CSCC11 Week 4 Notes

Logistic Regression Recap:

- Decision boundary, denoted as $d(x)$, splits the points into 2 classes, C_1 and C_2.

$d(x)=0 \leftarrow$ Decision Boundary

$d(x)>1 \leftarrow$ Correspond to C_1
$d(x)<1 \leftarrow$ Correspond to C_2

- If a point is on the decision boundary, then we are uncertain about the class for that point.

- If d is a linear function, then we have a linear decision boundary.

- Logistic regression has a linear decision boundary.

$$P(C=1|x) = \sigma(w^Tx) = \frac{1}{1+\exp(-w^Tx)}$$

$$P(C=2|x) = 1-P(C=1|x) = 1-\sigma(w^Tx)$$

K-Nearest Neighbour:

- In KNN, we classify an unknown point with the most common class "around" this point.
  Note: "around" means k closest points

- Can be used for both regression and clarification

- The set containing the k closest points to x in the training data is called the k-neighbourhood of x, denoted as $N_k(x)$.

- Let $y \in \{-1, 1\}$. Then: $f(x) = \text{sign}\left(\sum_{i \in N_k(x)} y_i\right)$

  Again, $N_k(x)$ is the set of k closest neighbours.

- There are 2 popular distance formulas to use for calculating "closeness":

  1. Manhattan Distance:
     - Let $x_1, x_2 \in X$ have p numeric features.
     - Then, the Manhattan Distance formula is:

       $$\sum_{j=1}^{p} |x_{1j} - x_{2j}|$$

     - E.g. Say $x_1 = (3,4)$, $x_2 = (5,3)$

       The Mahattan distance is:

       $|3-5| + |4-3|$
       $= 2+1$
       $= 3$

  2. Euclidean Distance:
     - Again, let $x_1, x_2 \in X$ have p numeric features
     - Then, the Euclidean Distance formula is:

       $$\sqrt{\sum_{j=1}^{p} (x_{1j} - x_{2j})^2}$$

     - E.g. Say $x_1 = (3,4)$ and $x_2 = (5,3)$. Then, we get:

       $\sqrt{(3-5)^2 + (4-3)^2}$
       $= \sqrt{(-2)^2 + 1^2}$
       $= \sqrt{4+1}$
       $= \sqrt{5}$

- Note: Both the Manhattan and Euclidean distances are part of the Minkowski distance or Lm Distance.

  Let a and b $\in X$ and have p numeric features. Then, the Minkowski distance formula is:

  $$\|a-b\|_q = \left(\sum_{j=1}^{p} |a-b|^q \right)^{1/q}$$

  When $q=1$, we have the L_1 distance or Manhattan distance.

  When $q=2$, we have the L_2 distance or Euclidean distance.

- Algorithm:
  1. For each test data, i, calculate the dist btwn data i and each training data point.

  2. Rank the dist from smallest to largest.

  3. Select the smallest k points.

  4. Calculate the frequency of these k chosen points in their classes.

  5. Return the class with the highest frequency as the predicted label.

- For classification in g groups, a majority vote is used:

$$\hat{h}(x) = \arg\max_{l \in \{1, \ldots, g\}} \sum_{i : x^{(i)} \in N_k(x)} (y^{(i)} = l)$$

Furthermore, posterior probabilities can be calculated with:

$$\hat{\pi}_l(x) = \frac{1}{k} \sum_{i : x^{(i)} \in N_k(x)} (y^{(i)} = l)$$

- We choose k based on hyperparameter search in validation.

In general:
1. As $k \rightarrow \infty$, the decision boundary becomes smoother.
2. As $k \leftarrow 1$, the decision boundary becomes rough and is susceptible to noisy data.

Rule of thumb: $k < \sqrt{N}$, where N is the number of points.

- Final notes on kNN:
1. kNN is a lazy classifier. It has no real training step, it simply stores the complete data which are needed during prediction.

2. Its parameters are the training data.

3. kNN requires storing the whole training set.

4. As the number of points increase, it becomes more computationally expensive. However, when $k=1$, we don't need to train.

Furthermore, as the number of parameters increase with the number of training points, we call kNN a non-parametric model.

5. kNN is not based on any distributional or functional assumption and can, in theory, model data situtations of any complexity.

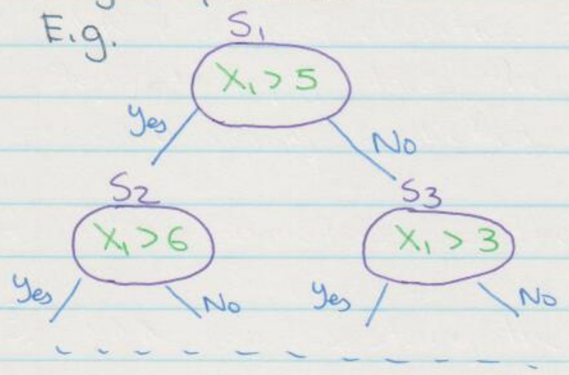
6. The smaller k is, the more rough/wiggly the decision boundary becomes.

7. The accuracy of kNN can be severely degraded by the presence of noisy or irrelevant features.

8. We often use the Euclidean distance to measure "closeness" but when the dimension is high, it will often become useless.

Decision Trees:

- Are usually binary trees but can be k-ary trees.
- The idea is to split the space in halves until we get a good prediction.
- E.g.



- Consider a binary tree:
  - Suppose it has M internal nodes. These are the split functions.
  - Suppose it has M+1 leaf nodes.
  - At an internal node, j, we define the split function $t_j(x) : R^d \to \{-1, 1\}$.

    If $t_j(x) = -1$, x is directed to the left child node
    If $t_j(x) = 1$, x is directed to the right child node

  - Leaf nodes have a categorical distribution over class label.

    Can represent as $P(y=c \mid \text{leaf node } j) = \frac{N_{jc}}{N_j}$

    where N_{jc} means the num of class c in node j and N_j means the num of points in node j.

    Note: $\frac{N_{jc}}{N_j}$ is the max likelihood estimate

- Learning the simplest/smallest decision tree is NP-hard or NP-complete problem.

  - We start from an empty decision tree.
  - We split on the best attribute.
  - Recurse

- To decide the best split value, we will choose it such that the data in the left/right children has the minimal possible uncertainty.

* There is a formula for conditional entropy, too.

$$H(y|X=x) = -\sum_y p(y|x) \log_2(p(y|x))$$

- Entropy is a measure of uncertainty.

$$H = -\sum_{c=1}^{k} p_c \log_2 p_c, \text{ where } p_c = P(y=c) \quad *$$

- Information gain (IG) is the reduction in entropy produced by partitioning the data according to a split test.

$$IG(y|x) = H(y) - H(y|x)$$

$$IG(D_j, t_j) = H(D_j) - \frac{N_L}{N_J} H(D_L) - \frac{N_R}{N_J} H(D_R)$$

NL is the num of data in the left child.
Nr is the num of data in the right child.

Note: Information gain is also called the mutual information of Y and X.

- E.g.

$X = \{\text{Raining, Not raining}\}$
$Y = \{\text{Cloudy, Not cloudy}\}$

	Cloudy	Not Cloudy
Raining	24/100	1/100
Not Raining	25/100	50/100

What is the entropy of cloudiness, given the knowledge of whether or not its raining?

Soln:

$$H(y|x) = \sum_{x \in X} P(x) \cdot H(y|X=x)$$

$$= P(X = \text{raining}) \cdot \overbrace{H(y|X=x)}^{\text{Conditional}} + P(X = \text{not raining}) \cdot \underbrace{H(y|X=x)}_{\text{Conditional}}$$

$$= \frac{25}{100} H(y|X=x) + \frac{75}{100} H(y|x=x)$$

$$\approx 0.75 \text{ bits}$$

$$H(y|\ \text{raining}) = -\sum_{y} P(y|x) \log_2(P(y|x))$$

$$= -\frac{24}{25} \cdot \log_2\left(\frac{24}{25}\right) - \frac{1}{25} \cdot \log_2\left(\frac{1}{25}\right)$$

$$\approx 0.24229$$

$$H(y|\text{Not raining}) = -\sum_{y} P(y|x) \cdot \log_2(P(y|x))$$

$$= -\frac{25}{75} \cdot \log_2\left(\frac{25}{75}\right) - \frac{50}{75} \cdot \log_2\left(\frac{50}{75}\right)$$

$$\approx 0.91829$$

How much info about cloudiness do we get by discovering whether it is raining?

Soln:

$IG(y|x) = H(y) - H(y|x)$

$= 0.25$

$$H(y) = -\sum_{c=1}^{k} P_c \cdot \log(P_c)$$

$= -P(\text{cloudy}) \cdot \log(P(\text{cloudy})) - P(\text{not cloudy}) \cdot \log(P(\text{not cloudy}))$

$= -\frac{49}{100} \cdot \log\left(\frac{49}{100}\right) - \frac{51}{100} \cdot \log\left(\frac{51}{100}\right)$

≈ 1

- Some attributes, such as age and height, are continous. In this case, to find the threshold for splitting, we do:

  1. Sort the continuous attribute in increasing order.
  2. Calculate the middle values between adj values.
  3. For each middle point, calculate the entropy.
  4. Choose the threshold with max reduction in entropy.

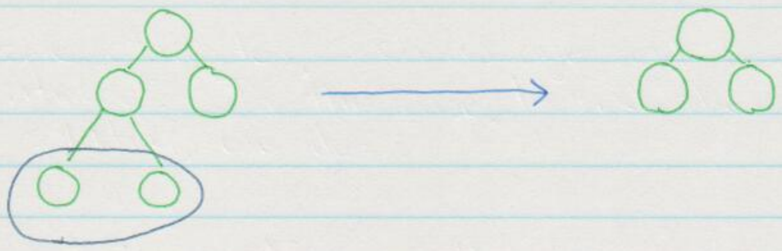
- As we grow the trees deeper, the model becomes more prone to overfitting.
  I.e. bias decreases while variance increases.

- There are 2 techniques we can use to reduce overfitting:
  1. Pruning the tree
  2. Ensemble method

- Pruning the tree:
  - One way we can prune the tree is by stop growing the tree if the info gain doesn't exceed a specific threshold. However, this is short sighted because a seemingly worthless split early on might be followed by a really good split later.

  - A better approach is to prune leaf nodes by combining them if the prediction accuracy on the validation set is not worse than the current accuracy.

  E.g.



If removing these 2 nodes don't worsen the accuracy, remove them.

Algo:
Start at the leaves and recursively eliminate splits.
1. Evaluate the performance of the tree on validation data.

2. Prune the tree if the performance does not worsen.

- Ensemble Method:
  - Also known as Random Forests
  - The idea is that we learn a bunch of trees and since each tree is randomly generated, each tree will provide a diff prediction. We aggregate the predictions by averaging.
  - Random forest is a bootstrapping method and does bootstrap aggregation/bagging.
  - Algo:
    1. For each tree:
       a) Bootstrap the data.
          I.e. Sample a subset of the training data with replacement.

       b) Build the tree using a subset of the data and only use a subset of features

    2. During prediction, each tree gives an output. We take the avg prediction. (aggregation)

  - Hyper parameters:
    1. Num of data to sample.
    2. Num of features to look at.
       Rule of thumb: If D is the number of features, we look at $\sqrt{D}$ features.
    3. Num of trees.

  - When we do bootstrapping, we increase the bias, although slowly, but we reduce the correlation of the trees and hence the variance.

- How to choose the hyperparameters:
  - Recall that we want to use the model to make the prediction. (Generalization)

  - First, we will use 1 of the following techniques to search over the hyperparameter space:

    1. Grid Search
       E.g. If the learning rate $\alpha \in (0, 1]$,
       we can do $\alpha = (0.1, 0.2, \ldots, 1)$.

    2. Random Search

    3. Grad Student Descent

  - Now, we'll go to validation.
    1. For a simple approach, we can use a metric that we care about.

       E.g. train using cross-entropy,
       validate using accurate

       The problem with this approach is that if we don't have many training data, it's hard to split.

    2. We can use K-Fold Cross Validation to bypass the limit/flaw of the first approach.

       Steps:
       1. Partition the training data into k partitions.
       2. For each partition, train on the remaining k-1 partitions.

3. Validate each partition.
   Say L_i is the validation of the ith partition.

4. Average L_i for overall performance.

- One special case of k-Fold Cross Validation is Leave One Out Cross-Validation (LOOCV).
  The idea behind this is with small training datasets, we have as much partitions as data points.
  I.e. If we have N training points,
  we have N partitions.
  The rest of LOOCV works like k-Fold Cross Validation.

  LOOCV is useful for small datasets but very expensive.

- In general, if there are m hyperparameters each taking C values, there are C^m models.
  If we do k-Fold Cross Validation, there are $k \cdot C^m$ models.

- While k-Fold Cross Validation is very simple and empirical way of comparing models, it has some issues:
  1. Can be time consuming and expensive.
  2. Because a reduced dataset is used for training, there must be sufficient training data so that all relevant phenomena of the problem exist in both the training and test data.
  3. It is safest to use a random partition to avoid the possibility that there are unmodeled correlations in the data.