COMP [56]630– Machine Learning

Lecture 4 – Regularized LR, Bias-Variance Tradeoff

Regularized Least Squares

Regularized Least Squares

- How do you prevent overfitting?

Regularized Least Squares

- How do you prevent overfitting?
- ***Regularization!***
- As model complexity increases, e.g., degree of polynomial or no. of basis functions, then it is likely that we overfit

Regularized Least Squares

- How do you prevent overfitting?
- ***Regularization!***
- As model complexity increases, e.g., degree of polynomial or no. of basis functions, then it is likely that we overfit
- One way to control overfitting is not to limit complexity but to add a regularization term to the error function E_D

$$E(\mathrm{w}) = E_D(\mathrm{w}) + \lambda E_W(\mathrm{w})$$

- where λ is the regularization coefficient that controls relative importance of data-dependent error $E_D(w)$ and regularization term $E_W(w)$

Regularized Least Squares

- Regularized least squares is $E(\mathrm{w}) = E_D(\mathrm{w}) + \lambda E_W(\mathrm{w})$
- Simple form of regularization term is $E_W(\mathrm{w}) = \frac{1}{2}\mathrm{w}^T\mathrm{w}$
- Hence total error is given by $E(\mathrm{w}) = \frac{1}{2}\sum_{n=1}^{N}\{\, t_n - \mathrm{w}^T\phi(x_n)\,\}^2 + \frac{\lambda}{2}\mathrm{w}^T\mathrm{w}$
- This regularization function is called *weight decay*
  - Weight values decay towards zero unless supported by training data examples

Regularized Least Squares

- Error function with weight decay (quadratic) regularizer is

$$\boxed{E(\mathrm{w}) = \frac{1}{2}\sum_{n=1}^{N}\{\ t_n - \mathrm{w}^T\phi(x_n)\ \}^2 + \frac{\lambda}{2}\mathrm{w}^T\mathrm{w}}$$

Regularized Least Squares

- Error function with weight decay (quadratic) regularizer is

$$\boxed{E(\mathrm{w}) = \frac{1}{2}\sum_{n=1}^{N}\{\, t_n - \mathrm{w}^T\phi(x_n)\,\}^2 + \frac{\lambda}{2}\mathrm{w}^T\mathrm{w}}$$

- Its exact minimizer can be found in closed form and is given by

$$\boxed{\mathrm{w} = (\lambda\mathrm{I} + \boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\mathrm{t}}$$

Regularized Least Squares

- Error function with weight decay (quadratic) regularizer is

$$E(\mathrm{w}) = \frac{1}{2}\sum_{n=1}^{N}\{ t_n - \mathrm{w}^T\phi(x_n) \}^2 + \frac{\lambda}{2}\mathrm{w}^T\mathrm{w}$$

- Its exact minimizer can be found in closed form and is given by

$$\mathrm{w} = (\lambda \mathrm{I} + \Phi^T\Phi)^{-1}\Phi^T\mathrm{t}$$

Simple extension of ordinary least squares solution

$$\mathrm{w}_{ML} = (\Phi^T\Phi)^{-1}\Phi^T\mathrm{t}$$

A General Regularizer

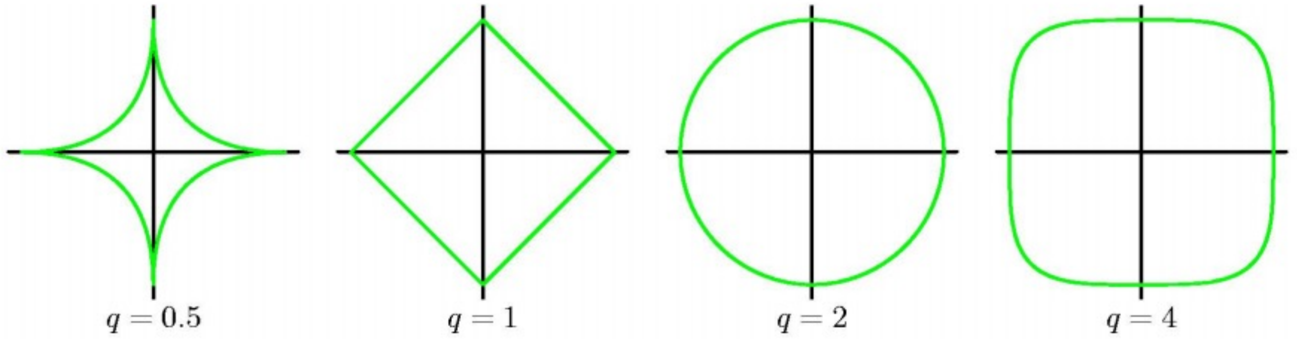
$$\frac{1}{2}\sum_{n=1}^{N}\{\, t_n - \mathbf{w}^T\phi(\mathbf{x}_n)\,\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|\, w_j\,|^q$$

- q=2 corresponds to the quadratic regularizer
- q=1 is known as lasso
  - Lasso has the property that if λ is sufficiently large some of the coefficients w_j are driven to zero
    - Leads to a sparse model in which the corresponding basis functions play no role

A General Regularizer

$$\frac{1}{2}\sum_{n=1}^{N}\{\ t_n - \mathrm{w}^T\phi(\mathrm{x}_n)\ \}^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|\ w_j\ |^q$$

$\sqrt{w_1} + \sqrt{w_2} = const$ $w_1 + w_2 = const$ $w_1^{\ 2} + w_2^{\ 2} = const$ $w_1^{\ 4} + w_2^{\ 4} = const$



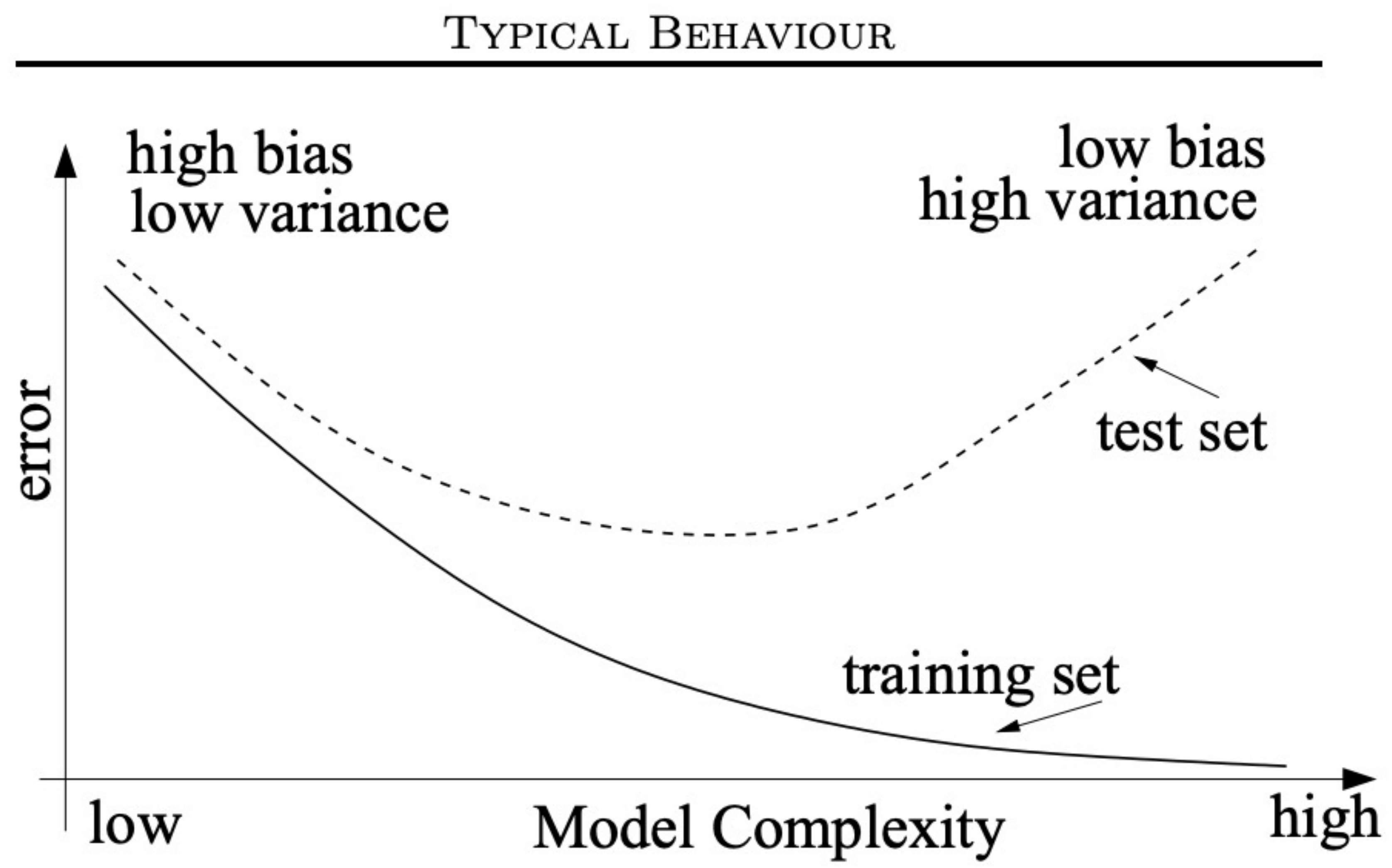
Summary

- Regularization allows
  - complex models to be trained on small data sets
  - without severe over-fitting
- It limits model complexity
  - i.e., how many basis functions to use?
- Problem of limiting complexity is shifted to
  - one of determining suitable value of regularization coefficient

Bias-Variance Tradeoff

Model Complexity and Overfitting

- We looked at linear regression where form of basis functions φ and number of functions M are fixed
- Using maximum likelihood (equivalently least squares) leads to severe overfitting if complex models are trained with limited data
  - However limiting M to avoid overfitting has side effect of not capturing important trends in the data
- Regularization can control overfitting for models with many parameters
  - But seeking to minimize wrt both w and λ leads to unregularized solution with λ=0


Typical Behaviour
high bias
low variance
low bias
high variance
error
test set
training set
low
Model Complexity
high

Some Definitions

- The *generalization* of a machine learning is the performance (classification, regression, density estimation) on test data, not used for training, but drawn from the same (joint) distribution as the training data. Often, our real goal is to get good generalization.
- When our model is too complex for the amount of training data we have, it memorizes parts of the noise as well as learning the true problem structure. This is called *overfitting or model variance*.
- When our model is not complex enough, it cannot capture the structure in our data, no matter how much data we give it. This is called *underfitting or model bias*.
- In statistics, there is an incomprehensible obsession with the minimum variance unbiased estimator. But if all we are about is generalization then we should be happy to introduce a little bit of bias if it reduces the variance a lot.

What is Bias and Variance?

- The ***bias error*** is an error from ***erroneous assumptions*** in the learning algorithm. ***High bias*** can cause an algorithm to miss the relevant relations between features and target outputs (***underfitting***).
- The ***variance*** is an error from ***sensitivity to small fluctuations*** in the training set. ***High variance*** can cause an algorithm to ***model*** the ***random noise*** in the training data, rather than the intended outputs (***overfitting***).

What are we trying to do?

- What basic problems are we trying to avoid with model selection?
- *Overfitting*: if we chose a model that is too complex, it will overfit to the noise in our training set. Another way of saying this is that the machine we end up with is very sensitive to the particular training sample we use. The model has a lot of variance across training samples of a fixed size.
- *Underfitting*: if we chose a model that is not complex enough, it cannot fit the true structure, and so no matter what training sample we use there is some error between the true function and our model approximation. The model has a lot of bias.
- Intuitively, we need the right balance. How can we formalize this?

Example

- Let us consider a supervised learning setup (scalar for now), with random noise (uncorrelated to inputs/outputs) and squared error:

$$y = g(x) + \text{noise}$$

$$y' = f(x)$$

$$\text{error} = (y - y')^2$$

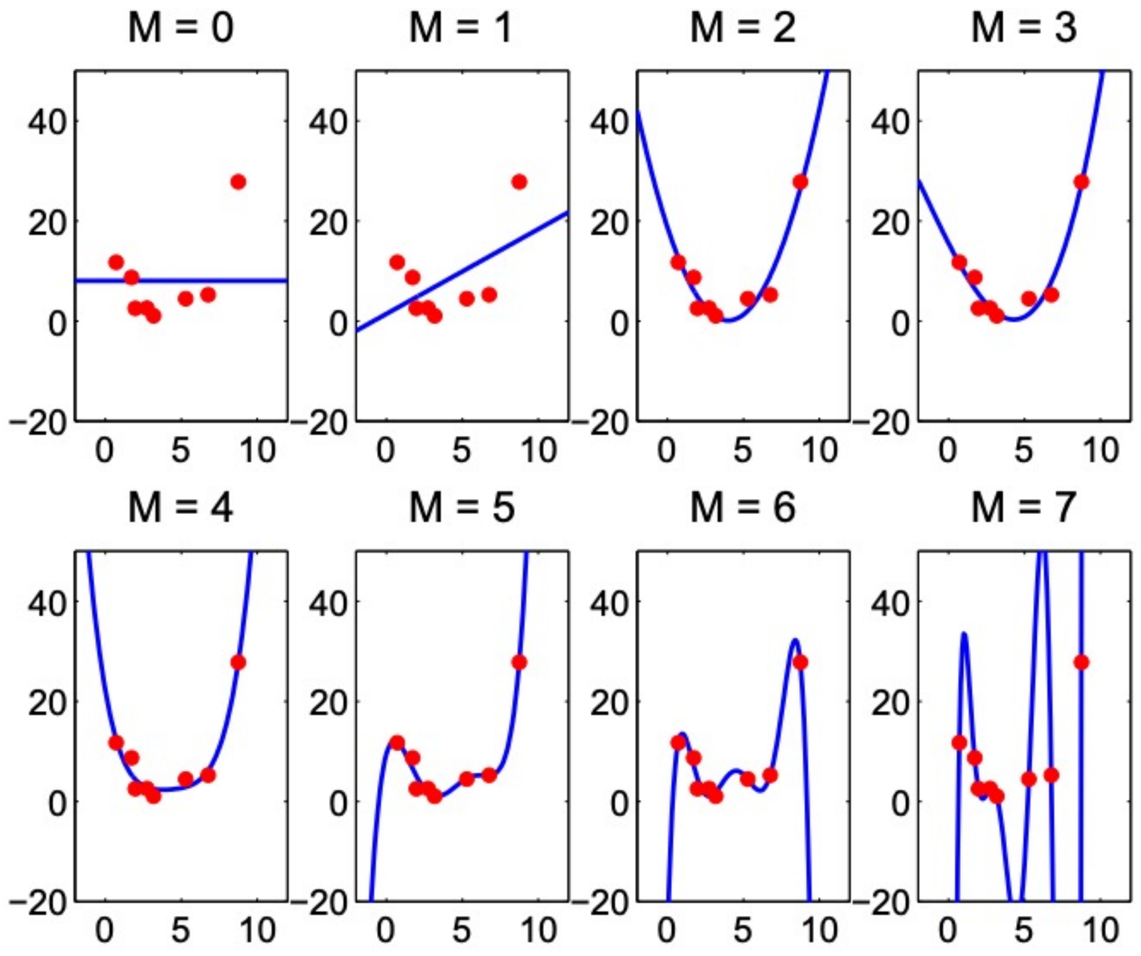
- Consider the expected error at a single test point x0, averaged over all possible training sets of size N, drawn from the joint distribution over inputs and outputs p(x, y) = p(x)p(y|x).

$$\begin{aligned} e(x_0) &= \langle (y_0 - \hat{y}_0)^2 \rangle \\ &= \langle (g(x_0) + \epsilon_0 - \hat{y}_0)^2 \rangle \\ &= \langle \epsilon_0^2 \rangle + (\langle f(x_0) \rangle - g(x_0))^2 + \langle f(x_0) - \langle f(x_0) \rangle \rangle^2 \\ &= \sigma^2 + (\text{mean}[f(x_0)] - g(x_0))^2 + \text{var}[f(x_0)] \\ &= \text{Unavoidable Error} + \text{Bias}^2 + \text{Variance} \end{aligned}$$

Model Selection & Performance Estimation

- Model Selection: out of a set of models (or continuum of model complexity), choose the model which will perform the best on future test data.
- Model Assessment: for the selected model, estimate its generalization error on new data.
- If we have lots of data, these two problems can be solved by dividing our data into 3 parts:
  - Training Data – used to train each model
  - Validation Data – used to measure performance of each trained model in order to select the best model
  - Assessment Data – used only once, on the final selected model, to estimate performance on future test data
- Typical split is 60% training, 20% validation, 20% assessment. So that's it, are we done?

Example: polynomial fitting



Approximations when Data is Limited

- Often, we don't have enough data to make 3 separate and reasonably sized training, validation and assessment sets.
- If we don't have very much data, we can try to *approximate* the results of validation and assessment.
- Two basic approaches for finite datasets:
  - Analytic methods: derive algebraic espressions which try to approximate the test error, e.g. using a complexity penalty which scales as the ratio between the number of parameters in the model and the number of training cases.
    Examples: BIC, AIC, MDL, VC-dimension.
  - Sample-recycling methods: try to estimate the test error computationally, using the same data that we trained on.
    Examples: jackknife,cross-validation, bootstrap.

Regularization and Capacity Control

- How can we improve generalization? Reduce either bias or variance!
- One obvious way: use more training data, and commensurately more complex models. If we scale up model complexity slowly enough, using more data reduces *both* bias and variance.
- But what if we can't get more data?
  Our goal should be to reduce variance (by using simpler models) while not increasing our bias too much (by not using *too* simple a model). We should not force ourselves to use unbiased (Bias=0) models, because we only really care about the sum Bias^2+Variance.
- We need a knob to control this tradeoff (e.g. by discretely constraining model *structure* or by continuously *regularizing* model complexity or smoothness) and a way to set the knob (i.e. decide on the right tradeoff balance).

Another Regularizer: Parameter Sharing

- Another way to control model complexity is to *tie together* or *share* various parameters. This allows us to have a complete model structure but not have to estimate a huge number of free parameters.
- This is used in mixtures of factor analyzers, to jointly estimate the sensor noises, in mixtures of Gaussians to jointly estimate cluster covariances (e.g. Fisher's discriminant is a class-conditional Gaussian model with shared covariances), in vision neural networks to learn translation-invariant receptive fields, etc.

More Regularization: Method with Local Support

- Yet another way to control model complexity is to restrict the amount of training data that can be used to predict the output on any new test case.
- Each test case prediction is only allowed to use a small fraction of the training data, typically the training points whose inputs are close to the input of the test case.
- This is known as a *locally weighted* method, e.g. nearest neighbour classification, Parzen density estimation, locally weighted regression.
- Local methods are related to "semi-parametric" models, which try to use the reservoir of training data to store most of the bits of their capacity, and only have a few "metaparameters" which control how that reservoir is used at test time.
  Examples: K-NN classifier, locally-weighted regression, parzen window density estimators

Regularization by adding a penalty term

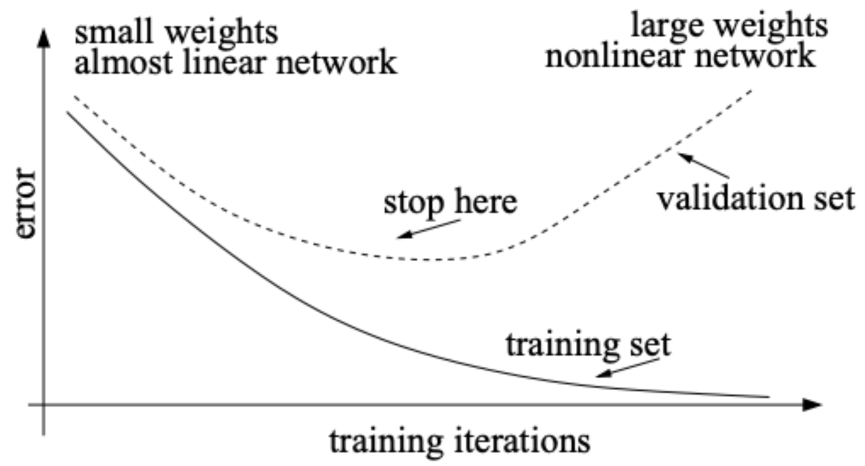
- Instead of discrete complexity controls, it is often useful to have a continuous range of complexity, set by one or more real valued "fudge-factors" or "hyperparameters".
- The most common way to achieve this is to add a "penalty term" to the cost function (error, log likelihood, etc) which measures in a quantitative and continuous way how complex/simple our model is:

$$\text{cost}(\theta) = \text{error}(\text{data}, \theta) + \lambda \text{penalty}(\theta)$$

- We can then weight this penalty term relative to the original error (or likelihood) and minimize the resulting penalized cost.
- The larger the penalty weight λ, the simpler our model will be.
- How can we set λ? If we have lots of data, we can use the performance on a held out set of validation examples to determine the correct value of the penalty weight.

Example Penalty: Early Stopping

- Another approach to regularization in models whose complexity grows with training time is to stop training early.
- This works quite well in neural networks, since small weights mean that the network is mostly linear (low complexity) and it takes a while for the weights to get bigger, giving nonlinear networks (high complexity). Essentially a penalty equal to # training iterations.
- A validation set can be used to detect stopping point.



Example Penalty: Weight Decay

- The most common regularization is the ridge regression penalty (weight decay) which discourages large parameter values in generalized linear models:

$$\text{cost}(\theta) = \text{error}(\text{data}, \theta) + \lambda \sum_k \theta_k^2$$

- This says: “don’t use big weights unless they really help to reduce your error a lot”. Otherwise, there is nothing to stop the model from using enormous positive and negative weights to gain a tiny benefit in error.
- Remember: on a finite training set, there will always be some tiny, accidental correlation between the noise in the inputs and the target values.

Practical Solutions

- Several simple ways to good generalization in practice.
- Use model classes with flexible control over their complexity. (e.g. ridge regression, mixture models)
- Employ regularization (capacity control) and (cross) validation,to match model complexity with the amount of data available.
- Build in as much reliable prior knowledge as possible, so algorithms don't have to waste data learning things we already know.
- Use cross-validation/bootstrap to make efficient use of limited data.
- Use subsampling or sparse methods to speed up algorithms on huge training sets, and keep them fast and small at test time.

Potential Pitfalls

- Several things can cause us trouble when we are trying to get good generalization from a learning algorithm:
  - we might not have enough training data to learn target concept
  - our testing might not *really* be from the same distribution as our training data
  - our model might not be complex enough, so it underfits
  - our model might be too complex, so it overfits
  - we have too much training data to run the algorithm in a reasonable amount of time or memory
- Sounds hopeless!
  What can we do?

Bias-Variance vs Bayesian

- Bias-Variance decomposition provides insight into model complexity issue
- Limited practical value since it is based on ensembles of data sets
  - In practice there is only a single observed data set
  - If there are many training samples then combine them
    - which would reduce over-fitting for a given model complexity
- Bayesian approach gives useful insights into over-fitting and is also practical

Bayesian Model Comparison

Remember: Curve Fitting

- Regression using basis functions and MSE:

$$y(\mathrm{x},\mathrm{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathrm{x}) = \mathrm{w}^T \phi(\mathrm{x})$$

$$E_D(\mathrm{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{ t_n - \mathrm{w}^T \phi(\mathrm{x}_n) \right\}^2$$

- Need an M that gives best generalization
  - M = No. of free parameters in model or model complexity
- With regularized least squares

$$E(\mathrm{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{ t_n - \mathrm{w}^T \phi(x_n) \right\}^2 + \frac{\lambda}{2}\mathrm{w}^T\mathrm{w}$$

- λ also controls model complexity (and hence degree of over-fitting)

Which model to use?

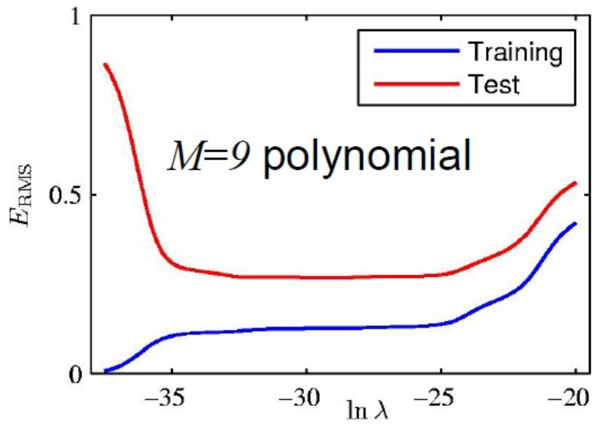
- There can be many models that you can use!
- Depending on basis functions, number of functions, etc…

Which model to use?

- There can be many models that you can use!
- Depending on basis functions, number of functions, etc…

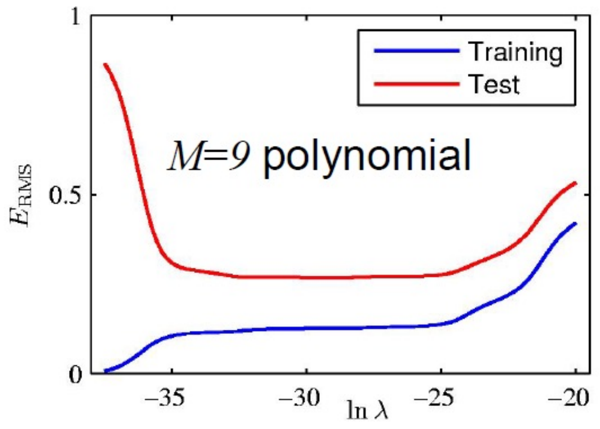
Choosing a model using data

- λ controls model complexity (similar to choice of M)
- Frequentist Approach:
  - Training set
    - To determine coefficients w for different values of (M or λ)
  - Validation set (holdout)
    - to optimize model complexity (M or λ)



Choosing a model using data

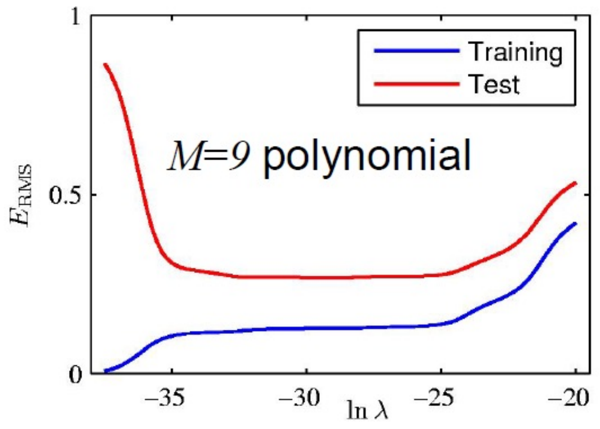
- λ controls model complexity (similar to choice of M)
- Frequentist Approach:
  - Training set
    - To determine coefficients w for different values of (M or λ)
  - Validation set (holdout)
    - to optimize model complexity (M or λ)



What value of λ minimizes error?
λ = -38 : low error on training, high error on testing set
λ = -30 is best for testing set

Choosing a model using data

- λ controls model complexity (similar to choice of M)
- Frequentist Approach:
  - Training set
    - To determine coefficients w for different values of (M or λ)
  - Validation set (holdout)
    - to optimize model complexity (M or λ)



What value of λ minimizes error?
λ = -38 : low error on training, high error on testing set
λ = -30 is best for testing set

$$E_{RMS} = \sqrt{2E(\mathrm{w}^*)/N}$$

Division by N allows different data sizes to be compared since E is a sum over N Sqrt (of squared error) measures on same scale as t

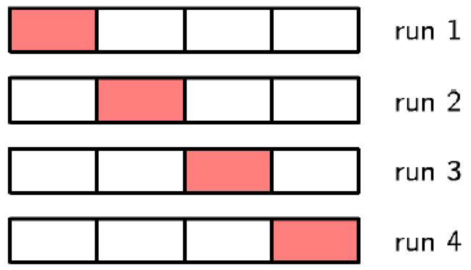
Use the Validation Set!

- Performance on training set is not a good indicator of predictive performance
- If there is plenty of data,
  - use some of the data to train a range of models Or a given model with a range of values for its parameters
  - Compare them on an independent set, called validation set
  - Select one having best predictive performance
- If data set is small then some over-fitting can occur and it is necessary to keep aside a test set

K-fold Cross Validation

- Supply of data is limited
- All available data is partitioned into K groups
- K-1 groups are used to train and evaluated on remaining group
- Repeat for all K choices of held-out group
- Performance scores from K runs are averaged

K=4



If K=N, then it is called the leave-one-out cross validation

Disadvantage of Cross-Validation

- No. of training runs is increased by factor of K
- Problematic if training itself is expensive
- Different data sets can yield different complexity parameters for a single model
  - E.g., for given M several values of λ
- Combinations of parameters is exponential
- Need a better approach
  - Ideally one that depends only on a single run with training data and should allow multiple hyperparameters and models to be compared in a single run

Potential Problems with Cross-Validation

- CV is awesome and it can be used on clustering, density estimation, classification, regression, etc.
- But intensive use of cross-validation can overfit, if you explore too many models, by finding a model that accidentally predicts the whole training set well (and thus every leave-one-out sample well).
- CV can also be very time consuming if done naively.
- Often there are efficient tricks for computing all possible leave-one-out cross validation folds, which can save you a lot of work over brute-force retraining on all N possible LOO datasets.
- For example, in linear regression, the term $(\sum_{n \neq \ell} \mathbf{x}_n \mathbf{x}_n^\top)^{-1}$ which leaves out datapoint ℓ can be computed using the matrix inversion lemma: $(\sum_n \mathbf{x}_n \mathbf{x}_n^\top - \mathbf{x}_\ell \mathbf{x}_\ell^\top)^{-1}$.
- This is also true of the Generalized Cross Validation (GCV) estimate of Golub and Wahaba. (see extra readings)

Model Averaging

- One last way to reduce variance, while not affecting bias too severely, is to average together the predictions of a bunch of different models.
- These models must be different in some way, either because they were trained on different subsets of the data, or with different regularization parameters, different local optima, or something.
- When we average them together, we would like to weight more strongly the models we believe are fitting the data better.
- Such systems are often called *committee machines*.
- Really, this is just a weak form of Bayesian learning.
  MAP = estimate of mode of posterior over models
  Bagging (next class) = estimate of mean of posterior over models
  BIC/AIC = estimates of correct predictive distribution

What do we need?

What do we need?

- Need to find a performance measure that depends only on the training data and which does not suffer from bias due to over-fitting
- Historically various information criteria have been proposed that attempt to correct the for the bias of maximum likelihood by the addition of a penalty term to compensate for the overfitting of more complex models

Akaike Information Criterion (AIC)

- AIC chooses model that maximizes $\ln p(D|\boldsymbol{w}_{\mathrm{ML}}) - M$
  - First term is best-fit log-likelihood for given model
  - M is no of adjustable parameters in model
- Penalty term M is added for over-fitting using many parameters
- Bayesian Information Criterion (BIC) is a variant of this quantity
- Disadvantages: • need runs with each model, prefers overly simple models

Bayesian Perspective

- Avoids over-fitting
  - By marginalizing over model parameters
    - sum over model parameters instead of point estimates
- Models compared directly over training data
  - No need for validation set
  - Allows use of all available data in training set
  - Avoids multiple training runs for each model associated with cross-validation
  - Allows multiple complexity parameters to be simultaneously determined during training
    - Relevance vector m/c is Bayesian model with one complexity parameter for every data point

How do you choose a model?

How do you choose a model?

- Based on its evidence!
- A model is a probability distribution over data D – E.g., a polynomial model is a distribution over target values *t* when input x is known, i.e., p(t|x,D)
- Uncertainty in model itself can be represented by probabilities
- Compare a set of models Mi , i=1,..L
- Given training set, wish to evaluate posterior

$$p(M_i|D) \quad \alpha \quad \underbrace{p(M_i)}_{\text{Prior expresses preference for different models}} \; \underbrace{p(D|M_i)}_{\text{Model Evidence (or Marginal Likelihood)}}$$

We assume equal priors

Model Evidence

- $p(D|M_i)$ is preference shown by data for model
- Called Model evidence or *marginal likelihood*
  - because parameters have been *marginalized*

Bayes Factor

- From model evidence p(D/Mi) for each model:
  - The ratio $\dfrac{p(D \mid M_i)}{p(D \mid M_j)}$ is called the Bayes Factor
    - Shows the preference for Model *I* over Model *j*

Predictive Distribution

- Given $p(M_i|D)$ the predictive distribution is given by

$$p(t \mid \boldsymbol{x}, D) = \sum_{i=1}^{L} \underbrace{p(t \mid \boldsymbol{x}, M_i, D)}_{\text{Prediction under model}} p(M_i \mid D)$$

  - This is called a *mixture:*
    - predictions under different models are weighted by posterior probabilities of models
- Instead of all models Mi , approximate with single most probable model
  - Known as *Model Selection*

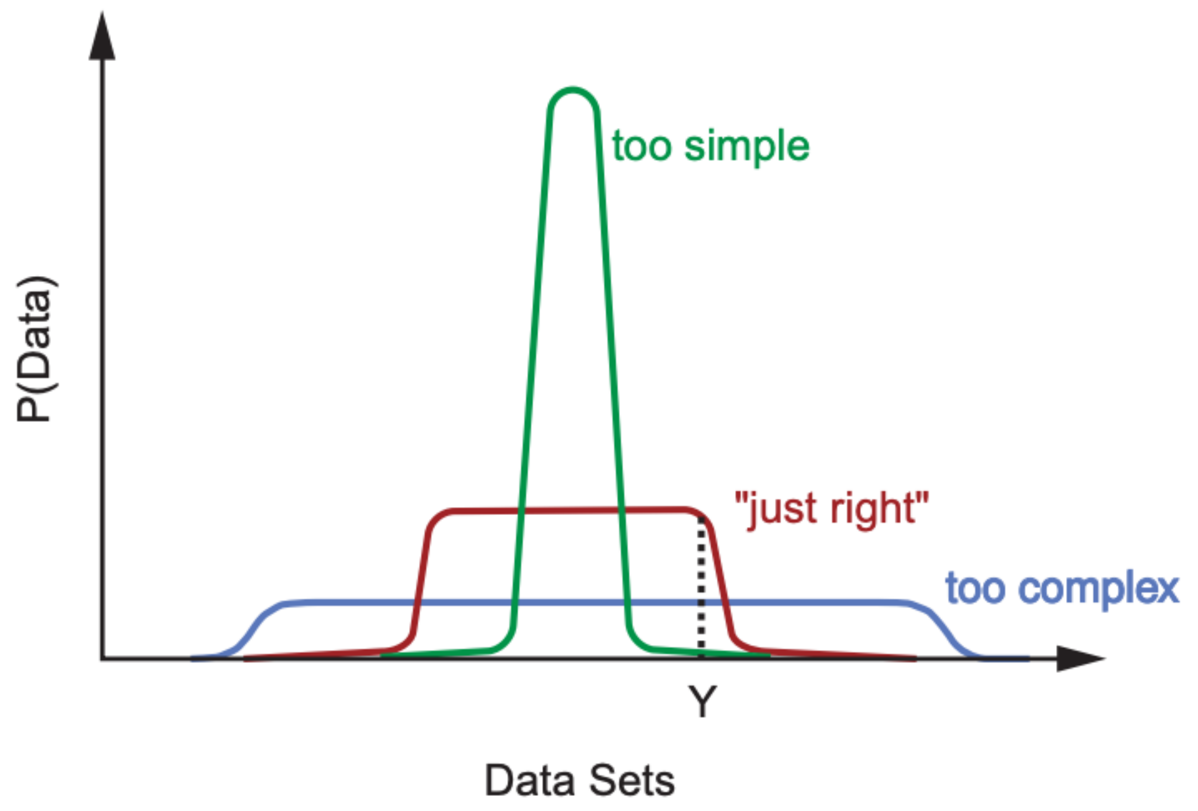
Bayes factor always favors correct model

- We are assuming the true model is contained among the Mi
- If this is so, Bayes model comparison will favor the correct model
- If M1 is the true model, and we average over the distribution of data sets, Bayes factor has the form

$$\int p(D \mid M_1) \ln \frac{p(D \mid M_1)}{p(D \mid M_2)} dD$$

- This is K-L Divergence which is always positive
  - Thus Bayes factor favors the correct model

Ockham's Razor



We want to use the simplest model which explains the data well.
[A now famous figure, first introduced by Mackay.]

No Free Lunch

- David Wolpert and others have proven a series of theorems, known as the "no free lunch" theorems which, roughly speaking, say that *unless you make some assumptions* about the nature of the functions or densities you are modeling, no one learning algorithm can *a priori* be expected to do better than any other algorithm.
- In particular, this lack of clear advantage includes any algorithm and any meta-learning procedure applied to that algorithm. In fact, "anti-cross-validation" (i.e. picking the regularization parameters that give the *worst* performance on the CV samples) is a priori just as likely to do well as cross-validation. Without assumptions, random guessing is no worse than any other algorithm.
- So capacity control, regularlization, validation tricks and meta-learning (next class) cannot *always* be successful.

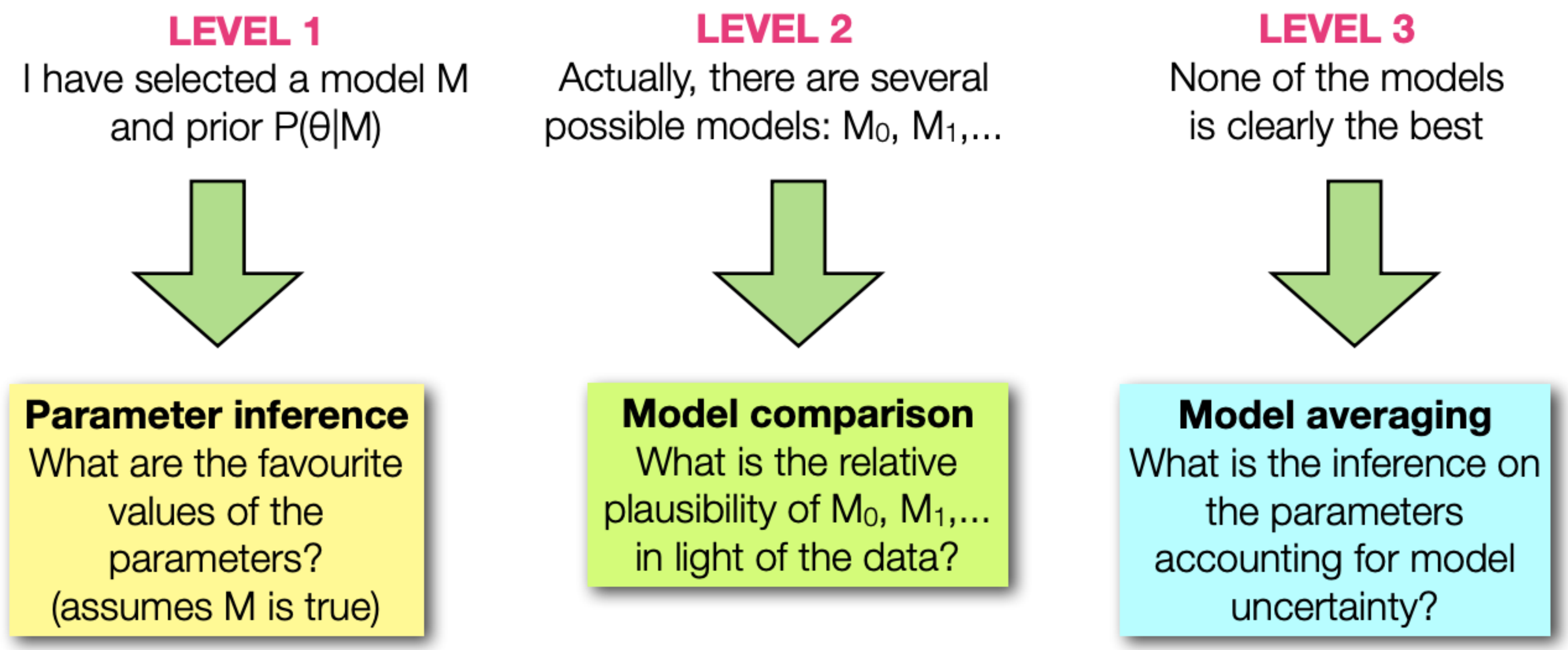
Generalization Error vs. Learning Error

- A key issue here is the difference between test error on a test set drawn from the same distribution as the training data (may contain duplicates) and *out of sample* test error.
- Remember back to the first class: learning binary functions.
  No assumptions == no generalization on out of sample cases.
- The only way to learn is to wait until you have seen the whole world and memorize it.
- Luckily, we *can* make some progress in real life.
- Why? Because the assumptions we make about function classes are often partly true.

Summary

- Avoids problem of over-fitting
- Allows models to be compared using training data alone
- However needs to make assumptions about form of model
  - Model evidence can be sensitive to many aspects of prior such as behavior of tails
- In practice necessary to keep aside independent test data to evaluate performance

3 levels of inference



LEVEL 1
I have selected a model M and prior P(θ|M)

Parameter inference
What are the favourite values of the parameters? (assumes M is true)

LEVEL 2
Actually, there are several possible models: M_0, M_1,...

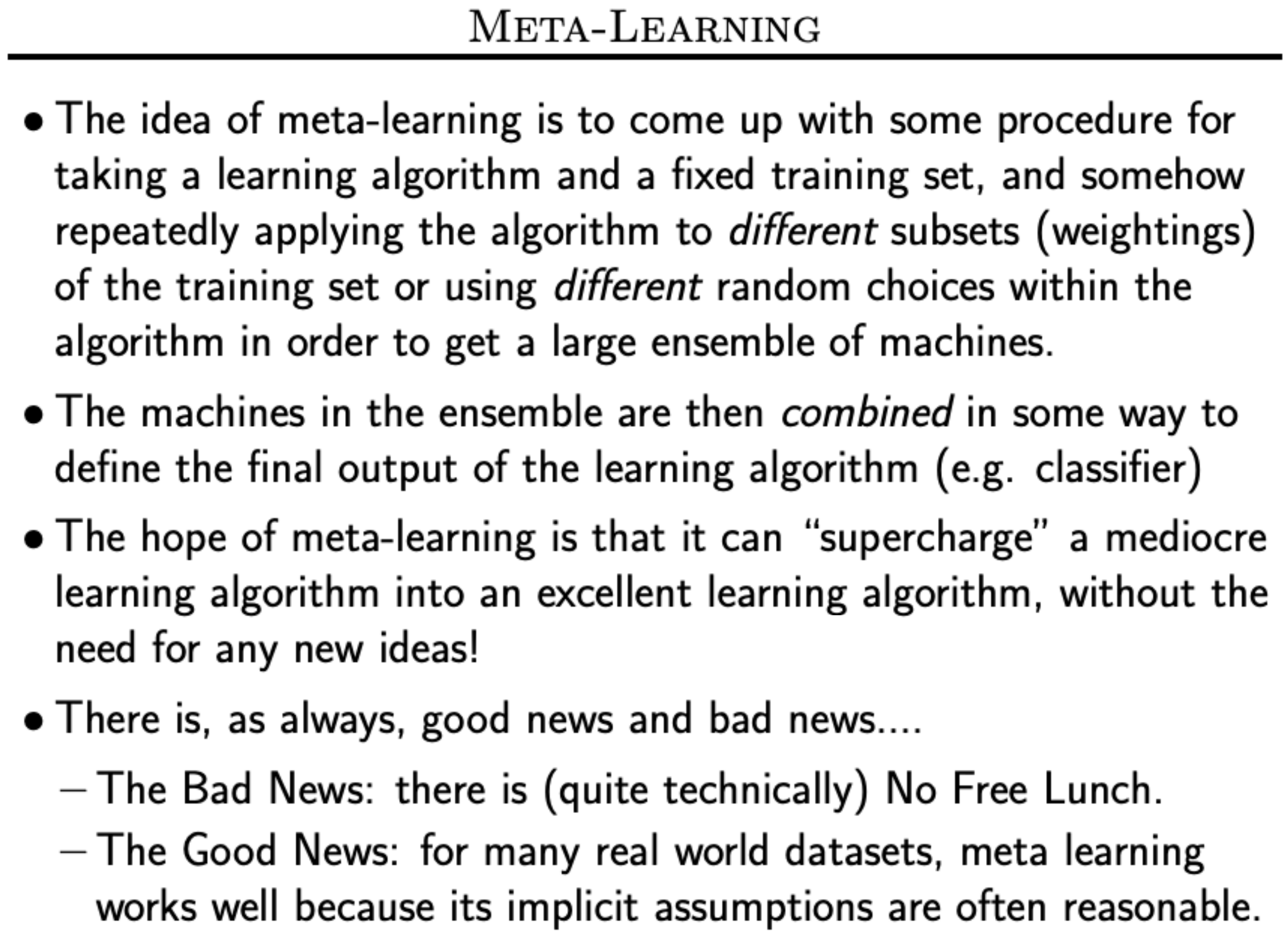
Model comparison
What is the relative plausibility of M_0, M_1,... in light of the data?

LEVEL 3
None of the models is clearly the best

Model averaging
What is the inference on the parameters accounting for model uncertainty?

Image Credits: Roberto Trotta

What is SOTA Research?



META-LEARNING

- The idea of meta-learning is to come up with some procedure for taking a learning algorithm and a fixed training set, and somehow repeatedly applying the algorithm to *different* subsets (weightings) of the training set or using *different* random choices within the algorithm in order to get a large ensemble of machines.
- The machines in the ensemble are then *combined* in some way to define the final output of the learning algorithm (e.g. classifier)
- The hope of meta-learning is that it can "supercharge" a mediocre learning algorithm into an excellent learning algorithm, without the need for any new ideas!
- There is, as always, good news and bad news....
  - The Bad News: there is (quite technically) No Free Lunch.
  - The Good News: for many real world datasets, meta learning works well because its implicit assumptions are often reasonable.

Image Credits: Roberto Trotta