UCK358E – INTR. TO ARTIFICIAL INTELLIGENCE

SPRING ‘23

LECTURE 2

MACHINE LEARNING BASICS

Instructor: Asst. Prof. Barış Başpınar

Learning Algorithms: Tasks

- A learning algorithm is an algorithm that performs a task by using data.
- A learning problem usually has three components: Task, Performance and Experience
- Here are some common learning tasks
  - **Classification**: Find a function that maps input data to one of k categories
    - Example: Object recognition
  - **Classification with missing inputs**: Find a function or a collection of functions that maps input data to one of k categories, where the input vector might have missing values
    - Example: Medical diagnosis
  - **Regression**: Find a function that predicts a continuous output
    - Example: Predict expected amount an insured person will make
  - **Transcription**: Observe some unstructured data and transcribe it into discrete, textual form
    - Example: Image translation, speech recognition

Learning Algorithms: Tasks

- Here are more learning tasks
  - **Machine Translation**: Convert from one language to another
    - Example: Translation for natural languages
  - **Structured Output**: Any task that involves the output as a vector with important relationship among its elements
    - Example: Break down a sentence into nouns, verbs, etc, image captioning
  - **Anomaly Detection**: Flag unusual or atypical objects
    - Example: Spam filtering, credit card fraud detection
  - **Data Completion**: Estimate missing values in data
    - Example: Product recommendation
  - **Denoising**: Recover a clean example from a corrupt one
    - Example: Image and video denoising
  - **Density Estimation**: Compute the $p(\boldsymbol{x})$ that generated the data
    - Example: Estimate the structure of a PGM

Learning Algorithms: Performance

- We need a quantitative measure P to measure how well our algorithm is performing on task T
- For classification and translation tasks, we measure the performance by accuracy,
  - Just divide the number of correctly classified examples to number of all examples
- Performance is evaluated on *test data*, data that has never been seen by the algorithm before
  - This set is usually different from the training data
- Sometimes selecting the correct performance criteria can be a complicated task itself
  - For translation, should we measure the accuracy on word by word basis or a sentence by sentence basis?
  - For regression, should we penalize small medium frequency mistakes, or large rare mistakes

Learning Algorithms: Experience

- Many learning algorithms are categorized as supervised or unsupervised based on the data that experience
  - Roughly speaking, unsupervised algorithms try to determine $p(\boldsymbol{x})$ from $\boldsymbol{x}$
  - Whereas the supervised algorithm try to determine $p(\boldsymbol{y} \mid \boldsymbol{x})$ from $(\boldsymbol{x}, \boldsymbol{y})$
- However, sometimes the lines between the two categories get blurry
  - In semi-supervised algorithms, where the data is only partially labelled
  - Unsupervised algorithms can also be used for supervised learning
    - Simply estimate $p(\boldsymbol{x}, \boldsymbol{y})$ and then marginalize
  - And vice versa:
    - Use chain rule $p(x) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1})$ to turn an unsupervised problem to a collection of n supervised problems

Learning Algorithms: Linear Regression Example

- Let's use a linear regression example to clarify these subjects
- In linear regression, our model is as follows:

$$\hat{y} = \boldsymbol{w}^{\top}\boldsymbol{x}$$

  - Where $\boldsymbol{w} \in R^n$ is a weight vector. Hence the task is to find $\boldsymbol{w}$ that does well according to some performance measure.

- A popular type of performance measure for these type of tasks is the mean squared error

$$\text{MSE}_{test} = \frac{1}{m}\sum_i \left(\hat{y}^{(\text{test})} - y^{(\text{test})}\right)^2$$

- The experience is gained by observing the dataset $\left(\boldsymbol{X}^{(\text{train})}, \boldsymbol{y}^{(\text{train})}\right)$
  - It can be shown that the optimal solution is found by $\nabla_{\boldsymbol{w}}\text{MSE}_{train} = 0$

$$\boldsymbol{w} = \left(\boldsymbol{X}^{(\text{train})\top}\boldsymbol{X}^{(\text{train})}\right)^{-1}\boldsymbol{X}^{(\text{train})\top}\boldsymbol{y}^{(\text{train})}$$

Learning Algorithms: Linear Regression Example

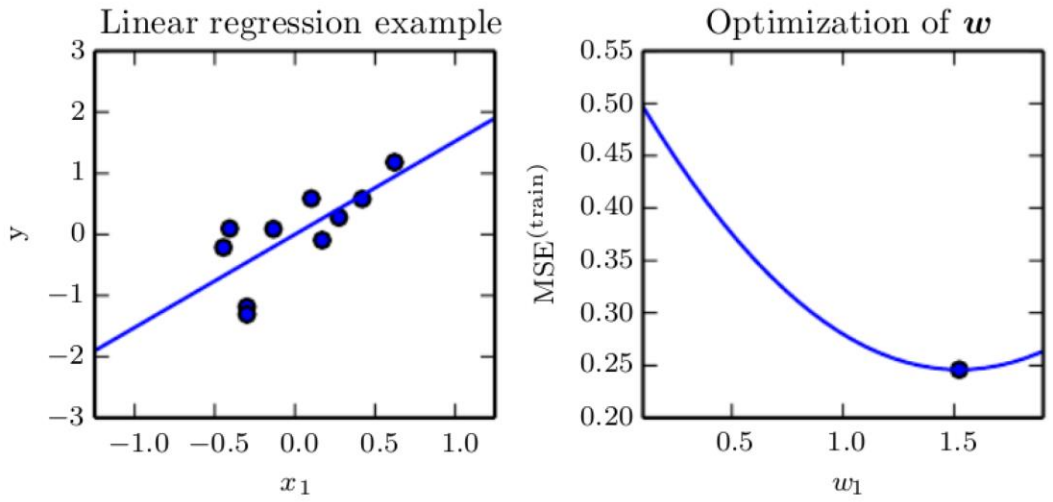


Figure 5.1: A linear regression problem, with a training set consisting of ten data points, each containing one feature. Because there is only one feature, the weight vector $\boldsymbol{w}$ contains only a single parameter to learn, w_1. *(Left)* Observe that linear regression learns to set w_1 such that the line $y = w_1 x$ comes as close as possible to passing through all the training points. *(Right)* The plotted point indicates the value of w_1 found by the normal equations, which we can see minimizes the mean squared error on the training set.

Capacity, Overfitting and Underfitting: Generalization

- A central challenge in machine learning is to perform well on new, **previously unseen** inputs.
  - This property is known as generalization. The error on the test data is referred to as *generalization error*
- But how can we say something about the test data, if all we see is training data?
  - If these datasets were generated arbitrarily, we can't...
  - However, if they come from the same distribution $p(\boldsymbol{x}, y)$, then we can say something! For instance, their mean should be equal
- In general, we want to find a $\boldsymbol{w}$ such that:
  I. Make the training error small
  II. Make the gap between training and test error small

  - Unfortunately, we rarely do both good at the same time
  - Failing on I, is known as underfitting, our model's capacity is not sufficient to capture $p(\boldsymbol{x}, y)$
  - Success on I but failing on II, is known as overfitting, our model's capacity is so high that it becomes overtuned to work only on training data

Capacity, Overfitting and Underfitting: Generalization

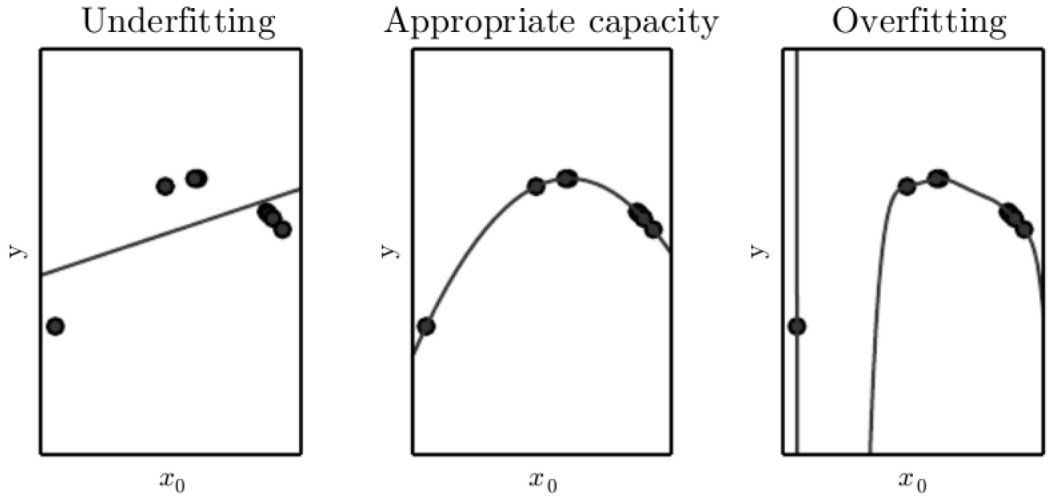


Figure 5.2: We fit three models to this example training set. The training data was generated synthetically, by randomly sampling x values and choosing y deterministically by evaluating a quadratic function. (*Left*) A linear function fit to the data suffers from underfitting—it cannot capture the curvature that is present in the data. (*Center*) A quadratic function fit to the data generalizes well to unseen points. It does not suffer from a significant amount of overfitting or underfitting. (*Right*) A polynomial of degree 9 fit to the data suffers from overfitting. Here we used the Moore-Penrose pseudoinverse to solve the underdetermined normal equations. The solution passes through all of the training points exactly, but we have not been lucky enough for it to extract the correct structure. It now has a deep valley in between two training points that does not appear in the true underlying function. It also increases sharply on the left side of the data, while the true function decreases in this area.

Capacity, Overfitting and Underfitting: Capacity

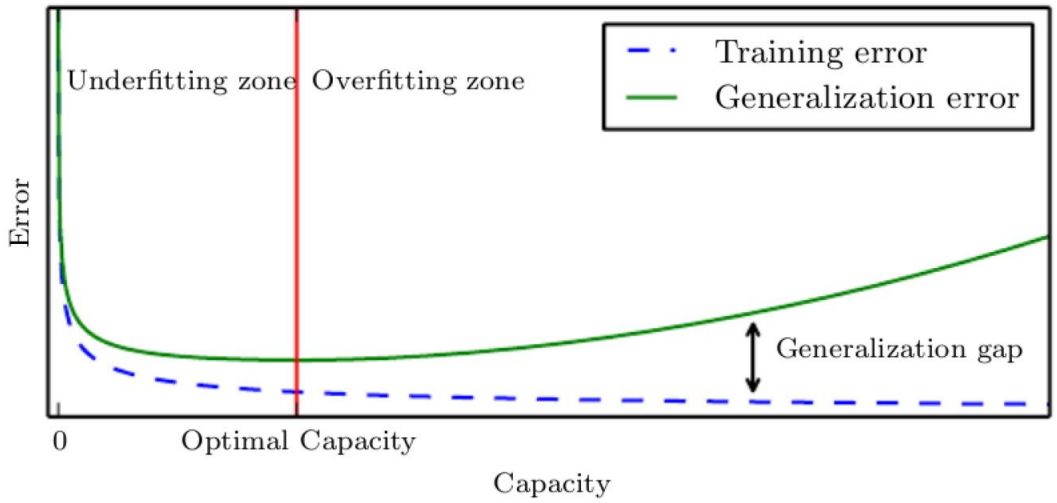
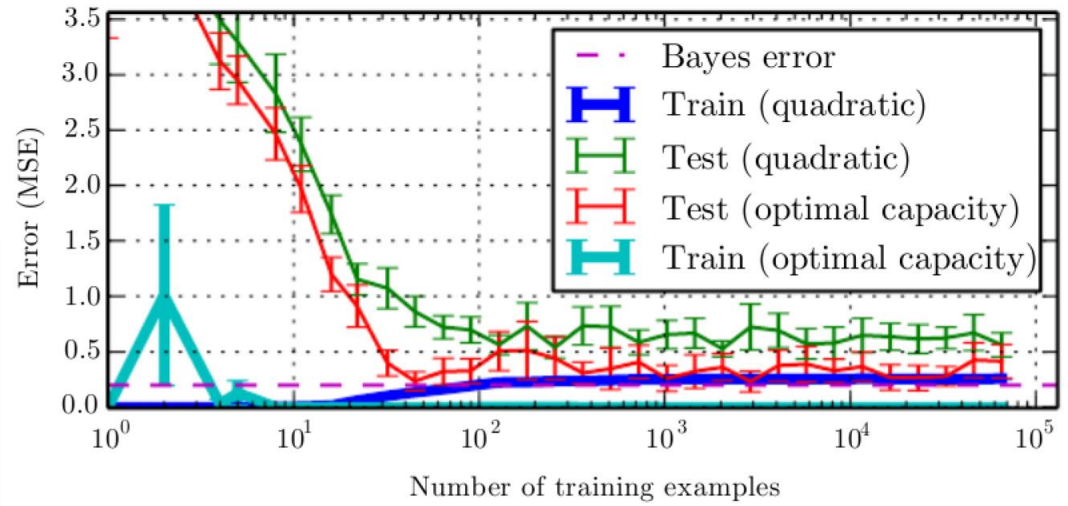


Figure 5.3: Typical relationship between capacity and error. Training and test error behave differently. At the left end of the graph, training error and generalization error are both high. This is the *underfitting regime*. As we increase capacity, training error decreases, but the gap between training and generalization error increases. Eventually, the size of this gap outweighs the decrease in training error, and we enter the *overfitting regime*, where capacity is too large, above the *optimal capacity*.

Capacity, Overfitting and Underfitting: Capacity



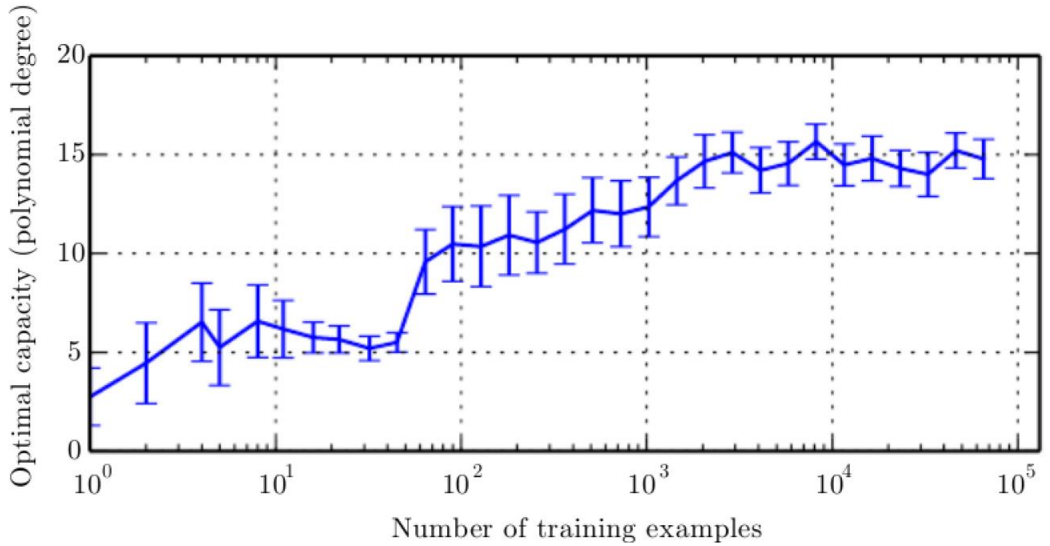
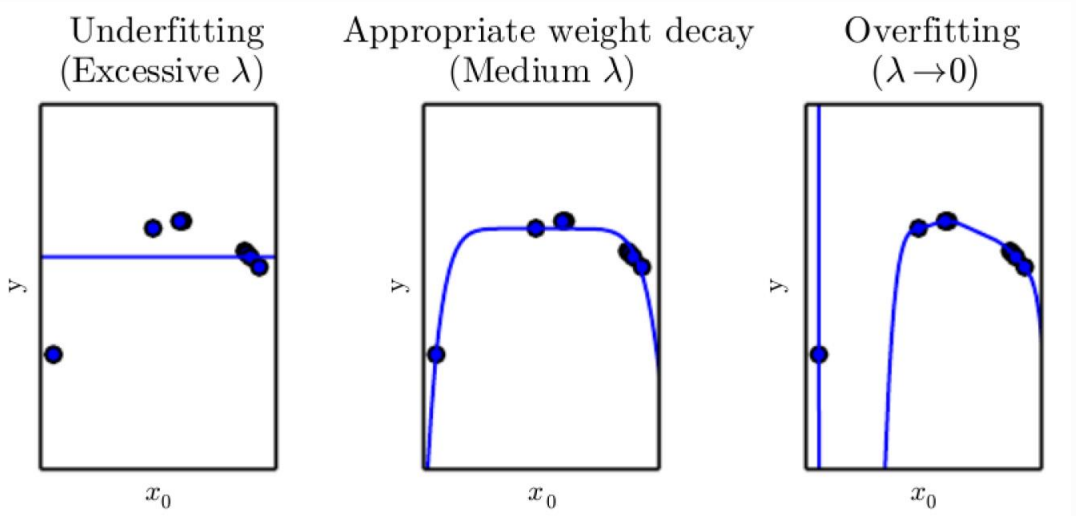


Figure 5.4: The effect of the training dataset size on the train and test error, as well as on the optimal model capacity.

Capacity, Overfitting and Underfitting: Regularization

- Regularization is any modification we make to the learning algorithm to reduce generalization error.
  - The most common form of regularization is to modify the loss function so that larger parameters are penalized

$$J(\boldsymbol{w}) = \mathrm{MSE}_{\mathrm{train}}(\boldsymbol{w}) + \lambda \boldsymbol{w}^{\top}\boldsymbol{w}$$



  - There are other forms of regularization, for instance ‘dropout‘ (randomly dropping some of the weights in your model) is very popular in deep learning

Hyperparameters and Validation Sets

- Hyperparameters are the parameters of our model that controls the capacity, as well as the behaviour of the algorithm
  - Examples: degree of a polynomial, number of layers in a neural net, structure of a PGM...
  - These are usually set by domain experts
- Can we 'learn' those parameters as well?
  - We can try optimizing them, this is called *hyperparameter optimization*
- However, using the whole training data does not make sense
  - The optimization procedure would simply select the parameters that result in highest capacity (i.e. overfitting)
- Similar to how we separated the whole data into test and training, we can further divide the training data into two disjoint subsets
  - The smaller part that is used for learning the hyperparameters is called the validation set

Estimators, Bias and Variance: Estimation

- How can we quantify the concept of overfitting/underfitting?
  - Classical statistics and estimation theory can help with that
  - We will take the frequentist view for the rest of this section
- Point estimation is an attempt to provide a single best estimate for a quantity of interest. We denote the point estimate of $\boldsymbol{\theta}$ as $\widehat{\boldsymbol{\theta}}$
- Let $\{x^{(1)}, \ldots, x^{(m)}\}$ be a set of m i.i.d. data points. A point estimator or statistic is any function of the data:

$$\hat{\boldsymbol{\theta}}_m = g(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)})$$

  - $\{x^{(1)}, \ldots, x^{(m)}\}$ is generated randomly, hence $\widehat{\boldsymbol{\theta}}_m$ is a random variable, even though $\boldsymbol{\theta}$ is not
  - We are interested in analyzing the relationship between $\widehat{\boldsymbol{\theta}}_m$ and $\boldsymbol{\theta}$, both for finite m and $m \rightarrow \infty$

Estimators, Bias and Variance: Bias

- The bias of an estimator is defined as:

$$\text{Bias}(\hat{\boldsymbol{\theta}}_m) = \mathbb{E}(\hat{\boldsymbol{\theta}}_m) - \boldsymbol{\theta}$$

  - The expectation is taken over the data
  - An estimator is *unbiased* if Bias($\widehat{\boldsymbol{\theta}}_m$) = 0 and *asymptotically unbiased* if $\lim_{m\to\infty}$ Bias ($\widehat{\boldsymbol{\theta}}_m$) = 0

- Let $x^{(i)}$ be generated from a Bernoulli distribution with mean θ. Then the estimator $\hat{\theta}_m = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$ is unbiased
  - The same estimator is also unbiased when $x^{(i)}$ are generated from a Gaussian distribution with mean μ
  - However, the estimator $\hat{\sigma}_m = \frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)} - \hat{\mu}_m\right)$ is only asymptotically unbiased. We can make it unbiased by setting $\hat{\sigma}_m = \frac{1}{m-1}\sum_{i=1}^{m}\left(x^{(i)} - \hat{\mu}_m\right)$

- It might seem like unbiased estimators are preferable, but this is not necessarily true, we will soon see that biased estimators possess some interesting properties

Estimators, Bias and Variance: Variance and Standard Error

- Another interesting property of an estimator is how much it varies as a function of the data sample. This is called *variance*

$$\text{Var}(\hat{\theta}_m) = E\left[(\hat{\theta}_m - E[\hat{\theta}_m])^2\right]$$

  - Low variance estimators give similar outputs irregardless of the sampled data. Variance is an excellent measure for quantifying the uncertainty in our estimations.
  - For instance, for Gaussian distributions we can say that our estimate of the mean $\hat{\mu}$ falls into the interval of

$$\left(\hat{\mu}_m - 1.96\sqrt{\text{Var}(\hat{\theta}_m)}, \hat{\mu}_m + 1.96\sqrt{\text{Var}(\hat{\theta}_m)}\right)$$ with 95% probability.

  - Hence ideally, we want both low bias and low variance.

- For Bernoulli distribution, variance of the estimator is $\frac{1}{m}\theta(1-\theta)$
  - The variance decreases as m increases, this is a common property of popular estimators

Estimators, Bias and Variance: Bias-Variance Decomposition

- Bias and variance are two difference sources of error in an estimator
  - Bias measures the expected deviation from the true value.
  - Variance measures deviation from the expected estimator value among different subsets of data.

- Should we choose a model with low bias or low variance?
  - The answer is given by the key idea that mean squared error can be decomposed as follows

$$\text{MSE} = \mathbb{E}[(\hat{\theta}_m - \theta)^2] = \text{Bias}(\hat{\theta}_m)^2 + \text{Var}(\hat{\theta}_m)$$

  - Hence for a fixed MSE, there is a trade-off between variance and bias
  - In general, increasing capacity increases the variance and decreases the bias.

Estimators, Bias and Variance: Bias-Variance Trade-off

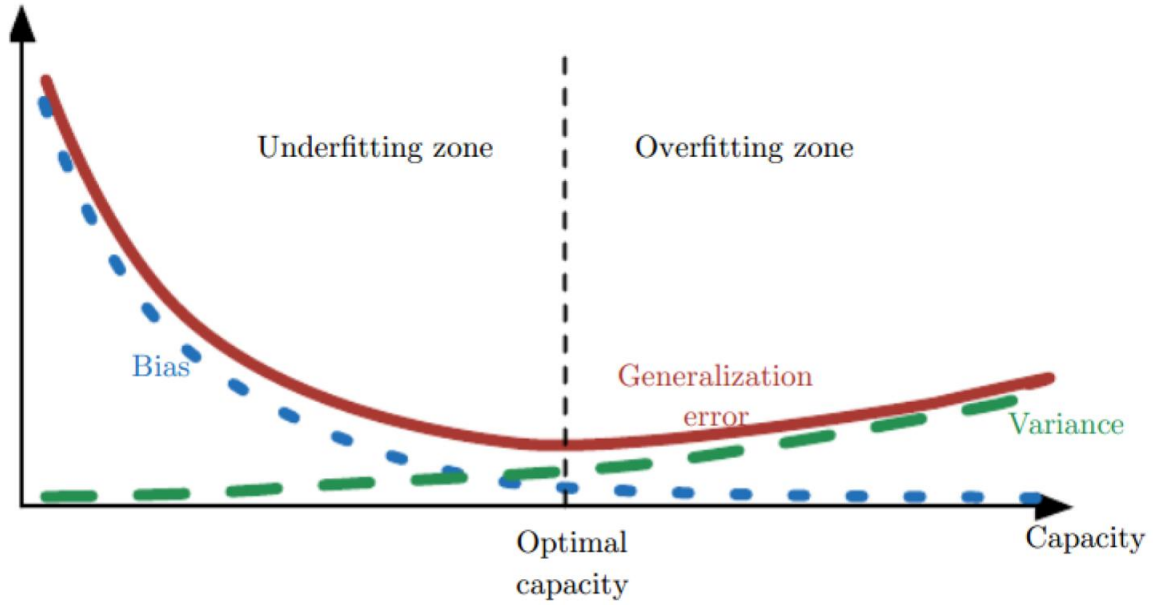


Figure 5.6: As capacity increases (x-axis), bias (dotted) tends to decrease and variance (dashed) tends to increase, yielding another U-shaped curve for generalization error (bold curve). If we vary capacity along one axis, there is an optimal capacity, with underfitting when the capacity is below this optimum and overfitting when it is above. This relationship is similar to the relationship between capacity, underfitting, and overfitting, discussed in Sec. 5.2 and Fig. 5.3.

Supervised Learning Algorithms: Probabilistic Supervised Learning

- The supervised learning is mainly about estimating the probability distribution $p(y \mid \boldsymbol{x})$
  - To use MLE for this job, we first define a family of probability distributions $p(y \mid \boldsymbol{x}; \boldsymbol{\theta})$
- Linear regression problem can be expressed by the family:

$$p(y|\boldsymbol{x}; \boldsymbol{\theta}) = \mathcal{N}(y; \boldsymbol{\theta}^\top \boldsymbol{x}, \boldsymbol{I})$$

  - It can be shown that in this case θ_{ML} has a closed form expression, which is the same as least squares solution!

- For binary classification, we can use the logistic sigmoid function:

$$p(y = 1|\boldsymbol{x}; \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \boldsymbol{x})$$

  - No closed form solution in this case, we need to use numerical optimization methods.

Supervised Learning Algorithms: Parametric Supervised Learning

- Not all supervised learning algorithms are probabilistic. We can simply define a parametric family of deterministic models $y = f(\boldsymbol{x}; \boldsymbol{w})$ and use some loss function to find an optimal $\boldsymbol{w}^*$
- One of the most famous approaches to this problem is the Support Vector Machines (SVMs)
  - For binary classification, we let $f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$
  - If $f(\boldsymbol{x}) > 0$ we predict that sample belongs to class 1 and vice versa
- SVM is useful for two reasons
  - Solving for optimal $\boldsymbol{w}$ is a convex optimization problem
  - It can be shown that:

$$\boldsymbol{w}^\top \boldsymbol{x} + b = b + \sum_{i=1}^{m} \alpha_i \boldsymbol{x}^\top \boldsymbol{x}^{(i)}$$

- The second part is the key results. We can replace $\boldsymbol{x}^T \boldsymbol{x}^{(i)}$ by any inner product and we can still use the same algorithm!
  - This is how we generalize to nonlinear classifiers: $b + \sum_{i=1}^{m} \alpha_i k(\boldsymbol{x}, \boldsymbol{x}^{(i)})$
  - The k is called a $kernel$

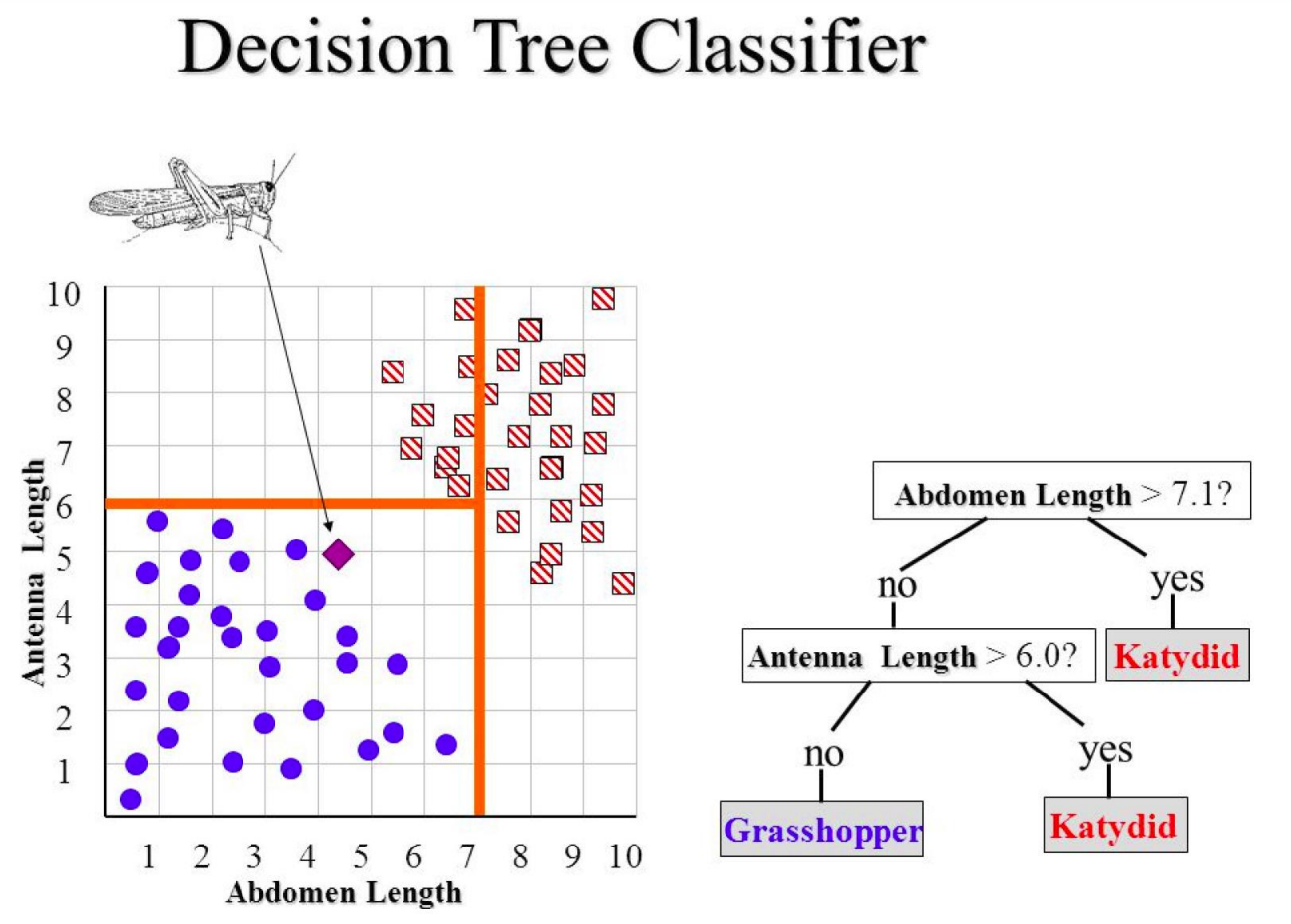
Supervised Learning Algorithms: Nonparametric Supervised Learning

- For most supervised learning algorithms we need to fix the form and number of parameters in the model beforehand.
  - Can we also learn those from the model?
  - Models that allow these are called *nonparametric*, in some sense, they let the data speak for itself

- One of the simplest nonparametric classifier is *k*-nearest neighborhood method
  - Fix a positive integer *k* (yes even nonparametric models have parameters)
  - For a new point, find the nearest *k* samples from the dataset, compute their class/output averages and simply predict this value for the new point

- There are other popular nonparametric classifiers as well, decision trees, random forests etc.

Supervised Learning Algorithms: Nonparametric Supervised Learning



Unsupervised Learning

- There are unsupervised learning algorithms for dimensionality reduction, e.g. PCA
  - PCA takes unlabeled and difficult to interpret data and transforms it into a much more manageable form

- Another popular form of unsupervised learning is clustering
  - How can we segment the data in a meaningful way?
  - k-means clustering is a very popular algorithm for this job.

- Generating association rules is another task that can be solved via unsupervised learning
  - An association rule is an unsupervised learning method which is used for finding the relationships between variables in the large database.

References

- I. Goodfellow, Y. Bengio and A. Courville, “Deep Learning”, 2016.