Lecture 2:

Parameters: Things that you learn during trainings

The decision tree algorithm primarily learns two things:

1.the best feature to split on

2.the threshold for the feature to split on at each node

These are called parameters of the decision tree model. When predicting on new examples, we need parameters of the model.

Hyper-parameters: Things that you set to control the training

**Decision trees:**

~ We can also use decision tree for regression problems.

<https://ubc-cs.github.io/cpsc330/lectures/02_decision-trees.html#decision-tree-for-regression-problems>

~ Decision stump: A decision tree with only one stump (5:30). Not necessarily all leaf nodes are pure in this case.

~ If we let the decision tree grow without limit, we does NOT always end up with pure leaf nodes! (Imagine two features have the exactly same value, then how to separate them?)

Lecture 3 ML Fundamentals

Fundamental tradeoff of supervised learning:

~ As you increase the model complexity, E\_train tends to go down but E\_valid - E\_train tends to go up (The gap goes up but not exactly E\_valid goes up!!!!).

Also bias vs variance tradeoff (only either one will be high)

**Bias (high=underfitting):**

**the tendency to learn the same wrong thing or failing to learn something important**

**Variance (high=overfitting):**

**the tendency to learn random things irrespective of the real sign**

Lecture 4 knn

**k-NN:**

~ k-nn selects no feature at all, it's gonna use up all features to calculate the euclidean tree (The decision tree on the other hand may be working at some small sets of a feature) (E.g. a tree stump (with depth = 1) has one split and 1 good feature)

~ training error is 0 when k = 1 (always find the closest one which is itself!)

~ We can also use knn to do regression problem (take the average of k neighbours)

Pros of k-NNs for supervised learning¶

~ Easy to understand, interpret.

~ Simple hyperparameter k (n\_neighbors) controlling the fundamental tradeoff.

~ Can learn very complex functions given enough data.

~ Lazy learning: Takes no time to fit

Cons of k-NNs for supervised learning¶

~ Can be potentially be VERY slow during prediction time, especially when the training set is very large.

~ Often not that great test accuracy compared to the modern approaches.

~ It does not work well on datasets with many features or where most feature values are 0 most of the time (sparse datasets).

Curse of dimensionality (More features might result in bad results for k-nn!):

~ Affects all learners but especially bad for nearest-neighbour.

~ k-NN usually works well when the number of dimensions d is small but things fall apart quickly as d goes up.

~ If there are many irrelevant attributes, k-NN is hopelessly confused because all of them contribute to finding similarity between examples.

~ With enough irrelevant attributes the accidental similarity swamps out meaningful similarity and k-NN is no better than random guessing.

**SVM RBF (Support Vector Machine with RBF kernel)**

Superficially, SVM RBFs are more like weighted k-NNs.

~ The decision boundary is defined by a set of positive and negative examples and their weights together with their similarity measure.

~ A test example is labeled positive if on average it looks more like positive examples than the negative examples.

The primary difference between k-NNs and SVM RBFs is that

~ Unlike k-NNs, SVM RBFs only remember the key examples (support vectors). So it’s more efficient than k-NN.

~ SVMs use a different similarity metric which is called a “kernel” in SVM land. A popular kernel is Radial Basis Functions (RBFs) They usually perform better than k-NNs!

Support vectors¶

~ Each training example either is or isn’t a “support vector”.

~ This gets decided during fit.

~ Main insight: the decision boundary only depends on the support vectors.

Key hyperparameters of rbf SVM are

~ gamma (larger gamma → more complex)

~ C (larger C → more complex)