Machine Learning: Big Picture

Scott Sanner
U. of Toronto, MIE

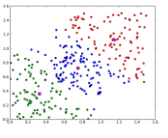
Machine Learning

- You have a data set **D** = **{(x_i,y_i)}**
  - $\mathbf{x_i}$ is a feature vector, $\mathbf{y_i}$ are labels
  - Data may be partially specified
- You want to learn **y** = **f(x)** from **D**
  - More precisely, you want to minimize some error **E(f,w,D)**

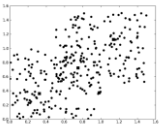
    Function class — Function parameters
- Majority of problems are either
  - Classification: **y** is discrete
  - Regression: **y** is continuous

Supervised vs. Unsupervised

- D = {($\mathbf{x_i}$,$\mathbf{y_i}$)}
- Supervised: $\mathbf{y_i}$ observed
  - Classification for discrete $\mathbf{y_i}$
  - Regression for continuous $\mathbf{y_i}$



- Unsupervised: $\mathbf{y_i}$ not observed
  - Learning method has to assign $\mathbf{y_i}$
  - E.g., clustering via K-means



Inductive Bias

- **Let's avoid making assumptions about f**
  - Assume for simplicity that **D** = **{(x_i,y_i)}** is noise free
  - $\mathbf{x_i}$'s in **D** only cover small subset of input space **x**
- **What's the best we can do?**
  - If we've seen **x=**$\mathbf{x_i}$ report **y=**$\mathbf{y_i}$
  - If we have not seen **x=**$\mathbf{x_i}$, can't say anything (no assumptions)
- **This is called rote learning… boring, eh?**
  - Key idea: *you can't generalize to unseen data w/o assumptions*!
- **Thus, key to ML is generalization**
  - To generalize, ML algorithm *must* have some **inductive bias**
  - Bias usually in the form of a **restricted hypothesis space**
  - Important to understand restrictions (and whether appropriate)

Parametric vs. Non-parametric

- **Parametric:** has parameters
  - Most Probabilistic Approaches
    - Gaussians
    - Bernoulli / Binomial / Multinomial
    - Graphical Models
  - Linear Regression / Classifiers
  - SVM with linear kernel

> **What assumptions?**
> Data is generated by given distribution.
> Linearly separable, error assumptions (e.g., Gaussian), etc.

- **Non/semi-parametric:** data oriented
  - Neighbor-based approaches
    - (K-)nearest Neighbor
    - Parzen Windows
  - SVM with RBF kernel

> **What assumptions?**
> Encoded in distance function & K / width.
> Smoothness… no abrupt changes!

Linear vs. Non-linear

- **y = f(x,w)**
  - **x** is your input vector
  - **w** is your parameter vector (weights)
- **Which f is linear in w?**

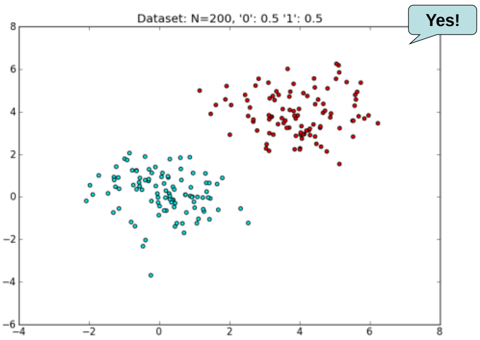
  i.e., **f(x,w)** = ⟨**w,x**⟩ (assume $\mathbf{x_0}$ = 1)
  - $\mathbf{f_1(x) = w_1\, x_1 + w_2\, x_2}$ ✔
  - $\mathbf{f_2(x) = w_1\, x_1^2 + w_2\, x_1x_2 + w_3\, x_2^2}$ ✔
  - $\mathbf{f_3(x) = w_1\, x_1 + w_2w_3\, x_2 + w_3^2\, x_3}$ ✘

> **Any** transformation of input **x** maintains linear function in **w**!

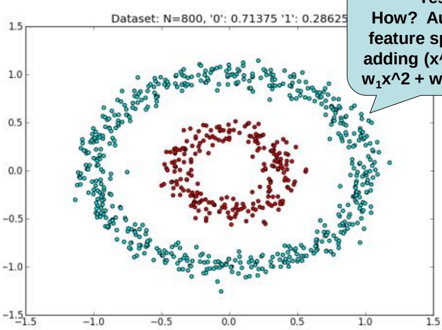
Your Linear Fun. Approx. Toolbox

- **Classification**
  - Naïve Bayes (simple)
  - Logistic Regression (better than NB for dependent features)
  - Perceptron (didactic, rarely used in practice)
  - SVM and Kernel Methods (very powerful)
- **Regression**
  - Linear Regression (closed-form solution)
  - SVR and Kernel Methods (very powerful)
- **Key Advantage**
  - All of above lead to convex optimization problems ➔ global optima will be found.

Classes Linearly Separable?



Classes Linearly Separable?



How Powerful are Linear Models?

- **Short answer:**
  - $\mathbf{y} = \langle \mathbf{w},\mathbf{x} \rangle = \sum_i \mathbf{w}_i \mathbf{x}_i$ is surprisingly powerful
  - Especially if **x** transformed: $\mathbf{x} \rightarrow \Phi(\mathbf{x})$
    - $\mathbf{y} = \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle$

      input space *feature* space
- **Can use expressive feature spaces $\Phi(\mathbf{x})$**
  - If data << features, beware of **overfitting**
    - More on overfitting later
  - Allows to fit virtually any nonlinear function
    - Optimization problem is still convex – **solve optimally**!

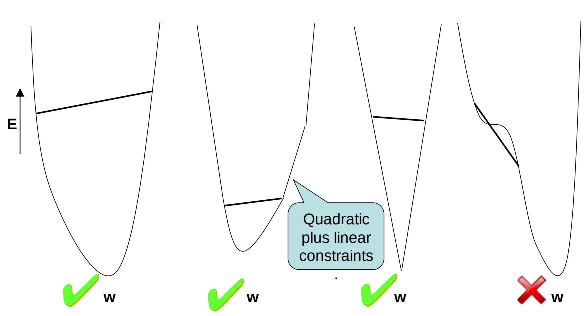
This is Worth Investigating…

- **Seems counterintuitive…**
  - Fit crazy non-linear functions & find global optimum?
  - WTF? Why's that fine?
- **Think about abstracted problem…**
  - **E(w)**=**E(f,w,D)** (b/c **f**,**D** fixed)
  - We change the weights **w**, we get different **E(w)**.
  - Which setting of weights optimizes **E(w)**?
- **Question: how does E(w) look w.r.t. weights?**
  - Linear regression: linear **f** & SSE, **E(w)** looks quadratic
  - In general, can show Hessian (2^{nd} derivative matrix) is positive semidefinite ⇒ **convex**…

Convexity

For convex problems, (sub)gradient descent gets us to **global minima**. It's that easy!

- Graphically… which of these is convex?



Empirical Risk Minimization (ERM)

- A **general framework** for function approximation
- Minimize E'(**w**) = Loss(**w**) + C*Regularizer(**w**) — Critical to avoid overfitting
- Loss functions penalize errors in different ways, e.g.,
  - Sum of squared error (SSE) — Linear Regression
  - Hinge loss — Why useful?
  - ε-insensitive loss — Why useful?
- Regularizer expresses preference on **w**, e.g.,
  - $||\mathbf{w}||_2$: assumes Gaussian prior (prefers small weights)
  - $||\mathbf{w}||_1$: can encourage sparsity
  - **w** · log **w**: maximizes entropy for prob. interpretation of **w**
- **Many E'(w) possibilities are convex for linear f!**

The Joy of Convex(ity)

All of these losses are convex for linear **f**:

Table 1: Scalar loss functions and their derivatives, depending on $f := \langle w, x \rangle$, and y.

	Loss $l(f, y)$	Derivative $l'(f, y)$
Hinge [20]	$\max(0, -yf)$	0 if $yf \geq 0$ and $-y$ otherwise
Squared Hinge [26]	$\frac{1}{2}\max(0, -yf)^2$	0 if $yf \geq 0$ and f otherwise
Soft Margin [4]	$\max(0, 1 - yf)$	0 if $yf \geq 1$ and $-y$ otherwise
Squared Soft Margin [10]	$\frac{1}{2}\max(0, 1 - yf)^2$	0 if $yf \geq 1$ and $f - y$ otherwise
Exponential [14]	$\exp(-yf)$	$-y\exp(-yf)$
Logistic [13]	$\log(1 + \exp(-yf))$	$-y/(1 + \exp(yf))$
Novelty [32]	$\max(0, 1 - f)$	0 if $f \geq 0$ and -1 otherwise
Least mean squares [43]	$\frac{1}{2}(f - y)^2$ ← **linear regression**	$f - y$
Least absolute deviation	$\|f - y\|$	$\mathrm{sgn}(f - y)$
Quantile regression [27]	$\max(\tau(f - y), (1 - \tau)(y - f))$	τ if $f > y$ and $\tau - 1$ otherwise
ϵ-insensitive [41]	$\max(0, \|f - y\| - \epsilon)$	0 if $\|f - y\| \leq \epsilon$ and $\mathrm{sgn}(f - y)$ otherwise
Huber's robust loss [31]	$\frac{1}{2}(f - y)^2$ if $\|f - y\| < 1$, else $\|f - y\| - \frac{1}{2}$	$f - y$ if $\|f - y\| \leq 1$, else $\mathrm{sgn}(f - y)$
Poisson regression [16]	$\exp(f) - yf$	$\exp(f) - y$

Table 2: Vectorial loss functions and their derivatives, depending on the vector $f := Wx$ and on y.

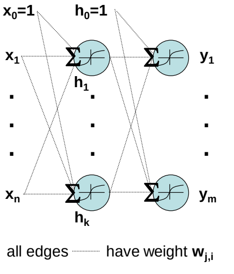
	Loss	Derivative
Soft Margin [38]	$\max_{y'}(f_{y'} - f_y + \Delta(y, y'))$	$e_{y^*} - e_y$, where y^* is the argmax of the loss
Scaled Soft Margin [40]	$\max_{y'} \Delta^\beta(y, y')(f_{y'} - f_y + \Delta(y, y'))$	$\Delta^\beta(y, y')(e_{y^*} - e_y)$, where y^* is the argmax of the loss
Softmax [14]	$\log \sum_{y'} \exp(f_{y'}) - f_y$	$\left[\sum_{y'} e_{y'} \exp(f'_y)\right] / \sum_{y'} \exp(f'_y) - e_y$
Multivariate Regression	$\frac{1}{2}(f - y)^\top M(f - y)$ where $M \succeq 0$	$M(f - y)$

Non-convexity

- Convexity was the rage from 2000-2010
- Non-convex approaches like neural networks could not guarantee optimal learning
  - Therefore no one used them
  - No one bothered to compare to them
  - Reviewers rejected papers on the topic
  - Until some folks recently tried using them in combination with Big Data + Many Layers + GPUs
- And they led to massive improvements!

Artificial Neural Networks

- **Neural Net:** non-linear weighted combination of *shared* sub-functions
- **Backpropagation:** to minimize SSE, train weights using *gradient descent* and *chain rule*
- **Deep nets:** have more than one hidden layer
  http://playground.tensorflow.org/

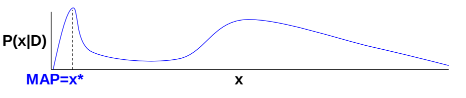


Being Bayesian

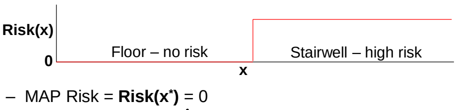
- Many (philosophical) interpretations of what it means to be Bayesian
  - All share a common characteristic
- Bayesian = maintaining a distribution over the most likely values of a (random) variable
  - Variable can be any quantity of interest
    - A location **(x,y)** for tracking
    - Parameters **w** of a linear classifier

Why be Bayesian? Risk!

- **Robot has Bayesian belief P(x|D) over position x**
  - **D** consists of noisy range finder readings



- **Associate Risk(x) with position x (e.g., stairs!)**



  - MAP Risk = **Risk(x*)** = 0
  - Full Bayesian Risk = $\int_x$ **Risk(x)p(x|D)** > 0
- **Which risk estimate would you use?**

Overfitting

- In brief: fitting characteristics of training data that do not generalize to future test data
- Central problem in machine learning
  - Particularly problematic if #data << #parameters

  … don't have enough data to "identify" parameters

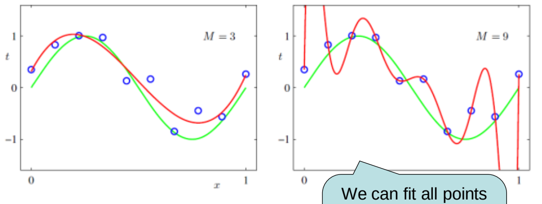
Overfitting in Classification

- Example: try to classify technology web pages {true, false}
  - Crawl Microsoft website as positive examples
  - Highly weighted features:
    - Bill Gates
    - support@microsoft.com
  - But do they generalize?

No, we've overfit to sampled-biased training data. Can we combat this?

Overfitting in Regression

- Green: Generate data (blue) plus noise
- Red: Fit a polynomial of degree M to the data



We can fit all points perfectly with a 9th degree polynomial… will it's predictions generalize?

Combatting Overfitting

- Careful "unbiased" selection of training data helps
  - Test data should be from same distribution as train data
- But unbiased data is not easy, nor enough
  - There are always spurious correlations to overfit
  - Best way to fight overfitting is by **restricting the hypothesis space**
    - For linear classifiers, do feature selection
    - Tune "hyperparameters" to avoid overfitting
      - Naïve Bayes has smoothing hyperparameter
      - SVMs and Logistic Regression have "C"
    - **Cross-validation is important for tuning**

How to fix problem on previous slide?

(Cross) Validation

Sometimes a single random split may do unusually good or bad… better to average over multiple splits.

- **Validation:**
  - Split your data into train and test

All Data	
Test Data	Train Data

- **K-fold Cross Validation:**
  - Reduce variance of test error estimate
  - 3-fold cross validation example

Test Data	Train Data	Train Data
Train Data	Test Data	Train Data
Train Data	Train Data	Test Data

Report avg & 95% CI

Repeated Random Sub-sampling Validation

- If have 1000 samples and need k=100
  - Then test case only has 10 samples
  - Too small!
- Instead, choose an X% train/test split k times:
  - For X = 90, randomly split data into 90% for train data and 10% test data
  - Take k=100 splits with test data containing 100 samples
  - Caveat: testing data overlaps, introduce correlations
  - Make sure to use a proper random permutation
    - numpy.random.shuffle or numpy.random.permutation

Nested Cross-Validation (NCV)

- Two levels of cross-validation
- 1st (Top) level CV
  - For **average performance**
  - Split data into {train, test}
- 2nd (Bottom) level CV
  - For **hyperparameter tuning**
  - Split train into {train', validation}
  - Choose hyperparameters that maximize avg performance on validation set, use in 1st level

Why NCV? Because must tune to fight overfitting, but cannot tune on the test set! This is cheating… why?

Can't tune on train data – need tuning to generalize to unseen data.

So separate out validation from train in a CV sub-level to tune hyperparameters.

If takes too long, tune on single train/val split at 2nd level, but less robust.

Aside: CI Common Mistake

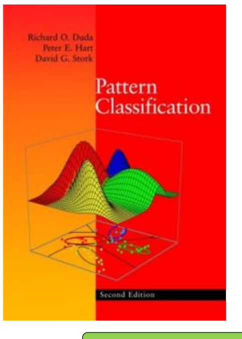
- When I ask for **avg ± CIs**
  - I want confidence interval (CIs) on the **avg**
    - 67% CIs are *roughly* **avg** ± $\sigma/\sqrt{n}$
    - 95% CIs are *roughly* **avg** ± $2\sigma/\sqrt{n}$
    - $\sigma/\sqrt{n}$ is standard error of the mean
      - https://en.wikipedia.org/wiki/Standard_error#Standard_error_of_the_mean_2
    - **95% CIs:** intervals for avg that should hold 95 out of 100 times if I rerun the randomized experiment
  - Students often give me **avg** ± σ (std deviation)
    - Sample deviation, not confidence in sample mean

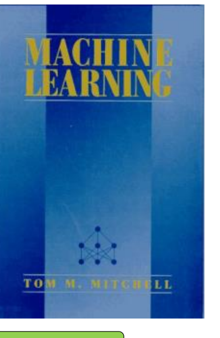
How confident are you for n=∞?

Model and Feature Selection

- Data set **D** = $\{(x_i,y_i)\}$
- Learn **y** = **f(x)** by minimizing **E(f,w,D)**
- Model Selection:
  - Which **f** to use
    - Linear / Non-linear
    - Parametric / Non- / Semi-parametric
    - Parameters within each model
- Feature Selection
  - Which **x** to use
    - Subset, transformed?
- How to choose between different models or feature sets
  - Can choose model / feature set with **lowest CV error**

Introductory Books to Consider





Chapters 1-6 in both books are useful.