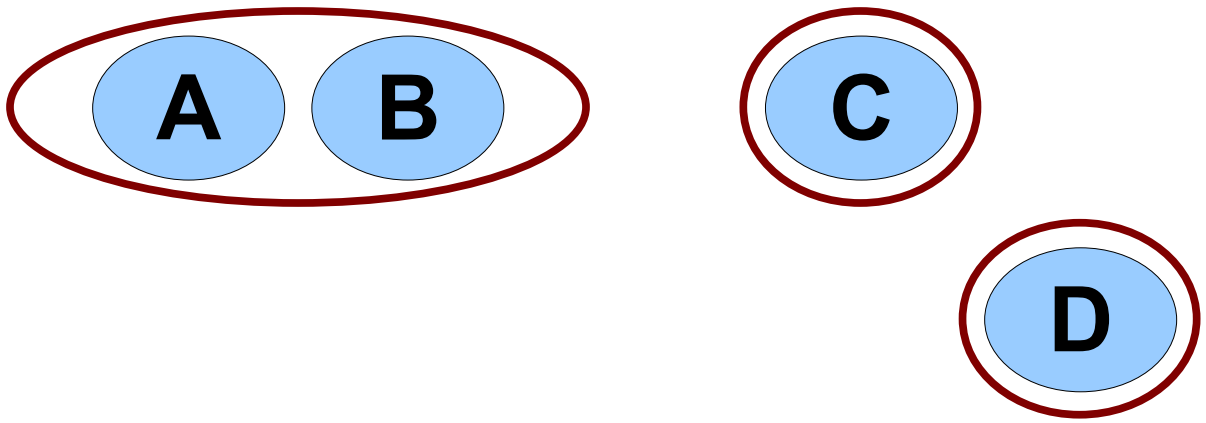
Clustering Example

- Hierarchical agglomerative clustering, Euclidean distance



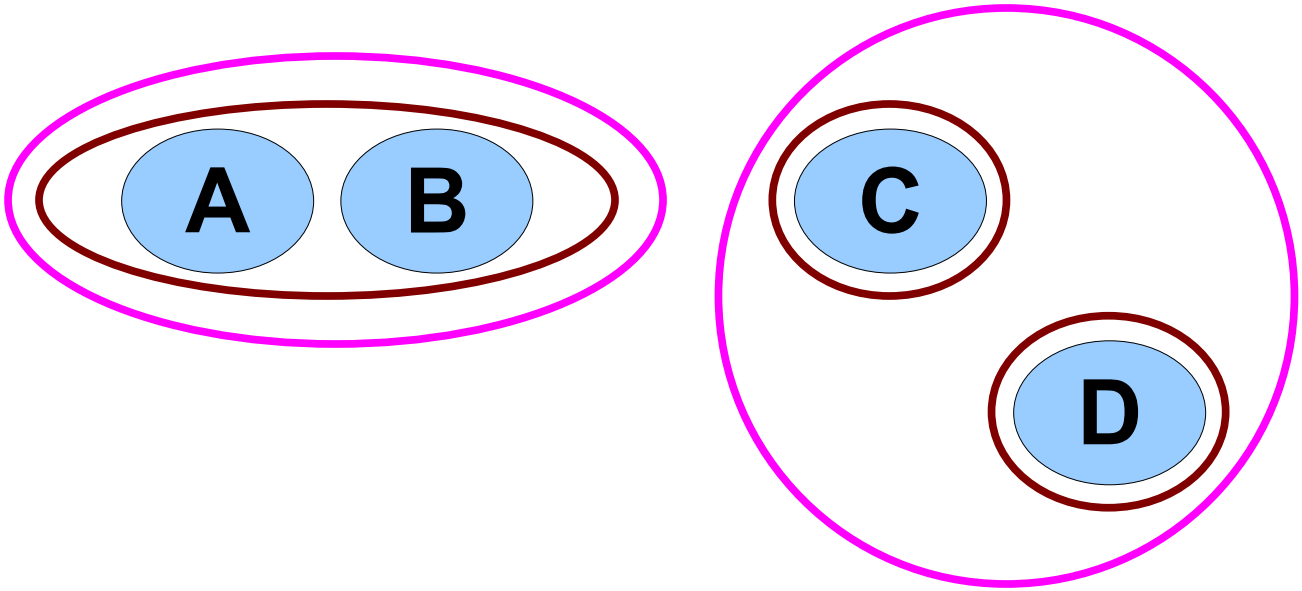
Clustering Example

- Hierarchical agglomerative clustering, Euclidean distance



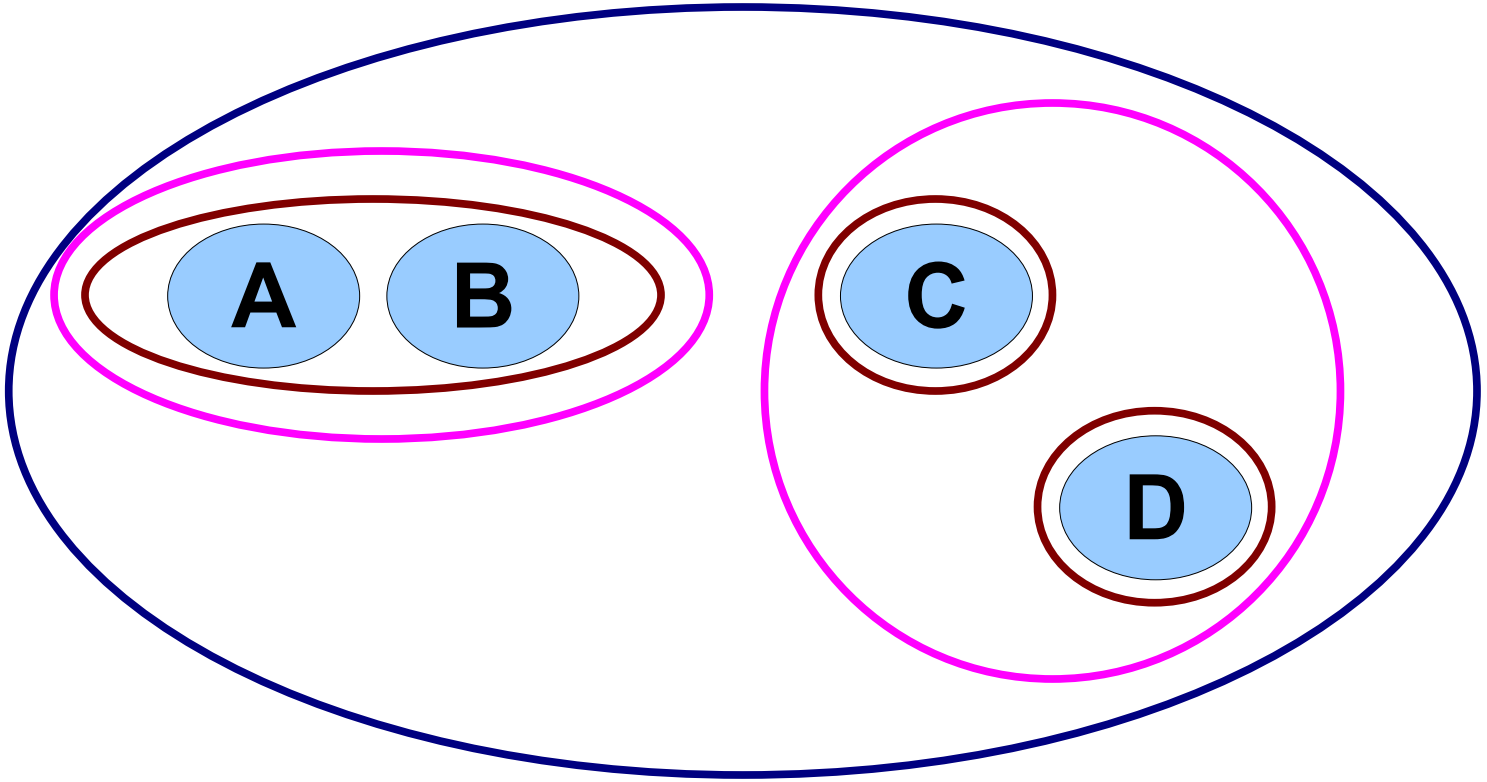
Clustering Example

- Hierarchical agglomerative clustering, Euclidean distance



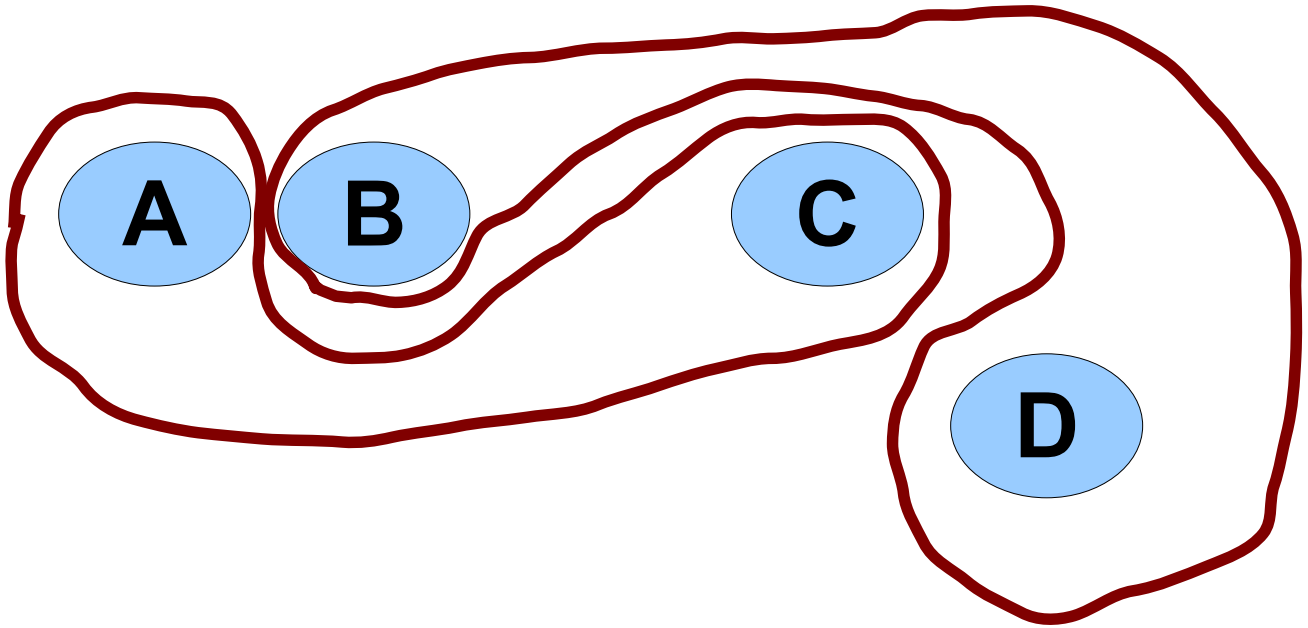
Clustering Example

- Hierarchical agglomerative clustering, Euclidean distance



Clustering Intuition

- Why is {A,C} {B,D} a bad clustering?



K-Means Clustering

- The objects in a cluster should be close to each other
- Given a cluster *C* and its mean point *m*, the badness (i.e., error or **intra-cluster variance**) of the cluster is the sum, over all objects *x* in *C*, of distance(*x*,*m*).
- The objective of the ***k*-means algorithm** is to partition objects into *k* clusters such that the sum of the intra-cluster variances is minimized

K-Means Algorithm

make *k* initial mean points somehow

each one is (will be) the center of a cluster!

assign each object to a cluster randomly

while you're not done

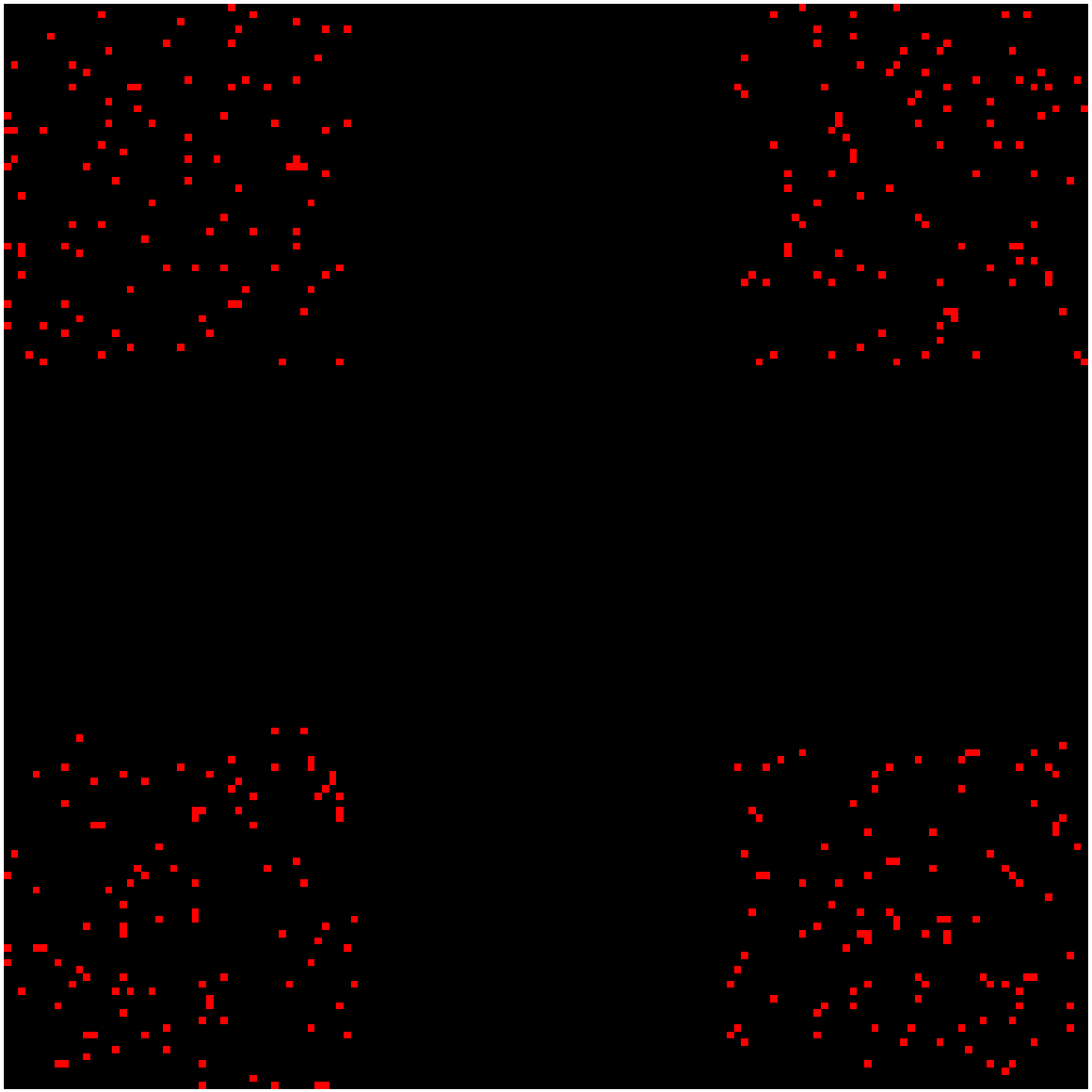
put each object in the cluster it is closest to

(i.e., in the cluster with the mean point it is closest to)

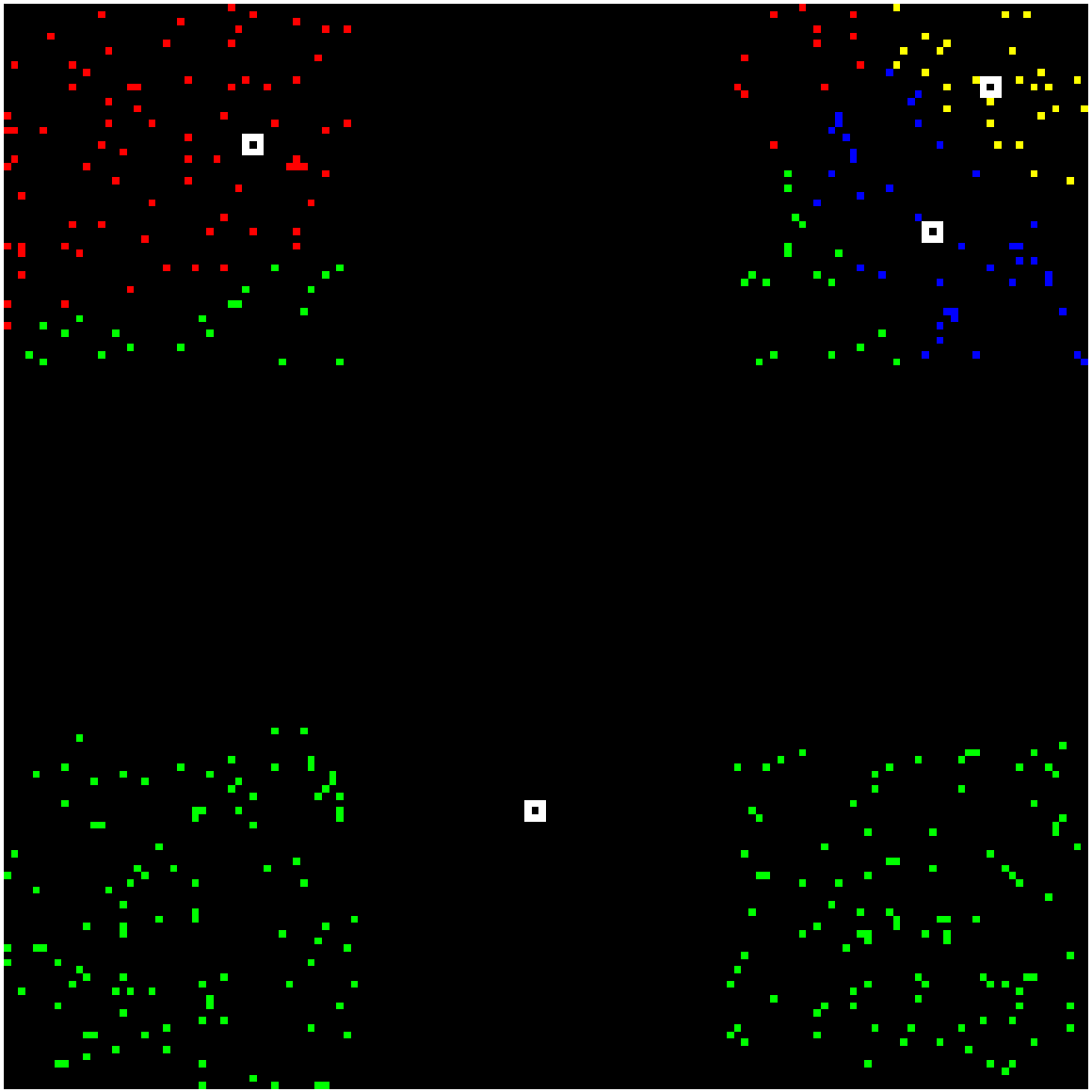
for each cluster, recalculate where the mean point is

(i.e., average all the objects now in the cluster)

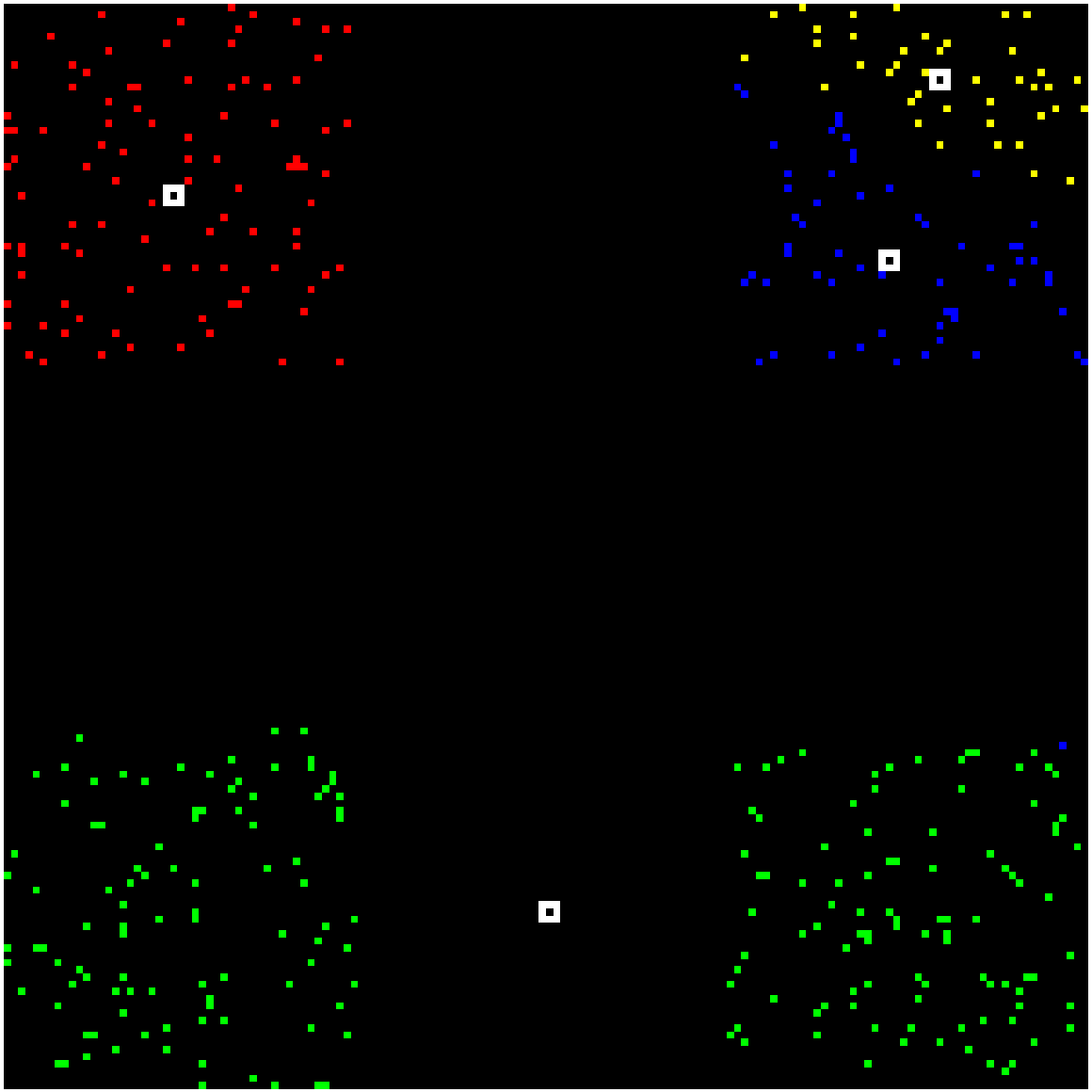
K-Means Example (01/10)



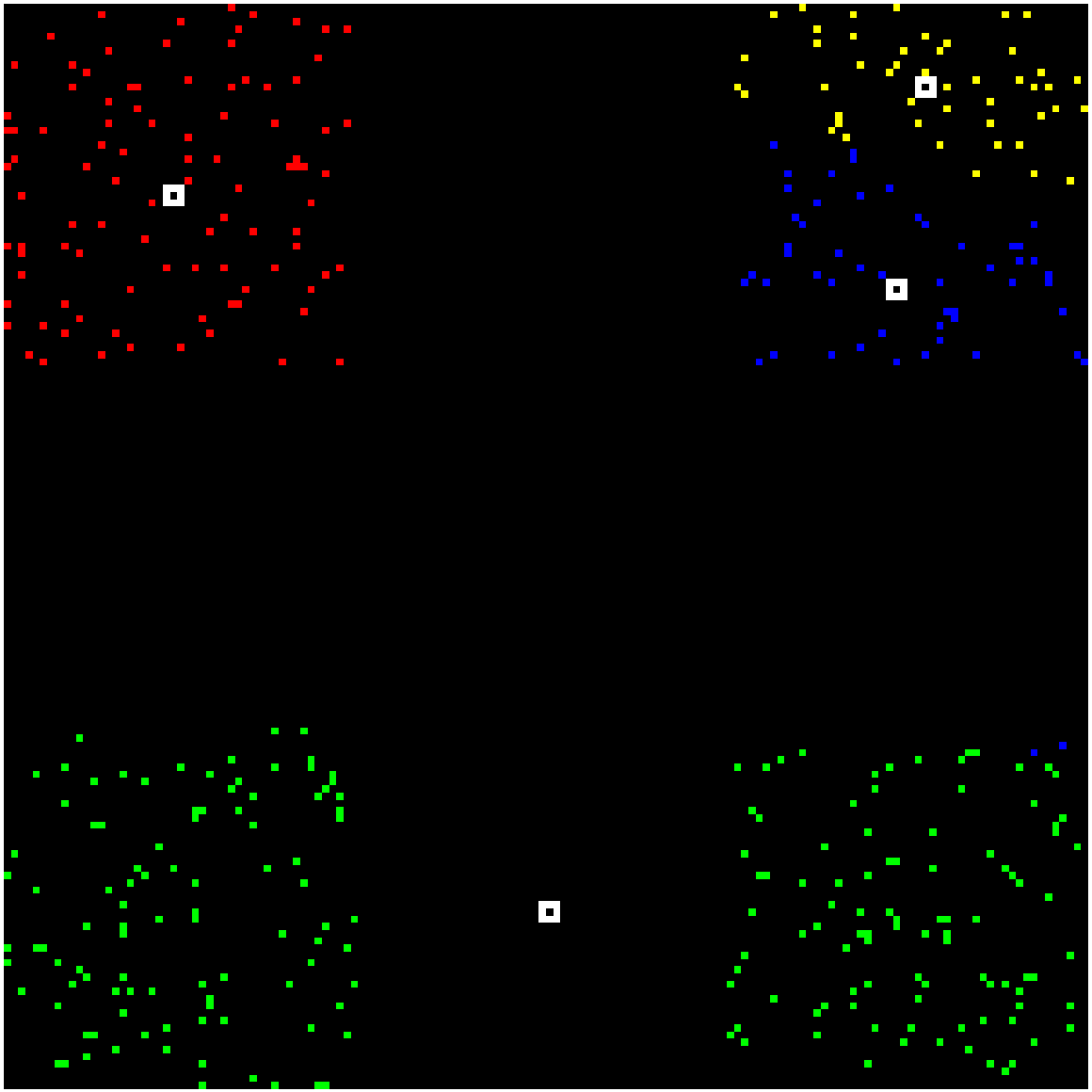
K-Means Example (02/10)



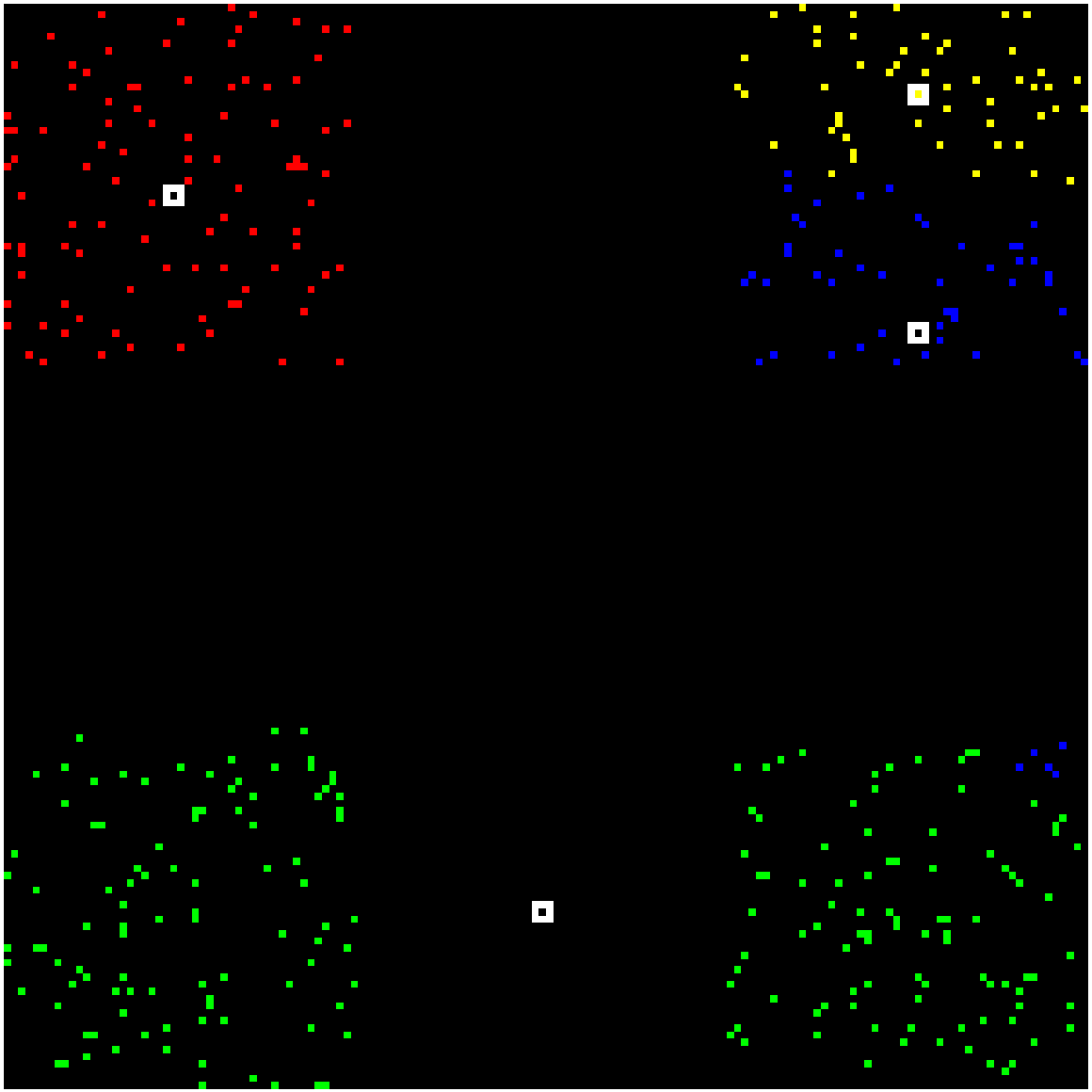
K-Means Example (03/10)



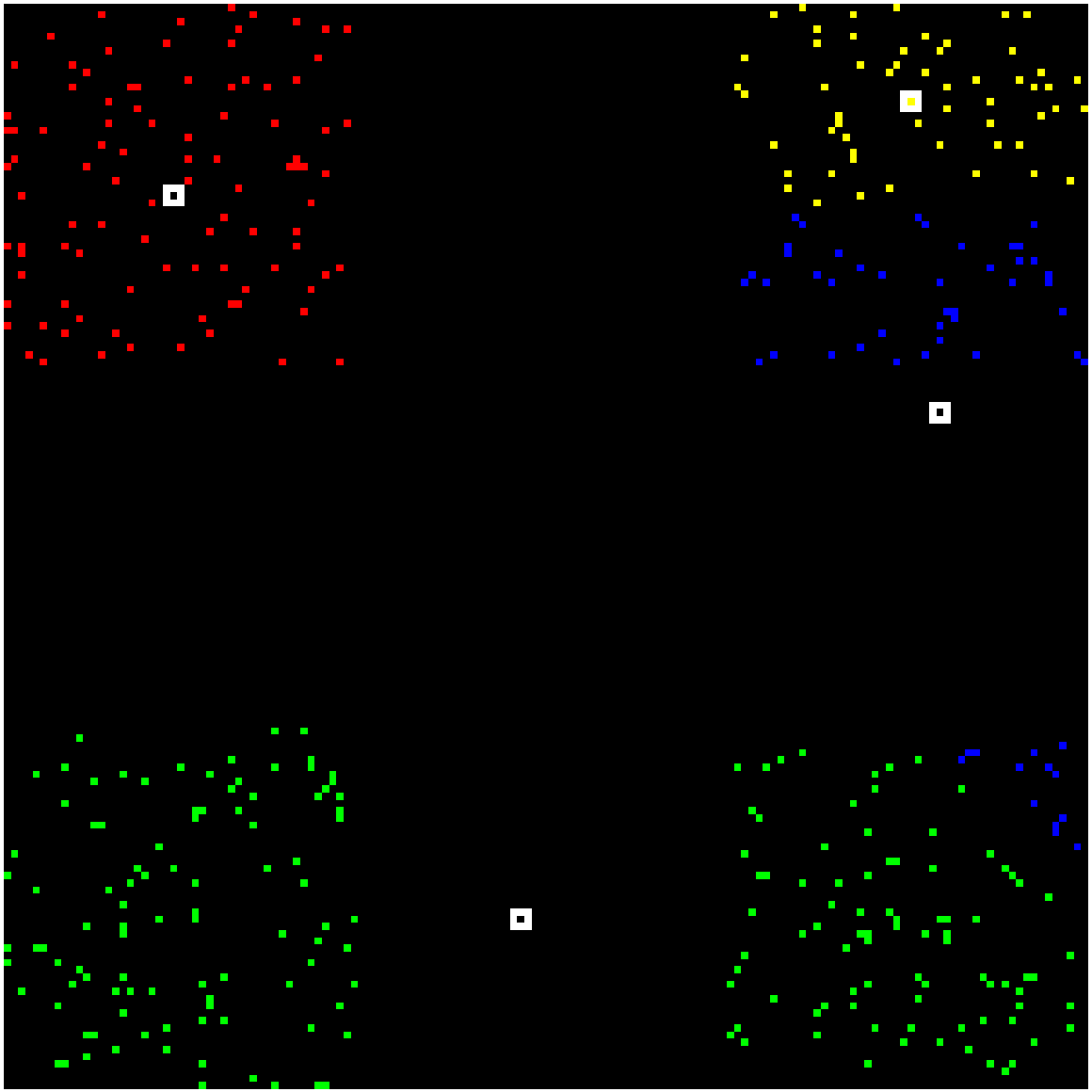
K-Means Example (04/10)



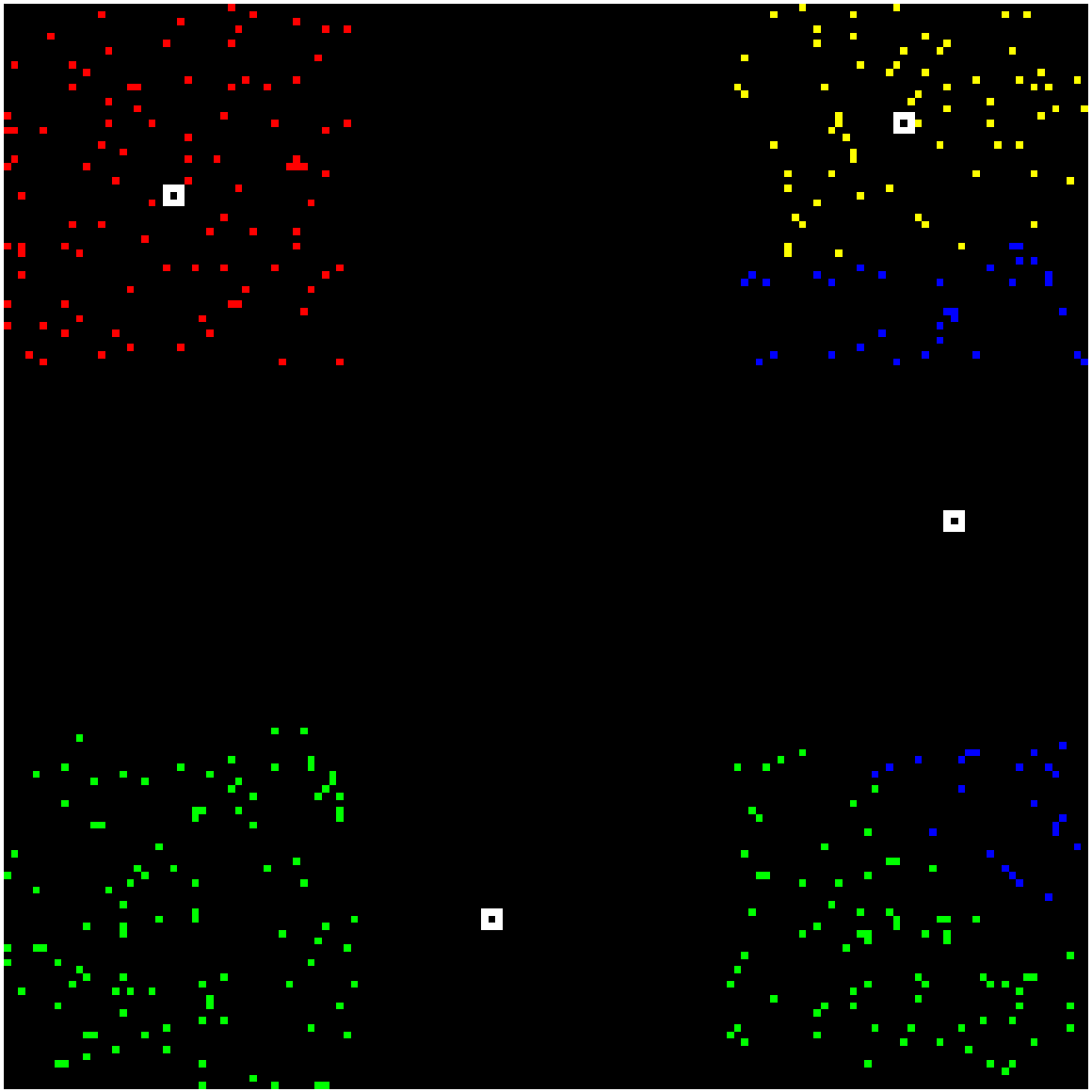
K-Means Example (05/10)



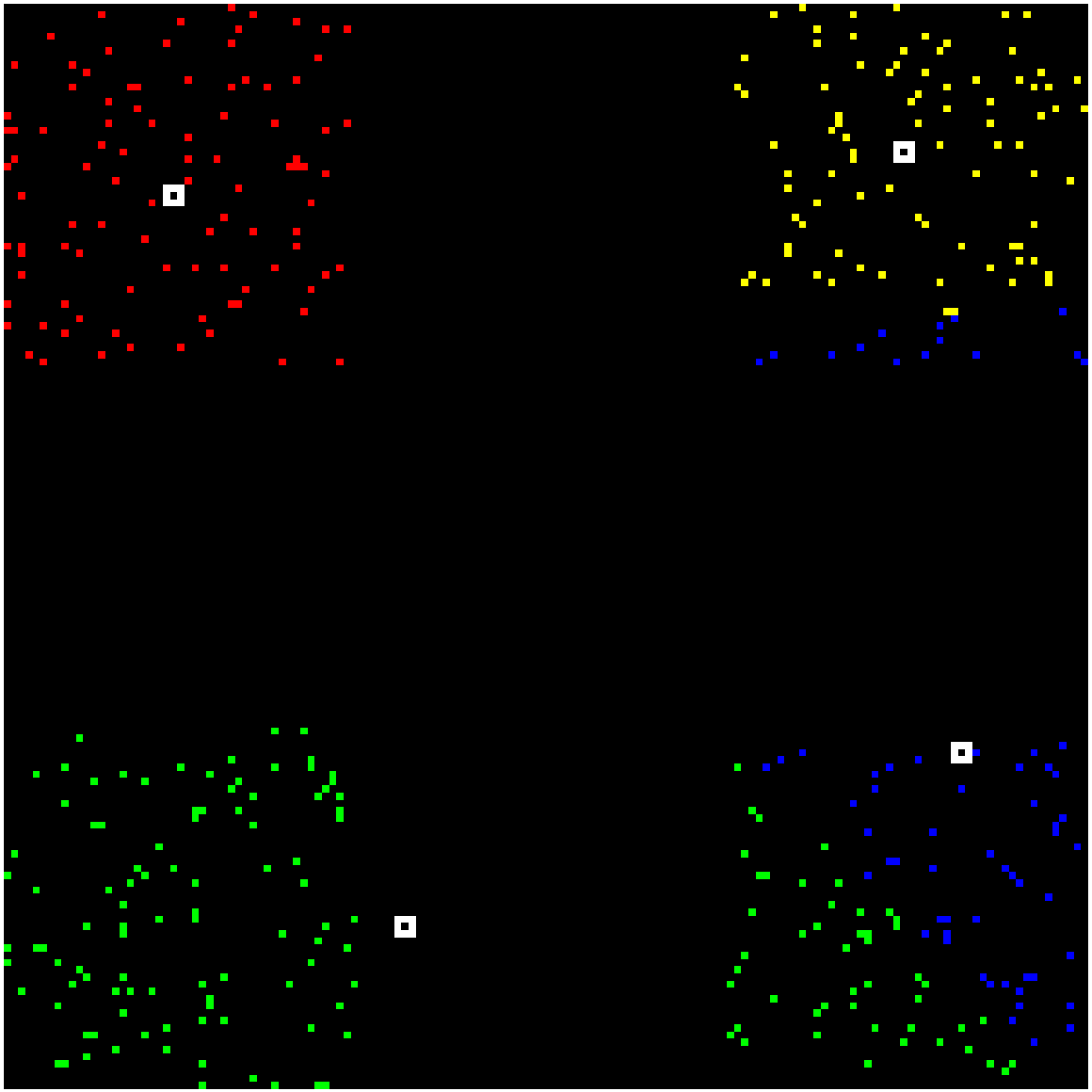
K-Means Example (06/10)



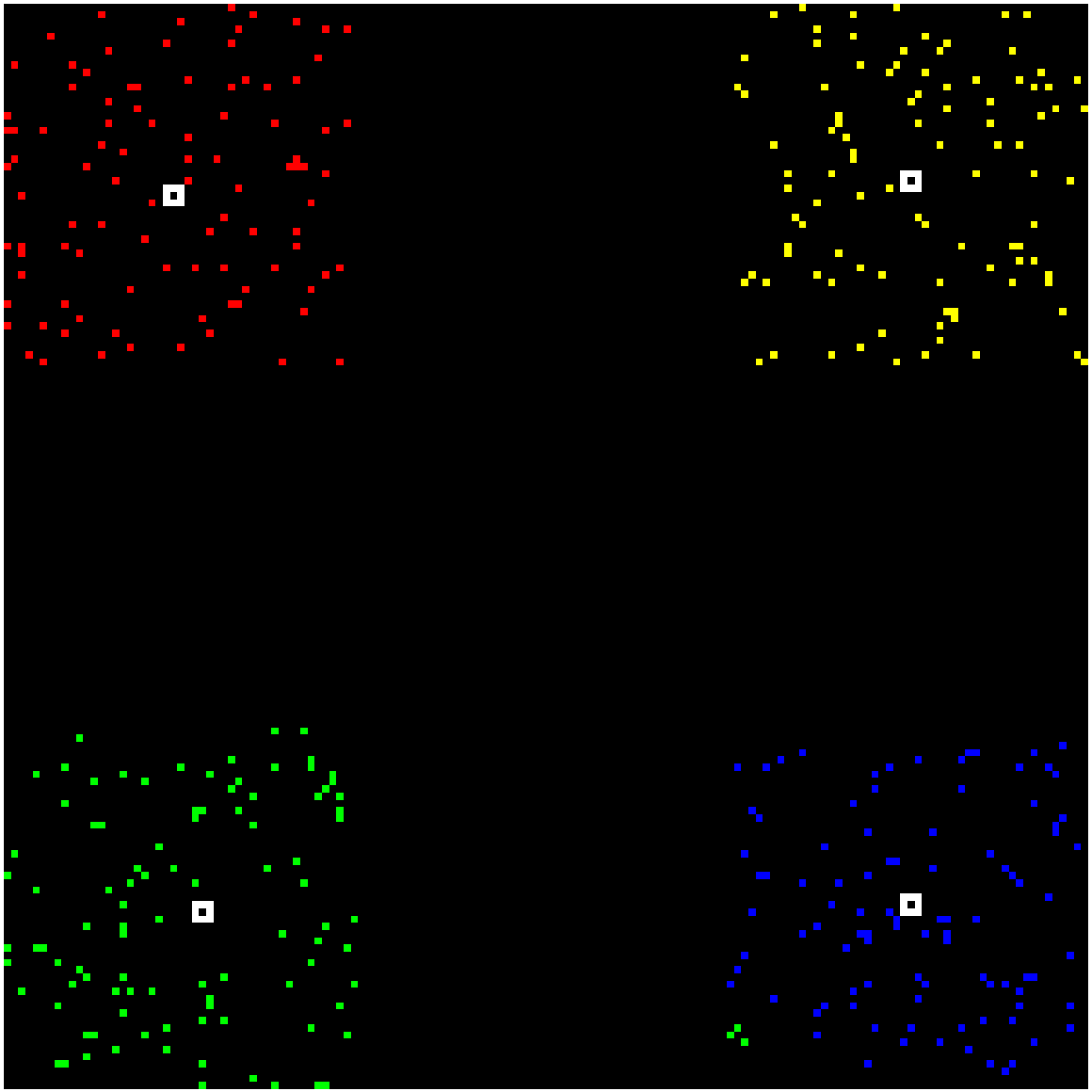
K-Means Example (07/10)



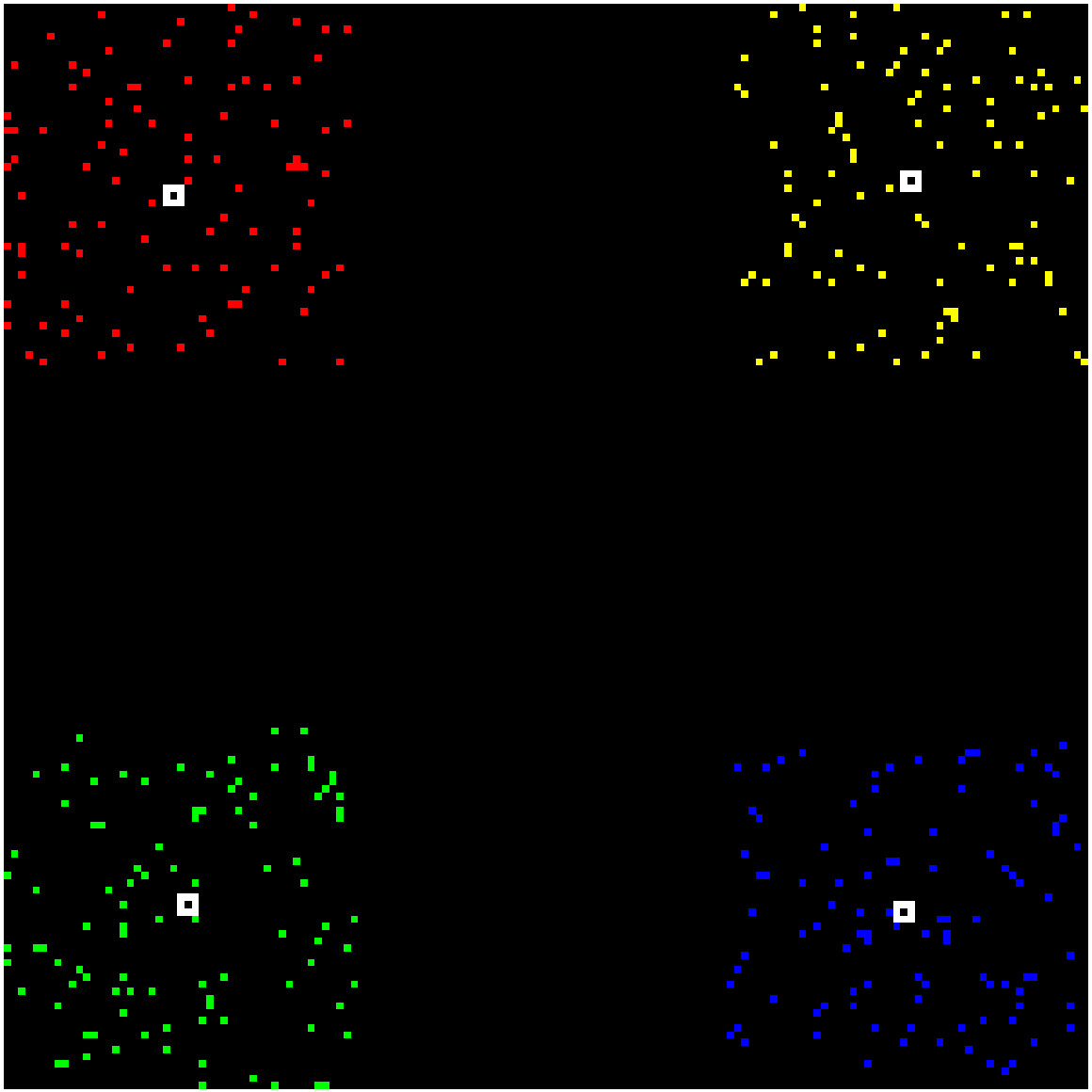
K-Means Example (08/10)



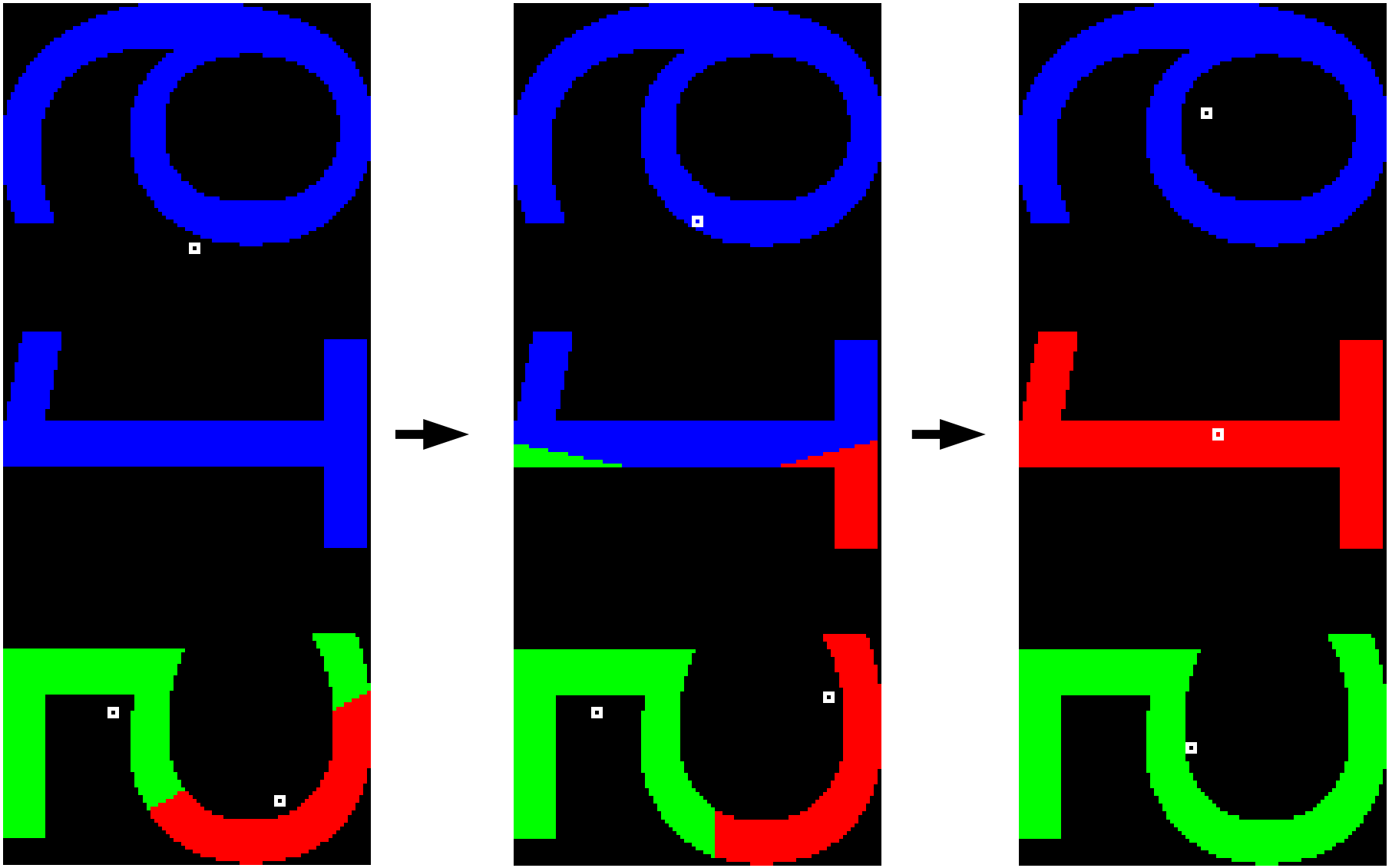
K-Means Example (09/10)



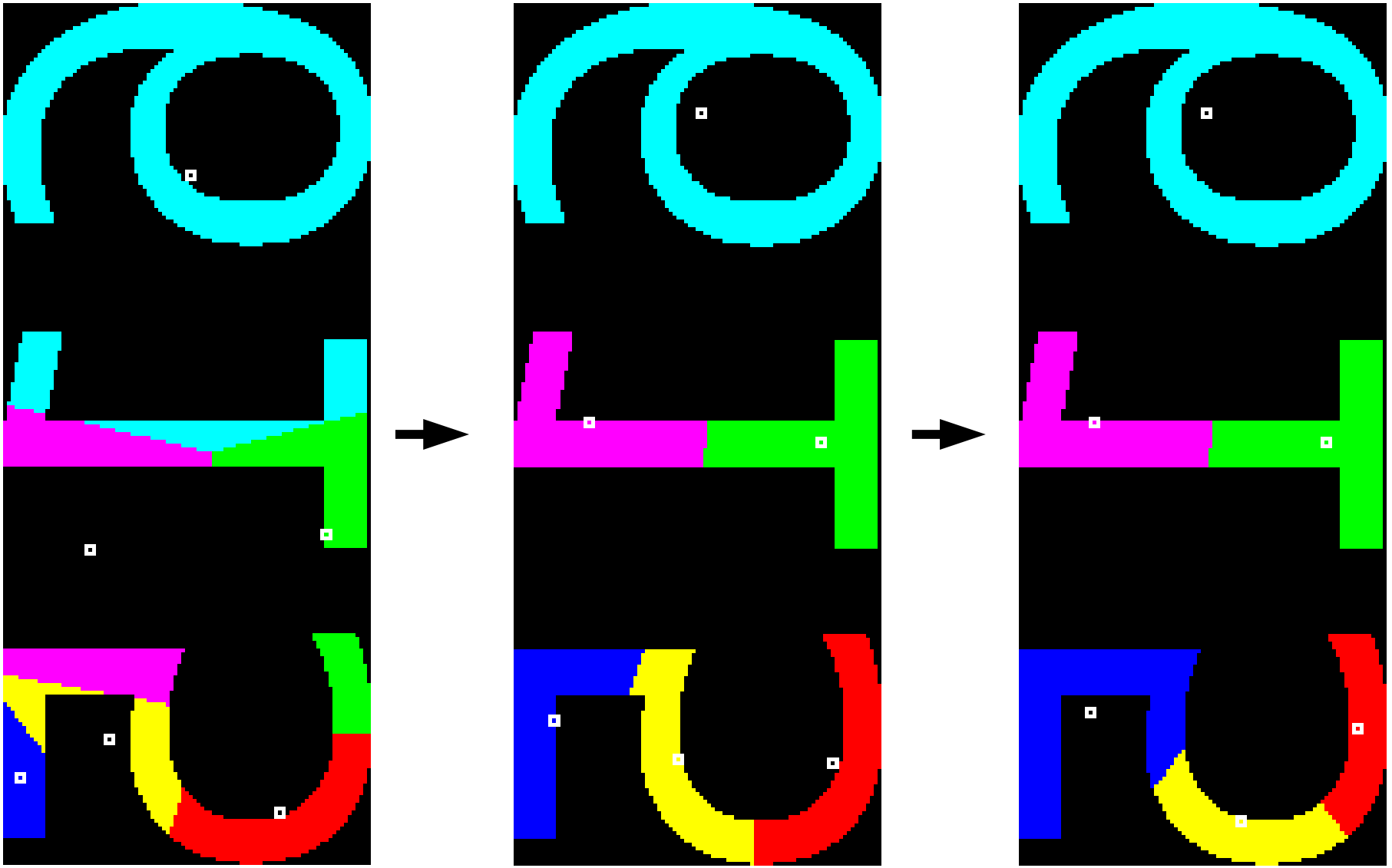
K-Means Example (10/10)



K-Means is Usually Decent



But What If You Don't Know *K*?



Parameter Selection

- Glenn Ammons, Rastislav Bodík, James R. Larus: Mining specifications. POPL 2002: 4-16

set of interactions `Selected` (line *) always has one element, and the recursion never branches. With an appropriate implementation of `OfLeastKinds` (which sorts the interactions), the algorithm runs in that case in time proportional to $n \log n$. In our experience, the time spent running `Standardize` is an insignificant part of the scenario extraction time.

4.2 Automaton learning

This section presents the algorithms and data structures used in learning the specification automaton. The automaton A is an NFA with edges labelled by standardized interactions, whose language includes the most common substrings of the scenario strings extracted from the training traces, plus other strings that the PFSA learner adds as it generalizes. Automaton learning has two steps. First, an off-the-shelf learner learns a PFSA. Then, the corer removes infrequently traversed edges and converts the PFSA into an NFA.

The PFSA learner is an off-the-shelf learner [22] that learns a PFSA that accepts the training strings, plus other strings. The learner is a variation on the classic k-tails algorithm [4]. Briefly, the k-tails algorithm works as follows. First, a retrieval tree is constructed from the input strings. The algorithm then computes all strings of length up to k (*k-strings*) that can be generated from each state in the trie. If two states q_a and q_b generate the same k-strings, they are merged. The process repeats until no more merges are possible. The PFSA learner modifies k-tails by comparing how likely two states are to generate the same k-strings.

The resulting PFSA accepts a *superset* of all the strings in the training scenarios, due to the generalizations performed by the learner. The parameter N that controls the size of the training scenarios is chosen by the user to be large enough to include all of the interesting behavior. It is therefore very likely that the ends of the training scenarios contain *uninteresting* behavior. This is in fact what we see experimentally: the typical PFSA has a "hot" core with a few transitions that occur frequently, with the core surrounded by a "cold" region with many transitions, each of which occurs infrequently. The corer whittles away the "cold" region, leaving just the "hot" core.

The corer can not simply drop edges with low weights. Consider the PFSA in Figure 19 (edge labels are not important and are omitted). Four edges have a weight of 5, which is low compared to the three edges with a weight of 10000. However, any string through this PFSA must traverse the edge out of the start state and the edge into the end state. Despite their low weight, a string is more likely to traverse these edges than it is to traverse the edges with a weight of 10000. Thus, a better measure of an edge's "heat" is its likelihood of being traversed while generating a string from the PFSA. The problem of computing this measure is known as the *Markov*

Linear Regression

- If only we could get something to pick those parameters for us!
  - Let's look at an algo that doesn't need them.
- **Linear regression** models the relationship between a **dependent variable** (what you want to predict) and a number of **independent variables** (**features** you can already measure) as a linear combination:
  - $Dep = c_0 + c_1 \times Indep_1 + \ldots + c_n \times Indep_n$
  - Linear regression finds $c_0 \ldots c_n$ for you

Linear Regression as Machine Learning

- Linear regression is a supervised learning task
  - You provide labeled training data, consisting of the values of the features *and* the dependent variable associated with a number of instances
- The output is a **linear model**
  - A function that, given values for all the features, produces a numeric value for the dependent variable
- How is this model produced?
  - Call SAS, Minitab, Matlab, R, take a Stats course ...

Regression Case Study: Bug Reports

- Software maintenance accounts for over $70 billion each year and is centered around **bug reports**. Unfortunately, 26-36% of bug reports are **invalid** or duplicates and must manually triaged and removed by developers. This takes time and money.
- If we could separate valid from invalid bug reports, we could save time and money.
- Goal: highlight some design decisions when using ML in practice

Regression Case Study: Bug Reports Preliminaries

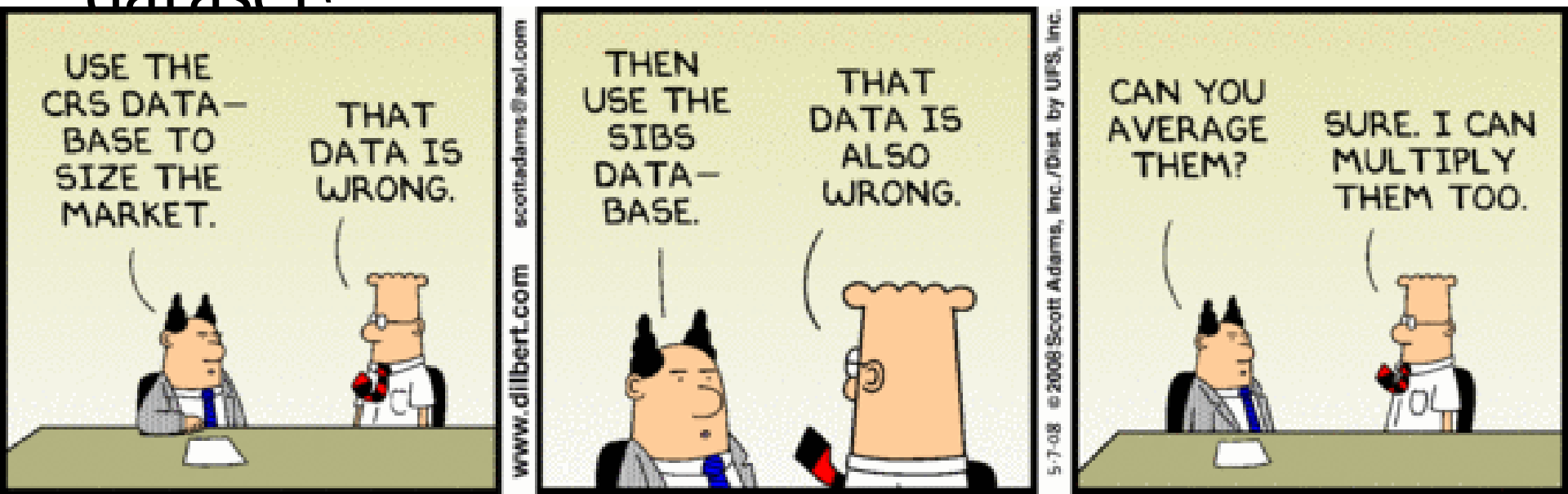
- Dependent Variable: We want to know **how long** (in minutes) it will take a bug report to be resolved.
  - Low quality or invalid reports that take more than 30 days to resolve (say) are an expensive use of developer time. If we could predict this, we'd win!
- Independent Variables:
  - self-reported severity, readability, daily load, submitter reputation, comment count, attachment count, operating system used, ...

Regression Case Study: Bug Reports Instances

- Gather all 27,984 non-empty bug reports between 01/01/2003 and 07/31/2005 (Firefox 1.5).
  - Each report is an **instance** (or **feature vector**)
  - Note the indep features (e.g., priority, readability)
  - Note the dependent feature (minutes to resolved)
- Feed to Linear Regression, get out coeffs
  - Are we done?
  - Let's look at some design decisions in using ML.

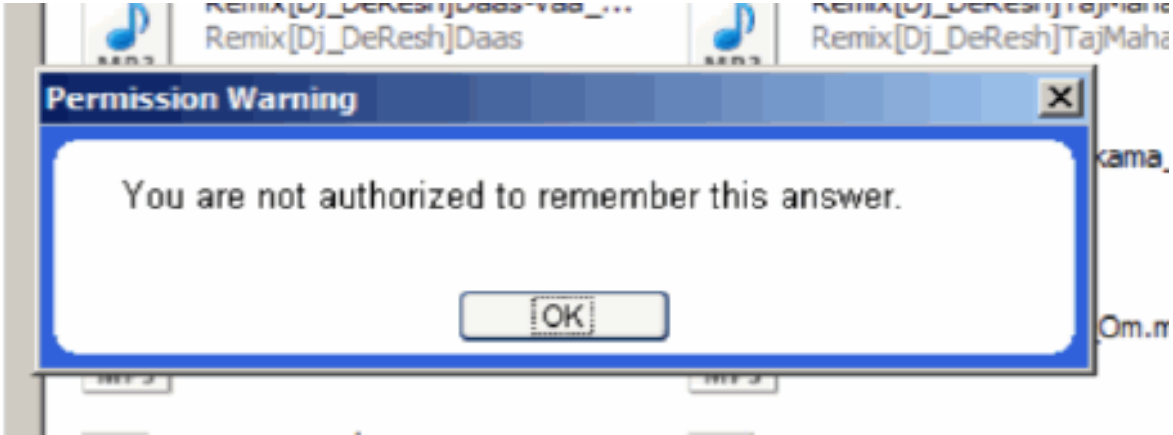
Regression Case Study: Input Dataset Threats

- Can I cherry-pick random bug reports?
- What if I take all reports 1 month after a beta release?
- What is the purpose of having a larger dataset?



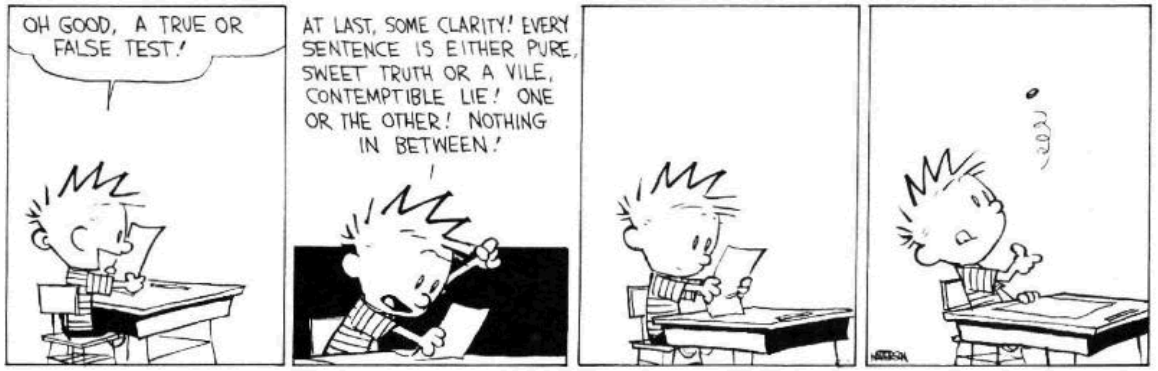
Regression Case Study: Independent Variables

- All features for linear regression are real-valued (see next lecture for discrete features)
  - Comment count is easy enough
  - 1-bit saturating comment count
- How to encode “high/medium/low priority”?
- How to encode “operating system used”?



Regression Case Study: Dependent Variable

- How would these be different:
  - Resolved in X minutes
  - Resolved in X days
  - Resolved within 30 days => 1, otherwise => 0
- Linear Models give continuous output!
  - If you want a binary classifier, may need to pick a cutoff (e.g., model < 0.7 => 0, otherwise => 1)



Regression Case Study: Evaluation

- You have a binary classifier for "will this report be resolved in <= 30 days"
- You have 27,984 reports with known answers
  - C = correct set of reports resolved in 30 days
  - R = set of reports the model returns
- **Precision** = $|C \cap R| \ / \ |R|$
- **Recall** = $|C \cap R| \ / \ |C|$
- F-Measure = $(2 \times Prec \times Rec) \ / \ (Prec + Rec)$

Regression Case Study: Evaluation Baselines

- Say you have 100 instances
- 50 yes instances, 50 no instances, at random
  - "Flip Fair Coin": Prec=0.5, Rec=0.5, F=0.5
  - "Always Guess Yes": Prec=0.5, Rec=1.0, F=0.66
- 70 yes instances, 30 no instances, at random
  - "Flip Fair Coin": Prec=0.7, Rec=0.5, F=0.58
  - "Flip Biased Coin": Prec=0.7, Rec=0.7, F=0.7
  - "Always Guess Yes": Prec=0.7, Rec=1.0, F=0.82
- May want to subsample to 50-50 split for evaluation purposes

Regression Case Study: Threats To Validity

- **Overfitting** occurs when you have learned a model that is too complex with respect to the data.
  - i.e., no actual abstraction has occurred
  - e.g., “memorize all input instances”
- **N-Fold Cross-Validation** can mitigate or detect the threat of overfitting
  - Partition instances into n subsets
  - Train on 2..n and test on 1
  - Train on 1, 3..n and test on 2, etc.

Regression Case Study: Final Results

- Given one day's worth of features, our best F-Measure for predicting “resolved within 30 days” was 0.76, and the industrial practice baseline was 0.73.
- F-Measure assumes false positives and false negatives are equally bad
  - For bug reports, missing a bug report is much worse than triaging an invalid one
- IR metrics are good, but relating your results back to the real world is key:
  - “For the purposes of comparison, however, if *Triage* is $30 and *Miss* is $1000, using our model as a filter saves between five and six percent of the development costs for this data set.”

Next Time

- How to design features!
- Which features mattered?
- More exotic ML algorithms!
- How should we pick parameters?
- Practical information!