ANU ML Workshop

Developing and Debugging Machine Learning Algorithms

Stephen Gould
stephen.gould@anu.edu.au

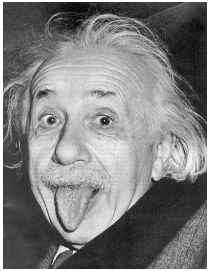
Australian National University

23 September 2011

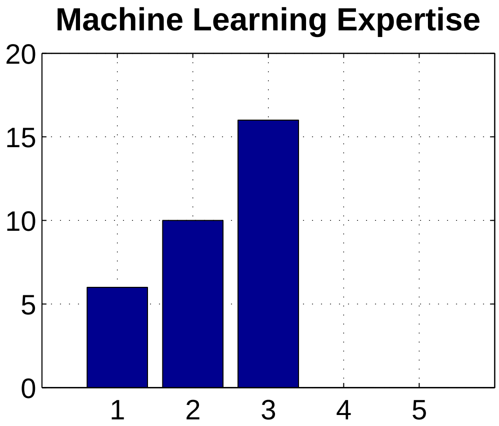
What is this workshop about?

"if we knew what we were doing it wouldn't be called research"

— Albert Einstein



Registration Expertise



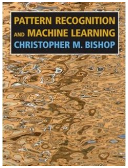
Purpose

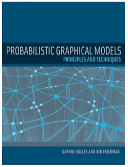
- The goal of this workshop is to provide researchers who are new to the field with some tools for debugging machine learning applications.
- The material is not mathematical. Rather we aim to develop intuitions for getting algorithms to work.
- Some of the material is debatable.
- The workshop is geared towards using supervised machine learning as a technology within real applications—the advice is not necessarily applicable to doing novel research in machine learning.
- This workshop is for you so make sure to ask lots of questions.

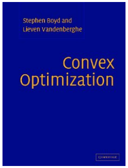
Schedule

- **Session 1 (10:00am–10:45am)**
  - Overview of supervised machine learning
  - Linear regression
  - Logistic regression and classification
- **Morning Tea**
- **Session 2 (11:15am–12:30pm)**
  - Getting started / Running experiments
  - Measuring performance and comparing algorithms
- **Lunch**
- **Session 3 (1:30pm–3:00pm)**
  - Diagnosing learning problems / Error analysis
  - Implementation tricks and approximations
- **Afternoon Tea**
- **Session 4 (3:30pm–4:15pm)**
  - Internet-scale image and video processing
- **Conclusion**

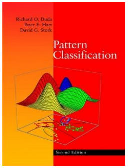
Machine Learning Books

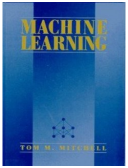


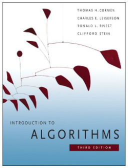

PROBABILISTIC GRAPHICAL MODELS
PRINCIPLES AND TECHNIQUES
DAPHNE KOLLER AND NIR FRIEDMAN

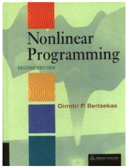


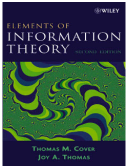
INTRODUCTION TO PROBABILITY
Dimitri P. Bertsekas | John N. Tsitsiklis

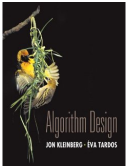


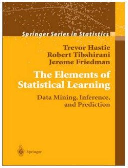


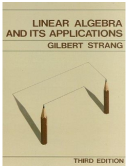


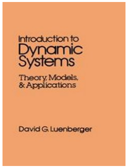






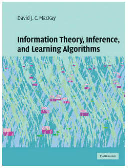

Springer Series in Statistics
Trevor Hastie
Robert Tibshirani
Jerome Friedman
The Elements of Statistical Learning
Data Mining, Inference, and Prediction
Springer




Introduction to Dynamic Systems
Theory, Models, & Applications
David G. Luenberger

MLaPP

Kevin Murphy



Machine Learning Courses

There are a number of wonderful machine learning courses with freely available lecture notes:

- Christfried Weber's "Introduction to Statistical Machine Learning" (ANU COMP4670)
- Andrew Ng's "Machine Learning" (Stanford CS229)
- Peter Christen and Lexing Xie's "Advanced Databases and Data Mining" (ANU COMP3420)

A number of other resources may be useful:

- `videolectures.net`
- Andrew Moore's tutorials

Attribution: Some of the technical material in this workshop is drawn from these sources.

Machine Learning Software

WEKA The University of Waikato

orange

SVM^{light}

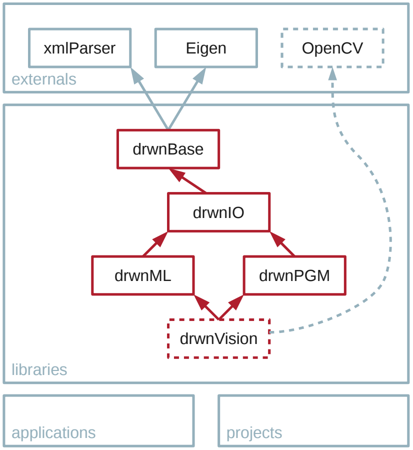
mloss.org machine learning open source software

hadoop

MiLDe Machine Learning Development Environment

Microsoft® Research

Darwin 0.4



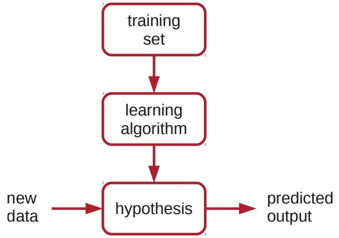
`http://users.cecs.anu.edu.au/~sgould/darwin/`

Machine Learning Algorithms Need Data

[play movie]

Supervised Machine Learning

- The task of supervised machine learning is, given a set of training examples $\mathcal{D} = \left\{(\mathbf{x}^{(t)}, y^{(t)})\right\}_{t=1}^{T}$, to learn a function $h : \mathcal{X} \to \mathcal{Y}$ so that $h(\mathbf{x})$ is a good predictor of y.



- When $y^{(t)}$ are continuous, we call the problem regression.
- When $y^{(t)}$ take on a small number of discrete values, we call the problem a classification problem.

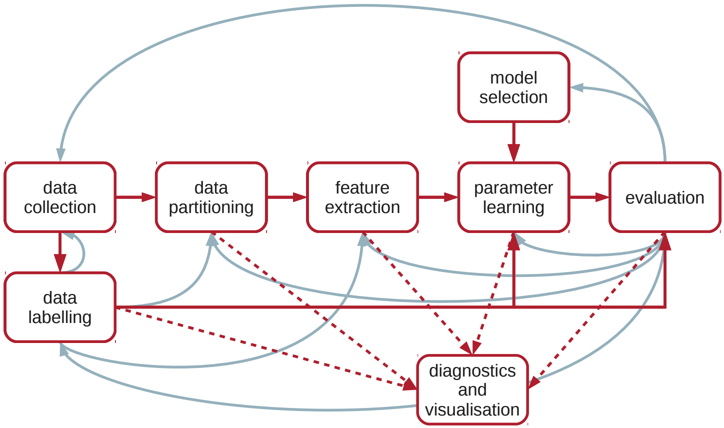
A Simple Machine Learning Problem

Australia vs. USA Rugby

Year	Score	Winner
1912	12–8	AUS
1976	24–12	AUS
1983	49–3	AUS
1987	47–12	AUS
1990	67–9	AUS
1999	55-19	AUS
2011	?	?

source: www.pickandgo.info

Machine Learning Pipeline



Linear Regression

Suppose the $\mathbf{x}^{(t)} \in \mathbb{R}^n$ and we choose our hypothesis function to approximate $y \in \mathbb{R}$ as a linear function of $\mathbf{x}$,

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x}$$

The learning task is to find the values for the parameters $\boldsymbol{\theta} \in \mathbb{R}^n$ so as to make $h_{\boldsymbol{\theta}}(\mathbf{x})$ close to y for the training samples. One way to define "close" is by square-error:

$$J(\boldsymbol{\theta}) = \frac{1}{2} \sum_{t=1}^{T} \left(h_{\boldsymbol{\theta}}(\mathbf{x}^{(t)}) - y^{(t)} \right)^2$$

The optimal parameters are then $\boldsymbol{\theta}^\star = \operatorname{argmin}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$.

Normal Equations

The parameters $\boldsymbol{\theta}^\star$ can be computed explicitly as

$$\boldsymbol{\theta}^\star = \left(X^T X\right)^{-1} X^T Y$$

where

$$X = \begin{bmatrix} — \left(\mathbf{x}^{(1)}\right)^T — \\ — \left(\mathbf{x}^{(2)}\right)^T — \\ \vdots \\ — \left(\mathbf{x}^{(T)}\right)^T — \end{bmatrix} \quad \text{and} \quad Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(T)} \end{bmatrix}.$$

This is fine in theory and for small problems but for large problems it is better to use iterative methods (more on this later).

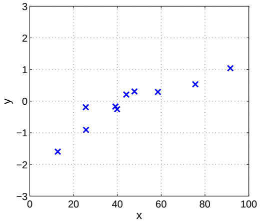
“Non-Linear” Regression

We can use linear regression to model non-linear functions by extending the input features $\mathbf{x} \in \mathbb{R}^n$ through a feature mapping, $\phi(\mathbf{x}) \in \mathbb{R}^m$. We then have

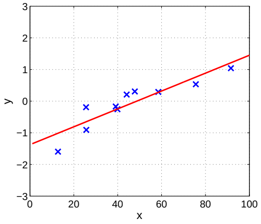
$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^T \phi(\mathbf{x})$$

Example: we can learn a quadratic function $y = ax^2 + bx + c$ with feature mapping $\phi(x) = (1, x, x^2)$.

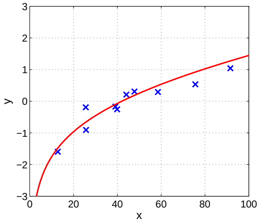
Linear Regression Example ($y = \boldsymbol{\theta}^T \phi(\mathbf{x})$)



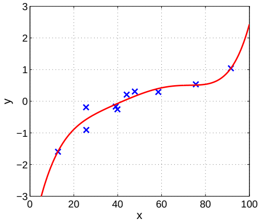
data



$\phi(x) = (x, 1)$



$\phi(x) = (x, \log(x), 1)$



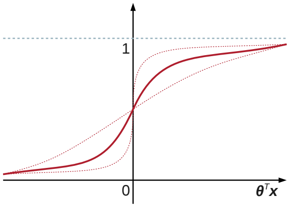
$\phi(x) = (x, x^2, x^3, x^4, x^5, 1)$

Logistic Regression

Assume we have a binary classification problem where $y \in \{0, 1\}$. We call $y = 1$ the positive class and $y = 0$ the negative class. Given a feature vector $\mathbf{x}^{(t)}$, the corresponding $y^{(t)}$ is also called the label for $\mathbf{x}^{(t)}$.

It turns out that the logistic function, or sigmoid, performs well as a hypothesis for binary classification,

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \mathbf{x}}}$$



Logistic Regression (2)

The probabilistic interpretation is

$$\mathrm{P}\left(y = 1 \mid \mathbf{x}; \boldsymbol{\theta}\right) = h_{\boldsymbol{\theta}}(\mathbf{x}) \quad \text{and} \quad \mathrm{P}\left(y = 0 \mid \mathbf{x}; \boldsymbol{\theta}\right) = 1 - h_{\boldsymbol{\theta}}(\mathbf{x})$$

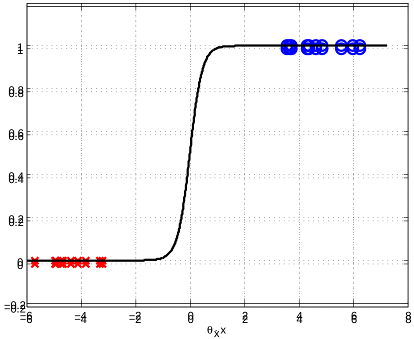
Assuming the training examples are generated independently, we can write the log-likelihood of the labels as

$$\log L(\boldsymbol{\theta}) = \sum_{t=1}^{T} y^{(t)} \log\left(h_{\boldsymbol{\theta}}(\mathbf{x}^{(t)})\right) + \left(1 - y^{(t)}\right) \log\left(1 - h_{\boldsymbol{\theta}}(\mathbf{x}^{(t)})\right)$$

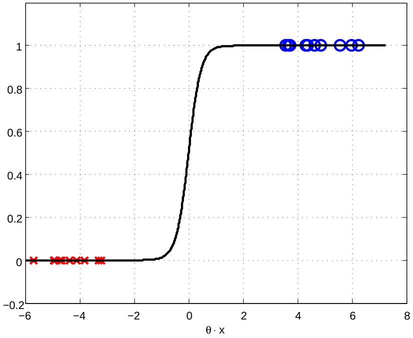
We can now take gradients and maximize with respect to $\boldsymbol{\theta}$.

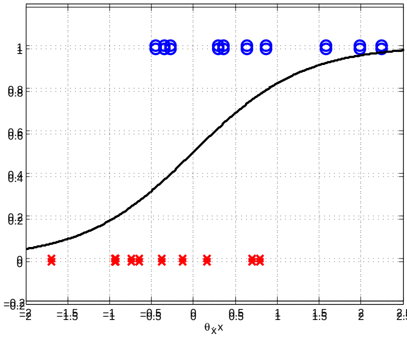
Logistic Regression Example

Label	Feature
0	-3.83
0	-4.37
1	6.25
0	-4.92
1	4.36
1	5.58
0	-4.94
0	-3.20
1	4.64
1	4.86
0	-4.64
1	3.55
1	4.29
1	3.65
0	-5.69
0	-3.30
0	-4.94
0	-3.20
0	-4.73
0	-4.12
1	3.73
0	-4.73
0	-4.12
1	5.98
0	-4.12



Logistic Regression Example (2)





Multi-Class Logistic Regression

The logistic, or log-linear, model can be extended to a multi-class classifier ($y \in \{1, \ldots, K\}$) as follows,

$$\mathrm{P}\left(y = k \mid \mathbf{x}; \boldsymbol{\theta}\right) = \frac{e^{\boldsymbol{\theta}_k^T \mathbf{x}}}{Z}$$

where $Z = \sum_{k=1}^{K} e^{\boldsymbol{\theta}_k^T \mathbf{x}}$ (known as the partition function).

Learn parameters $\boldsymbol{\theta}$ by maximum-likelihood

$$\log L(\boldsymbol{\theta}) = \sum_{t=1}^{T} \sum_{k=1}^{K} [\![y^{(t)} = k]\!] \boldsymbol{\theta}_k^T \mathbf{x}^{(t)} - \log Z.$$

Regression and Classification Summary

	Linear Regression	Logistic Regression	Multi-class Logistic
features	$\mathbb{R}^n$	$\mathbb{R}^n$	$\mathbb{R}^n$
targets	$\mathbb{R}$	$[0, 1]$	Δ^{K-1}
hypothesis	$h(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x}$	$\mathrm{P}(1) = \frac{e^{\boldsymbol{\theta}^T \mathbf{x}}}{1+e^{\boldsymbol{\theta}^T \mathbf{x}}}$	$\mathrm{P}(k) = \frac{e^{\boldsymbol{\theta}_k^T \mathbf{x}}}{Z}$
loss	ℓ_2	- log-likelihood	- log-likelihood

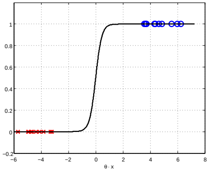
Regularization

Purely optimizing for the loss function (e.g., maximum-likelihood) may overfit the model to our training data. To avoid overfitting we often use regularization (which we can motivate from a Bayesian perspective).

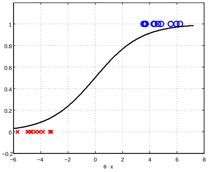
$$\boldsymbol{\theta}^\star = \operatorname{argmin}_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}) + \lambda r(\boldsymbol{\theta})$$

The meta-parameter $\lambda \geq 0$ controls how much we regularize the parameters.

λ small



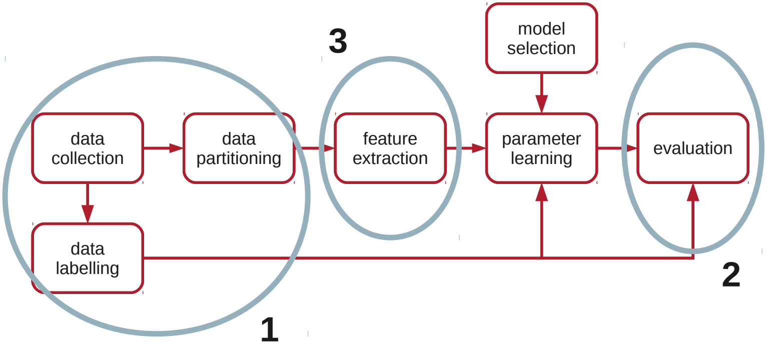
λ large



Session 2

(11:15am–12:30pm)

Getting Started: ML Pipeline



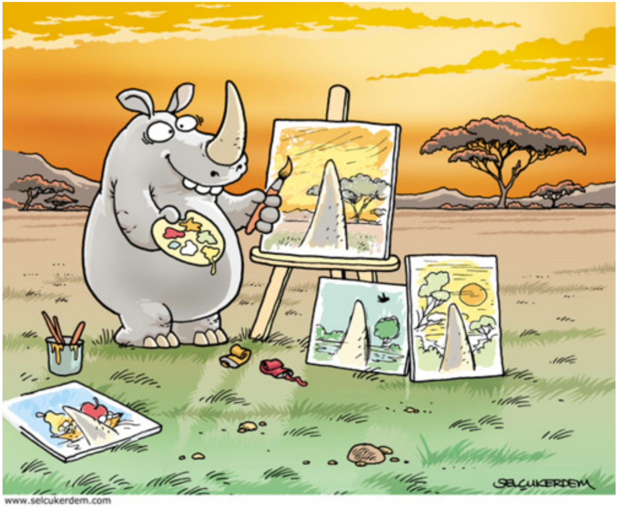
Getting Started: Be Organized

A useful directory structure:

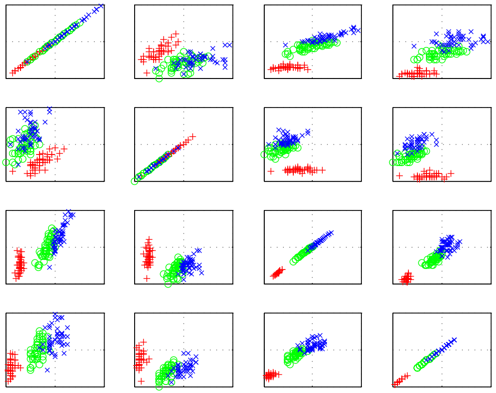
```
cached/
data/
models/
output/
clean.sh
pipeline.sh
configuration.xml
trainList.txt
valList.txt
testList.txt
experiment.log
```

Note: no code (which should be revision controlled elsewhere)

Getting Started: Plot Your Data

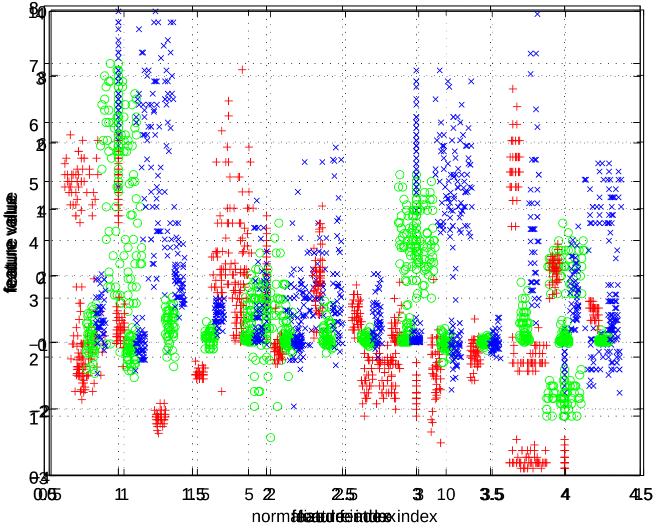


Scatter Plots

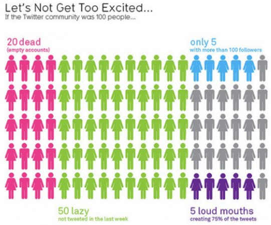


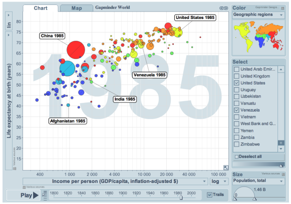
can also visualize arbitrary projections or as 3D point clouds

Dither Plots



Other Data Visualization

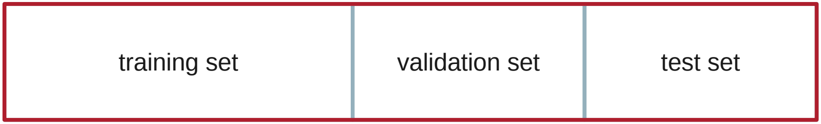

Let's Not Get Too Excited...
If the Twitter community was 100 people...
20 dead
(empty accounts)
only 5
with more than 100 followers
50 lazy
not tweeted in the last week
5 loud mouths
creating 75% of the tweets



data
prices
time
interest
Google
map
look
like
Olympic
Thanks
probably
gas
India
SimCity
search
series
stuff
medal
sure
line

source: google images

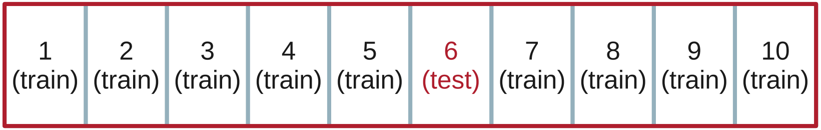
Dataset Partitioning



- **training set:** learn model parameters
- **validation set:** tune meta-parameters
- **testing/evaluation set:** report performance
  - *ideally used exactly once*

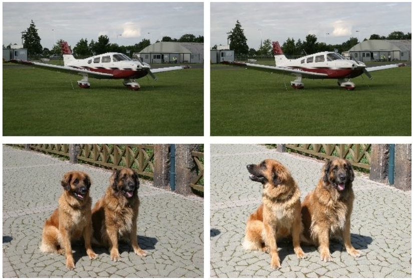
Cross-Validation

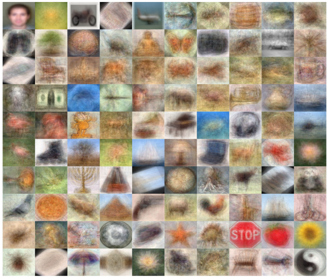
Cross-validation is a common method used to estimate how well a model generalizes to unseen data.



- **K-fold**: Split the data into K sets of roughly equal size. For the k-th fold, train the model on $K-1$ parts and test on the k-th part. We can now use all the data to estimate the prediction error. (How?)
  - How do we choose K?
- **leave-one-out (LOOCV)**: set K to the size of the dataset

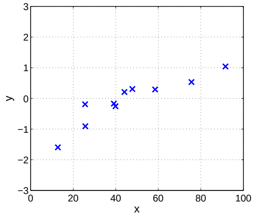
Dataset Bias


21-class MSRC

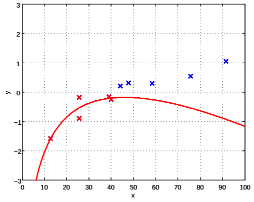


Caltech 101

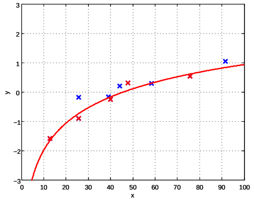
Training Set (Fold) Sampling Strategies



data



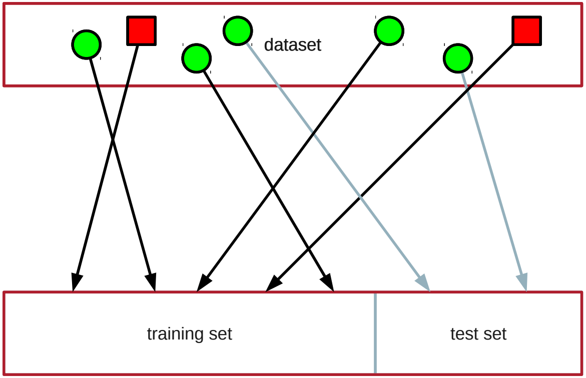
$\phi(x) = (x, \log(x), 1)$



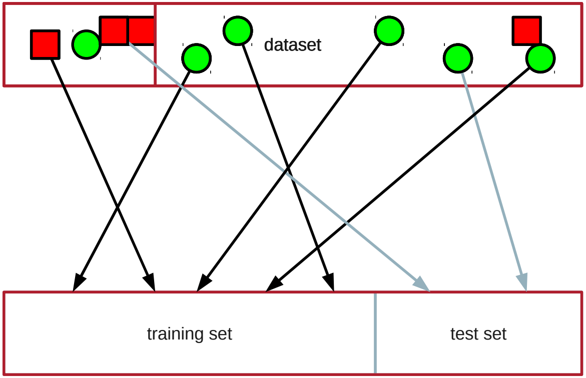
$\phi(x) = (x, \log(x), 1)$

- random sampling
  - Example: classifying pixels in images
  - Example: classifying frames in a video
- unbalanced datasets
  - stratified sampling
  - data weighting (re-sampling or modifying loss function)

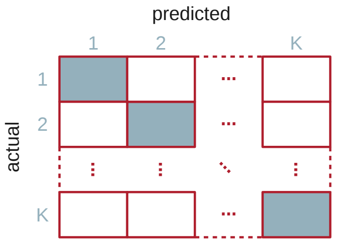
Unbalanced Datasets (Random Sampling)



Unbalanced Datasets (Stratified Sampling)



Confusion Matrix



- **(i,j) entry:** number of examples of class *i* that were predicted as class *j*
- **row sum:** number of ground-truth examples of class *i*
- **column sum:** number of examples predicted as class *j*
- **diagonal sum:** number of correctly classified examples
- **total sum:** number of total examples

The number of rows does not need to equal number of columns!

Warning: sometimes you will see the matrix transposed.

Accuracy: Macro vs. Micro Averaging

Often we care about overall classification accuracy. This is an example of micro-averaging,

$$\text{accuracy}_{\text{micro}} = \frac{\text{number of correct classifications}}{\text{total number of examples}}$$

However, sometimes we have an unbalanced dataset and wish to treat each class equally. This is an example of macro-averaging,

$$\text{accuracy}_{\text{macro}} = \frac{1}{K} \sum_{k=1}^{K} \frac{\text{number of correct classifications for class } k}{\text{total number of examples for class } k}$$

More generally, we may also want to compute weighted accuracy.

Precision and Recall

		predicted	
		0	1
actual	0	TN	FP
	1	FN	TP

Terminology:

TERMINOLOGY	EQUATION
true positive, hit, detection	TP
true negative, correct rejection	TN
false positive, false alarm, Type I error	FP
false negative, miss, Type II error	FN

Precision and Recall (2)

actual \ predicted	0	1
0	**TN**	**FP**
1	**FN**	**TP**

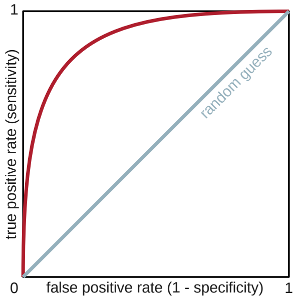
Derived statistics:

Statistic	Equation
recall, true positive rate, sensitivity, hit rate	TP / (TP + FN)
positive predictive power, precision	TP / (TP + FP)
true negative rate, specificity	TN / (TN + FP)
accuracy	$\frac{(TP+TN)}{(TN+FP+FN+TP)}$
F_1-score	$2\frac{(precision \times recall)}{(precision+recall)}$

Others: false alarm rate, false positive rate, fall-out; Jaccard coefficient, area-of-overlap; false discovery rate; F_β-score; etc.

Receiver Operator Characteristics (ROC) Curves

Visualizes how correctly classified positive examples varies with the number of incorrectly classified negative examples.



Not good if large skew in class distribution.

ROC Numerical Example

Consider an application where we are trying to detect cancer. We have a test set of 1000 patients who are normal and 10 patients who have cancer.

Our algorithms gives the following confusion matrix:

	N	C
N	990	10
C	5	5

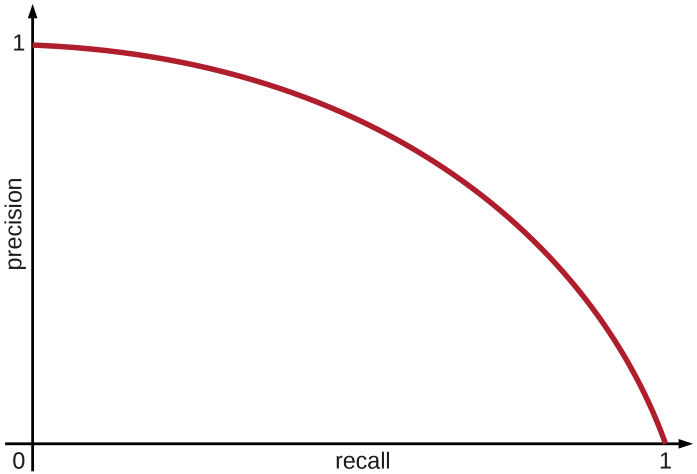
recall (TPR) = 50%
TNR = 99%
precision = 33%

A different algorithm has confusion matrix:

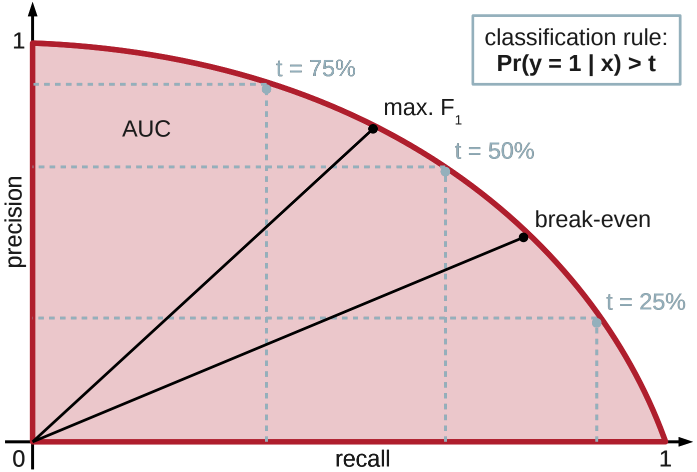
	N	C
N	972	28
C	3	7

recall (TPR) = 70%
TNR = 97.2%
precision = 20%

Precision-Recall (PR) Curves



Precision-Recall Curve Operating Points



classification rule:
Pr(y = 1 | x) > t

Efficiently Computing Precision-Recall Curves

Computing PR Curve (Method 1)

- for each threshold t
  - classify samples using rule $h_{\boldsymbol{\theta}}(\mathbf{x}) > t$
  - build confusion matrix
  - compute precision
  - compute recall
  - plot point

Computing PR Curve (Method 2)

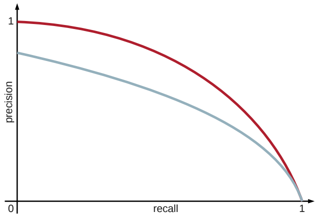
- construct a sorted table

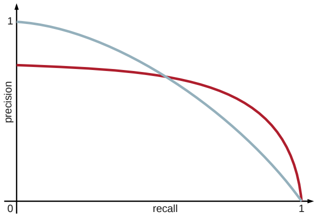
SCORE	#P	#N
$-\infty$		
$\vdots$		
∞		

- compute cumulative sums
- ...

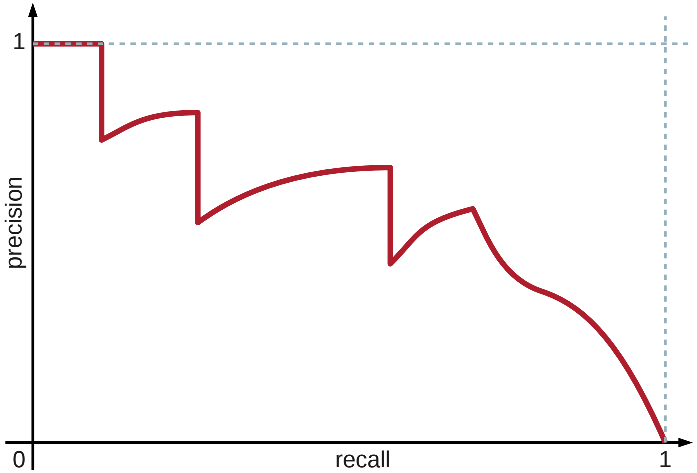
Comparing Precision-Recall Curves

Which algorithm is better?

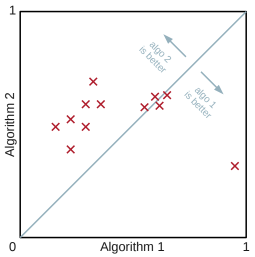


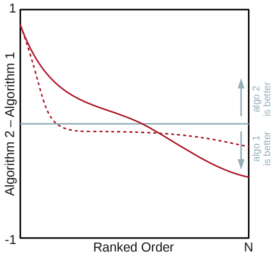


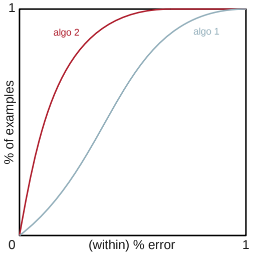
Real Precision-Recall Curves



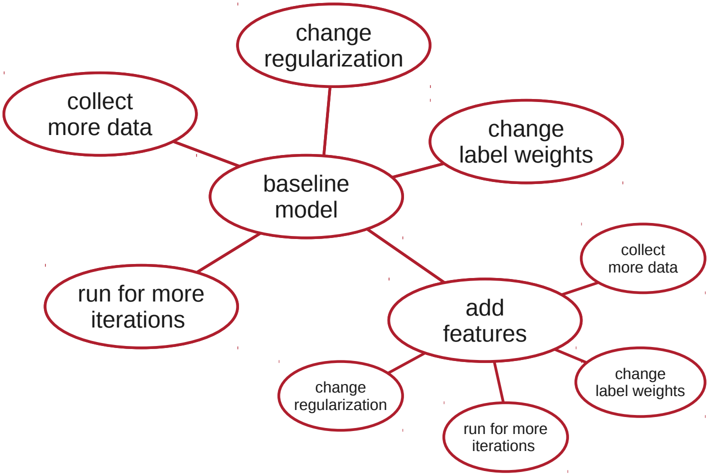
Other Ways to Compare Algorithms







Exploring Features and Meta-Parameters



Feature Selection

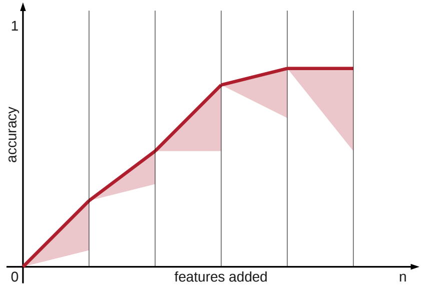
Often the number of available features is very large but there are only a small number of relevant features. We want to choose a good subset of features, but given n features there are 2^n possible subsets. How can we choose the best one?

- **Filter Methods:** Use a computationally cheap heuristic to evaluate features, e.g., mutual information between a features and class label.
- **Wrapper Methods:** Incrementally add the best feature to the feature set (forward feature selection) or remove the worst from the feature set (backward feature selection).

Note. Some recent methods use sparsity inducing priors in an attempt to perform joint feature selection and parameter learning.

Example: Forward Feature Selection

- Start with an empty feature set, $\mathcal{F} = \{\}$
- Repeatedly try each feature $i \notin \mathcal{F}$, create $\mathcal{F}_i = \mathcal{F} \cup \{i\}$, and use cross-validation to evaluate $\mathcal{F}_i$. Set $\mathcal{F}$ to be the best $\mathcal{F}_i$.



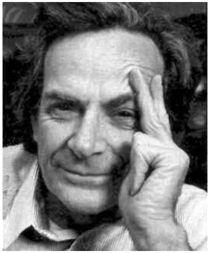
Session 3

(1:30pm–3:00pm)

Diagnostics

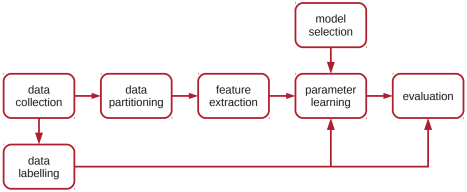
"I write down the question, I think very hard, and then I write down the solution."

— Richard Feynman



Diagnosing Machine Learning Problems

Assume you've written some code for your machine learning application and you're not getting the performance you want. What could the problem be?



- problem statement
- data
- features
- algorithm/model
- implementation
- something else

We need diagnostics to help narrow down the problem...

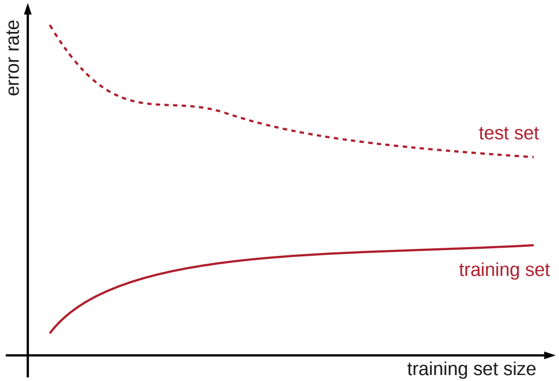
Diagnosing Machine Learning Problems (2)

Example: Suppose that our test error is unacceptably high and we suspect the problem is either that the model overfitting or the features are not good enough.

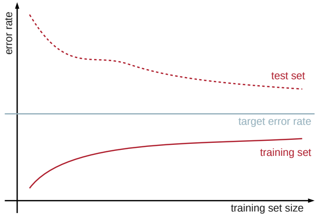
Diagnostics:

- The first hypothesis (overfitting) suggests that the training error will be much lower than the test error.
- The second hypothesis (features) suggests that the training error and test error will both be high.

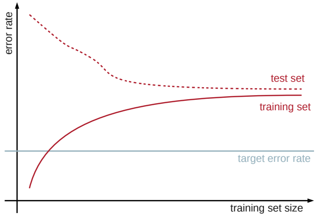
Learning Curves



Learning Curves: Bias vs. Variance

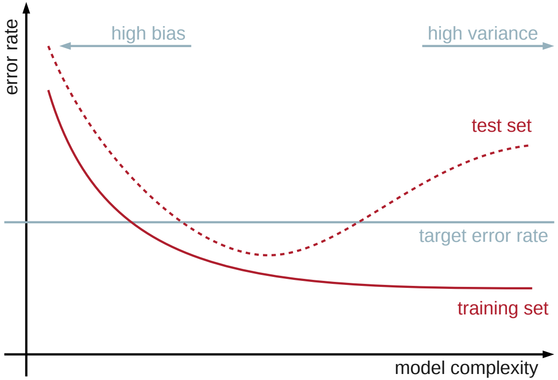


high variance



high bias

Bias/Variance Trade-Off



Fixes for Bias/Variance Problems

Diagnosing bias and variance problems provides us with hints as to what to try next.

For bias problems:

- try a larger set of features
- try a richer model class

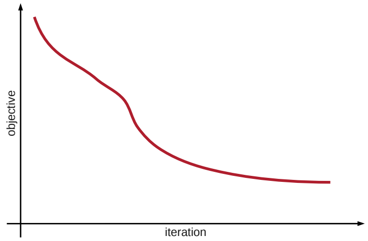
For variance problems:

- try getting more training examples
- try a smaller set of features

Objective/Optimization Problems

We may suspect that our poor performance is due to either a problem with our optimization algorithm (e.g., not running for long enough) or with our objective.

Unfortunately it is often very difficult to determine whether or not an iterative algorithm has converged.



Diagnosing Optimization Problems

Suppose the thing that we care about is weighted accuracy, i.e.,

$$\mathrm{acc}(\boldsymbol{\theta}) = \sum_{t=1}^{T} [\![y^{(t)} \neq h_{\boldsymbol{\theta}}(\mathbf{x}^{(t)})]\!] w^{(t)}$$

(where higher is better).

Our learning algorithm is trying to optimize $J(\boldsymbol{\theta}) = \ell(\theta) + \lambda r(\boldsymbol{\theta})$ (where lower is better).

Let $\boldsymbol{\theta}^\star$ be the model parameters returned by our learning algorithm and let $\hat{\boldsymbol{\theta}}$ be any other parameters (e.g., guessed or obtained from a different learning algorithm).

Diagnosing Optimization Problems (2)

	$\text{acc}(\hat{\boldsymbol{\theta}}) > \text{acc}(\boldsymbol{\theta}^\star)$	$\text{acc}(\hat{\boldsymbol{\theta}}) < \text{acc}(\boldsymbol{\theta}^\star)$
$J(\boldsymbol{\theta}^\star) < J(\hat{\boldsymbol{\theta}})$	wrong objective	no problem (?)
$J(\boldsymbol{\theta}^\star) > J(\hat{\boldsymbol{\theta}})$	bad optimization	

- $\boldsymbol{\theta}^\star$: parameters from our algorithm
- $\hat{\boldsymbol{\theta}}$: competing parameters
- $J(\boldsymbol{\theta})$: what we are optimizing (lower is better)
- $\text{acc}(\boldsymbol{\theta})$: what we care about (higher is better)

Fixes for Optimization/Objective Problems

Diagnosing optimization versus objective problems provides us with hints as to what to try next.

For optimization problems:

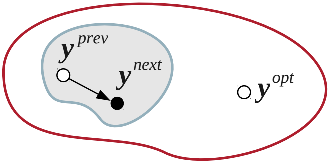
- try running for more iterations
- try using a different algorithm (e.g., Newton's method as opposed to gradient descent)
- try random restarts (for non-convex objectives)
- try smoothing (e.g., L_1-approximation (see later))

For objective problems:

- try different regularization
- try weighting training examples
- try a different loss function
- change the model

Approximate Search Algorithms

- initialize a solution, $\hat{\mathbf{y}}$
- repeat (until convergence)
  - define a neighbourhood $\mathcal{N}(\hat{\mathbf{y}})$
  - find best local solution $\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \in \mathcal{N}(\hat{\mathbf{y}})} \operatorname{score}(\mathbf{y}; \mathbf{x})$



How can we tell whether the problem is with our search algorithm or the scoring function?

For those who work with graphical models, how can we tell whether the problem is with our inference algorithm or our energy function?

Diagnosing Search/Score Problems

Suppose we are trying to maximize score. Let $\hat{\mathbf{y}}$ be the solution found by our search algorithm and let $\mathbf{y}^\star \neq \hat{\mathbf{y}}$ be the ground-truth solution.

- If $\mathrm{score}(\hat{\mathbf{y}}) > \mathrm{score}(\mathbf{y}^\star)$ then the problem is with our scoring function.

Otherwise, initialize the search algorithm with the ground-truth solution, $\mathbf{y}^\star$.

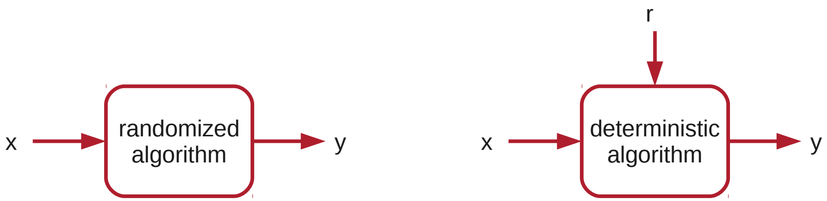
- If the search algorithm moves away from the ground-truth solution then the problem is also with our scoring function.
- Otherwise the problem is with our search algorithm.

Diagnostics Summary

- Diagnostics are an important tool when developing your machine learning algorithm.
  - We showed examples for **bias/variance**, **search/score**, and **optimization/objective**, but there are many others.
  - They can save a lot of wasted effort by guiding your choice of what to try next.
  - They also allow you to develop insights into your particular application and justify your design decisions.
- Diagnostics often involve repeated experiments with different parameter settings while keeping everything else fixed (see next slide).
- Another important diagnostic tool is that of error analysis, i.e., understanding where your errors are coming from.

De-randomization

Comparing different runs of an algorithm is difficult if the algorithm is stochastic.



- transform random algorithm $A(\mathbf{x})$ into deterministic $A'(\mathbf{x}, \mathbf{r})$ where $\mathbf{r}$ is a sequence of random numbers
- use random seeds
  - (e.g., `srand()` in C/C++, `rng()` in Matlab R2011a)

Error Analysis

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

- **How much error is due to various different machine learning components in the application?**

  Plug the ground-truth (if available) into each component of the application and see how it affects accuracy. Alternatively, we could add noise to each component and, again, see how it affects accuracy.

- **Does the algorithm fail on a particular subclass of examples?**

  Visualize the data and results (see previous session).

Ablative Analysis

Ablative analysis tries to explain the different between some baseline performance and the current performance.

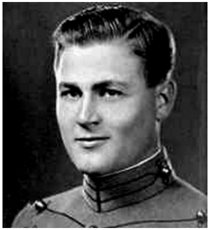
Example: You've been working on your application for the past several months and now have a number of sophisticated features that you pass to a logistic regression classifier. Which features account for the good performance of your classifier over some baseline logistic regression model with some simple features?

Ablative analysis would remove features from the application one at a time and see which results in the biggest decrease in performance—similar to backward feature selection. **Note:** The order of feature removal matters.

Implementation Issues

"Anything that can go wrong will go wrong."

— Edward A. Murphy, Jr.



Diagnosing Implementations

So you've just finished implementing the first version of your new whizz-bang deep-kernalized-logistic-SVM-GP classifier. How do you test it?

- small synthetic test case
- ground-truth features
- random features
- boundary cases
- re-use known working components

Numerical Tricks

Numerical calculations on a computer are always subject to errors. These can be due to

- limited precision arithmetic
- algorithmic limitations (e.g., generating true random numbers)
- careless implementation
  - we will see some examples soon
- bugs

Example. What is $16777216 + 1$?

```
float x = 16777216.0;
float y = x + 1.0f;
assert(x != y);
```

Feature Scaling

Numerical algorithms work best on well-scaled data. We usually scale our input feature vectors to have zero mean and unit variance (sometimes called feature whitening), e.g.,

$$x_i^{(t)} \mapsto \frac{x_i^{(t)} - \hat{\mu}_i}{\hat{\sigma}_i} \sim \mathcal{N}(0, 1)$$

or

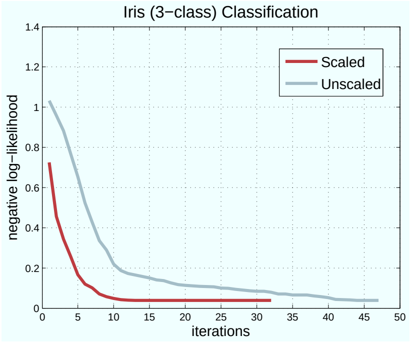
$$\mathbf{x}^{(t)} \mapsto \hat{\Sigma}^{-\frac{1}{2}} \left(\mathbf{x}^{(t)} - \hat{\boldsymbol{\mu}} \right) \sim \mathcal{N}(\mathbf{0}_n, \mathbf{I}_n)$$

Note. Parameters are learned from the set of training examples.

Question: for which classifiers (i.e., learning algorithms) would feature scaling not have any effect?

Feature Scaling Example

- **Dataset:** Iris [Fisher, 1936]: three classes, four features, 50 examples per class
- **Feature vector:** squared raw features plus bias term
- **Classifier:** multi-class logistic



Effect of Scaling on Classification Accuracy

- Feature scaling does not affect the “strength” of the classifier—however, it does help with convergence during training
- For linear models there is a direct mapping between optimal parameter vectors. **Example:** for binary classifier

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \mathbf{x} + \theta_0}}$$

with $\tilde{\mathbf{x}} = \hat{\Sigma}^{-\frac{1}{2}}(\mathbf{x} - \hat{\boldsymbol{\mu}})$ we have

$$\tilde{\boldsymbol{\theta}}^\star = \hat{\Sigma}^{\frac{1}{2}} \boldsymbol{\theta}^\star \quad \text{and} \quad \tilde{\theta}_0^\star = \theta_0^\star + \hat{\boldsymbol{\mu}}^T \tilde{\boldsymbol{\theta}}^\star$$

Standard Deviation Calculations

The empirical standard deviation of a feature is defined as

$$\hat{\sigma}_i = \sqrt{\frac{\sum_{t=1}^{T}\left(x_i^{(t)} - \mu_i\right)^2}{T-1}}$$

where $\mu_i = \frac{1}{T}\sum_{t=1}^{T} x_i^{(t)}$.

This calculation requires two passes through the data. A seemingly better approach is to perform equivalent calculation

$$\hat{\sigma}_i = \sqrt{\frac{T\sum_{t=1}^{T}\left(x_i^{(t)}\right)^2 - \left(\sum_{t=1}^{T} x_i^{(t)}\right)^2}{T(T-1)}}$$

which only requires one pass.

What can go wrong with this implementation?

logsumexp

Often, e.g., in computing the (log-)normalization constant for logistic regression, we would like to perform a computation like

$$Z = \log\left(\sum_{i=1}^{n} \exp\left(\alpha_i\right)\right)$$

Problem: numerical overflow and underflow

Solution: set $\alpha^{\max} = \max\left\{\alpha_i : i = 1, \ldots, n\right\}$, then

$$Z = \alpha^{\max} + \log\left(\sum_{i=1}^{n} \exp\left(\alpha_i - \alpha^{\max}\right)\right)$$

L_2-norm Calculations

$$\|\mathbf{x}\|_2 = \sqrt{x_1^2 + \ldots + x_n^2}$$

Problem: numerical overflow and underflow

Solution: set $x^{\max} = \max_i \{|x_i|\}$, then

$$\|\mathbf{x}\|_2 = x^{\max} \sqrt{\sum_{i=1}^{n} \left(\frac{x_i}{x^{\max}}\right)^2}$$

Speed Comparison

$$x = Ab$$

```
memset((void *)x, 0.0, ...);
for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
        x[i] += A[i][j] * b[j];
```

running time: 1.8ms.
(N = 1000)

$$x = A^T b$$

```
memset((void *)x, 0.0, ...);
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++)
        x[i] += A[j][i] * b[j];
```

running time: 8.4ms.

```
memset((void *)x, 0.0, ...);
for (int j = 0; j < N; j++)
    for (int i = 0; i < N; i++) {
        x[i] += A[j][i] * b[j];
```

running time: 2.2ms.

both operations take approximately 1.5ms using Eigen.

Matlab Vectorization

Despite what most people claim, Matlab is very fast. You just need to use it for what it's good at.

Example: Consider the following code for computing the Mahalanobis distance, $d = (\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})$, between each vector $\mathbf{x}^{(t)}$ in our dataset and some reference vector $\boldsymbol{\mu}$.

```
for t = 1:T,
  d(t) = (X(t, :) - mu') * inv(Sigma) * (X(t, :)' - mu);
end;
```

A faster version...

```
z = X - repmat(mu', T, 1);
d = sum((z * inv(Sigma)) .* z, 2);
```

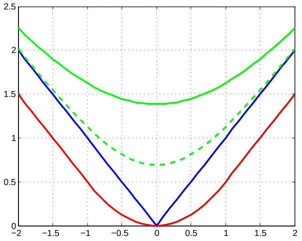
L_1 Approximations

The L_1-norm penalty is often used as a robust regularizer or sparsity inducing prior. Unfortunately it is non-smooth and cannot be easily optimized.

$$r_{L1}(x) = |\theta|$$

$$r_{L1}(x; M) \approx \begin{cases} \frac{1}{2}x^2 & \text{for } x \leq M \\ M(|x| - \frac{1}{2}M) & \text{otherwise} \end{cases}$$

$$r_{L1}(x; \alpha) \approx \frac{1}{\alpha}\Big(\log(1 + e^{-\alpha x}) + \log(1 - e^{\alpha x})\Big)$$



Other Implementation Tips

- **software is written for people, not for machines**
- check for `NaN` and `Inf`
- assert pre- and post-conditions
- write simple test cases when debugging (and keep them for future regression testing)
- use existing code where possible
  - but don't get bogged down gluing together third-party code
- use source control (e.g., SVN, Git, etc.)
  - do not store files than can be reproduced or downloaded
- print out debugging information (but not within tight loops)
- use `top`, Task Manager, or `valgrind` to check for memory leaks
- run on small examples before your entire dataset

Session 4

(3:30pm–4:15pm)

Conclusion

Final Advice

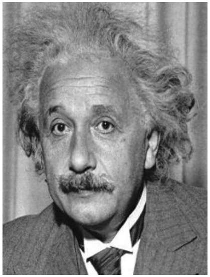
- **VISUALIZE YOUR DATA AND RESULTS!!!**
- two design strategies:

careful design or build-and-fix

- implement and test as you go
  - keep notes of what to do later

"every system should be made a simple as possible and no simpler"

— Albert Einstein



Topics Not Covered

It is impossible to cover all the practical issues of machine learning in one day. Here is a partial list of topics that were not covered:

- curse of dimensionality (and dimensionality reduction, i.e., feature selection)
- choice of classifier (i.e., model selection)
  - Occam's razor
  - no free lunch theorem
- dealing with missing data
- unsupervised learning and models with latent variables
- structured prediction problems (e.g., Markov random fields)

thank you

(if you have feedback please email me)