Contents

Chapter 1

Introduction to Machine Learning

Chapter based on UIUC CS 498 Fall 2020 8/26 Lecture: `http://slazebni.cs.illinois.edu/fall20/`

1.1 History of AI Techniques

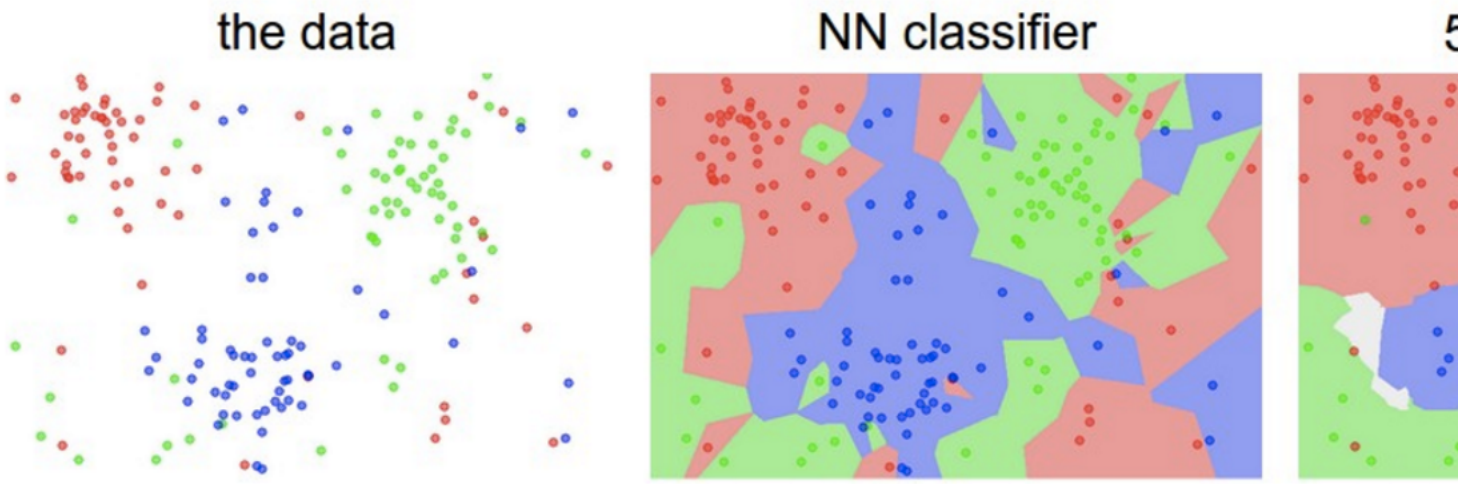
Old Techniques Before statistical learning, 'old-fashioned AI' was the norm. The main idea was to program expertise into the agent. For example, to build a translator using old AI techniques, one would incorporate many different language rules based on experts. This would require putting in many different rules for each type of word, for each type of sentence for each language. While there was some success with such expert engines, they generally never worked.

Statistical Learning Instead of creating expert agents, the modern idea was to program the agent the ability to improve performance based on experience. The agent should be able to gain 'experience' from training data or demonstrations. This technique is much more flexible. For example, to build an expert agent to recognize an everyday object like a chair, one would have to come up with a specific definition that generally fits all chairs. Alternatively, using statistical learning techniques, we can show an agent many examples of different types of chairs and the agent can 'learn' what defines a chair without a human having to actually define it. By learning, we mean that we want to optimize the performance of the agent on training data and then hope that it *generalizes* to unseen examples. This 'generalization' is not theoretically given and is a current area of research.

1.2 General Learning Procedure

Learning Pipeline Training samples –¿ Feature Extraction –¿ Training –¿ Learned Model
The general learning procedure first starts off with a group of training samples (not necessarily labelled). Historically, before the actual training begins, some type of *feature extraction* has to take place. Feature extraction usually involves some type of transformation of the data into the input form that goes into model. For example, a form of feature extraction for natural language processing could involve removing commonly used words. After the feature extraction takes place, the training process takes place and the result is the learned model. To test the quality of the learned model, the model is evaluated (the input is passed through the model without 'saving' any of the information from the test data) on a test set which includes data that the model has not seen before during training.

Figure 1.1: k=1 vs k=5 nearest neighbors



Basic Supervised Learning Framework Mathematically, the framework described above can be seen as follows: $y = f(x)$ where y is the output, f is the prediction function and x is the input. Training or learning means given a training set of labelled examples $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, a predictor f is created. Testing means applying f to a new test example x and output the predicted value y.

1.3 Simple Classification Models

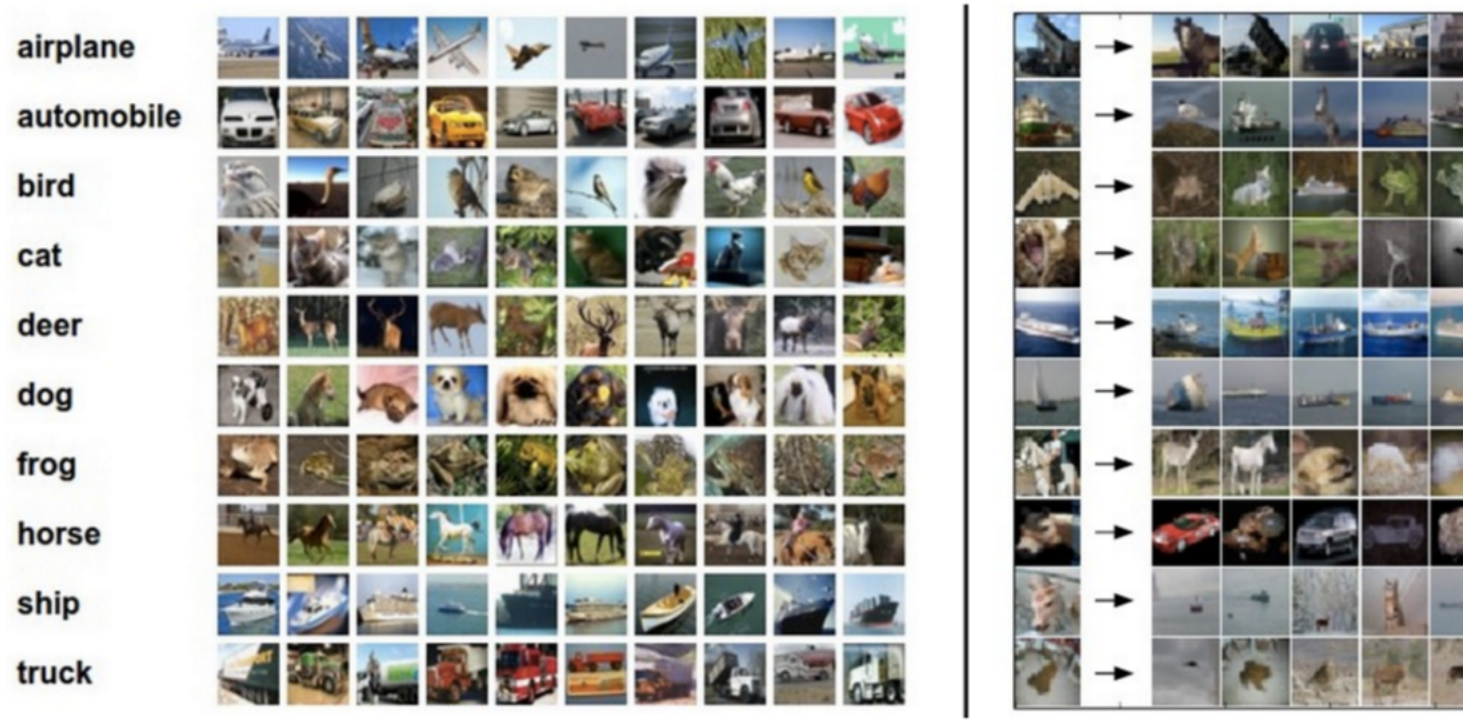
Nearest Neighbor Classifier Training points are labelled with a class. The label of a test point is the label of the training example nearest to the test point. Nearest can mean Euclidean distance, Manhattan distance, etc. This is a simple classifier since it only needs the distance function and no training is required. A variant of nearest neighbor, is k-nearest neighbor where for a new point, find the k closet points from training data. Higher k-nearest neighbor is generally more robust to outliers and can create more complex decision boundaries as shown in the image below. Nearest neighbor methods does have many limitations. An example of running k-nearest neighbors on pixels of images is seen below.

Linear Classifier For linear classifiers, we want to find a linear function to separate the classes. Linear classifiers are parametric meaning there are parameters that represent the (ideally a compressed form of) training data. Linear classifiers have as many parameters as training points. Each training point x_i, has a weight w_i. To determine where classes are separate (note that this linear classifier can only handle two classes), we use the following parametric equation to identify a hyperplane: $f(x) = sgn(w_1x_1 + w_2x_2 + \ldots + w_Dx_D + b) = sgn(w\dot{x} + b)$. Points that are one side of the hyperplace are classified as one class and points on the other side are classified as the other. Note that there will be 'mis-classified' points whose actual label is different than the label assigned by the classifier.

At this point we have not discussed how we learn the parameters of the linear classifier. Learning parameters of different models will be the subject of many of the next chapters.

Pros and Cons of Nearest-Neighbor and Linear Classifiers Pros of nearest neighbor methods include that it is simple to implement, decision boundaries are not necessarily linear, works for any number of classes and it is a nonparametric method (no parameters to learn). The cons are that it requires are

Figure 1.2: Nearest neighbor on images. A simple distance metric does not capture the real differences between images. One would have to engineer a quite complicated distance function to accurately use nearest neighbor for images. It is easier to learn functions that represent the images.



Left: Example images from the CIFAR-10 dataset. Right: first column shows a few test images and top 10 nearest neighbors in the training set according to pixel-wise difference.

good distance function and is slow. Linear classifier pros include that it is a low-dimensional parametric representation and is very fast at test time. Cons are that it only works for two classes, we have to train the classifier and if the data are not linearly separable (we cannot draw a straight line through the data) then the method is not accurate (we will expand on this later).

1.4 Taxonomy of Statistical Learning Problems

Classification is a form of *supervised* learning, where we are given the labels for our training data. Regression and structured prediction are also forms of supervised learning. The difference between these three problems is the form of the output. Classification output is discrete (like names of animals), regression output is continuous and structured prediction has complicated structures like parsing trees.

Supervised Learning So far we have seen classification, where the output values we are predicting are discrete. In **Regression** problems, the output values are continuous. In **structured prediction**, the outputs are more complicated structures like parsing trees.

Unsupervised Learning *Unsupervised* learning is a situation where there is unlabelled data as input and the goal is to learn some structure of the data. The goal is less clearly defined than in supervised learning. Examples of unsupervised learning include clustering, where the goal is try to find groups of similar data points, quantization or data compression where the goal is to encode the data into a more compact form, and dimensionality reduction where the goal is to discover a lower-dimensional surface on which the data lives. Another type of unsupervised learning is concerned with learning the data distribution which can be solved with density estimation (finding a function that approximates the probability density of the distribution and learning to sample where you produce samples that mimic the training distribution.

Other Forms There are forms of learning in-between unsupervised and supervised. *Semi-supervised* learning contains labels for a small portion of the training data and *weakly supervised* has noisy labels or labels are not exactly for the task of interest. In *self-supervised* learning, the model uses part of the data to predict other parts of the data. In *reinforcement learning*, the model learns from rewards in a sequential environment. Reinforcement learning is the learning behind many games. In *active learning*, the learning algorithm can choose its own training examples or ask a 'teacher' for an answer on selected inputs. *Lifelong learning* is focused on learning over time. There are many other subfields of learning but these are some of the major ones.

Chapter 2

Deep Learning Theory

Notes based on blog by Desh Raj https://desh2608.github.io/.

2.1 Optimization

Neural networks can be viewed as trying to minimize a loss function. The reason this is difficult is because the loss function is non-convex. Non-convex functions can have many local minima, saddle points, flat regions, and varying curvature which makes non-convex optimization at least NP-Hard (https://www.cs.cornell.edu/courses/cs6787/2017fa/Lecture7.pdf). If this is the case, why can deep networks find approximately reasonable solutions?

Almost all neural networks are trained by moving the parameters in the direction of a non-zero gradient. The goals of such descent are to find a critical point $\nabla = 0$ and to find local optimum $\nabla = 0$ and $\nabla^2 > 0$ (aka Hessian is positive semi-definite).

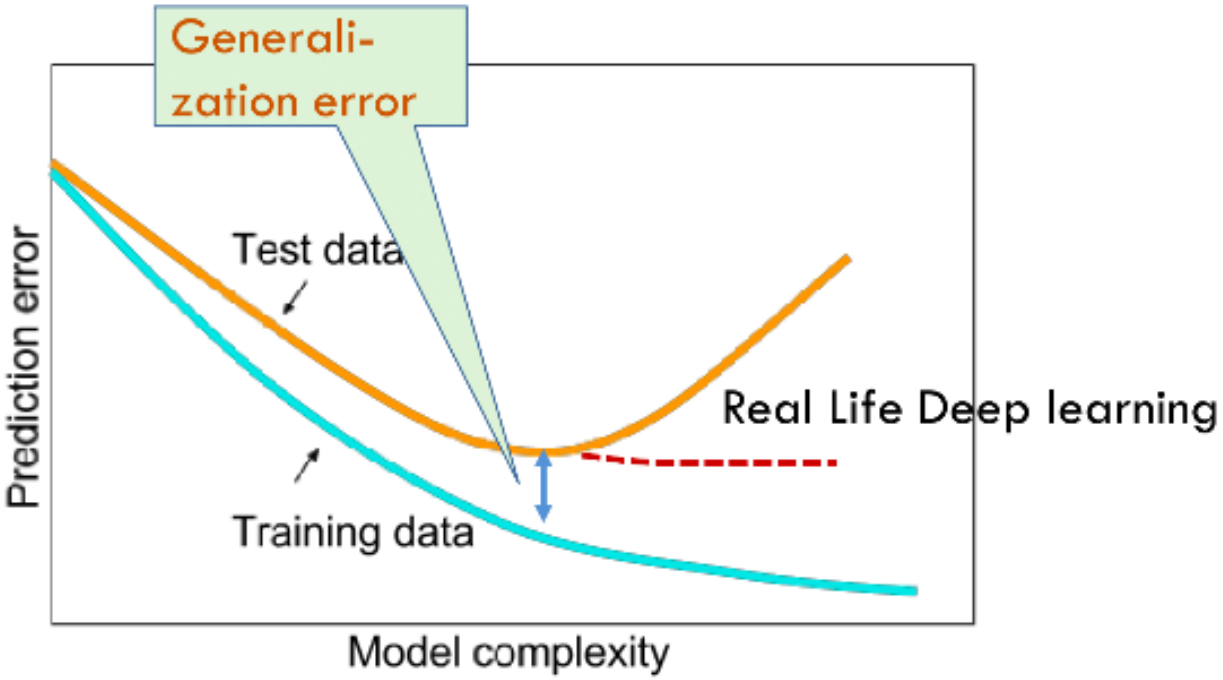
Critical Points Parameter θ, usually update in the following form: $\theta_{n+1} = \theta_n - \eta \nabla f(\theta_n)$. The hyper-parameter in the update equation is the learning rate, η. We know that we want it to be small but how do we know that η is small enough? We utilize the Hessian aka ∇^2. Suppose there exists some β such that $-\beta I \leq \nabla^2 f(\theta) \leq \beta I$. Then a higher β means ∇^2 varies more so the learning rate should be lower. It can be more formally proved that in $O(\frac{\beta}{\epsilon^2})$, θ will arrive at a critical point.

Saddle Points One issue that such a descent may run into is the issue of saddle points. Several results (http://proceedings.mlr.press/v40/Ge15.pdf, https://arxiv.org/pdf/1602.04915.pdf, https://arxiv.org/pdf/1703.00887.pdf) show that gradient descent can be done by evading saddle points. The main ideas from these papers are that there are many saddle points but it is hard to converge to one and while Hessians can be used to avoid the saddle points, we do not actually have to because a noisy form of gradient descent converges to a local optimum in a polynomial number of steps. The last paper discusses how perturbed gradient descent does a better job of escaping saddle points than regular gradient descent.

2.2 Generalization

What is impressive of most modern deep learning frameworks is that many generalize very well to test cases even when the number of parameters is greater than the number of training samples. This is counter-intuitive because one would think that deep neural networks would overfit with a small number of samples.

A common 'folklore' experiment (discussed in cs.princeton.edu/~rlivni/files/papers/LivnComputational.pdf) 'is to show that for sufficiently over-specified networks, global optima are ubiquitous and in general



computationally easy to find.' To understand this conceptually, think of fixing a 2 layer neural network with n hidden nodes. We provide it random inputs and collect the outputs. We then use these input/output pairs to train randomly initialized (2 layer) neural network. With only 2 layers, this would be quite difficult but it becomes easier when we increase the number of hidden nodes.

Capacity The capacity of a learning model can be viewed as the complexity of training samples that it can fit. A quadratic regression will have more capacity than linear regression. It can be proved that the following is true:

Test Loss - training loss $\leq \sqrt{\frac{N}{m}}$ where N is the effective capacity and m is the number of training samples. There are other measures such as Rademacher complexity and VC dimension that relate error to capacity. However, these fail for deep neural networks because the upper bounds from these measures are greater than 1.

Excess Capacity From the figure we see that in general deep neural networks are able to generalize relatively well. For it was believed that the performance of deep neural networks did not necessarily mean high capacity and that a combination of stochastic gradient descent and regularization eliminates the excess capacity of a neural network. However, this was proved wrong in a paper from 2017 (`https://arxiv.org/abs/1611.03530`). The experimental findings of the paper include that a classic convolutional neural net, like Alexnet, could be trained with random labels and still achieve high accuracy on training data. The paper proved that 'there exists a two-layer neural network with RELu activations and $2n + d$ weights that can represent any function on a sample size n in d dimensions. Recent work has shown that excess capacity can also exist in Kernel methods.

2.3 Other notions of Generalizability

There are some other correlations of good generalizability with flat minima and noise stability implying compressibility. The idea of flat minima is an old one but noise stability is relatively new. The idea of noise stability starts with measuring capacity using Gaussian noise. What happens is that zero-mean Gaussian noise is added at an intermediate output of a neural network which is reduced in higher layers. The capacity of the network to fit this random noise can be measured. If a network has the capacity for the noise then it is considered to have noise stability. Noise stability implies compressibility, which is defined as 'given a network C with N parameters and some training loss, compression means obtaining a new network C that has fewer parameters and the training loss effectively remains the same which would mean that the new network would have better generalization capability. There is a more formal definition of noise capacity involving the distribution of singular values when noise is added to a neural-network.

2.4 Bounding Capacity

PAC-Bayes The PAC-Bayes theorem informally bounds expected error of a classifier chosen from a distribution Q in terms of KL-Divergence from a priori fixed distribution P. KL-Divergence is a metric that compares the similarity between probability distributions:

$$KL(p||q) = \mathbb{E}(logp(x) - logq(x)) = \sum_{i=1}^{N} p(x_i)(logp(x_i) - logq(x_i))$$

PAC-Bayes relates to the following bound:

$$KL(\hat{e}(Q, S_m)||e(q)) \leq \frac{KL(Q||P) + log\frac{m}{\delta}}{m-1}$$

where $\hat{e}(Q, S_m)$ is the empirical loss of Q with respect to some i.i.d sample S_m and $e(Q)$ is the expected loss. PAC-Bayes bounds the test error of a binary variant of MNIST in 16-22% where the actual bound is around 3%.

Compressibility Arora et al introduced boundaries using compressions where the loss is bounded by q parameters having at most r discrete values.

2.5 Depth of Neural Networks

The proofs for these are relatively involved but it can be shown that 'radial functions can be approximated by depth-3 networks but not with depth-2 networks and that functions expressible by $\theta(k^3)$-depth networks of constant width cannot be approximated by $O(k)$-depth networks with polynomial width. Recent work might also suggest that increasing depth can speed up optimization.