Summary

General challenges

- Model complexity++ => Training Error– & Test Error– then: (sweet spot!) Test Error++
  - *Training set*: Fitting the models
  - *Validation set*: Determine hyperparameters
  - *Test set*: General assesment of the model
- **Curse of dimensionality**: as dimensions grow, data points more sparse
- **Bias-Variance** tradeoff: Error = Variance + Bias^2
  - **Bias** measures average estimation with true function
  - **Variance** measures dependency of classifier on random sampling in training set
- **Conclusion**:
  - Match the model complexity to the data resources, not to the target complexity

Nearest Neighbour methods

- Compute distance between new point and all samples
- Pick k neighbours that are nearest to x (majority vote to decide)
- *k << High Variance, Low Bias & k >> Low Variance, High Bias*
- **Pros**: Good performance, effective in low dimension data
- **Cons**: Memory requirement and costly to compute distances

Decision Trees

- **if** Stop growing tree when unique label else split recursively nodes by best information gain
- **Entropy**: **-pi*log2(pi)** where pi is proportion of label in dataset
- **Information gain**: **Gain(D, A) = Ent(S)— Ent(Sv)*|Sv|/|S|** *where Sv is subset of dataset D with values A*
- **Gini impurity**: **1- pi^2** (replacement for Entropy, more sensitive to equal probabilities)
- **Prune tree** by removing unnecessary branches, using:
  - *validation set* to get un-biased pruning
  - *test set* to see how the pruned tree generalize on new data

Regression

- *Goal: Predict target associated to any arbitrary new input*
- **Least Squares**: Minimize squared error between target and input
- **RANSAC**: Robust fit of model to data set S which contains outliers
- **k-NN**: To predict Y from X, take k closest points to X in training data and take the average of the responses
  - f(x) = 1/k* yi
  - Larger values of k provide a smoother and less variable fit (lower variance!)
  - In higher dimensions k-NN often preforms worse than linear regression.
- **PARAM vs Non-PARAM**:
  - *PARAM* better if close to the true form of *f* or *high dimension*
  - Since Parametric are more interpretable, more prefered if error is similar or slighly lower
- **Ridge Regression**: Reduce useless features to almost zero with *shrinkage penalty*
- **The Lasso**: Can reduce useless features to exactly zero with *l1-norm of shrinkage penalty*

- **Effect on MSE, Variance, Bias**: As we increase *shrinkage penalty* (lambda or s)
- Variance slowly decreases then decreases rapidly
- Bias stays the same then rapidly increases
- MSE decreases to a minima then rapidly increases
- **Interpretation**: As we increase lambda (penalty), variance steadily decreases and bias increases. Training error always decreases, but test error decreases at first, then as we go over a tipping point, the models gets too simple and loses all features (reduced to 0) => high test error. For s, it does the contrary for bias and variance.

Probalistic Reasoning

- **Probility Methods** make results interpretable, and define a unified ML theory
- **Bayes decision theory**:
- yMAP = argmax P(x|y)P(y) where P(x|y) is **likelihood distribution** and P(y) is **prior distribution**

Probalistic Learning Framework

- **Maximum Likelihood Estimate**: Maximizing likelihood of data
- Problem: More features => More difficult to model
- Solution: **Naive Bayes** => All features are regarded as independent
- **Maximum a Posteriori**: Model parameters are probabilities
- To classify data points, we need to know max ci P(xi|c=ci)
- In generative approaches, each distribution will not be affected by the data from other class
- We can try to fit a **mixture** of K distributions:
  - **K-Means** finds centroids with neighboring points
    - ++ Guaranteed to converge
    - − Sensitive to initial conditions
    - − Euclidian distance favors spherical clusters
- Expectation Maximization w/ **Mixture of Gaussians** * ++ Can describe complex multi-modal probability distributions * − Data points are smoothly used to update all parameters
- **Conclusion**:
  - Better ways of comparing models involve estimating the posterior of the model given the data by integrating over all possible values of the parameters. This way, more complex models will have a much better fit to the data but only for a much more restricted domain of the parameter space, and therefore, more simple models have a chance to win the comparison.

Classification with Separating Hyperplanes

- y = sign(xi*wi)
- Finding the best weights to separate data
- **Perceptron Learning**: Incremental learning
  - Initialize weights to zeros(training_dim+1) and add 1-element to samples
  - wi ← wi -+ x (only change when output is wrong!)
- **Delta Rule**: Minimize ||t −wTx||^2
  - wi ← w − gradw||t − w Tx||^2 (for each sample)
  - wi ← wi + (t − wT x)xi
- Minimization of structural risk = Maximization of the margin

Support Vector Machines

- We can separate almost everything in higher dimensions
  1. Transform Input in high-dimension w/ function
  2. Choose unique separating hyperplane
  3. Classify new data using hyperplane
- Popular kernels
  - Linear: xT.y + 1
  - Polynomial: (xT.y + 1)^p
  - Radial: exp -(x-y)$^{2/2\text{sigma}}$2
  - Sigmoid: tanh(kxT.y - lamb)
- Kernel trick can solve inefficiency by avoid calculating higher dimenions
- Maximize **i—0.5* i j ti tj (xi)T (xj)** under **0 i C** i:
  1. C = +inf for NO slack, C € R for allowing slack
  2. Choose kernel function and find i
  3. Classify data point x via i ti K(x,xi) > 0
- **Advantages**: Work well with small data + Fast + Generalize well

Artificial Neural Networks

- 1-Layer NN can implement **any linear function**, 2-Layer NN **any function**
- Perceptron/delta impossible as we lose info of weight in each neuron
- **Trick**: Using continuous threshold-like functions
- *Goal*: Minize error as function of *all* weights
  1. Compute direction where total error increases most
  2. Back-propragate weights in opposite direction wi ← wi − * E/ wi
- In deep networks, gradients **vanish** as they become really small
  - We can avoid by having non-squashing activation (Rectified Linear Unit aka ReLU)
- **Convolutional Networks**: Alternating convolution and sub-sampling layers (pooling)
- **Dropout**: Randomly *mute* neurons in network to avoid overfitting
  - Dropout helps the network to generalize better and increase accuracy since the (possibly somewhat dominating) influence of a single node is decreased by dropout.

Ensemble Learning

- Combining knowledge from **multiple** classifiers

Bagging

- Use replicates of training set by sampling with replacement
- Weak classifiers must be *better than random* and *enough independence* between them
- **Bias– Variance++ ==> Variance–**

Forest

- Bagging + Random feature selection at *each* node
- Two kinds of randomness:
- Generating bootstrap replicas
- Feature selection at each node

- Trees are less correlated i.e even higher variance between learners
- Suited for multi-class problem

Boosting

- The selection method encourages classifiers to be diverse, de-correlated
- Creating single strong classifier from a set of weak learners
  1. Apply learner to weighted samples
  2. Increase weights of misclassified examples
- Adaboost Algorithm:
  1. Set uniform weight to all samples
  2. Train T classifiers and select the one minimizing training error
  3. Compute **reliability coefficient**: t =log(t/1 - t)
  4. Update weights using **reliability coefficient**
  5. Normalize all weights

Dimensionality Reduction

Principal Component Analysis

- Reduce the number of variables by selecting components of larger varainces
- The eigenvectors help find the *direction* of the data in lower dimensions

Discriminant function

- *Idea*: Vectors in same class are localized close to each other
- Computes **closeness** in vectors by computing different types of similarities
  - **Angle**: Computes cosine between vector (closer to 1, more likely to be in certain label)
- **Subspace Method**: exploit localization of pattern distributions
  1. *Idea*: Samples in same cass are localized in same subspace
  2. Determine class of new input by calculating **projection length** to subspace