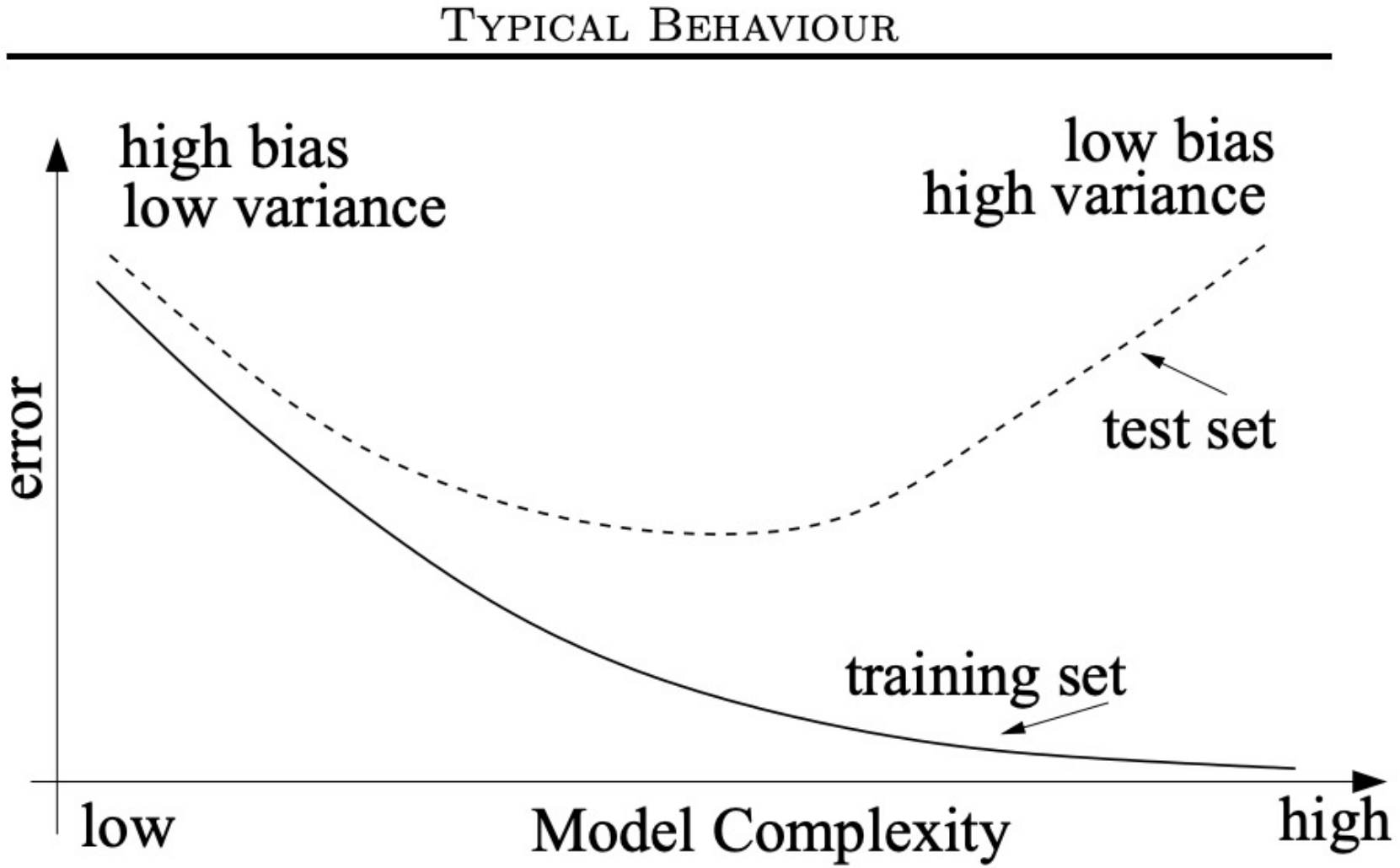
COMP [56]630– Machine Learning

Lecture 6 – Bias-Variance Tradeoff, Model Selection, Logistic Regression Pt. 1



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***).

Mitigation Techniques

- Model selection
  - Use model assessment to pick the best model
  - Use validation data if available. Otherwise, K-Fold Cross validation
- Add more training data
- Regularization
- Parameter sharing
- Locally weighted or non-parametric models to use less data
- Early stopping

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

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

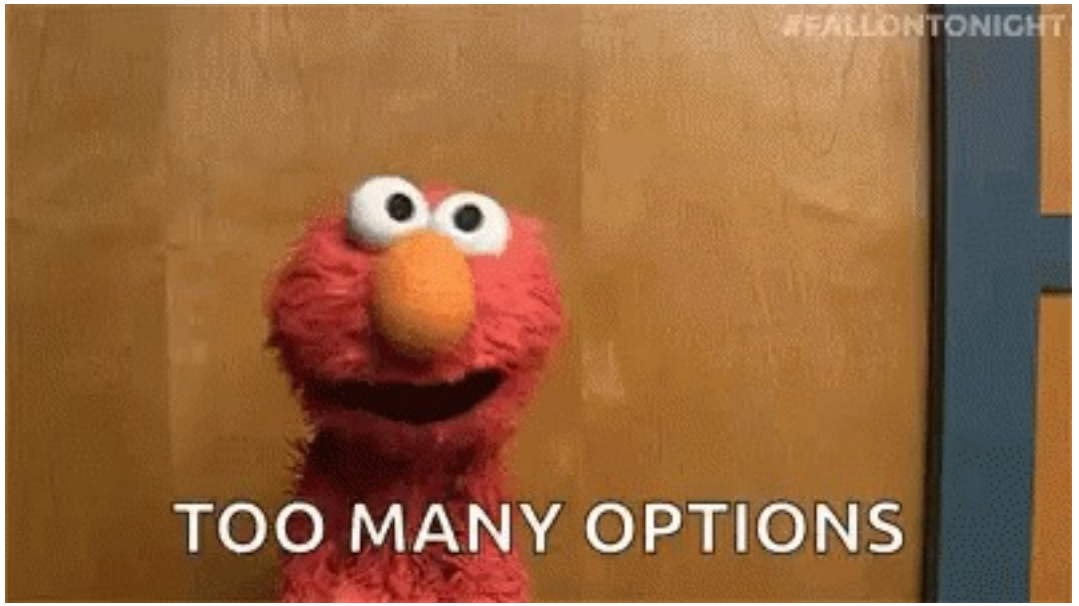
- λ also controls model complexity (and hence degree of over-fitting)

Which model to use?

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

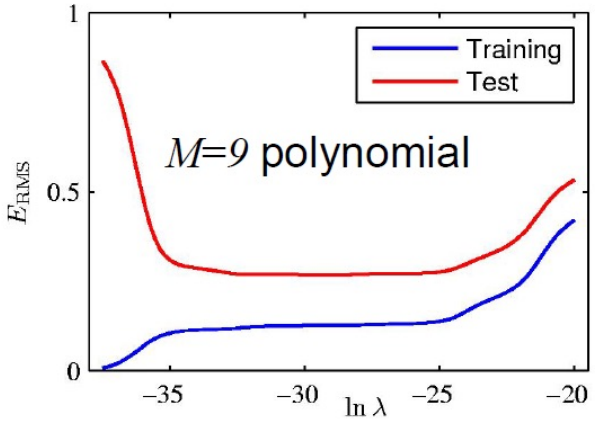
Which model to use?

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



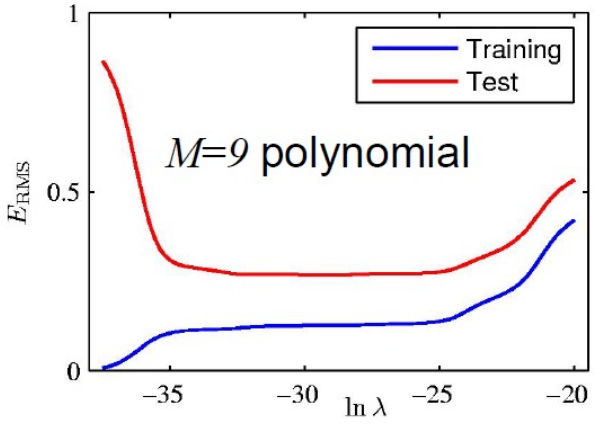
Choosing a model using data

- 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

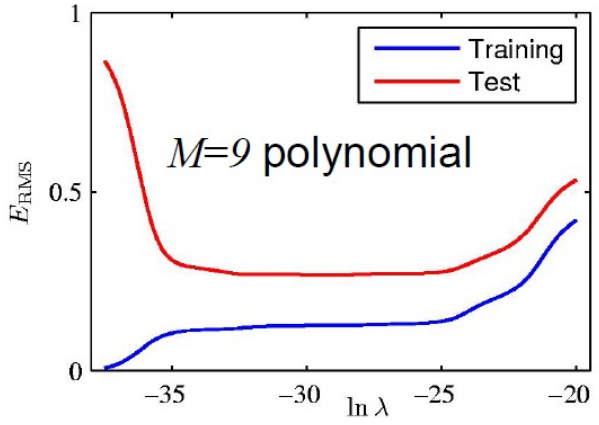
- λ controls model complexity (similar to choice of M)
- Frequentist Approach:
  - Training set
    - To determine coefficients w for different values of (M or λ)
  - Validation set (holdout)
    - to optimize model complexity (M or λ)



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

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

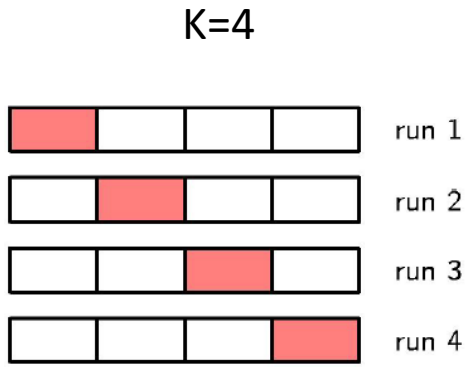
Division by N allows different data sizes to be compared since E is a sum over N Sqrt (of squared error) measures on same scale as t

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



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

$$p(M_i|D) \quad \alpha \quad p(M_i)\ p(D|M_i)$$

Prior expresses preference for different models

We assume equal priors

Model Evidence (or Marginal Likelihood)

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 $\frac{p(D \mid M_i)}{p(D \mid M_j)}$ 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

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

Prediction under model

  - 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*

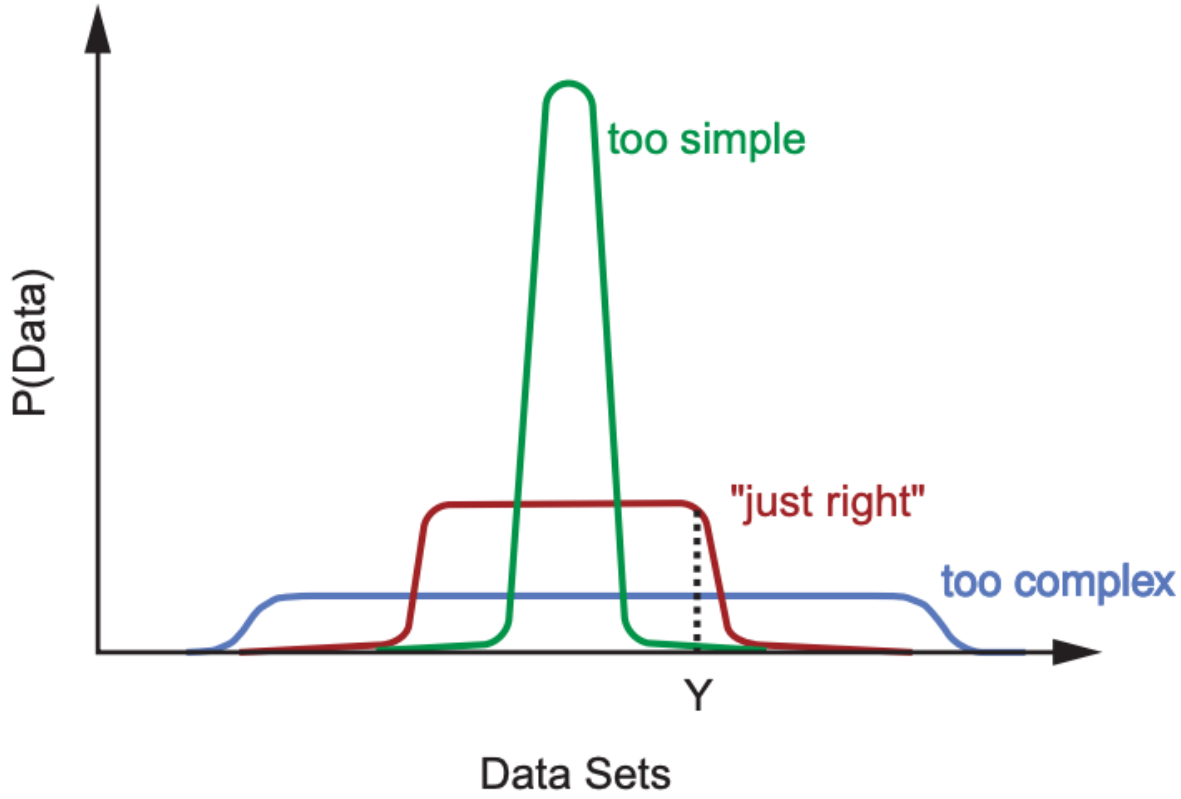
Bayes factor always favors correct model

- 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 $P(\theta|M)$

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.

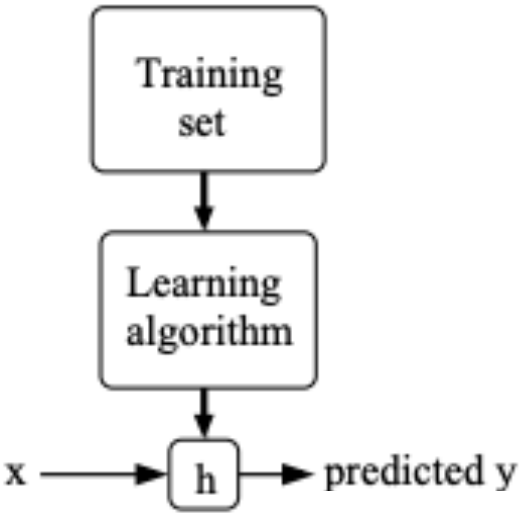
Image Credits: Roberto Trotta

Logistic Regression

On to Classification!

The basics

- Input: a set of inputs X = {x_1, x_2, … x_n}, also called ***features***
- Output: a set of expected outputs or ***targets*** Y = {y_1, y_2, … y_n}
- Goal: to learn a function *h : X → Y* such that the function h(x_i) is a good predictor of the corresponding value y_i
  - h(x) is called the ***hypothesis***
- If the target is continuous the problem setting is called ***regression.***
- If the target is discrete or categorical, the problem is called ***classification.***



Example

- Suppose we have a dataset giving the living areas and prices of 47 houses from Stillwater, OK

Living area (ft^2)	# bedrooms	Price (1000$s)	House Type
1643	4	256	Condo
1356	3	202	Apartment
1678	3	287	House
…	…	…	
3000	4	400	House

Logistic Regression

- Goal: formulate a hypothesis function h(x) which will model the 3-d input feature (size, # bedrooms, price) and produce the expected target value (type of home i.e. condo, apartment, house, etc.).
- Let us consider a 2-class problem i.e. a binary classifier that says whether the given home is a house or not a house.
  - Represent as 0 and 1
  - 0 → negative class
  - 1 → positive class

  Discrete outputs!
- Given $x^{(i)}$, the corresponding $y^{(i)}$ is also called the label for the training example.

Logistic Regression

- We could approach the classification problem ignoring the fact that y is discrete-valued, and use our old linear regression algorithm to try to predict y given some features x

Logistic Regression

- We could approach the classification problem ignoring the fact that y is discrete-valued, and use our old linear regression algorithm to try to predict y given some features x
  - Does not work all the time.
  - Does not make sense to have predictions greater than 1 or less then 0.
    - Since the categories are 1 and 0 and $y \in \{0, 1\}$.

Logistic Regression

- We could approach the classification problem ignoring the fact that y is discrete-valued, and use our old linear regression algorithm to try to predict y given some features x
  - Does not work all the time.
  - Does not make sense to have predictions greater than 1 or less then 0.
    - Since the categories are 1 and 0 and y ∈ {0, 1}.
- New hypothesis function:

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

Logistic Regression

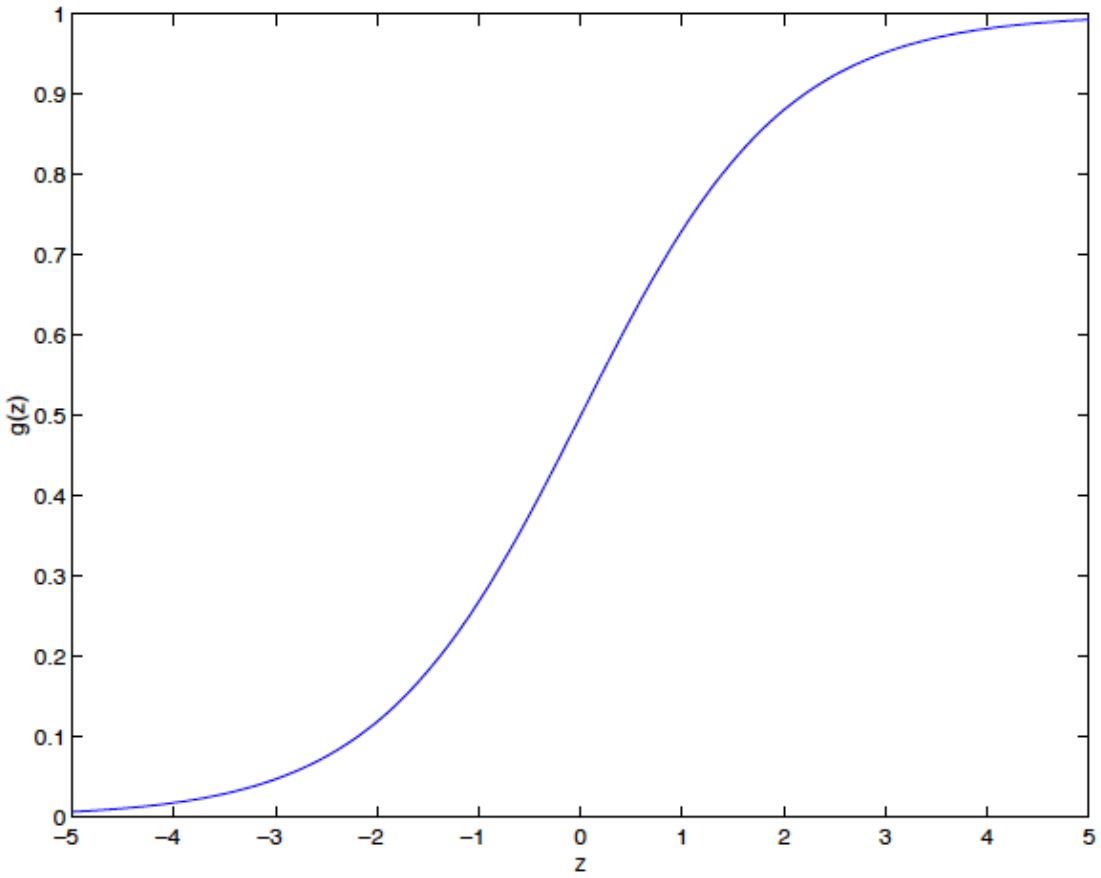
- We could approach the classification problem ignoring the fact that y is discrete-valued, and use our old linear regression algorithm to try to predict y given some features x
  - Does not work all the time.
  - Does not make sense to have predictions greater than 1 or less then 0.
    - Since the categories are 1 and 0 and y ∈ {0, 1}.
- New hypothesis function:

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

- where $g(z) = \frac{1}{1 + e^{-z}}$ ← **Logistic Function**

$$\theta^T x = \theta_0 + \sum_{j=1}^{n} \theta_j x_j$$

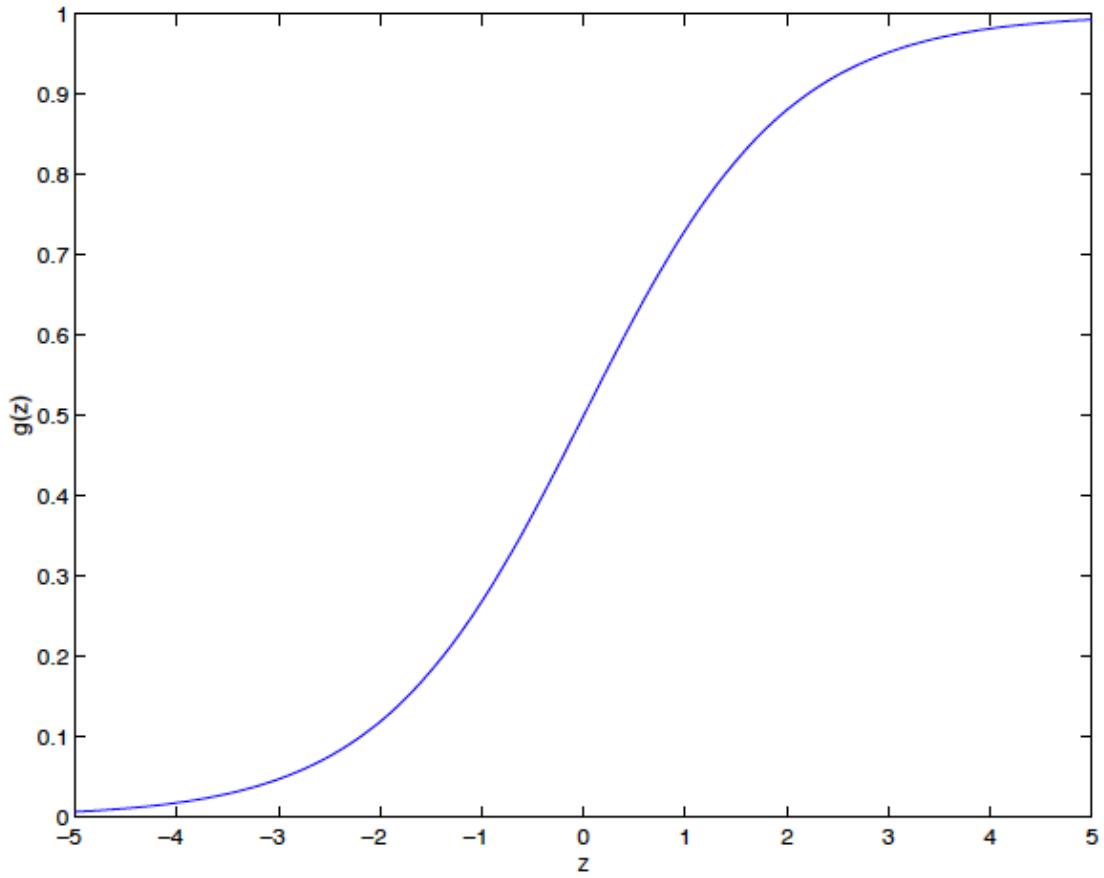
Logistic Function



- g(z) tends towards 1 as $z \to \infty$
- g(z) tends towards 0 as $z \to -\infty$.

- What does this tell us?

Logistic Function



- g(z) tends towards 1 as $z \to \infty$
- g(z) tends towards 0 as $z \to -\infty$.

- What does this tell us?
- **g(z), and hence also h(x), is always bounded between 0 and 1.**

How do we get θ?

- Gradient Descent!
- What do we need for gradient descent?
  - An objective function J(θ)
  - A Learning rate α
  - An initial "guess" for θ called θ_j
- Then, update θ_j until convergence as follows

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Defining the Objective Function J(θ)

- Let us say that

$$\begin{aligned} P(y = 1 \mid x; \theta) &= h_\theta(x) \\ P(y = 0 \mid x; \theta) &= 1 - h_\theta(x) \end{aligned}$$

- Or, more concisely:

$$p(y \mid x; \theta) = (h_\theta(x))^y \, (1 - h_\theta(x))^{1-y}$$

Defining the Objective Function J(θ)

- Given: $$p(y \mid x; \theta) = (h_\theta(x))^y \, (1 - h_\theta(x))^{1-y}$$

- We want to estimate θ that will capture the dependency between y and x. When we wish to explicitly view this as a function of θ, we will instead call it the ***likelihood function*** that maximizes p(y|X; θ) and is given by

$$L(\theta) = \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)}; \theta)$$

Defining the Objective Function J(θ)

- Given: $$p(y \mid x; \theta) = (h_\theta(x))^y \, (1 - h_\theta(x))^{1-y}$$

- We want to estimate θ that will capture the dependency between y and x. When we wish to explicitly view this as a function of θ, we will instead call it the ***likelihood function*** that maximizes p(y|X; θ) and is given by

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)}; \theta) \\ &= \prod_{i=1}^{m} \left(h_\theta(x^{(i)})\right)^{y^{(i)}} \left(1 - h_\theta(x^{(i)})\right)^{1-y^{(i)}} \end{aligned}$$

Defining the Objective Function J(θ)

- Now, given this probabilistic model how can we get θ?

Defining the Objective Function J(θ)

- Now, given this probabilistic model how can we get θ?
- The principal of ***maximum likelihood*** says that we should should choose θ that makes the data as high probability as possible. i.e., we should choose θ to maximize L(θ).

Defining the Objective Function J(θ)

- Now, given this probabilistic model how can we get θ?
- The principal of ***maximum likelihood*** says that we should should choose θ that makes the data as high probability as possible. i.e., we should choose θ to maximize L(θ).
- Instead of maximizing L(θ), we can also maximize any strictly increasing function of L(θ). So we will maximize the log likelihood ℓ(θ):

$$\ell(\theta) = \log L(\theta)$$

Defining the Objective Function J(θ)

- Now, given this probabilistic model how can we get θ?
- The principal of ***maximum likelihood*** says that we should should choose θ that makes the data as high probability as possible. i.e., we should choose θ to maximize L(θ).
- Instead of maximizing L(θ), we can also maximize any strictly increasing function of L(θ). So we will maximize the log likelihood ℓ(θ):

$$\ell(\theta) = \log L(\theta)$$

$$= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))$$

How do we get θ?

- Gradient Descent!
- Then, update θ_j until convergence as follows

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- So what is $\frac{\partial}{\partial \theta_j} J(\theta)$?

How do we get θ?

- Gradient Descent!
- Then, update θ_j until convergence as follows

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- Hence our new update rule for *logistic* regression is

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

How do we get θ?

- Gradient Descent!
- Then, update θ_j until convergence as follows

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- Hence our new update rule for *logistic* regression is

Looks familiar?

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

How do we get θ?

- Gradient Descent!
- Then, update θ_j until convergence as follows

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- Hence our new update rule for *logistic* regression is

Looks familiar?

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

Identical to Linear Regression Update rule!