Advice for applying Machine Learning

Andrew Ng

Stanford University

Today's Lecture

- Advice on how getting learning algorithms to different applications.
- Most of today's material is not very mathematical. But it's also some of the hardest material in this class to understand.
- Some of what I'll say today is debatable.
- Some of what I'll say is not good advice for doing novel machine learning research.
- Key ideas:
  1. Diagnostics for debugging learning algorithms.
  2. Error analyses and ablative analysis.
  3. How to get started on a machine learning problem.
     - Premature (statistical) optimization.

Debugging Learning Algorithms

Debugging learning algorithms

Motivating example:

- Anti-spam. You carefully choose a small set of 100 words to use as features. (Instead of using all 50000+ words in English.)

- Bayesian logistic regression, implemented with gradient descent, gets 20% test error, which is unacceptably high.

$$\max_{\theta} \sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)}, \theta) - \lambda||\theta||^2$$

- What to do next?

Fixing the learning algorithm

- Bayesian logistic regression:

$$\max_{\theta} \sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)}, \theta) - \lambda||\theta||^2$$

- Common approach: Try improving the algorithm in different ways.
  - Try getting more training examples.
  - Try a smaller set of features.
  - Try a larger set of features.
  - Try changing the features: Email header vs. email body features.
  - Run gradient descent for more iterations.
  - Try Newton's method.
  - Use a different value for λ.
  - Try using an SVM.

- This approach might work, but it's very time-consuming, and largely a matter of luck whether you end up fixing what the problem really is.

Diagnostic for bias vs. variance

Better approach:

- Run diagnostics to figure out what the problem is.
- Fix whatever the problem is.

Bayesian logistic regression's test error is 20% (unacceptably high).

Suppose you suspect the problem is either:

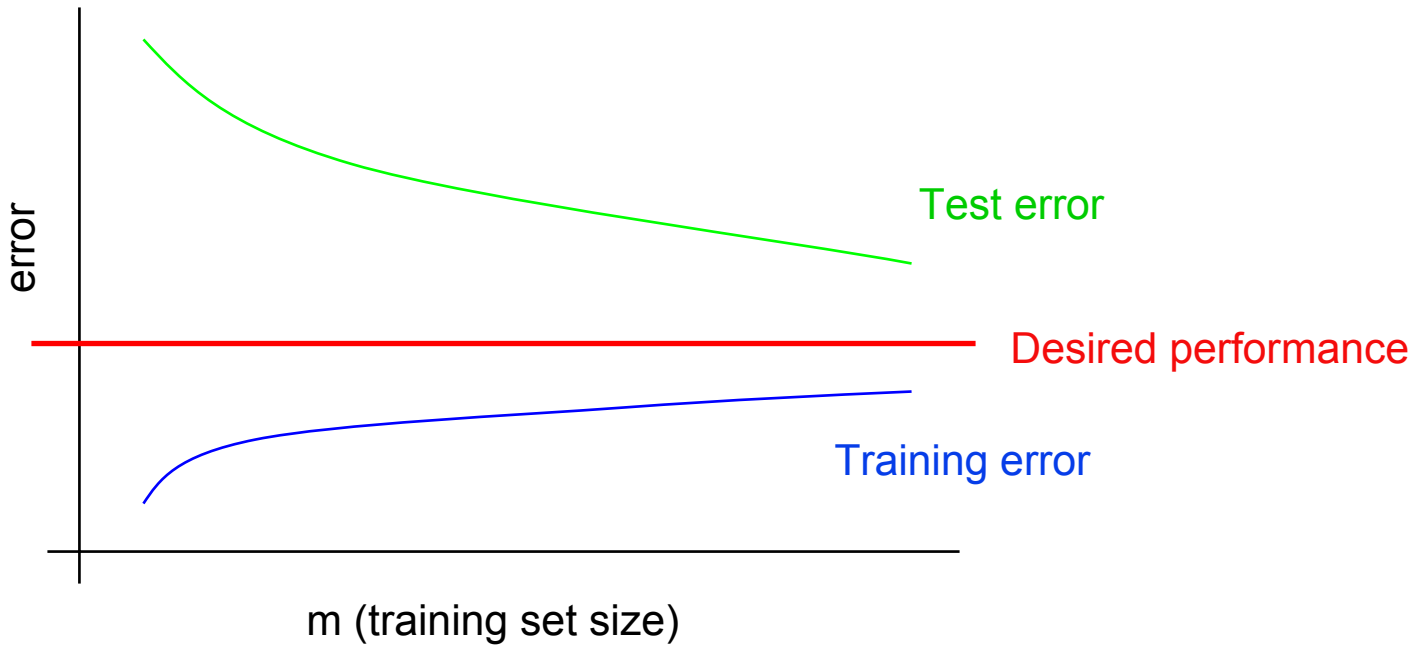
- Overfitting (high variance).
- Too few features to classify spam (high bias).

Diagnostic:

- Variance: Training error will be much lower than test error.
- Bias: Training error will also be high.

More on bias vs. variance

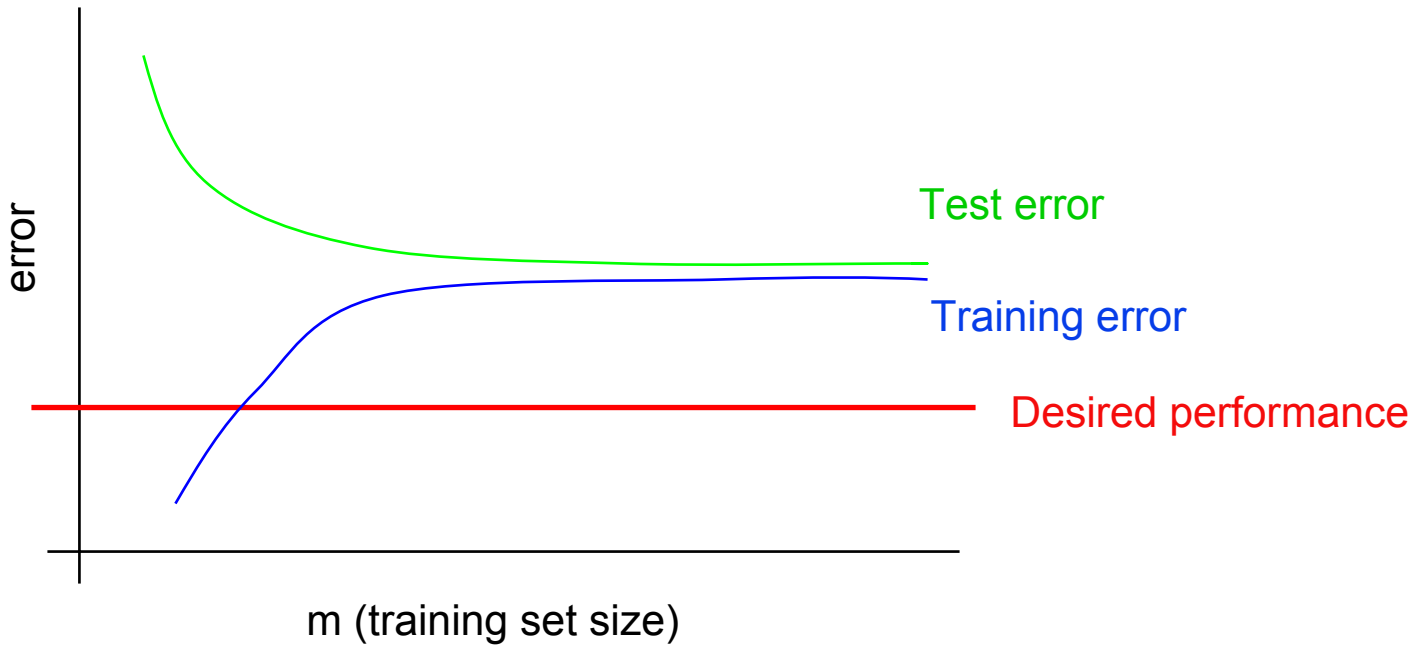
Typical learning curve for high variance:



• Test error still decreasing as m increases. Suggests larger training set will help.
• Large gap between training and test error.

More on bias vs. variance

Typical learning curve for high bias:



- Even training error is unacceptably high.
- Small gap between training and test error.

Diagnostics tell you what to try next

Bayesian logistic regression, implemented with gradient descent.

Fixes to try:

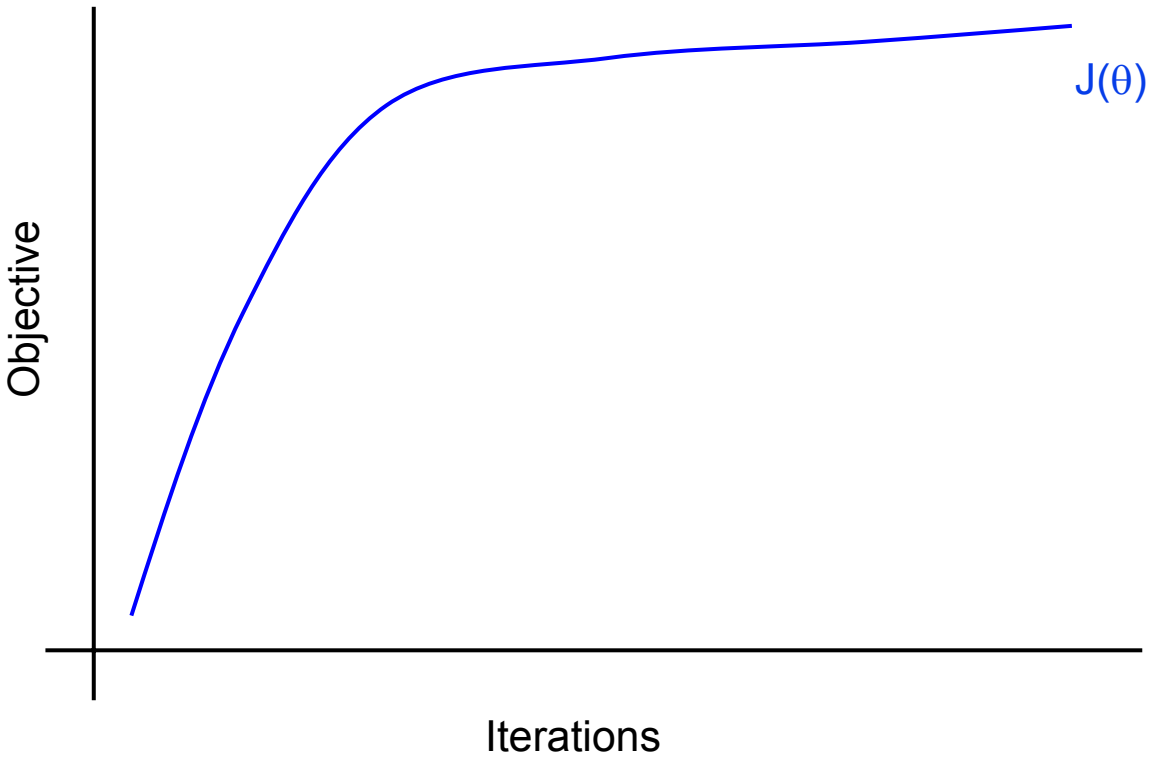
- Try getting more training examples. — Fixes high variance.
- Try a smaller set of features. — Fixes high variance.
- Try a larger set of features. — Fixes high bias.
- Try email header features. — Fixes high bias.
- Run gradient descent for more iterations.
- Try Newton's method.
- Use a different value for λ.
- Try using an SVM.

Optimization algorithm diagnostics

- Bias vs. variance is one common diagnostic.
- For other problems, it's usually up to your own ingenuity to construct your own diagnostics to figure out what's wrong.
- Another example:
  - Bayesian logistic regression gets 2% error on spam, and 2% error on non-spam. (Unacceptably high error on non-spam.)
  - SVM using a linear kernel gets 10% error on spam, and 0.01% error on non-spam. (Acceptable performance.)
  - But you want to use logistic regression, because of computational efficiency, etc.
- What to do next?

More diagnostics

- Other common questions:
  - Is the algorithm (gradient descent for logistic regression) converging?



It's often very hard to tell if an algorithm has converged yet by looking at the objective.

Diagnostics tell you what to try next

Bayesian logistic regression, implemented with gradient descent.

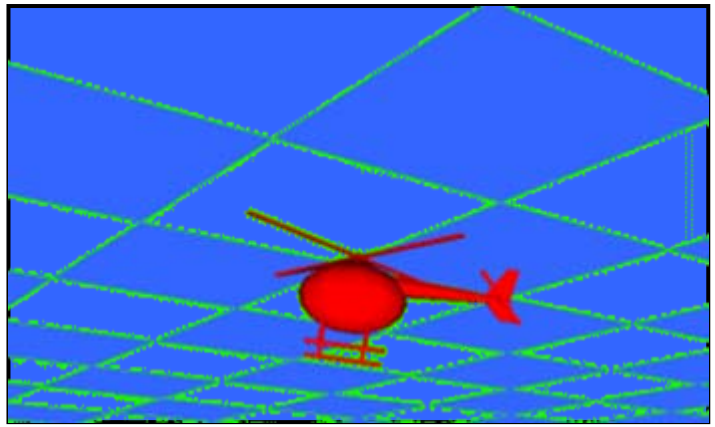
Fixes to try:

– Try getting more training examples.	Fixes high variance.
– Try a smaller set of features.	Fixes high variance.
– Try a larger set of features.	Fixes high bias.
– Try email header features.	Fixes high bias.
– Run gradient descent for more iterations.	Fixes optimization algorithm.
– Try Newton's method.	Fixes optimization algorithm.
– Use a different value for λ.	Fixes optimization objective.
– Try using an SVM.	Fixes optimization objective.

The Stanford Autonomous Helicopter

Payload: 14 pounds
Weight: 32 pounds

Machine learning algorithm



Simulator

1. Build a simulator of helicopter.

2. Choose a cost function. Say $J(\theta) = ||x - x_{desired}||^2$ (x = helicopter position)

3. Run reinforcement learning (RL) algorithm to fly helicopter in simulation, so as to try to minimize cost function:

$$\theta_{RL} = \arg\min_{\theta} J(\theta)$$

Suppose you do this, and the resulting controller parameters θ_{RL} gives much worse performance than your human pilot. What to do next?

Improve simulator?
Modify cost function J?
Modify RL algorithm?

Debugging an RL algorithm

The controller given by θ_{RL} performs poorly.

Suppose that:

1. The helicopter simulator is accurate.
2. The RL algorithm correctly controls the helicopter (in simulation) so as to minimize $J(\theta)$.
3. Minimizing $J(\theta)$ corresponds to correct autonomous flight.

Then: The learned parameters θ_{RL} should fly well on the actual helicopter.

Diagnostics:

1. If θ_{RL} flies well in simulation, but not in real life, then the problem is in the simulator. Otherwise:
2. Let θ_{human} be the human control policy. If $J(\theta_{human}) < J(\theta_{RL})$, then the problem is in the reinforcement learning algorithm. (Failing to minimize the cost function J.)
3. If $J(\theta_{human}) \geq J(\theta_{RL})$, then the problem is in the cost function. (Maximizing it doesn't correspond to good autonomous flight.)

More on diagnostics

- Quite often, you'll need to come up with your own diagnostics to figure out what's happening in an algorithm.

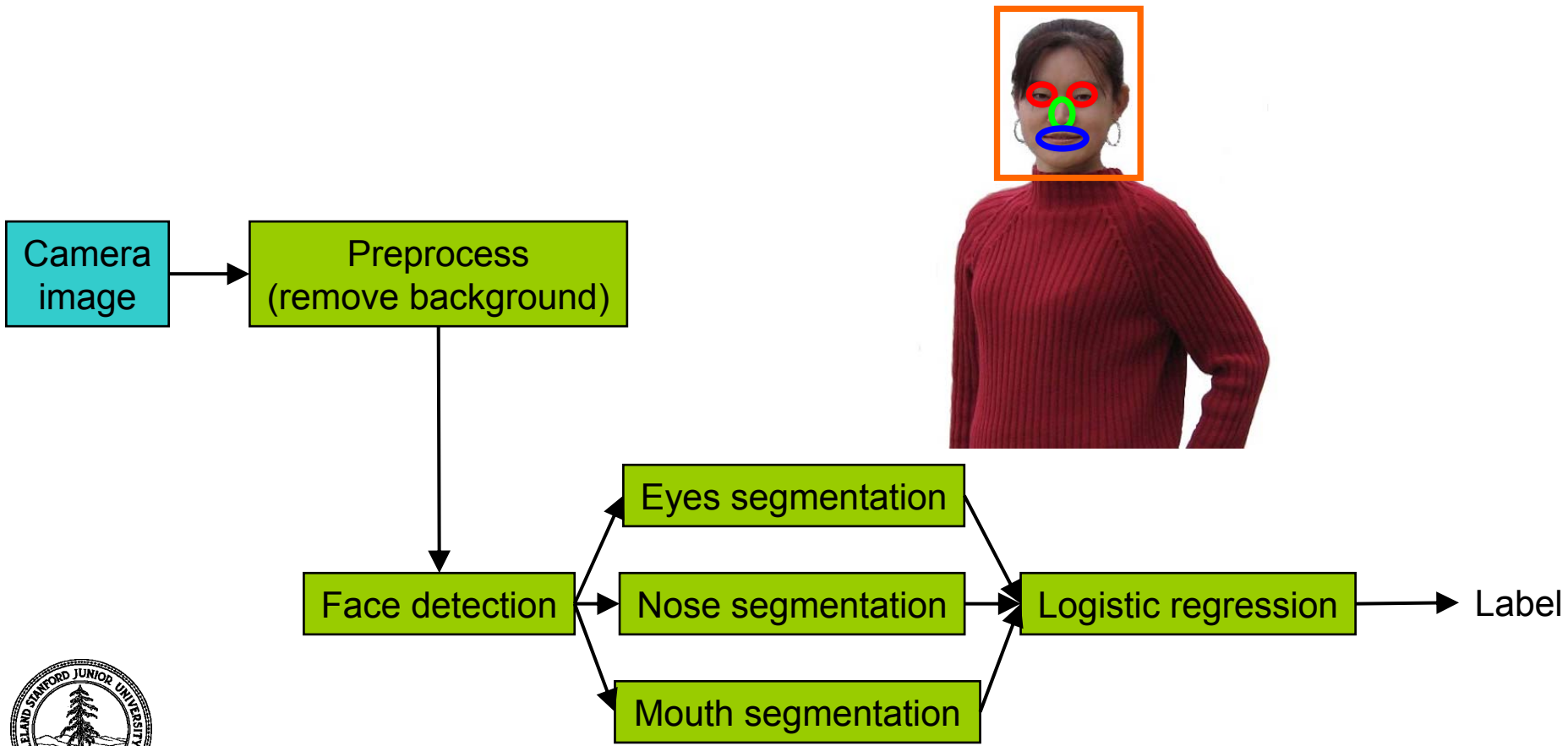
- Even if a learning algorithm is working well, you might also run diagnostics to make sure you understand what's going on. This is useful for:
  - Understanding your application problem: If you're working on one important ML application for months/years, it's very valuable for you personally to get a intuitive understand of what works and what doesn't work in your problem.
  - Writing research papers: Diagnostics and error analysis help convey insight about the problem, and justify your research claims.
  - I.e., Rather than saying "Here's an algorithm that works," it's more interesting to say "Here's an algorithm that works because of component X, and here's my justification."

- Good machine learning practice: Error analysis. Try to understand what your sources of error are.

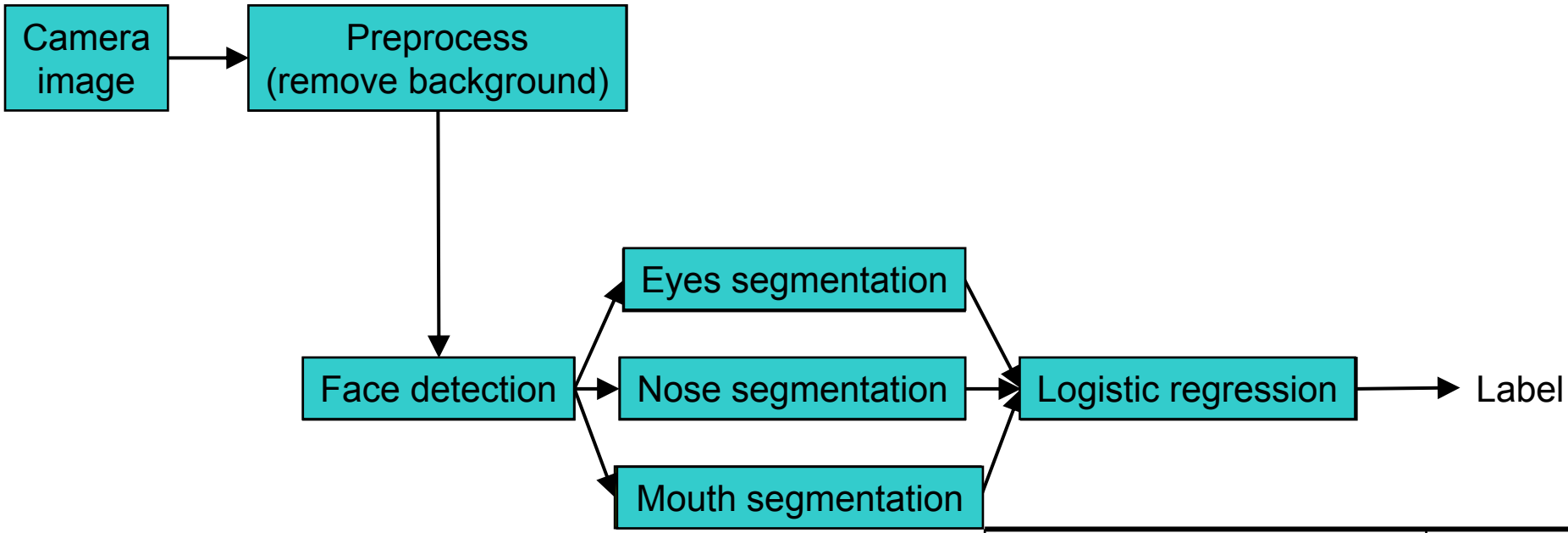
Error Analysis

Error analysis

Many applications combine many different learning components into a "pipeline." E.g., Face recognition from images: [contrived example]



Error analysis



How much error is attributable to each of the components?

Plug in ground-truth for each component, and see how accuracy changes.

Conclusion: Most room for improvement in face detection and eyes segmentation.

Component	Accuracy
Overall system	85%
Preprocess (remove background)	85.1%
Face detection	91%
Eyes segmentation	95%
Nose segmentation	96%
Mouth segmentation	97%
Logistic regression	100%

Ablative analysis

Error analysis tries to explain the difference between current performance and perfect performance.

Ablative analysis tries to explain the difference between some baseline (much poorer) performance and current performance.

E.g., Suppose that you've build a good anti-spam classifier by adding lots of clever features to logistic regression:

- Spelling correction.
- Sender host features.
- Email header features.
- Email text parser features.
- Javascript parser.
- Features from embedded images.

Question: How much did each of these components really help?

Ablative analysis

Simple logistic regression without any clever features get 94% performance.

Just what accounts for your improvement from 94 to 99.9%?

Ablative analysis: Remove components from your system one at a time, to see how it breaks.

Component	Accuracy	
Overall system	99.9%	
Spelling correction	99.0	
Sender host features	98.9%	
Email header features	98.9%	
Email text parser features	95%	
Javascript parser	94.5%	
Features from images	94.0%	[baseline]

Conclusion: The email text parser features account for most of the improvement.

Getting started on a learning problem

Getting started on a problem

Approach #1: Careful design.

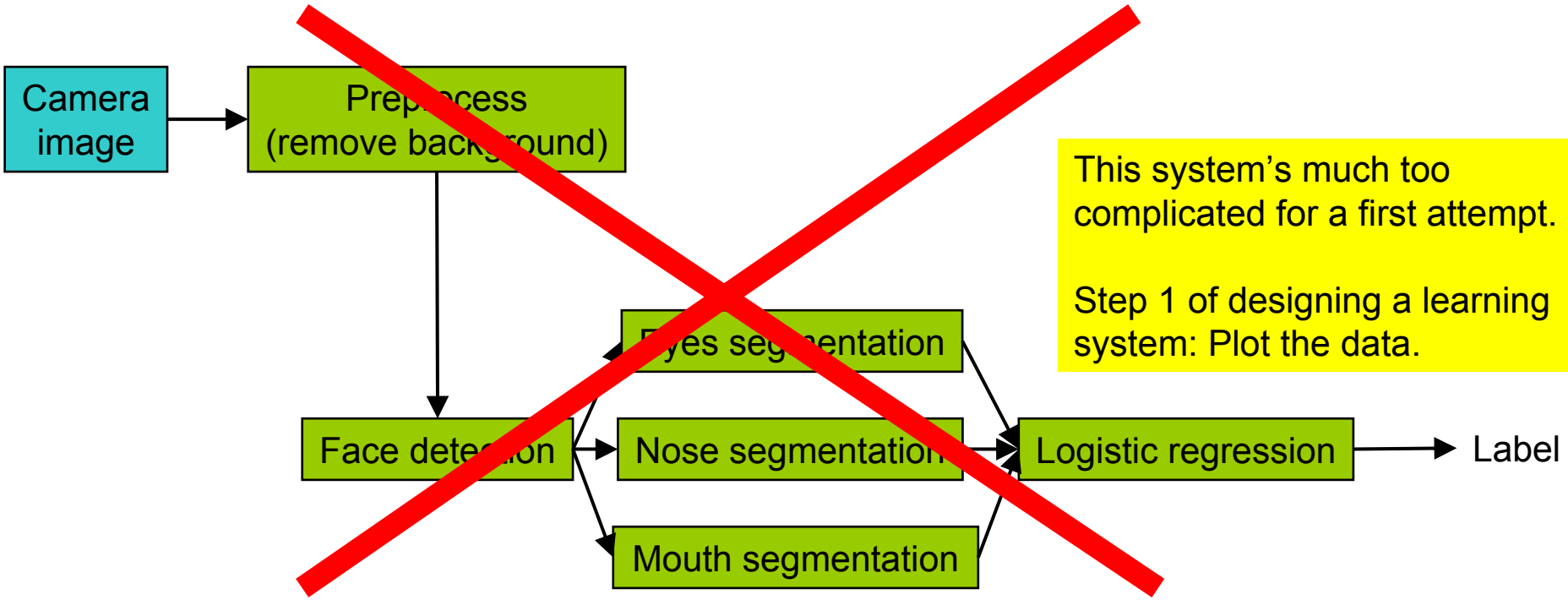
- Spend a long term designing exactly the right features, collecting the right dataset, and designing the right algorithmic architecture.
- Implement it and hope it works.
- Benefit: Nicer, perhaps more scalable algorithms. May come up with new, elegant, learning algorithms; contribute to basic research in machine learning.

Approach #2: Build-and-fix.

- Implement something quick-and-dirty.
- Run error analyses and diagnostics to see what's wrong with it, and fix its errors.
- Benefit: Will often get your application problem working more quickly. Faster time to market.

Premature statistical optimization

Very often, it's not clear what parts of a system are easy or difficult to build, and which parts you need to spend lots of time focusing on. E.g.,



The only way to find out what needs work is to implement something quickly, and find out what parts break.

[But this may be bad advice if your goal is to come up with new machine learning algorithms.]

Summary

Summary

- Time spent coming up with diagnostics for learning algorithms is time well-spent.
- It's often up to your own ingenuity to come up with right diagnostics.
- Error analyses and ablative analyses also give insight into the problem.
- Two approaches to applying learning algorithms:
  - Design very carefully, then implement.
    - Risk of premature (statistical) optimization.
  - Build a quick-and-dirty prototype, diagnose, and fix.