Digithead's Lab Notebook

Mad science in silico...

Friday, December 23, 2011

Practical advice for applying machine learning

Sprinkled throughout Andrew Ng's machine learning class is a lot of *practical advice for applying machine learning*. That's what I'm trying to compile and summarize here. Most of Ng's advice centers around the idea of making decisions in an empirical way, fitting to a data-driven discipline, rather than relying on gut feeling.

Training / validation / test

The key is dividing data into training, cross-validation and test sets. The test set is used only to evaluate performance, not to train parameters or select a model representation. The rationale for this is that training set error is not a good predictor of how well your hypothesis will generalize to new examples. In the course, we saw the cross-validation set used to select degrees of polynomial features and find optimal regularization parameters.

Model representation

The representation of the hypothesis, the function *h*, defines the space of solutions that your algorithm can find. The example used in the class was modeling house price as a function of size. The model tells you what parameters your algorithm needs to learn. If we've selected a linear function, then there are two parameters, the slope and intersect of the line.

About

Digithead's lab notebook is a place for tutorials, cheat-sheets and stray thoughts on topics related to programming, bioinformatics, machine-learning and visualization.

Learning R - tutorials and crib-notes I cooked up whilst flailing my way up the learning curve of R

Using R for Introductory Statistics - a very slow journey through the book

Machine learning class - Andrew Ng's free online class on machine learning

Hipster programming languages - a silly post, but my most popular, ever, probably based solely on the cheeky title.

By J. Christopher Bare.

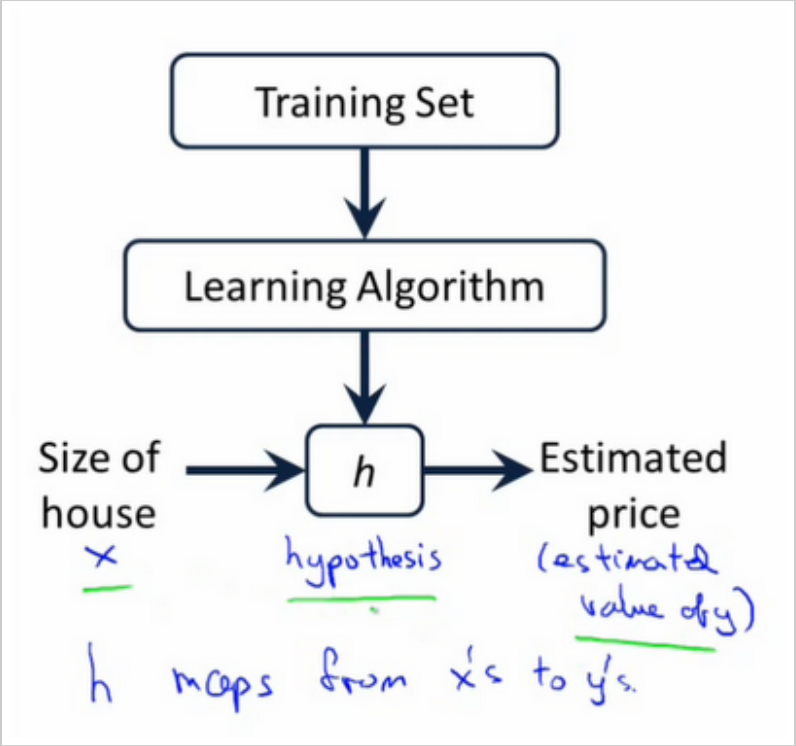
Blog Archive

► 2013 (17)

► 2012 (32)

▼ 2011 (44)

▼ December (6)



Feature selection and treatment

Are the given features sufficiently informative to make predictions? Asking whether a human expert can confidently predict the output given the input features will give a good indication.

At times, it may be necessary to derive new features. Polynomial features, for example, can let linear regression fit non-linear functions. Computing products, ratios, differences or logarithms may be informative. Creativity comes in here, but remember to test the effectiveness of your new features on the cross-validation set.

Features are on different scales may benefit from **feature scaling**. Mean normalizing and scaling to a standard deviation of one puts features on an even footing.

Gathering data might be expensive. Another option is **artificial data synthesis**, either creating new examples out of whole cloth or by transforming existing examples. In text recognition, a library of digital fonts might be used to generate examples, or existing examples might be warped or reflected.

Overfitting

Often, a learning algorithm may fit the training data very well, but perform poorly on new examples. This failure to generalize is called **overfitting**.

The classic example is fitting a high degree polynomial, which can lead to a very curvy line that closely fits a large number of data points. Our hypothesis is complex and might be fitting noise rather than an underlying relationship and will therefore generalize poorly.

One way to combat this problem is to use a simpler model. This is valid, but might be limiting. Another option is **regularization**, which penalizes large parameter values. This prioritizes solutions fitting the training data reasonably well without curving around wildly.

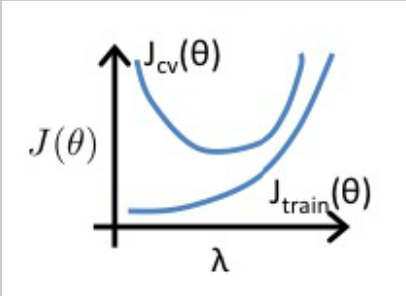
Practical advice for applying machine learning
SDCube and hybrid data storage
Neural Networks
Effective Data Visualizations
K-means
International Open Data Hackathon

► November (3)
► October (5)
► September (3)
► August (3)
► July (2)
► June (5)
► May (2)
► April (5)
► March (4)
► February (4)
► January (2)

► 2010 (36)
► 2009 (44)
► 2008 (60)
► 2007 (13)

Labels

Bioinformatics (48)
links (44)
R (42)
Java (32)
rant (31)
software engineering (29)
analytics (21)
Programming languages (20)
visualization (20)
Ruby (15)
books (15)



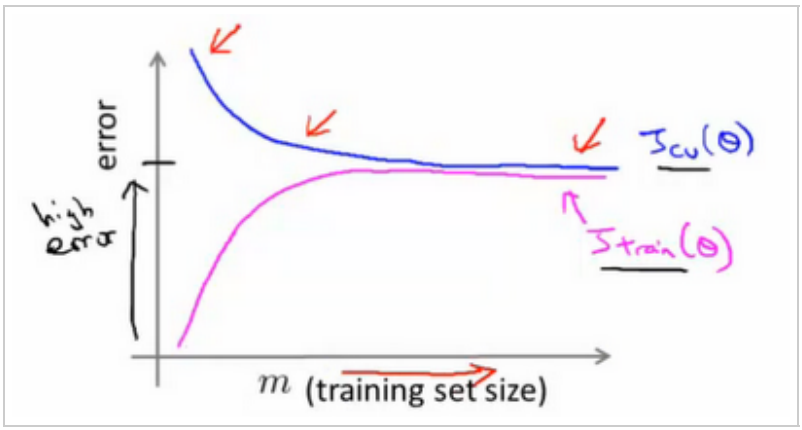
Regularization can be tuned by plotting training set error and validation set error as functions of the regularization parameter, lambda.

Tuning the trade off between bias vs variance.

The steps we take to improve performance depend on whether our algorithm is suffering from bias or variance. A **learning curve** is a diagnostic that can tell which of these situations we're in, by plotting training error and validation error as a function of training set size. Look for high training and cross-validation error indicating high bias or a steadily decreasing validation error, with a gap between validation and training error indicating high variance.

Bias

A high **bias** model has few parameters and may result in underfitting. Essentially we're trying to fit an overly simplistic hypothesis, for example linear where we should be looking for a higher order polynomial. In a high bias situation, training and cross-validation error are both high and more training data is unlikely to help much.

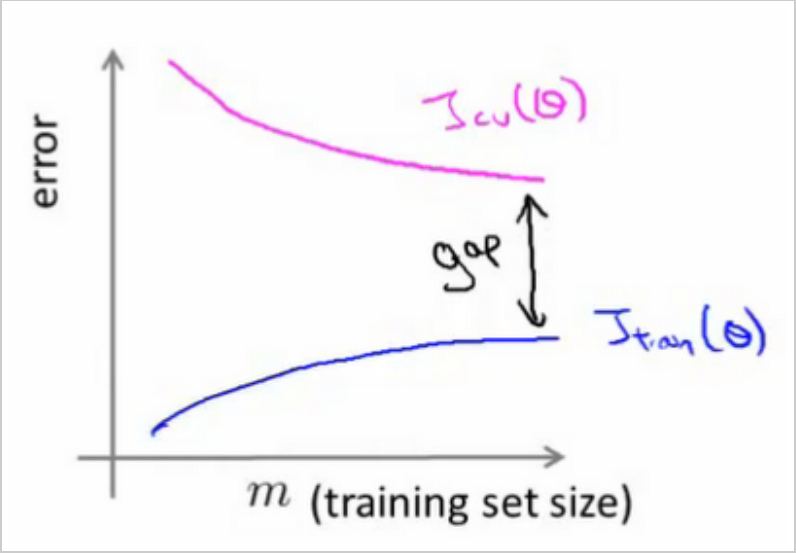


- find more features
- add polynomial features
- increase parameters (more hidden layer neurons, for example)
- decrease regularization

Variance

Variance is the opposite problem, having lots of parameters, which carries a risk of overfitting. If we are overfitting, the algorithm fits the training set well, but has high cross-validation and testing error. If we see low training set error, with cross-validation error trending downward, then the gap between them might be narrowed by training on more data.

Javascript (14)
machine learning (14)
UsingR (12)
technology (12)
tutorial (12)
Python (11)
biology (11)
crackpot theory (11)
gaggle (11)
scientific computing (11)
stats (10)
db (9)
data integration (8)
reference (8)
Firefox extension (7)
interoperability (7)
seattle (7)
software architecture (7)
Swing (6)
lectures (6)
Computer science (5)
clojure (5)
concurrency (5)
data mining (5)
graphics (5)
NoSQL (4)
data science (4)
probabilistic graphical models (4)
programming (4)
sematic web (4)
Scala (3)
bug (3)
education (3)
messaging (3)
microformats (3)
networks (3)
apis (1)
complex systems (1)
gurus (1)



- more training data
- reduce number of features, manually or using a model selection algorithm
- increase regularization

Error analysis

To improve performance of a machine learning algorithm, one helpful step is to manually examine the cases your algorithm gets wrong. Look for systematic trends in the errors. What features would have helped correctly classify these cases?

For multi-step machine learning pipelines, **ceiling analysis** can help decide where to invest effort to improve performance. The error due to each stage is estimated by substituting labeled data for that stage, revealing how well the whole pipeline would perform if that stage had no error. Stepping through the stages, we note the potential for improvement at each one.

Precision/Recall

It's helpful to have a single number to easily compare performance. Precision and recall and the F1 statistic can help when trying to classify very skewed classes, where one class is rare in the data. Simply taking a percentage of correct classifications can be misleading, since always guessing the more common class means you'll almost always be right.

precision: true positives / predicted positives, predicted positives = true positives + false positives (Of all the patients we predicted to have cancer, what fraction actually has cancer?)

recall: true positives / actual positives, actual positives = true positives + false negatives (Of all the patients that really had cancer, how many did we correctly predict?)

F1 *= 2·p·r / (p + r)*

Miscellaneous tips

Principle component analysis (**PCA**) can help by **reducing dimensionality** of high-dimensional features. Collapsing highly correlated features can help learning algorithms run faster.

open science (1)
video (1)
yak-shaving (1)

Cheat Sheets

MySQL
HTML, CSS and Javascript
Ruby
Python
PostgreSQL
Git

Feedz

Posts
Comments

About Me

CHRISTOPHER BARE

View my complete profile

Featured on

R Bloggers

Tip Jar

Often, incorrectly implemented machine learning algorithms can appear to work, producing no obvious error, but simply converging slower or with more error than a correct implementation. **Gradient checking** is a technique for checking your work, applied in the class to back propagation, but probably more generally applicable.

The recommended approach

Quickly testing ideas empirically and optimizing developer time is the approach embodied in these steps:

- First, implement a simple quick and dirty algorithm, plot learning curves, and perform error analysis.
- Create a list of potential ideas to try to improve performance. Then, start trying promising ideas, using the validation set to test for improvement.
- Use a learning algorithm with many parameters and many features - low bias. Get a very large training set.

Supporting that last point is the findings of Banko and Brill, 2001.

> *"It's not who has the best algorithm that wins. It's who has the most data."*

- diagrams by Andrew Ng
- ..more on the machine learning class

Posted by Christopher Bare at 2:09 PM

Labels: machine learning

3 comments Google+

Add a comment

Top comments

Anonymous 1 year ago

You do have permission to use these Coursera images don't you? At least one other blog has been shut down for copyright infringement recently. "All content or other materials available on the Sites, including but not limited to code, images, text, layouts, arrangements, displays,

Christopher Bare 1 year ago

My hope is that I'm on the right side of fair use. These are essentially my notes as a student taking the class. I make clear attribution and am not trying to claim anyone else's work as my own. Anyway, it's just

Anonymous 1 year ago

A good summarization, thanks.

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)