Lecture 2

* Structured label
  + {1,2,…,K}^N
* Multiclass
  + {1, 2, 3, …, K}
  + Only 1 class per instance
* Multilabel
  + Multiple labels per instance
* Questions
  + How to define space
  + Pick the right model
* Hypothesis space
  + Function space (input -> output)
* For n binary input function, there are 2^n^n possible functions to define (count all possible output configurations)

Lecture 3

* For label space Y and instance space X, there are |Y|^|X| possible functions f(x)
* The considered functions in this space is the hypothesis space
  + E.g., hypothesis space for all Boolean functions of 4 variables could be only conjunctive functions: x1 and x2, x1 and x3, …
  + E.g., m-of-n rule functions
  + E.g., neural networks, decision trees, nested collections, etc.
  + E.g., Linear classification
* Hypothesis: a specific function in the hypothesis space == model
* Goal: find best hypothesis in hypothesis set that fits the data
  + Hope it generalizes well to data never seen before
* Overfitting
  + Due to high complexity of model
  + Making choices based on very little data
  + Good performance on training data, but does not generalize well to testing data
* Prevent overfitting
  + Simpler, less-expressive models
  + Regularization
    - Promote simpler models – quantitatively define what a simple model is first
  + Data perturbation – add noise in training
  + Stop the optimization process earlier – don’t stop at the best one (that fits perfectly on training data)
* How to learn: how to find a good hypothesis/model form the hypothesis space?
  + Determine a hypothesis space.. use brute-force to find the best hypothesis/model/function
  + But the hypothesis space could be infinite
    - Use gradient descent, or other techniques
    - Local search
      * Start with a linear threshold function (random?)
      * See how well the function is and compute a gradient
      * Correct it
      * Repeat until converge
* K-Nearest Neighbor
* Algo
  + Learning: store all training instances
  + Predication of an input instance x: Average of k nearest neighbors to x (add a distance weight)
* K, p are hyperparameters
  + A model with hyperparameter k (and specific way of computing distance (e.g., p-norm)) is a hypothesis in the k-NN hypothesis space
  + If K is too high, then we choose the majority label every time – underfitted
  + If K is too low, then we look too closely at a specific data point – overfitted
* Distance
  + 1-norm: Manhattan distance
  + 2-norm: Euclidean distance
  + Hamming distance: Number of features that have a different value (used in discrete cases)
* Hyperparameter tuning
  + Best hyperparameter is determined by empirical studies based on dataset
  + Methods
    - Test/dev (validation)/train
      * KNN: Store data points in test for the actual nearest neighbor algo
      * Use data in dev set to see how accurate labels of dev set are predicated given current hyperparameters (p and K)
      * Adjust p and K and reiterate
      * Pick best p and K. Retrain model on (Train U Dev)
      * Finally, test performance on test set
    - N-fold cross validation
      * Split data into 2 sets (train and test)
      * For each hyperparameter K:
        + Perform cross validation on train:

Split train into N groups

N-1 are train and 1 is dev

Train data on N-1 and measure performance on dev

Continue until all folds are exhausted

Compute average accuracy of each iteration

* + - * Pick most accurate hyperparameters
      * (Optional) Retrain model with the entire train set
      * Evaluate model on Test set
* Decision boundary
  + KNN effectively divides space with decision boundaries
  + KNN algo builds an implicit function based on these boundaries
  + 1-NN with 2-norm distance will have linear boundaries
* Curse of Dimensionality
  + As number of dimensions increases, most of the volume of the “sphere” will be far away from the center
  + Which explains why at higher dimensions, distance between all points and a random point in a data set will be very similar (there is more space far away from the center of a sphere (e.g., the random point))
  + Data is not uniformly distributed at higher dimensions
  + Solution: reduce dimensionality
* Data preprocessing
  + Normalization
    - Rescale features so no one feature is the sole decider of the predicted label
    - Each feature x\_i = (x\_i – x\_i\_mean) / x\_i\_std\_dev
    - Do this for the entire dataset (train and test)

Lecture 4

* Decision Trees
  + Tree chars
    - Internal nodes are features
    - Branches are values
    - Leaves are labels
  + Decision boundaries can also be drawn for decision trees
  + Learning
    - Need all data in a batch before learning the tree (adding data points may change the tree structure)
      * In KNN, you can just add more data and feature queries will use the extra data points
    - Finding the minimum decision tree is NP-hard
    - Instead, we use a greedy approach to get close to optimal
    - Pick the feature with the lowest entropy
    - Entropy
      * In average, how many bits to send one character
      * #bits / length of message
    - To avoid overfitting, decide a max depth D. Once D is reached, pick the majority label in the current data subset

Lecture 6

* Linear model
  + Hypothesis space: H = {h | h(x) = w^T x + b}
  + Train: find best parameters (w, x)
  + Learning algorithms to derive parameters w, x
    - Perceptron
      * Label space = {-1, 1}
      * An online algorithm
        + Data given in a stream – processes one example at a time
      * Converges if data is separable – mistake bound
      * Learn by making mistakes
      * The algo
        + Initialize w = 0
        + For each epoch 1…T: (T -> hyperparameter)

For each (x, y) in D

~y = sgn(w^T x)

If ~y is not y: (equivalent to y(w^T x) <= 0)

w = w + nyx (n -> hyperparameter)

* + - * Mistake on positive:
        + w’ = w + x
        + w’^Tx = (w + x)^Tx = w^Tx + x^T x

w^T x becomes more positive for input x -> higher chance we will predict the right label for points near x next time

* + - * Only linearly separable functions can be learned
        + E.g., parity functions (XOR) cannot be learned
      * Convergence theorem
        + If the model is linearly separable, the model can learn parameters that separate all data in finite number of steps
        + The exact number of steps depends on the difficulty of the model (low difficulty -> high margin of data set)
      * This model does not perform well when data has noise (e.g., not linearly separable)
      * Margin of a hyperplane
        + Distance between the hyperplane and the closest data point
      * Margin of a data set
        + Maximum margin possible (distance between nearest point to best hyperplane)
      * Data set with larger margin is easier to classify (algo converges faster)