CE 395 Special Topics in Machine Learning

Assoc. Prof. Dr. Yuriy Mishchenko

Fall 2017

STATISTICAL LEARNING

REVIEW

Local Methods in High Dimensions

- Last time we examined two simple approaches to prediction: linear model and K-nearest neighbors algorithms
  - Linear model: $y\hat{}=x^T\theta$ – was a very simple prediction method with a few parameters $\theta_{1..p}$ that could be very accurately estimated by pulling together all observations (x_i,y_i) in RSS minimization
  - KNN method: $y\hat{}=k^{-1}\Sigma_{l\in Nk(x)}y_{(l)}$ – was a simple local average-approach that due to its flexibility could produce in the limit $k,n\rightarrow\propto$ the exact (that is, optimal) prediction rule for regression $Y\hat{}(x)=E[Y|X=x]$

Local Methods in High Dimensions

- One could expect that with large datasets theoretically one could approximate the optimal decision rule by a method like kNN averaging at some point.
- Unfortunately, this intuition is not true in high dimensions due to the phenomenon that is oftentimes called **the curse of dimensionality.**

Local Methods in High Dimensions

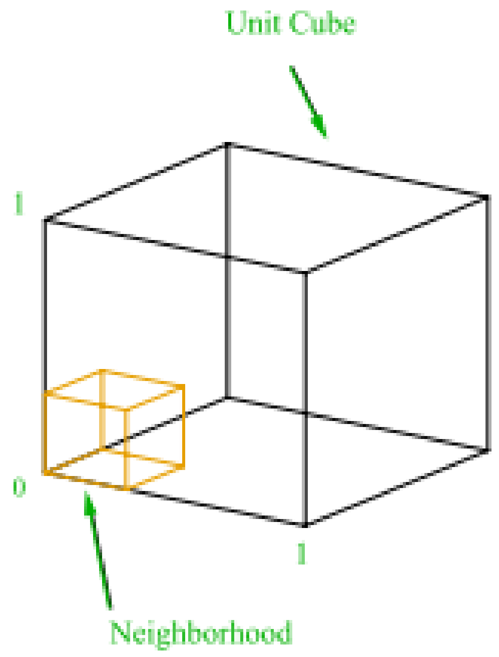
Curse of dimensionality:

- *The average nearest-neighbor distance between points uniformly distributed in a p-dimensional cube* $[0,1]^p$, $(1/n)^{1/p}$, *tends to 1 exponentially fast as p gets large*
  OR
- *The number of points uniformly distributed in a p-dimensional cube* $[0,1]^p$ *needed to achieve a fixed linear density (spacing)* Δx, $(1/\Delta x)^p$, *grows exponentially fast as p gets large*

Local Methods in High Dimensions

- Try it for yourself – construct a distributions of data points covering each linear direction in a cube $[0,1]^p$ at spacing $\Delta x=0.01$, in $p=1$, $p=3$, $p=10$, $p=100$ dimensions

Local Methods in High Dimensions



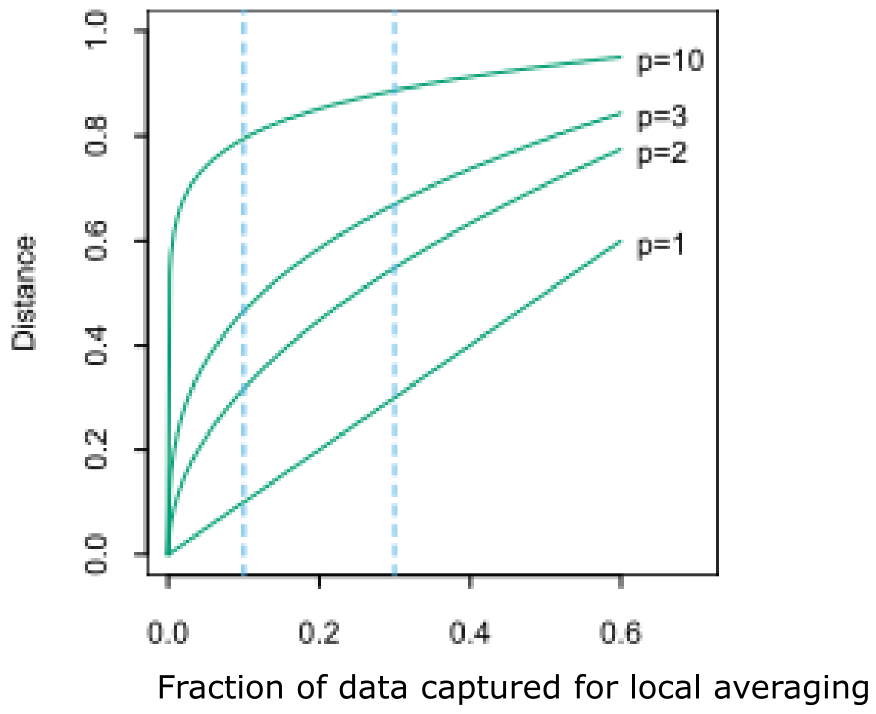


FIGURE 2.6. *The curse of dimensionality is well illustrated by a subcubical neighborhood for uniform data in a unit cube. The figure on the right shows the side-length of the subcube needed to capture a fraction r of the volume of the data, for different dimensions p. In ten dimensions we need to cover 80% of the range of each coordinate to capture 10% of the data.*

Local Methods in High Dimensions

Curse of dimensionality invariably causes local methods to fail as greater and greater proportion of the data range need to be included in local averages to garner the same number of points K needed to estimate the averages, eventually resulting in models that are neither local nor average

Local Methods in High Dimensions

Example - n=1000 data points uniformly distributed in $[0,1]^p$ used to estimate local average f(x)=E[Y|X=x] with K=100 points:

- In p=1, k=100 neighbors are found in L=0.1 neighborhood of any wanted x, resulting in averages, f^(x), approximating the true values E[Y|X=x] well
- In p=10, at least one of such 100 neighbors typically will have one x_i as large as ± 0.3, resulting in f^(x) that averages Y on the range of space X as large as ΔL=0.6 – this average is no longer local, and if E[Y|X=x] changes significantly with x the KNN average f^(x) will become way off target !!!
- In p=1000, even with n=1,000,000 data points, the typical averaging neighborhood for f^(x) for K even as small as 1 will be $(1/n)^{1/p}=0.986$ – that is, almost the entire cube !!!

Local Methods in High Dimensions

The other side of curse of dimensionality:

- In high dimensions the nearest data point to any internal point x is almost as far as away as the external boundary of the entire dataset !
- In fact, any point in dataset X is *closer to the boundary than any other point* in X !

Local Methods in High Dimensions

- We will now consider in detail an educational example: approximate the following function by using K=1NN

$$Y = f(X) = e^{-8|X|^2}$$

  (no randomness)
- The object of interest here is the **Mean Square Error (MSE)** of the predictions (averaged over different training datasets, whereas T represents different randomly selected training datasets of n points used to perform KNN):

$$MSE(x_0) = E_T[(f(x_0) - \hat{y}_0)^2]$$

Local Methods in High Dimensions

- **Note:** Even though Y=*f*(X) itself is not random here, KNN predictions y^ are random, because the dataset points, from which KNN selects the nearest neighbor for constructing *f*^(x), are random. KNN predictions, therefore, are random from that point of view.
- We indicate training dataset-related randomness via subscript T, and want to evaluate the square-loss of a large number of different KNN predictions for a single point x_0, with respect to random training dataset selection

Local Methods in High Dimensions

- Not random: $y_0 = f(x_0)$
- KNN is random: $\hat{y}_{0;T} = \hat{f}_T(x_0) = f(x_{(1)})$
- KNN Square-Loss: $MSE(x_0) = E_T[(f(x_0) - \hat{y}_0)^2]$

Local Methods in High Dimensions

Bias-variance decomposition is the standard transform in ML:

$$
\begin{aligned}
MSE(x_0) &= E_T[(f(x_0) - \hat{y}_0)^2] \\
&= E_T[(\hat{y}_0 - E_T[\hat{y}_0])^2] + (E_T[\hat{y}_0] - f(x_0))^2 \\
&= \mathrm{var}_T(\hat{y}_0) + \mathrm{Bias}^2(\hat{y}_0)
\end{aligned}
$$

– Variance represents how much y_0^ is changing from trial to trial, in relation to random selection of n training points T, and is the statistical (that is random) error in the predictions

– Bias shows how much the average prediction is off

Local Methods in High Dimensions

How to correctly understand the result of bias-variance decomposition:

$$
\begin{aligned}
MSE(x_0) &= E_T[(f(x_0) - \hat{y}_0)^2] \\
&= E_T[(\hat{y}_0 - E_T[\hat{y}_0])^2] + (E_T[\hat{y}_0] - f(x_0))^2 \\
&= \mathrm{var}_T(\hat{y}_0) + \mathrm{Bias}^2(\hat{y}_0)
\end{aligned}
$$

– **Variance** is how much predictions can change randomly due to chance choice of training data – when training data is collected from P(X,Y), such data can be assumed to come randomly from data model. Therefore, predictions constructed using such data and any prediction algorithm also should be viewed as random, dependent on and different every time a different dataset is collected for the same type of problem (for example, face recognition)

Local Methods in High Dimensions

How to correctly understand the result of bias-variance decomposition:

$$
\begin{aligned}
MSE(x_0) &= E_T[(f(x_0) - \hat{y}_0)^2] \\
&= E_T[(\hat{y}_0 - E_T[\hat{y}_0])^2] + (E_T[\hat{y}_0] - f(x_0))^2 \\
&= \mathrm{var}_T(\hat{y}_0) + \mathrm{Bias}^2(\hat{y}_0)
\end{aligned}
$$

- **Bias** is the error of the average prediction, in which the predictions' randomness was done away by means of averaging a large number of trials. After we do away with statistical randomness by averaging predictions obtained with many different datasets, the bias is what is left in *f*^(x). Bias cannot be removed any more by simply collecting or inspecting more data (but sometimes can be estimated and removed theoretically if the data model is well understood)

Local Methods in High Dimensions

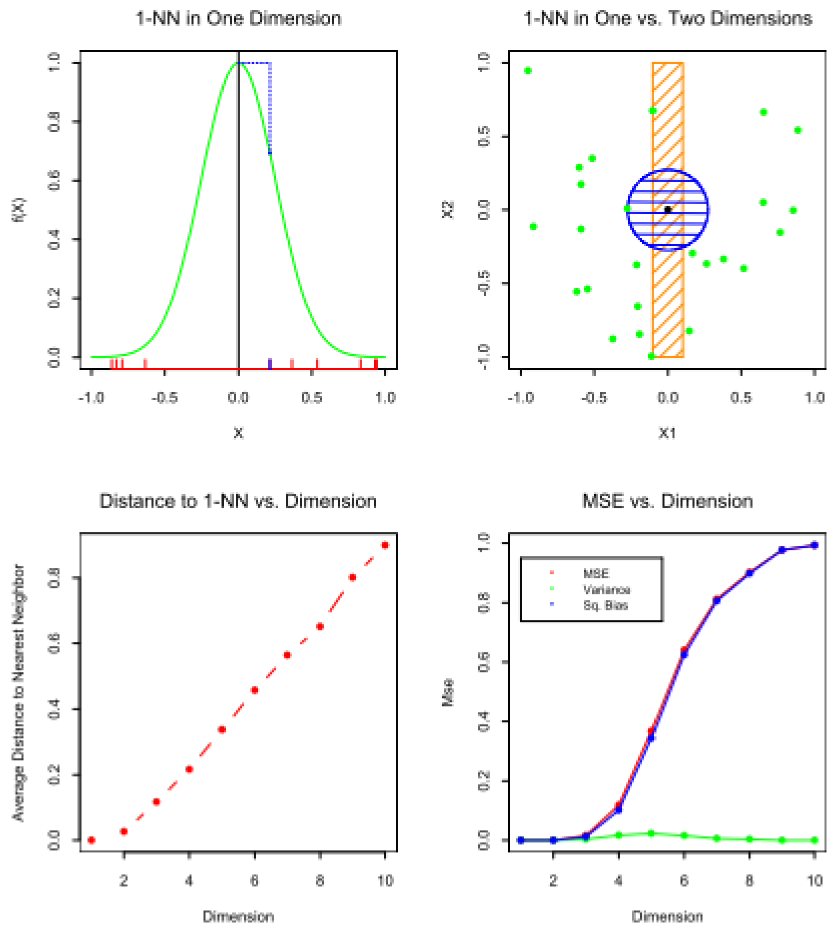


FIGURE 2.7. *A simulation example, demonstrating the curse of dimensionality and its effect on MSE, bias and variance. The input features are uniformly distributed in $[-1,1]^p$ for $p = 1,\dots,10$ The top left panel shows the target function (no noise) in $\mathbb{R}$: $f(X) = e^{-8||X||^2}$, and demonstrates the error that 1-nearest neighbor makes in estimating $f(0)$. The training point is indicated by the blue tick mark. The top right panel illustrates why the radius of the 1-nearest neighborhood increases with dimension p. The lower left panel shows the average radius of the 1-nearest neighborhoods. The lower-right panel shows the MSE, squared bias and variance curves as a function of dimension p.*

Matlab-sim (p=1,…,10):

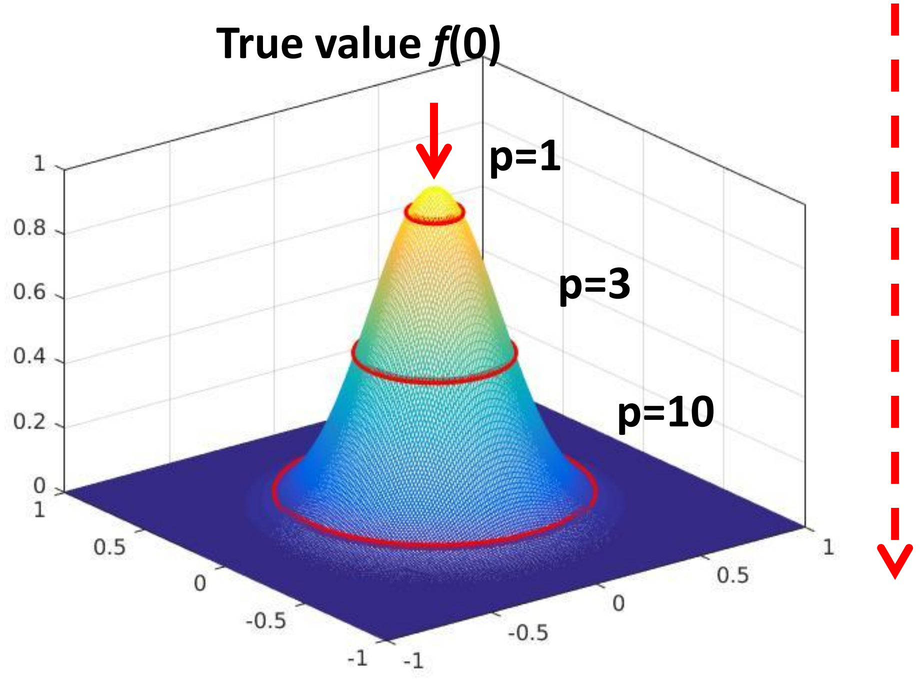
```
X=randn([p,1000,1000]);
X2=sum(X.^2,1);
X2nn=min(X2,[],2);
Ynn=exp(-8*X2nn(:));
figure,hist(sqrt(X2nn(:)),25);
figure,hist(Ynn,25);
[mean(Ynn);var(Ynn)]
```

1000 datasets of 1000 points of size px1

px1 points closest to 0

Local Methods in High Dimensions

What is your expectation?

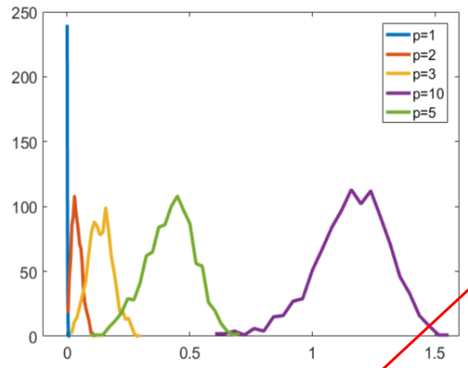


<u>Statistical variation</u> – the variance – of nearest neighbor's value ?

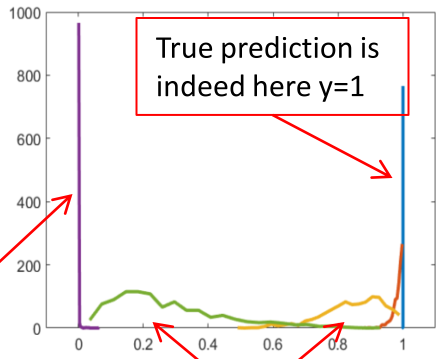
The <u>difference</u> of nearest neighbor value from *f*(0) – the bias ?

Local Methods in High Dimensions

Distribution of nn-distances



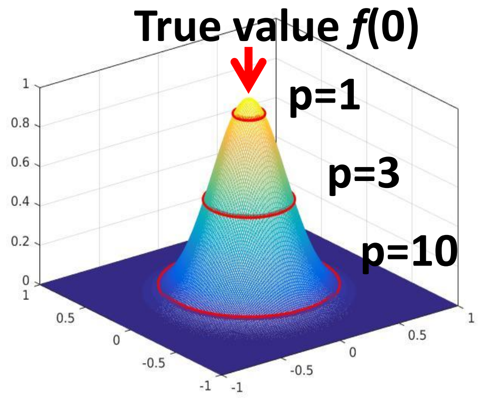
Distribution of nn-predictions y_T^



True prediction is indeed here y=1

Bias of predictions due to nearest neighbor being far in high-D situations

Variance of predictions due to spread in position of nearest neighbor in random training data

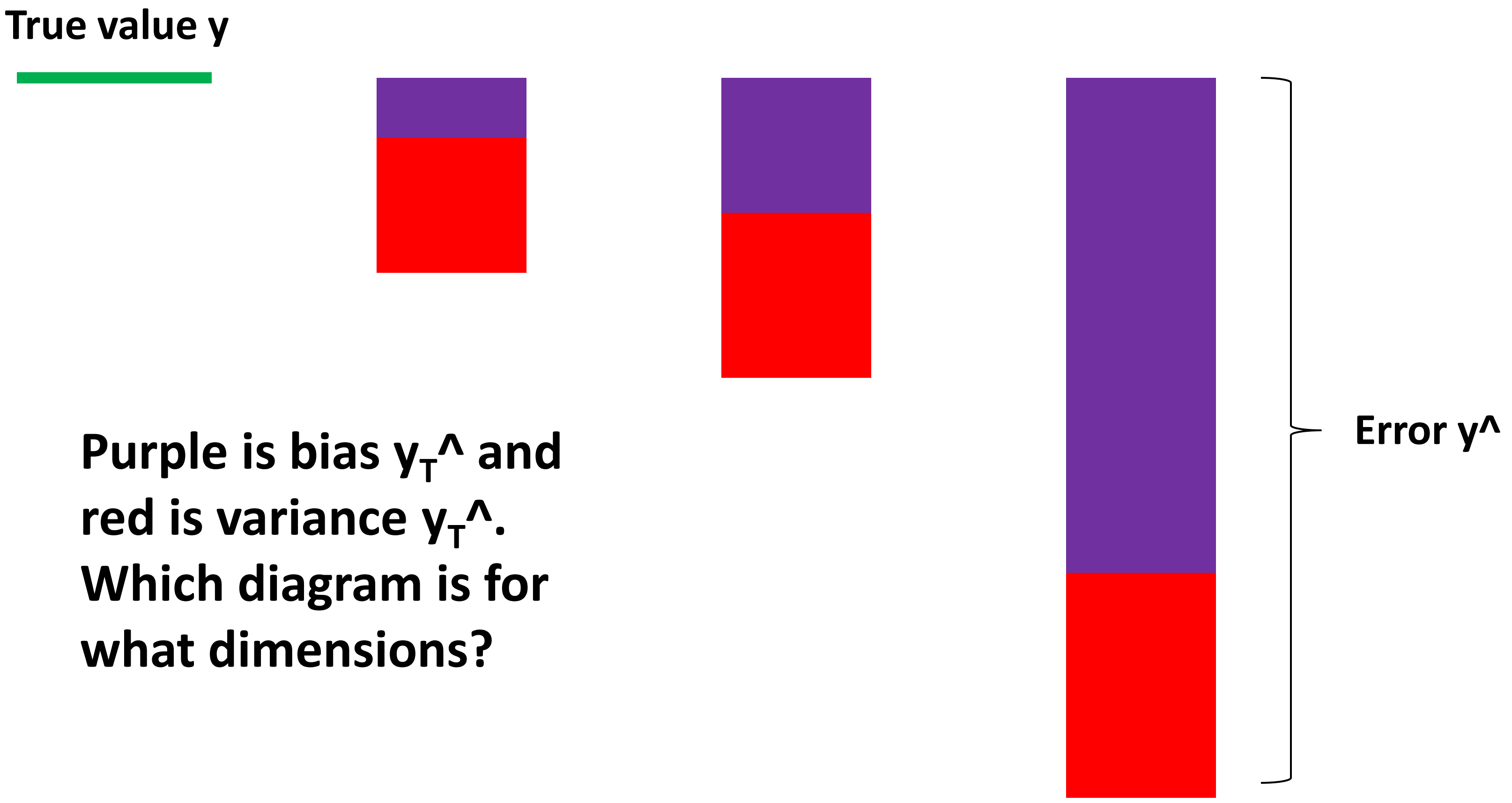


In low dimensions, the nearest neighbor is close, and almost all errors come from randomness in the nearest neighbor to point 0 in the random training data

In high dimensions, the nearest neighbor is very far and $f(x_{(1)})$ is very different from actually the $f(0)$: Practically all predictions f^(0) differ dramatically from $f(0)$ creating a large bias in the estimates y_T^ for all T.

Local Methods in High Dimensions

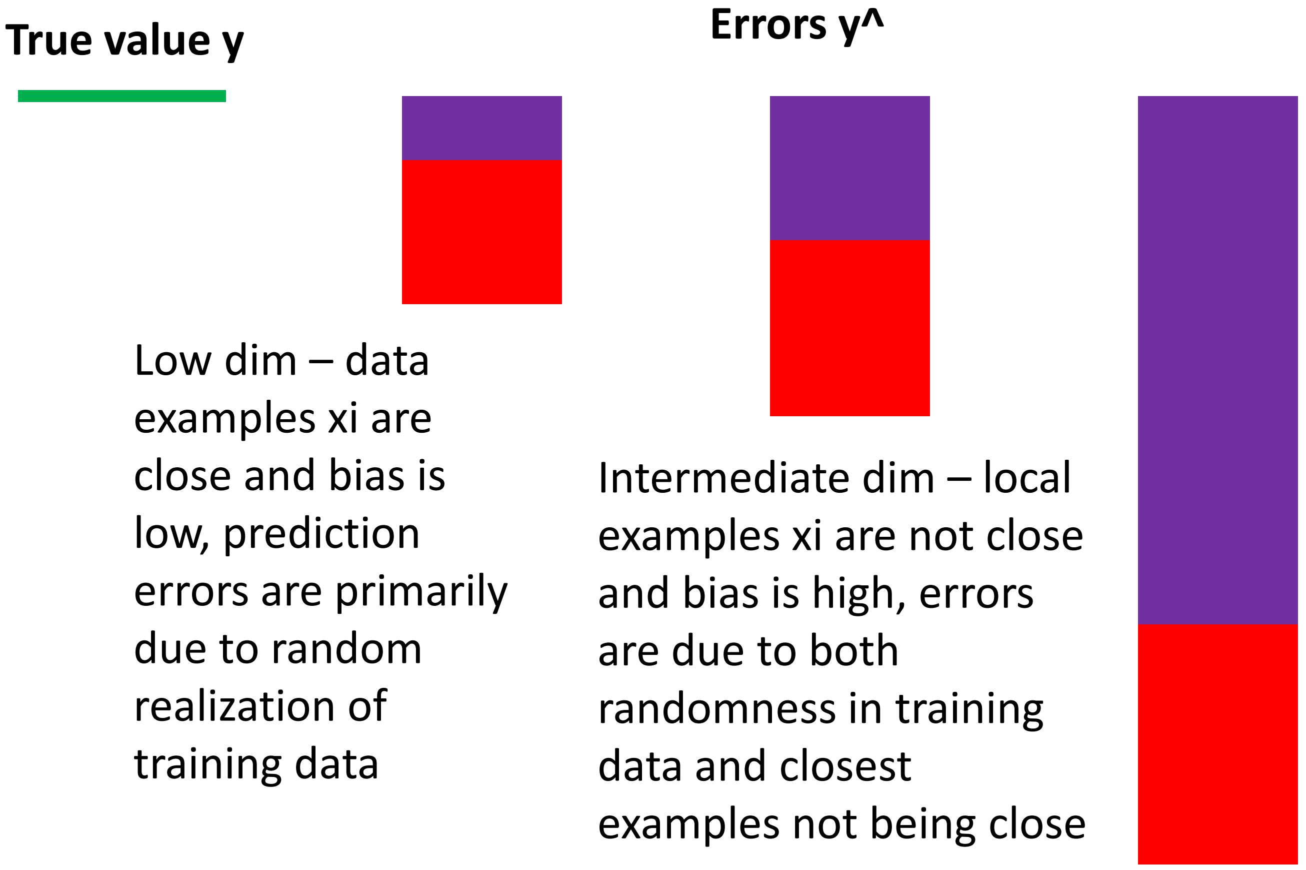
Curse of dimensionality in bias-variance decomposition:



Purple is bias y_T^ and red is variance y_T^. Which diagram is for what dimensions?

Local Methods in High Dimensions

Curse of dimensionality in bias-variance decomposition:



Low dim – data examples xi are close and bias is low, prediction errors are primarily due to random realization of training data

Intermediate dim – local examples xi are not close and bias is high, errors are due to both randomness in training data and closest examples not being close

High dim – "local" examples xi are **far**, bias is very high, errors are primarily due to best examples not being anything like the target x

Local Methods in High Dimensions

How can the problem associated with the curse of dimensionality be cured?

What if we knew the data are coming from a simple generative model with few parameters?

$$\underbrace{Y = X^T\theta}_{\text{Simple model Y(X)}} + \underbrace{\varepsilon, \varepsilon \sim N(0,\sigma^2)}_{\substack{\text{Randomness/noise/} \\ \text{not controlled variability}}}$$

Local Methods in High Dimensions

For linear model, we can estimate EPE theoretically to be

$$
\begin{aligned}
EPE(x_0) &= E_{y_0|x_0} E_T (y_0 - \hat{y}_0)^2 \\
&= (E_T \hat{y}_0 - x_0{}^T \theta)^2 + E_T (\hat{y}_0 - E_T \hat{y}_0)^2 + E(y_0 - x_0{}^T \theta)^2 \\
&= \mathrm{Bias}(\hat{y}_0)^2 + \mathrm{var}_T(\hat{y}_0) + \mathrm{var}(y_0) \\
&= 0 + \frac{p}{n}\sigma^2 + \sigma^2
\end{aligned}
$$

Here, the 1st term is **the bias** which is 0 because our prediction model is exact (so that when $n \rightarrow \propto$ we get exact E[Y|X=x]), the 2nd term is **the variance of the predictions due to random training data**, and the 3rd term is **the irreducible error due to Y being random** to begin with

Local Methods in High Dimensions

- Unlike KNN, the error in linear regression grows very slowly with p – proportionally to σ^2/n. If n is large or σ^2 is small, this is essentially negligible:

$$EPE(x_0) = \sigma^2 \frac{p}{n} + \sigma^2$$

- On the surface conclusion: linear model appears to be superior in high-dimensional situations

Local Methods in High Dimensions

One way to understand this - information principle:

- In KNN and similar ***local methods***, a limited number of data is used to estimate a large number of independent values $f(x)$ at possibly $N \approx (1/\Delta x)^p$ different points. That results in very few data points present per prediction $\hat{f}(x)$, $\approx (n \cdot (\Delta x)^p)$, and that makes estimation very difficult – think about trying to estimate the mean of a highly varying variable from single realization of it - only weak predictions can be made in that case.
- In linear regression and similar ***restricted estimator methods***, a limited number of data samples is used to estimate a small number of generative data model parameters θ. There is many data per each parameter, $\approx (n/\dim(\theta))$, and that allows accurate estimations to be made – think about estimating the mean of a highly varying variable from 1000 its realizations.

Local Methods in High Dimensions

Matlab-sim (local method):

```
X=10*randn(1);
m=mean(X)
```

Random variable with mean 0 and standard deviation 10

Matlab-sim (restricted est):

```
X=10*randn(100,1);
m=mean(X)
```

Estimate the mean by single look at X.

Estimate the mean from 100 examples of X.

- Note also that KNN and similar local methods can be seen as parametric methods with $N \approx (1/\Delta x)^p$ effective parameters – one for each value $f(x)$ estimated independently from the rest.

Function Estimation

Let’s delve into this problem a little deeper now.

Suppose our goal is to find useful approximation f^(x) for a noisy and high-dimensional function f(x) underlying a predictive relationship between input and output of certain data P(X,Y). (In the context of optimal decision making, we may want to approximate f(x)=E[Y|X=x] for P(X,Y), for example.)

Function Estimation

- We propose a following view on data model:

  $$Y = f(X) + \varepsilon$$

  X is seen as r.v. coming from some P(X) and Y is deterministically determined by Y plus ε - an independent random noise with zero mean – E[ε]=0.
- This model is known as **additive error model** and describes a deterministic relationship Y(X) corrupted by noise or uncontrolled factors ε

Function Estimation

$$\underbrace{Y = f(X)}_{\text{Deterministic relationship}} + \underbrace{\varepsilon}_{\text{Random variability}}$$

Note that by construction *f*(x)=E[Y|X=x] because E[ε]=0. So finding the true *f*(x) here also gives us the optimal prediction function E[Y|X=x].

Function Estimation

- **Supervised learning** is the approach for inferring *f*(x) through a series of its examples (xi,yi) either collected experimentally or given by a teacher.
- The observed input and output values are fed into a learning algorithm, which thus produces estimated outputs for training data *f*^(xi) as well as the model *f*^ that can be used for predicting y for new data x.
- **Note that** *f*^(xi) doesn't need to exactly match all yi because of assumed noise εi.

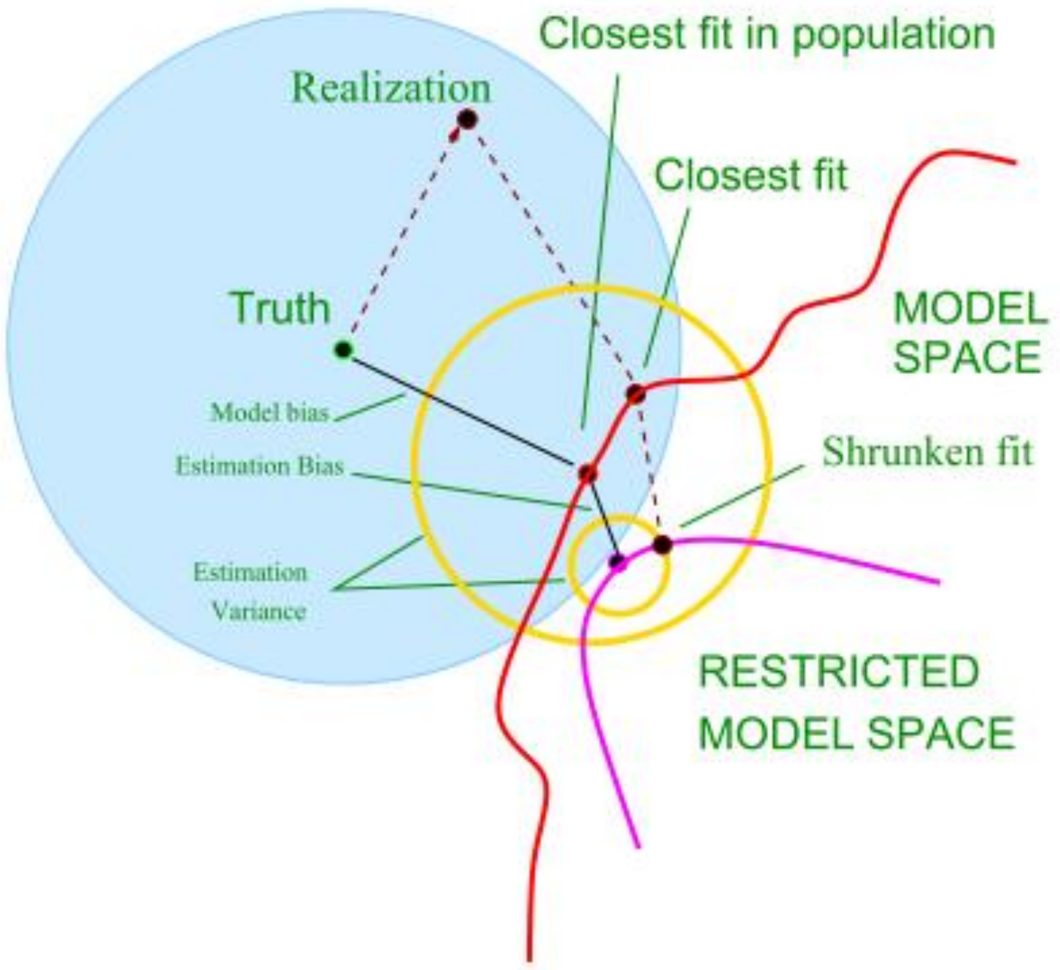
Function Estimation

- In applied statistics, supervised learning is seen as the problem of **function estimation** or **function approximation** for $f(\mathrm{x})$ of the true *generative data model* $\mathrm{y}=f(\mathrm{x})+\varepsilon$.
- Each training example {xi,yi} is viewed as a point in p+1-dimensional space, where the part of the points in p-dimensional “input” subspace xi is related to the p+1 “output” subspace yi via $\mathrm{y}_i=f(\mathrm{x}_i)+\varepsilon_i$

Function Estimation

- The goal is to produce a useful approximation to *f* from that set of examples $\{x_i,y_i\}$, which is thus called the **training set T**.
- If the process of producing such a useful approximation is restricted to remain within a family of some parametrized models $f_\theta(x)$, then such learning is called **parametric**; an example of parametric approach is the linear regression.
- **Note that** the data may or may not *really come* from a member of family $f_\theta(x)$ – in that case *learning produces a best approximation to true f(x) inside the class* $f_\theta(x)$.

Function Estimation



Function Estimation

- If the process of producing a useful approximation to *f*(x) does not operate within a family of parametrized models, such learning is called **nonparametric**; an example of this approach is the KNN regression
- In KNN regression, the predictions *f*^(x) are produced by inspecting directly the original data via calculation of the local averages, and no parameter sets are used in between $T \rightarrow y$^.

Restricted function estimators

- **Parametric models** are known as **restricted estimators** – the point being that a restriction is imposed on the structure of possible models for $f(x)$ that will be tried, in the form of forcing those to be in the parametric form of $f_\theta(x)$.
- **There is a great variety of restricted estimator models in ML and SL, with linear models being the simplest.**

Restricted function estimators

- A must know type of restricted estimators is the **linear basis expansion:**

$$f_\theta(x) = \sum_{k=1}^{K} \phi_k(x)\theta_k$$

- In terms of estimation, the linear basis expansion is trivially reduced to the linear regression by substituting variables $x \rightarrow \phi_k(x) = \phi_k$. At the same time, it allows capturing very complex nonlinear relationships in terms of x, by exploiting various basis functions ϕ_k which also may contain parameters that can be adjustable on their own **[example on whiteboard]**.

Restricted function estimators

Basis expansion estimation:

$$f_\theta(x) = \sum_{k=1}^{K} \phi_k(x)\theta_k \rightarrow$$

$$x \rightarrow \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \ldots \end{bmatrix} = \phi \Rightarrow y = \phi^T \theta$$

$$RSS = \min |\, y - \hat{y}\,|^2 \Rightarrow \hat{\theta} = (\Phi^T \Phi)^{-1} \Phi^T y$$

Restricted function estimators

Other classes of restricted estimators:

- **Roughness penalty estimators** and **Bayesian estimators** are obtained by penalizing (aka regularizing) the RSS with a penalty adjusted via a penalty (aka regularization) parameter λ:

$$PRSS(f;\lambda) = RSS(f) + \lambda J(f)$$

Restricted function estimators

Example – smooth function estimator:

$$PRSS(f;\lambda) = \sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda \int | f''(x) |^2 \, dx$$

Restricted function estimators

Other classes of restricted estimators:

- **Kernel methods** and **local regression** construct predictions for x_0 by emphasizing data that is local to x_0, according to a certain local-neighborhood criterion, typically expressed in terms of **kernel functions** such as **the Gaussian kernel**:

$$K_\lambda(x_0, x) = \frac{1}{\lambda^{p/2}} \exp\left[-\frac{|x - x_0|^2}{2\lambda} \right]$$

Restricted function estimators

The kernel expresses similarity between data points (in the feature space), and the way that is done may be controlled via various parameter such as the scale parameter λ in the Gaussian kernel – defining roughly the width of the local-neighborhood there.

Another popular kernel is **the cosine kernel**, which has been widely used in text processing and search:

$$\cos(x_0, x) = \frac{{x_0}^T \cdot x}{|x_0 \| x|}$$

Restricted function estimators

Local regression for predicting y(x0) is then defined by weighing RSS with the kernel

$$RSS(\theta, x_0) = \sum_{i=1}^{n} K_\lambda(x_0, x_i)(y_i - f_\theta(x_i))^2$$

thus prioritizing the nearby points in the prediction of $y(x_0)$.

For example, choice $f_\theta(x)=x^T\theta$ with Gaussian kernel gives the popular local linear regression method

Restricted function estimators

Example – local linear regression $f_\theta(x)=x^T\theta$

$$RSS(\theta, x_0) = \sum_{i=1}^{n} K_\lambda(x_0, x_i)(y_i - x_i{}^T\theta)^2$$

$$\Rightarrow \theta(x_0) \Rightarrow \hat{y}(x_0) = x_0{}^T\theta(x_0)$$

Restricted function estimators

Other classes of restricted estimators:

- **Basis functions** and **dictionary methods** restrict models to the form

$$f_\theta(x) = \sum_{m=1}^{M} \theta_m h_m(x)$$

where each h_m is a predetermined function and the model *f*(x) is linear in h_m.

Restricted function estimators

Examples

- Radial Basis Function Networks (K is the Gaussian kernel from earlier)

$$f_\theta(x) = \sum_{m=1}^{M} \theta_m K_\lambda(\mu_m, x)$$

- Feed-forward neural network, single layer (aka perceptron)(σ is the sigmoid function $e^x/(1+e^x)$)

$$f_\theta(x) = \sum_{m=1}^{M} \theta_m \sigma({\alpha_m}^T x + b_m)$$

Restricted function estimators

Another example is the single-layer feed-forward neural network, where the parameters of the basis functions α and b are also adaptively adjusted by optimization

$$f_\theta(x) = \sum_{m=1}^{M} \theta_m \sigma(\alpha_m{}^T x + b_m)$$

Restricted function estimators

- In both examples, the parameters of the basis functions λ, α or b, can be also adaptively adjusted via nonlinear optimization.
- When the basis functions are chosen adaptively, depending on the training data (instead of fixed beforehand), such methods are known as **dictionary methods** and *the sets of the derived basis functions are said to be* ***dictionaries***, in terms of which that is the target function is being decomposed/represented.

Restricted function estimators

- Generally, general numerical optimization algorithms need to be used to estimate the parameters of restricted estimator models with the rare exceptions such as the linear model or the linear basis expansion

$$\min RSS(\theta) = \min \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2$$

Bias-Variance Tradeoff in ML models

- Many models that we discussed so far contain a complexity or smoothing parameter such as λ – the multiplier in the penalty term or the width of the kernel, or M – the number of basis functions.
- In all the discussed cases a particular limit of complexity produces a mathematically "exact" function approximation method: if $\lambda \to 0$ (no smoothness or 0-width kernel) or $M \to \propto$ (for example Fourier series) such methods have the power to approximate any $y=f(x)$ in existence

Bias-Variance Tradeoff in ML models

In a sense, as we change λ or M between 0 and $\propto$, we interpolate between the linear model and the KNN algorithm. It is proper, therefore, to wonder at what point $f_{\lambda}(x)$ stops being a restricted estimator (lesser problems in high-dimensional data, that is linear model) and starts being a local method (big problems in high-dimensional data, that is KNN)

Bias-Variance Tradeoff in ML models

The issue at hands is all-permeating in ML and is known in ML as **the bias-variance tradeoff**:

- **Variance:** As complexity of a class of models increases, in the sense that more arbitrary functions can be described as the members of $f_\theta(x)$, the total error of the model caused by variation or noise in training sets increases without bounds, making such models unusable for predicting real data.
- **Bias:** As complexity of a class of models decreases, the true functions $f(x)$ no longer can be realized amongst $f_\theta(x)$, resulting in increased bias error $|f(x)-f_{\theta*}(x)|$, where $f_{\theta*}(x)$ is the best model for $f(x)$ in the class $f_\theta(x)$, also rendering such models unusable for predicting real data.

Bias-Variance Tradeoff in ML models

Bias-variance decomposition for bias-variance tradeoff

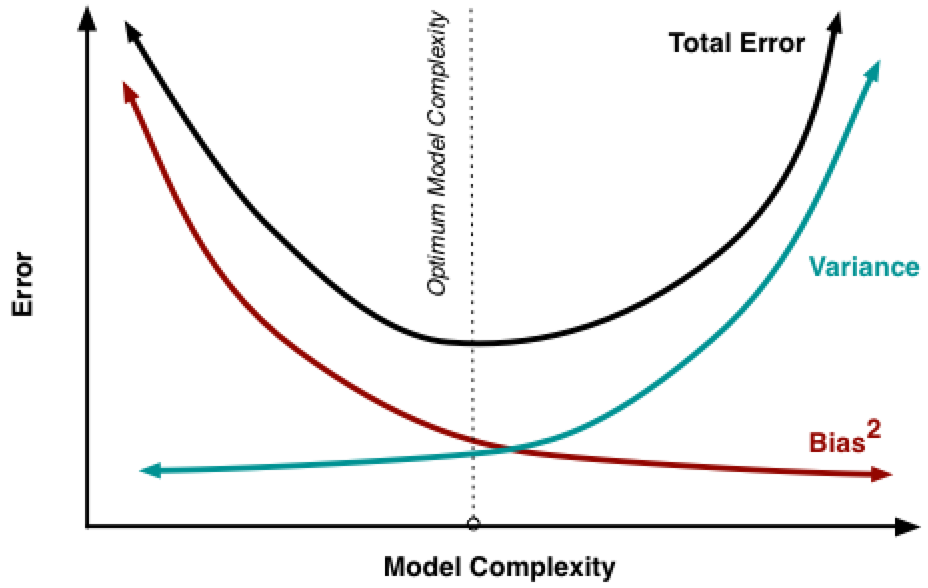
$$\begin{aligned} EPE_{\lambda}(x) &= E_T[(Y - \hat{f}_{\lambda}(x))^2 \mid X = x] \\ &= E(Y - f(x))^2 + (f(x) - \hat{f}_{\lambda}(x))^2 + E_T(\hat{f}_{\lambda;T}(x) - \hat{f}_{\lambda}(x))^2 \\ &= \sigma^2 + \mathrm{Bias}(\hat{f}_{\lambda}(x))^2 + \mathrm{var}(\hat{f}_{\lambda}(x)) \end{aligned}$$

Observe the 3 terms in EPE:

- σ^2 - uncontrolled variation of y at fixed x – **irreducible error**
- Error in the average prediction *f*^$_{\lambda}$ (x) from *f*(x) – **bias**
- Error in the individual predictions *f*^$_{\lambda;T}$ (x) due to random realization of training data T – **variance**

Bias-Variance Tradeoff in ML models

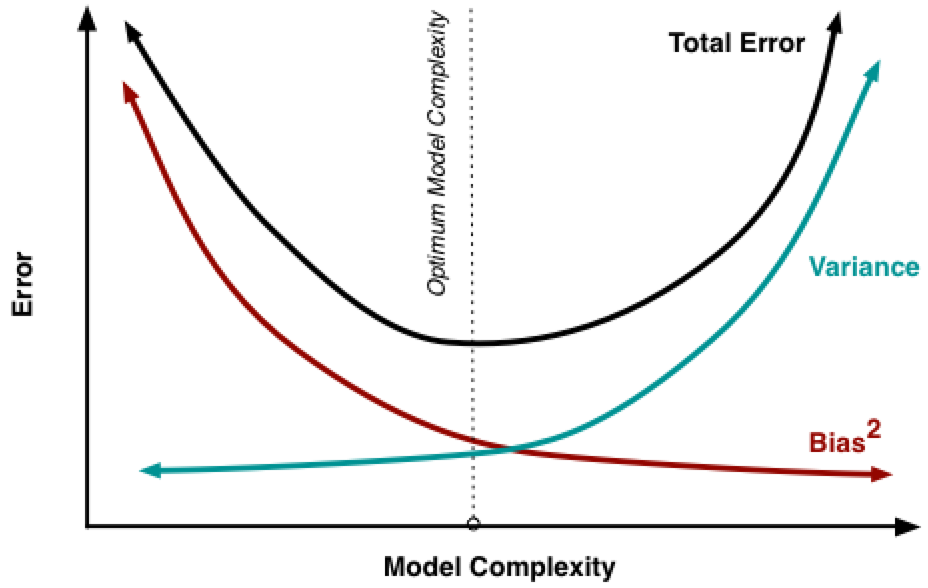
Bias-variance tradeoff #1: as the model's complexity decreases, the error's variance tends to decrease but the squared bias tends to increase.



$$\begin{aligned} EPE_\lambda(x) &= E_T[(Y - \hat{f}_\lambda(x))^2 \mid X = x] \\ &= \sigma^2 + \mathrm{Bias}(\hat{f}_\lambda(x))^2 + \mathrm{var}(\hat{f}_\lambda(x)) \end{aligned}$$

Bias-Variance Tradeoff in ML models

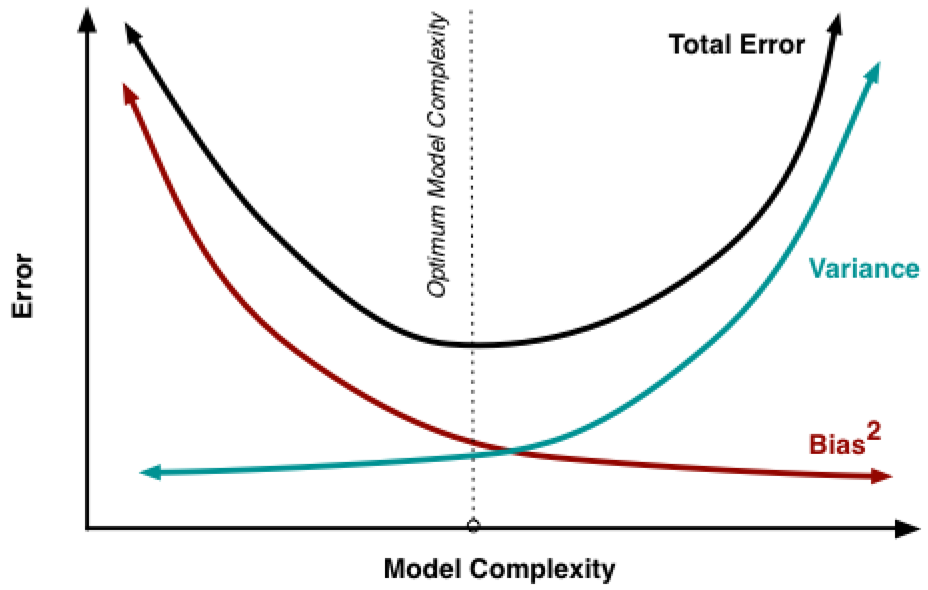
Bias-variance tradeoff #2: as model's complexity increases, the error's variance tends to increase while the squared bias tends to decrease, resulting in a U-shaped behavior of the total generalization error or EPE.



$$EPE_\lambda(x) = E_T[(Y - \hat{f}_\lambda(x))^2 \mid X = x]$$
$$= \sigma^2 + \text{Bias}(\hat{f}_\lambda(x))^2 + \text{var}(\hat{f}_\lambda(x))$$

Bias-Variance Tradeoff in ML models

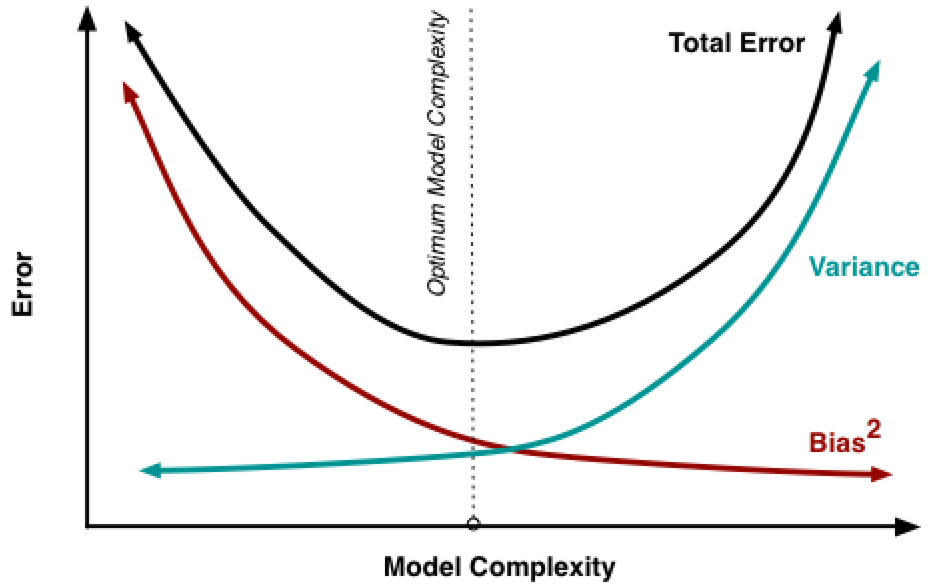
- It is currently thought to be impossible to achieve at the same time a small bias and small variance in ML models.
- Therefore, a trade-off is required between the bias and the variance by setting the complexity parameter λ to a point where **the sum of the two is the smallest**.



$$
\begin{aligned}
EPE_{\lambda}(x) &= E_T[(Y - \hat{f}_{\lambda}(x))^2 \mid X = x] \\
&= \sigma^2 + \mathrm{Bias}(\hat{f}_{\lambda}(x))^2 + \mathrm{var}(\hat{f}_{\lambda}(x))
\end{aligned}
$$

Bias-Variance Tradeoff and learning performance assessment

Question: how to select the optimal complexity point? How can we estimate where on the curve we reside?



$$
\begin{aligned}
EPE_{\lambda}(x) &= E_T[(Y - \hat{f}_{\lambda}(x))^2 \mid X = x] \\
&= \sigma^2 + \mathrm{Bias}(\hat{f}_{\lambda}(x))^2 + \mathrm{var}(\hat{f}_{\lambda}(x))
\end{aligned}
$$

Learning performance assessment

Fact: The residual-sum-of-squares obtained in the learning is not an accurate estimator of EPE aka generalization accuracy

Learning performance assessment

Think about the difference:

- Expected prediction error – average over all possible inputs and outputs:

$$EPE = \sum_{(X,Y)} L(y, f_{\hat{\theta}}(x))P(x,y) = E[L(y, f_{\hat{\theta}}(x))]$$

- Training-data prediction error – …

$$TPE = \sum_{(x_i,y_i)\in T} L(y_i, f_{\hat{\theta}}(x_i)) = E[L(y, f_{\hat{\theta}}(x)) \mid T]$$

Learning performance assessment

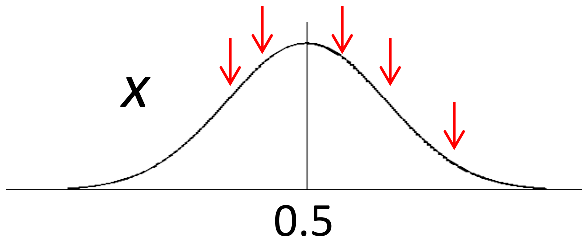
In ML, TPE is not a good predictor of EPE, in fact we will see shortly that TPE always underestima-tes (and often by far!) EPE

$$TPE < EPE$$

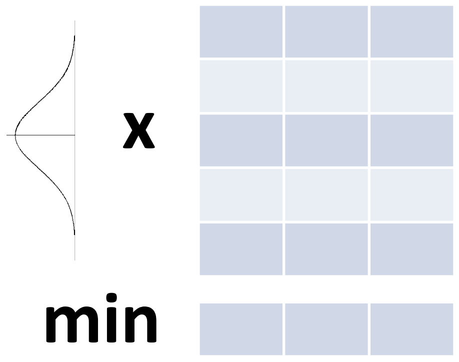
Learning performance assessment

Optimism bias

Consider Normal r.v. on which we want to select min (we can think that *x* represents the error of a ML algorithm with a certain fixed θ on random sets of training data T):

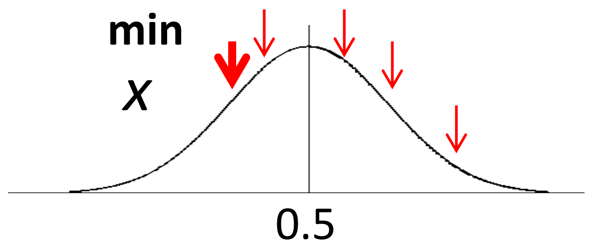


Choose *X*=min ($x_{1..n}$) (**the best alg.**); what can be said about X?

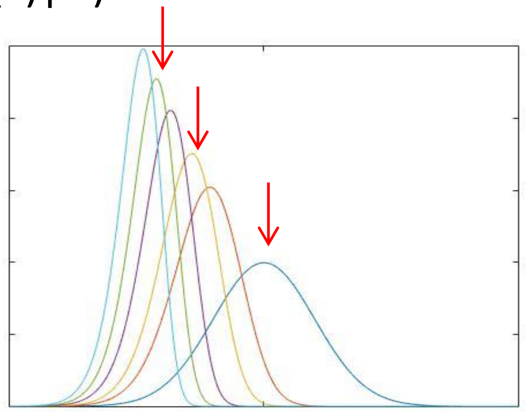


Learning performance assessment

X is nearly always smaller than m=E[x] - why?



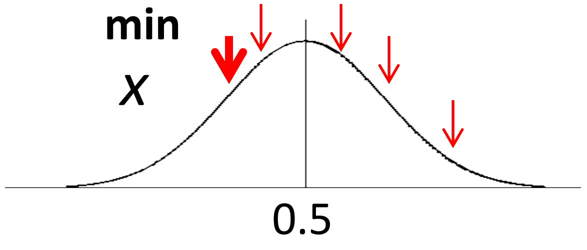
P($\min_N$(x)|N)

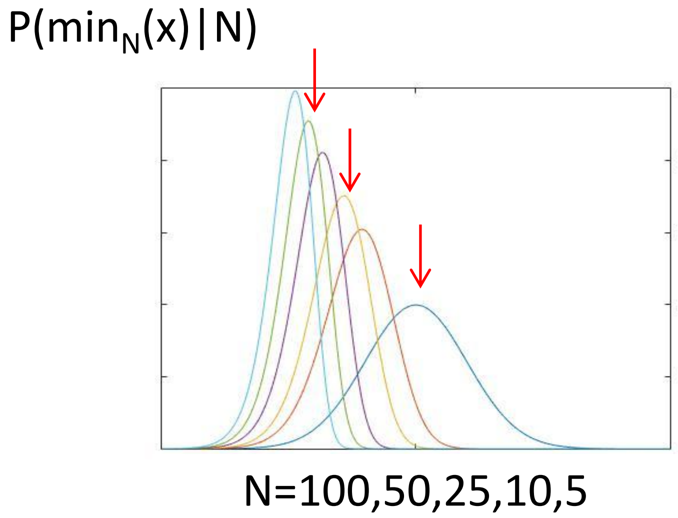


N=100,50,25,10,5

Learning performance assessment

This situation is known in ML as **optimism bias**: the expected value of variable obtained by min'ing a sample of other r.v. always **underestimates** the mean of that r.v.: E[minerr]<E[err]





Learning performance assessment

Why the above discussion is simplified and very rough, it is in fact quite accurate for the following situation:

- Consider a hypothetical ML model f_θ(x) for a classification y(x)=0|1 which is very bad – that is for any θ and x it essentially produces a random guess at y(x) consisting of 0 and 1 with probability 50%. The true EPE of that model is clearly EPE=50%.
- If we didn't know about that, we could try to "optimize" θ* by considering a number of different models θ and choosing the one with the best "accuracy". As each model θ basically randomly guesses yi(xi) of all xi in the training data, the average number of correct guesses of such models will be close-to-Normally distributed with mean 50% and variance depending on how many points there was in the training dataset.
- Just as we saw before, selecting the "best" model θ* will produce an impression that f_θ(x) actually learns and has the accuracy better than 50%! In fact, the more models are min'ed over, the larger the performance gain of such "best" selected model is going to be!
- Of course, should we try that model on any **new data**, its performance will drop immediately close to 50%! In ML we say that the model **fails to generalize**!

Learning performance assessment

Example: you can consider this example at home with Matlab or Octave: X here represents the result of choosing the best out of 100 random data models xk for some training data – X=max $x_{1..100}$. E[X]>E[xk] by nearly 2.5STD !

MATLAB-sim (opt. bias):

```
x=randn(100,1000);
disp(mean(x(:)))
X=max(x,[],1);
disp(mean(X(:)))
```

RESULT:

```
mean(x): -7.917E-4
mean(X):  2.5033
```

QUESTIONS FOR SELF-CONTROL

- What is the curse of dimensionality?
- How many data points are needed to cover a [0,1] cube uniformly with density $\Delta x=0.1$ in each direction in p=10 dimensions?
- What is the average nearest-neighbor distance of n=100,000 uniformly distributed data points in [0,1] cube in p=10 dimensions? (Rough estimate will suffice)
- What MSE (as in error) stands for? Define via formula.
- Explain why the result of learning a relationship Y=f(X) from a sample of its examples should be considered random even if the relationship f(.) itself is deterministic?
- What do we mean when we say "average over random training data"?
- Explain how the KNN estimate of a value $y_0=f(x_0)$ using a random selection of examples $\{(x_i,y_i=f(x_i))\}$ is random?
- Derive the bias-variance decomposition formula for MSE.
- Describe qualitatively how the bias and the variance of estimate f^(x) changes with growing number of dimensions of x, p?
- What does the bias of estimate f^(x) stand for?
- What does the variance of estimate f^(x) stand for?

- Why predictive capability of linear model is higher than that of KNN in high dimensions p?
- Define the additive error model via its general formula.
- What is parametric estimation of functions? Give example.
- What is nonparametric estimation of functions? Give example.
- Write down the general formula for linear basis expansion function estimators.
- How can linear basis expansion be estimated by using linear regression?
- Give some examples of restricted function estimators.
- What is Gaussian kernel? Write down the formula.
- What is local linear regression? Write down the formula.
- Explain bias-variance tradeoff.
- How does the bias and the variance of f^(x) change with respect to model complexity?
- How one should choose a model’s complexity?
- What is EPE and TPE? Illustrate the difference between the two.
- Why can’t we use the training error TPE to select a model’s complexity?
- What is optimism bias?

ADVANCED

Weighted regression methods

- In the context of additive error model, generally we assume that variance of errors ε does not depend on X (so called homoscedastic errors).
- Minimizing sum-square errors then emerges as a natural criterion for data model estimation.
- Errors ε that are non-uniform, that is whose variance depends on X (so called heteroscedastic errors), can be rather simply included as well by weighing different terms in square-loss function in relation to input x (weights themselves may need to be estimated iteratively as part of optimization):

$$L(Y, f(X)) = E[\sigma(x)^{-2}(y - f(x))^2]$$

Log-likelihood loss function

- Alternative to least squares and a more general theoretically loss function is related to Maximum Likelihood Estimation or MLE model fitting (Advanced, Probability)
- In there, the loss function is defined as the log of the probability of the training data being produced by the generative model f_θ(x):

$$L(\theta) = \sum_{i=1}^{n} \log P_\theta(x_i, y_i) \to \max$$

Log-likelihood loss function

- For additive error models with the error distributed according to a homoscedastic normal distribution, the square-loss and the log-likelihood loss trivially are the same

$$P_\theta(y \mid x) = N(f_\theta(x), \sigma^2) \Rightarrow$$

$$L(\theta) = -\frac{n}{2}\log(2\pi) - n\log\sigma - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - f_\theta(x_i))^2$$

Log-likelihood loss function

- A more interesting example is multi-class classification P(y|x), where MLE yields the following loss function also known in ML as **the cross-entropy loss**

$$L(\theta) = \sum_{i=1}^{n} \log P_\theta(y_i \mid x_i)$$

Log-likelihood loss function

- Cross-entropy has a number of advantageous properties in relation to other classification loss functions, such as being very smooth and relatively less sensitive to statistical outlier or anomalies.

$$L(\theta) = \sum_{i=1}^{n} \log P_{\theta}(y_i \mid x_i)$$

Bayesian estimation and penalty-restricted estimators

- Penalty restricted estimator methods express our belief that a solution should have a certain type of smooth behavior. That can be cast also in terms of Bayesian estimation (Advanced, Probability) by using prior distribution over models:

$$\mathrm{Prior}_\lambda(f) = \exp(-\lambda J(f))$$

- MAP with Normal additive errors then immediately yields PRSS as discussed in the main lecture.