

Practical Advice for Building Machine Learning Applications

Lecture 16

Machine Learning
Fall 2015



This lecture

- Making ML work in the world
- Mostly experiential advice
 - Also based on what other people have said
 - See readings on class website

ML and the world

- Diagnostics of your learning algorithm
- Error analysis
- Injecting machine learning into *Your Favorite Task*

ML and the world

- Diagnostics of your learning algorithm
- Error analysis
- Injecting machine learning into *Your Favorite Task*

Debugging machine learning

Suppose you train an SVM or a logistic regression classifier for spam detection

You obviously follow best practices for finding hyper-parameters (such as cross-validation)

Your classifier is only 75% accurate

What can you do to improve it?

Different ways to improve your model

1. More training data

2. Features

1. Use more features
2. Use fewer features
3. Use other features

Tedious!

And prone to errors, dependence on luck

3. Better training

1. Run for more iterations
2. Use a different algorithm
3. Use a different classifier
4. Play with regularization

Let us try to make this process more methodical

But first, diagnostics

Easier to fix a problem if you know where it is

Some possible problems:

1. Over-fitting (high variance)
2. Under-fitting (high bias)
3. Your learning does not converge
4. Your loss function is not good enough (if we want to build a classifier, we should aim for the 0-1 loss)

Detecting over or under fitting

Over-fitting: The training accuracy is much higher than the test accuracy

- The model explains the training set very well, but poor generalization

Under-fitting: Both accuracies are unacceptably low

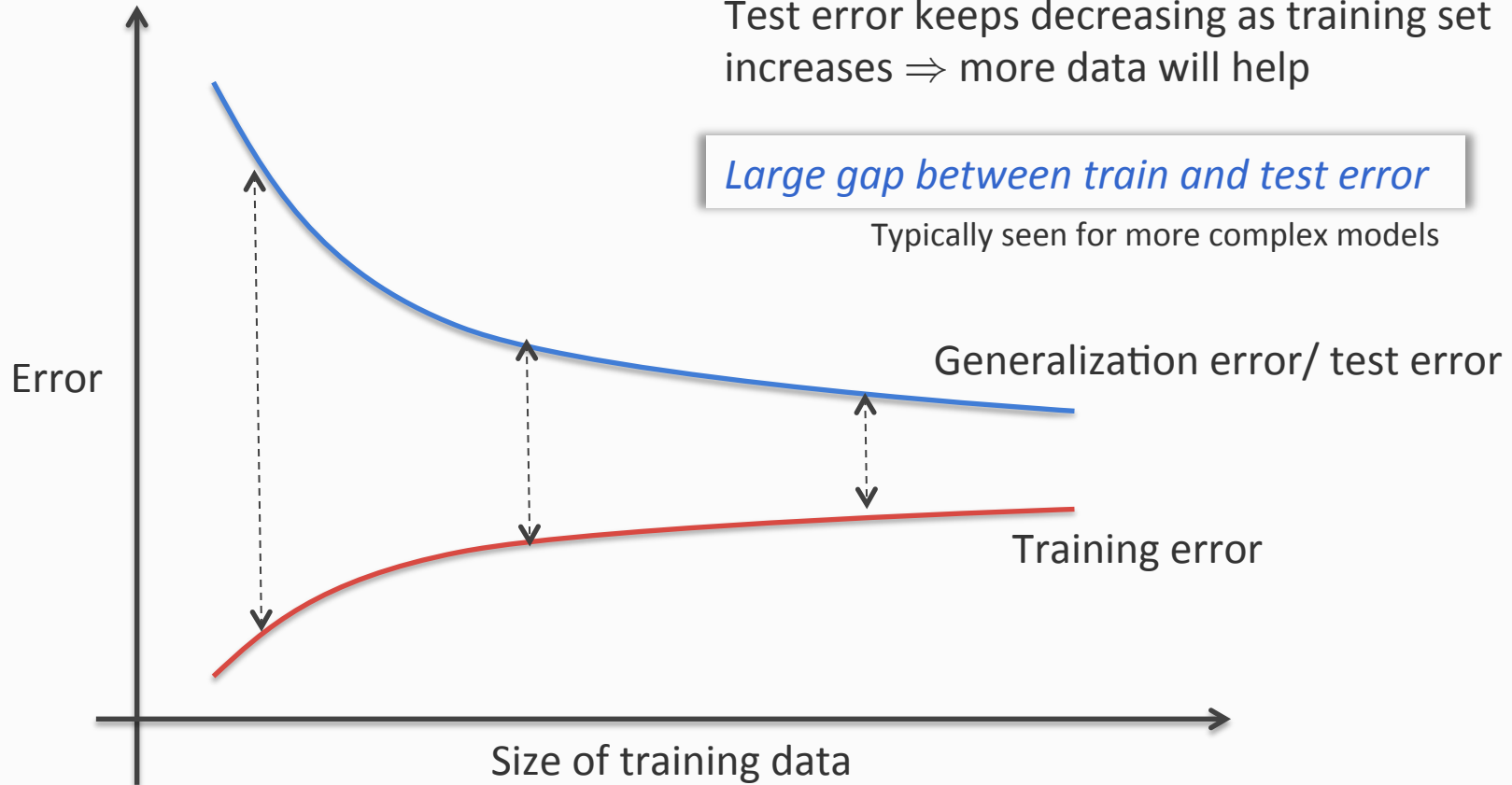
- The model can not represent the concept well enough

Detecting high **variance** using learning curves

Test error keeps decreasing as training set increases \Rightarrow more data will help

Large gap between train and test error

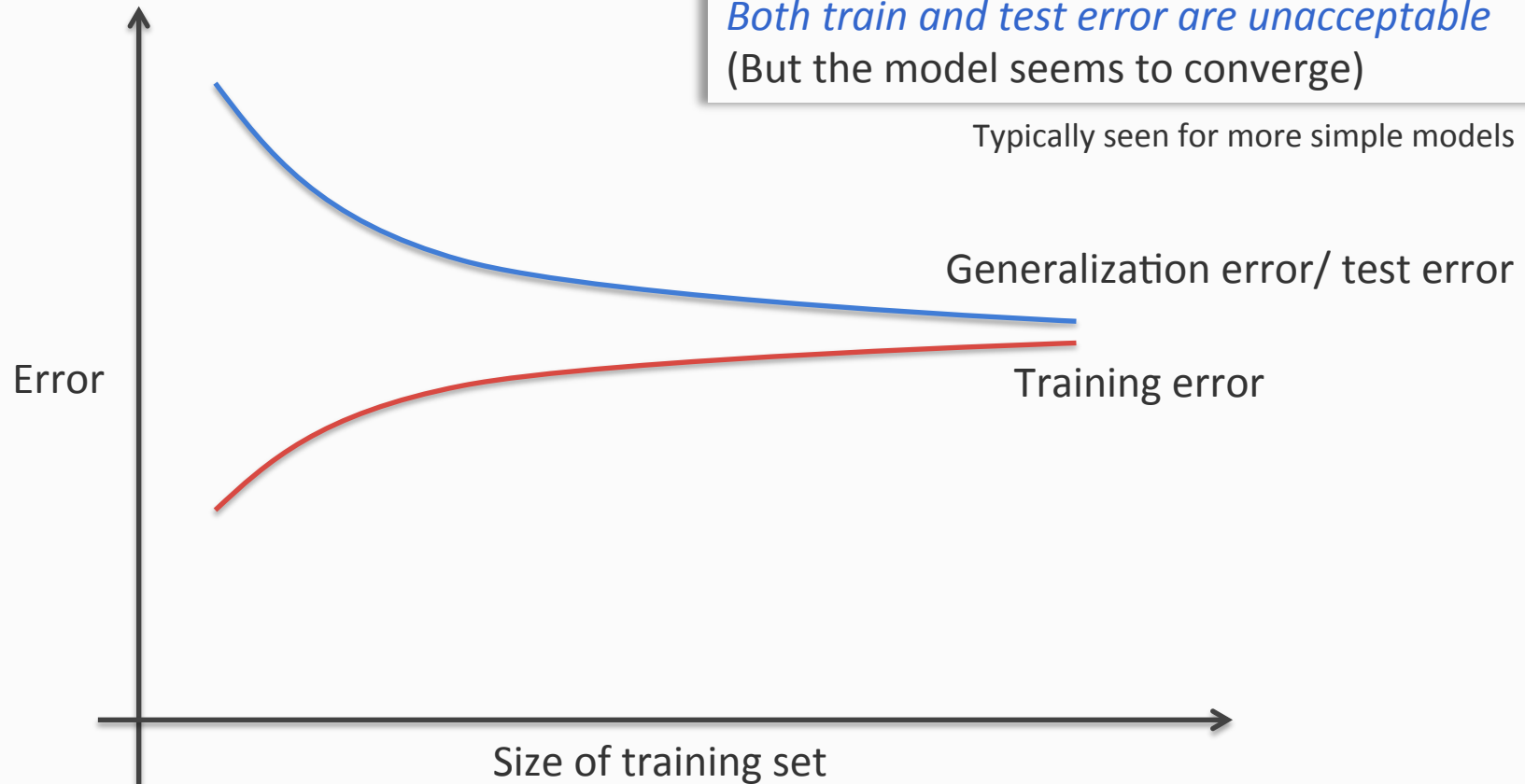
Typically seen for more complex models



Detecting high **bias** using learning curves

Both train and test error are unacceptable
(But the model seems to converge)

Typically seen for more simple models



Different ways to improve your model

1. More training data [Helps with over-fitting](#)
2. Features
 1. Use more features [Helps with under-fitting](#)
 2. Use fewer features [Helps with over-fitting](#)
 3. Use other features [Could help with over-fitting and under-fitting](#)
3. Better training
 1. Run for more iterations
 2. Use a different algorithm
 3. Use a different classifier
 4. Play with regularization [Could help with over-fitting and under-fitting](#)

But first, diagnostics

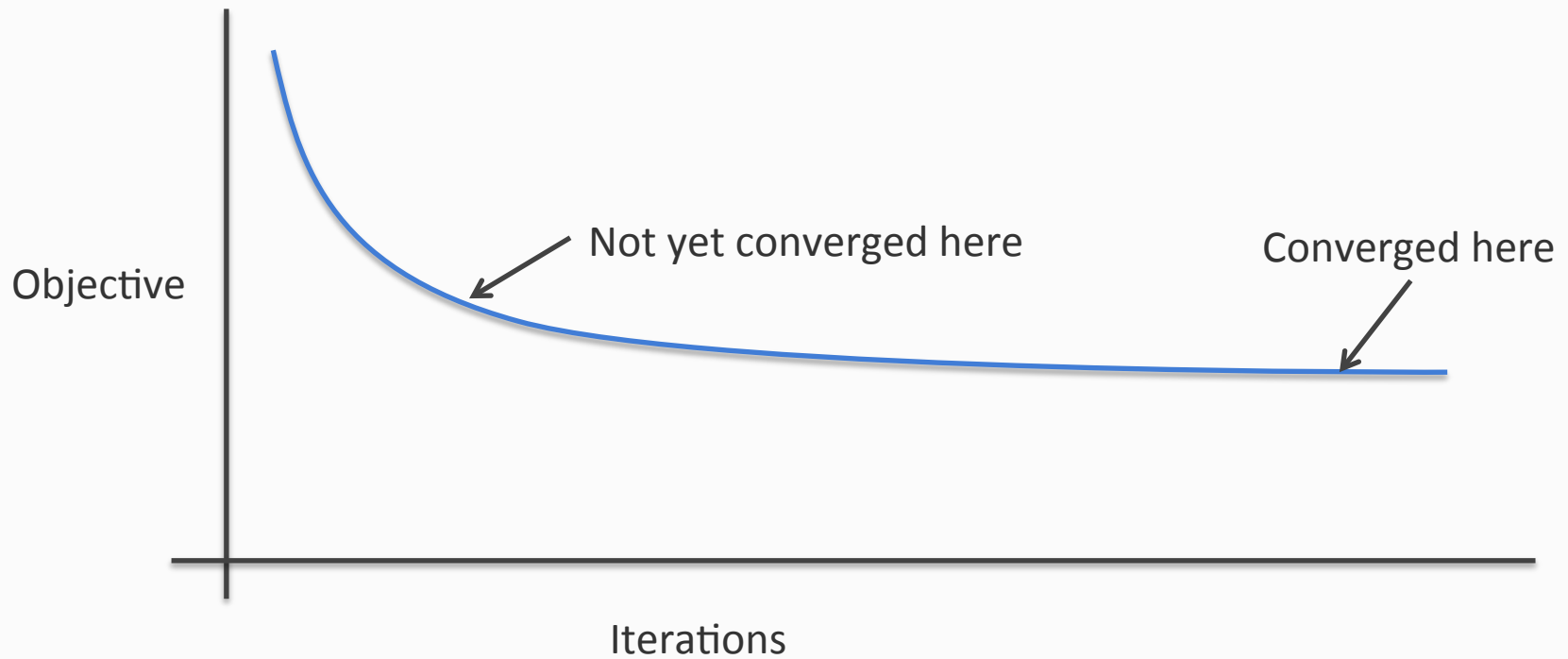
Easier to fix a problem if you know where it is

Some possible problems:

- ✓ Over-fitting (high variance)
 - ✓ Under-fitting (high bias)
3. Your learning does not converge
 4. Your loss function is not good enough (if we want to build a classifier, we should aim for the 0-1 loss)

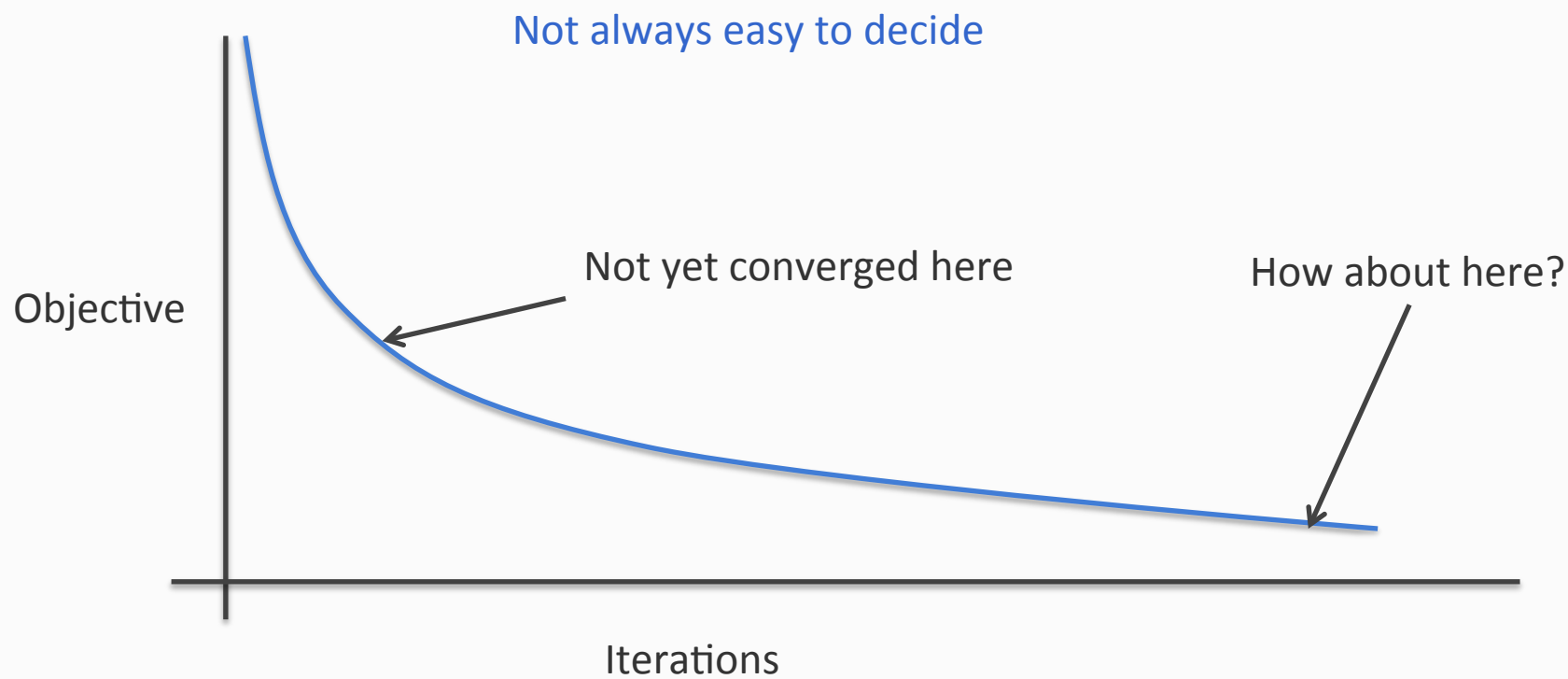
Does your learning algorithm converge?

If learning is framedd as an optimization problem, track the objective



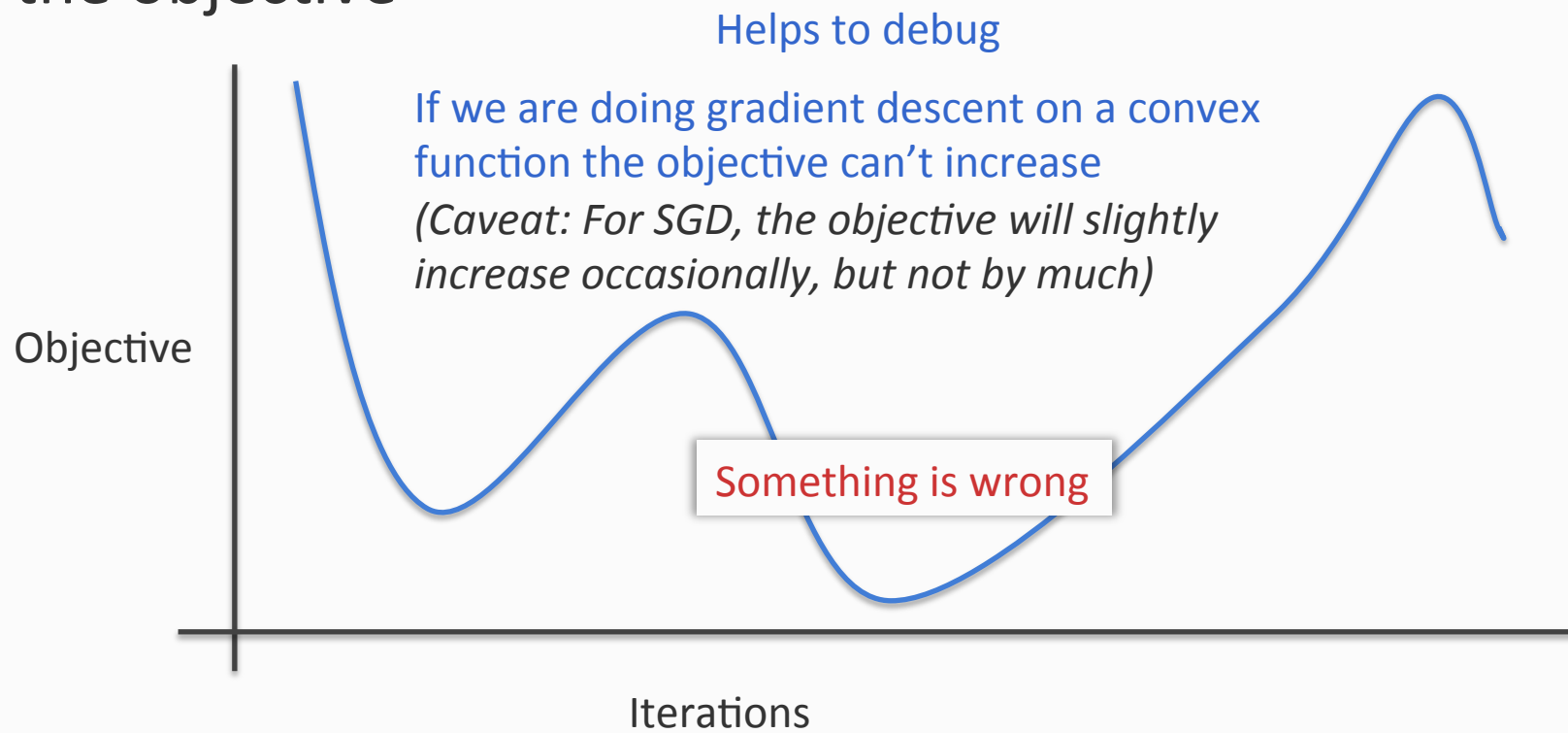
Does your learning algorithm converge?

If learning is framedd as an optimization problem, track the objective



Does your learning algorithm converge?

If learning is framedd as an optimization problem, track the objective



Different ways to improve your model

1. More training data [Helps with overfitting](#)
2. Features
 1. Use more features [Helps with under-fitting](#)
 2. Use fewer features [Helps with over-fitting](#)
 3. Use other features [Could help with over-fitting and under-fitting](#)
3. Better training
 1. Run for more iterations
 2. Use a different algorithm [Track the objective for convergence](#)
 3. Use a different classifier
 4. Play with regularization [Could help with over-fitting and under-fitting](#)

But first, diagnostics

Easier to fix a problem if you know where it is

Some possible problems:

- ✓ Over-fitting (high variance)
- ✓ Under-fitting (high bias)
- ✓ Your learning does not converge
- ✓ Your loss function is not good enough (if we want to build a classifier, we should aim for the 0-1 loss)

What if a different objective is better?

Try out both objectives A and B (eg: SVM and logistic regression)

Run to both convergence

- Remember that lower is better because we are minimizing
- That is, we hope that the lower objective gives better performance

If optimum value of $A >$ optimum value of B

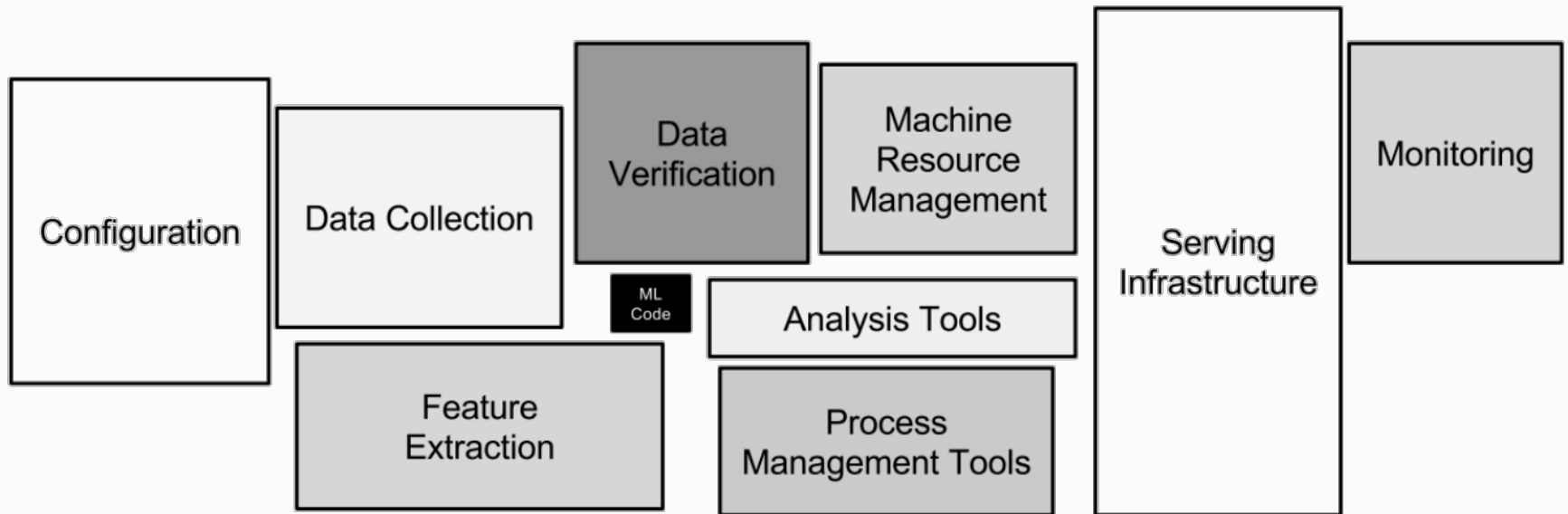
But the generalization error of $A <$ generalization error of B

Then, we know that B does not capture the problem well enough

ML and the world

- Diagnostics of your learning algorithm
- Error analysis
- Injecting machine learning into *Your Favorite Task*

ML in context



Error Analysis

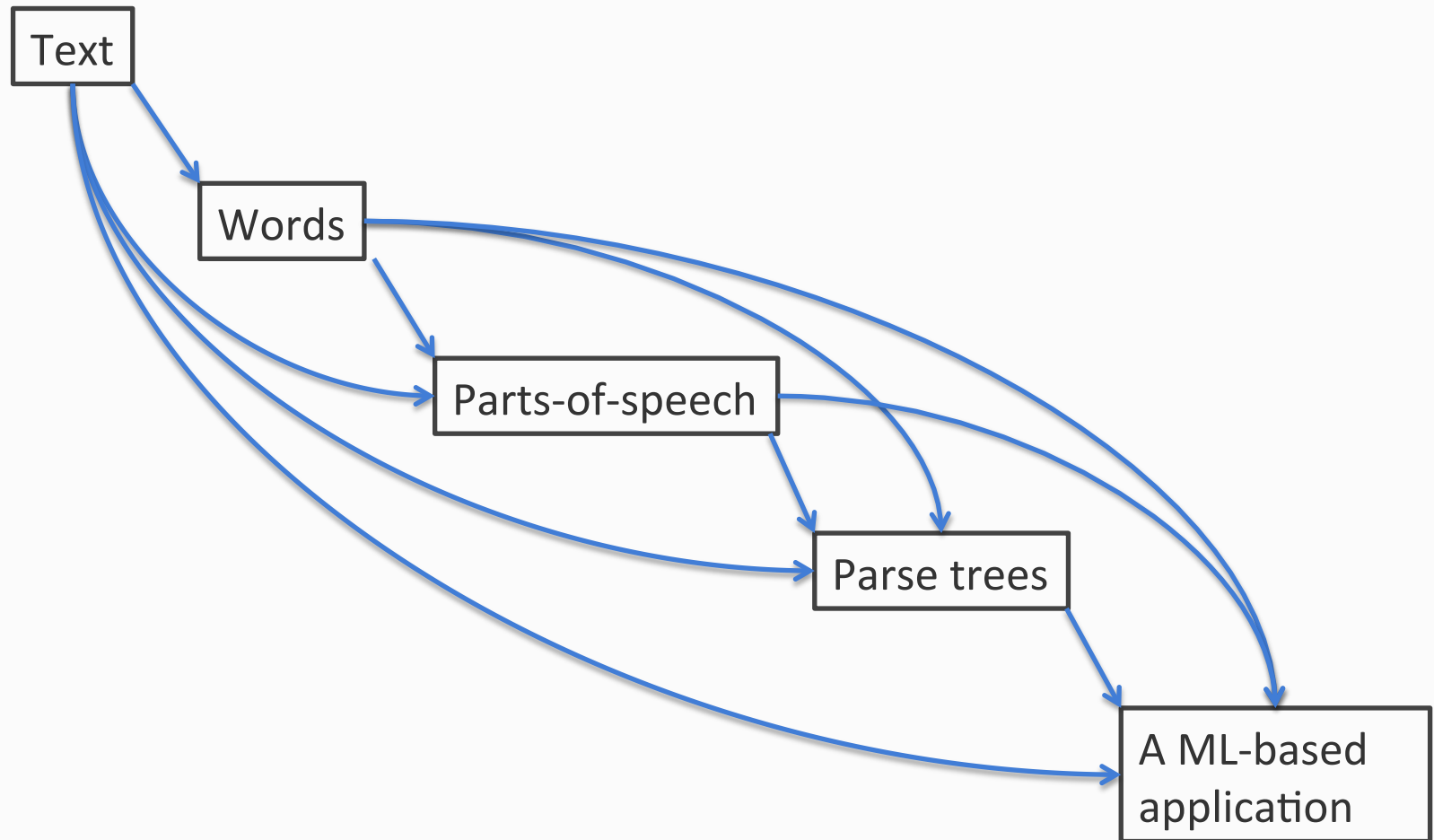
Generally machine learning plays a small role in a larger application

- Pre-processing
- Feature extraction (possibly by other ML based methods)
- Data transformations

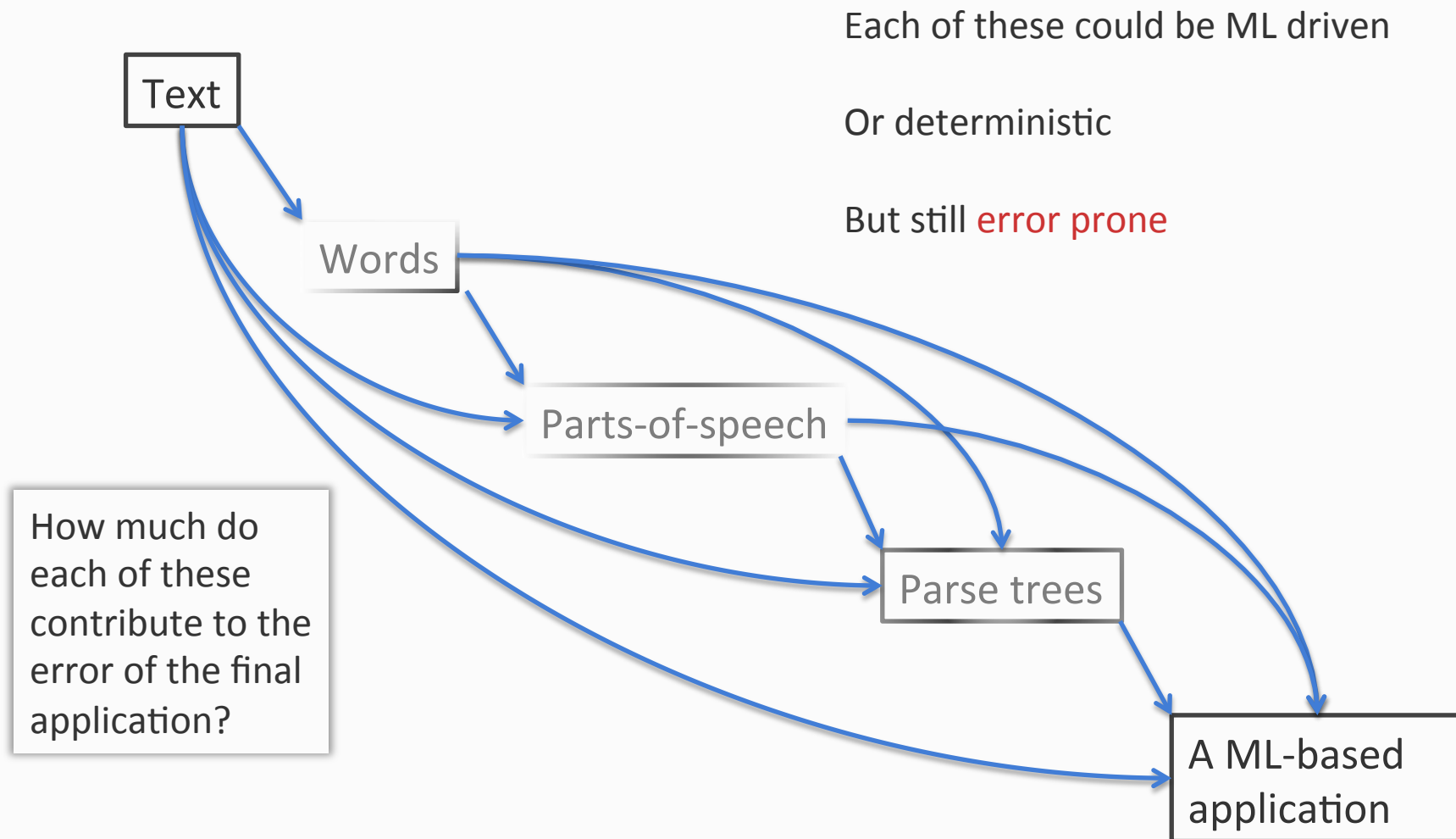
How much do each of these contribute to the error?

Error analysis tries to explain why a system is not performing perfectly

Example: A typical text processing pipeline



Example: A typical text processing pipeline




Tracking errors in a complex system

Plug in the ground truth for the intermediate components and see how much the accuracy of the final system changes

System	Accuracy
End-to-end predicted	55%
With ground truth words	60%
+ ground truth parts-of-speech	84 %
+ ground truth parse trees	89 %
+ ground truth final output	100 %

Error in the part-of-speech component hurts the most



Ablative study

Explaining difference between the performance between a strong model and a much weaker one (a baseline)

Usually seen with features

Suppose we have a collection of features and our system does well, but we don't know which features are giving us the performance

Evaluate simpler systems that progressively use fewer and fewer features to see which features give the highest boost

ML and the world

- Diagnostics of your learning algorithm
- Error analysis
- Injecting machine learning into *Your Favorite Task*

Classifying fish

Say you want to build a classifier that identifies whether a real physical fish is salmon or tuna

How do you go about this?

The slow approach

1. Carefully identify features, get the best data, and the software architecture
2. Implement it and hope it works

Advantage: Perhaps a better approach, maybe even a new learning algorithm. Research.

The hacker's approach

1. First implement something
2. Use diagnostics to iteratively make it better

Advantage: Faster release, will have a solution for your problem quicker

Classifying fish

Say you want to build a classifier that identifies whether a real physical fish is salmon or tuna

How do you go about this?

The slow approach

1. Carefully identify features, get the best data, and the software architecture
2. Implement it and hope it works

Advantage: Perhaps a better approach, maybe even a new learning algorithm. Research.

The hacker's approach

1. First implement something
2. Use diagnostics to iteratively make it better

Advantage: Faster release, will have a solution for your problem quicker

Be wary of premature optimization

Be wary of prematurely committing to a bad path

What to watch out for

- Do you have the right evaluation metric?
 - And does your loss function reflect it?
- Beware of contamination: Ensure that your training data is not contaminated with the test set
 - Learning = generalization to new examples
 - Do not see your test set either. You may inadvertently contaminate the model
 - (Be suspicious of perfect predictors)

What to watch out for

- Be aware of bias vs. variance tradeoff (or over-fitting vs. under-fitting)
- Be aware that intuitions do not work in high dimensions
 - No proof by picture
 - Curse of dimensionality
- A theoretical guarantee may only be theoretical
 - May make invalid assumptions (eg: if the data is separable)
 - May only be legitimate with infinite data (eg: estimating probabilities)
 - Experiments on real data are equally important

Big data is not enough

But more data is always better

- Cleaner data is even better

Remember that learning is impossible without some bias that simplifies the search

- Otherwise, no generalization

Learning requires **knowledge** to guide the learner

- *Machine learning is not a magic wand*

(Also, do not trust anyone who claims to perform magic)

What knowledge?

- Which model is the right one for this task?
 - Linear models, decision trees, kernels, etc
- Which learning algorithm?
- Feature engineering is crucial
- Implicitly, these are all claims about the nature of the problem

Miscellaneous advice

- Learn simpler models first
 - If nothing, at least they form a baseline that you can improve upon
- Ensembles seem to work better
- Think about whether your problem is learnable at all
 - Learning = generalization

ML and system building

Several recent papers about how ML fits in the context of large software systems

Machine Learning: The High-Interest Credit Card of Technical Debt

D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov,
Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young
{dsculley, gholt, dgg, edavydov}@google.com
{toddphillips, ebner, vchaudhary, mwyoung}@google.com
Google, Inc

Hidden Technical Debt in Machine Learning Systems

D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips
{dsculley, gholt, dgg, edavydov, toddphillips}@google.com
Google, Inc.

Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, Dan Dennison
{ebner, vchaudhary, mwyoung, jfcrespo, dennison}@google.com
Google, Inc.

Making machine learning matter

Machine Learning that Matters

Kiri L. Wagstaff

KIRI.L.WAGSTAFF@JPL.NASA.GOV

Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91109 USA

Challenges to the greater ML community

1. A law passed or legal decision made that relies on the result of an ML analysis
2. \$100M saved through improved decision making provided by an ML system
3. A conflict between nations averted through high quality translation provided by an ML system
4. A 50% reduction in cybersecurity break-ins through ML defenses
5. A human life saved through a diagnosis or intervention recommended by an ML system
6. Improvement of 10% in one country's Human Development Index attributable to an ML system

A retrospective look at the course

Learning as generalization

“A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E .”

Tom Mitchell (1999)



We saw different “models”

Or: what kind of a function should a learner learn

- Linear classifiers
- Decision trees
- Non-linear classifiers, kernels
- Ensembles of classifiers

Different learning protocols

- Supervised learning
 - A *teacher* supplies a collection of examples with labels
 - The *learner* has to learn to label new examples using this data
- Unsupervised learning
 - No *teacher*, *learner* has only unlabeled examples
 - Data mining
- Semi-supervised learning
 - *Learner* has access to both labeled and unlabeled examples

Learning algorithms

- **Online algorithms:** Learner can access only one labeled at a time
 - Perceptron, Winnow
- **Batch algorithms:** Learner can access to the entire dataset
 - Naïve Bayes
 - Support vector machines, logistic regression
 - Decision trees and nearest neighbors
 - Boosting
- **Unsupervised/semi-supervised algorithms**
 - Expectation maximization
 - K-Means

Representing data

What is the best way to represent data for a particular task?

- Features
- **Dimensionality reduction** (we didn't cover this, but do look at the material if you are interested)

The theory of machine learning

Mathematically defining learning

- Online learning
- Probably Approximately Correct (PAC) Learning
- Bayesian learning

Representation, optimization, evaluation

The three components of learning algorithms.

Representation	Evaluation	Optimization
Instances	Accuracy/Error rate	Combinatorial optimization
<i>K</i> -nearest neighbor	Precision and recall	Greedy search
Support vector machines	Squared error	Beam search
Hyperplanes	Likelihood	Branch-and-bound
Naive Bayes	Posterior probability	Continuous optimization
Logistic regression	Information gain	Unconstrained
Decision trees	K-L divergence	Gradient descent
Sets of rules	Cost/Utility	Conjugate gradient
Propositional rules	Margin	Quasi-Newton methods
Logic programs		Constrained
Neural networks		Linear programming
Graphical models		Quadratic programming
Bayesian networks		
Conditional random fields		

What we did not see

Machine learning is a large and growing area of scientific study

We did not cover

- Hidden Markov models
- Multiclass support vector machines
- Topic models
- Structured models
- Multi-layer neural networks
- Deep learning
-

But we saw the foundations of *how to think about machine learning*

This course

Focus on the **underlying concepts** and **algorithmic ideas** in the field of machine learning

Not about

- Using a specific machine learning tool
- Any single learning paradigm

What we saw

1. A broad theoretical and practical understanding of machine learning paradigms and algorithms
2. Ability to implement learning algorithms
3. Identify where machine learning can be applied and make the most appropriate decisions (about algorithms, models, supervision, etc)