Multiple inputs $x$ (can be continuous, discrete or both).

Single discrete output $y$.

Goal: predict output on future unseen inputs.

From a probabilistic point of view, we are using Bayes rule:

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)} = \sum_{y'} p(x|y)p(y)$$
Voronoi Tessellation, Decision Surfaces

- For continuous inputs, we can view the problem as one of segmenting the input space into regions which belong to a single class, i.e. constant output.
- Such a segmentation is the “Voronoi tessellation” for our classifier.
- The boundaries between regions are the “decision surfaces”.
- Training a classifier == defining decision surfaces.

\[
\begin{align*}
\text{\textbf{K-Nearest-Neighbour}}
\end{align*}
\]

- Finally: a real algorithm!
- To classify a test point, chose the most common class amongst its \( K \) nearest neighbours in the training set.

\[
\text{Algorithm K-NN}
\]

\[
\begin{align*}
\text{c-test} & \leftarrow \text{KNN}(K, x\text{-train}, c\text{-train}, x\text{-test}) \\
d(m, n) & = \text{distance between } x\text{-train}(m) \text{ and } x\text{-test}(n) \\
n(n, l) & = \text{index of } l\text{-th smallest entry of } d(:, n) \ [\ast] \\
c(n, 1) & = c\text{-train}(n(n, 1)) \\
c\text{-test}(n) & = \text{most common value in } c(n, 1:K) \ [\ast\ast]
\end{align*}
\]

- If ties at \( \ast \) when \( l = K \), increase \( K \) for that \( n \) only.
- If ties at \( \ast\ast \), decrease \( K \) for that \( n \) only.
- confidence \( \approx \) (\#votes for class) / \( K \)
- Q: How should we select \( K \)? A: Cross-Validation (coming soon).

\[
\begin{align*}
\text{Probabilistic Model, Bayes Error Rate}
\end{align*}
\]

- Model original data as coming from joint pdf \( p(x, y) \).
- Classification == trying to learn conditional density \( p(y|x) \).
- Even if we get the perfect model, our error rate may not be zero. Why? Classes may overlap.
- The best we could ever do if our cost function is number of errors is to guess \( y^{*} = \arg\max_{y} p(y|x) \).
- (The error rate of this procedure is known as the “Bayes error”.)

\[
\begin{align*}
\text{More on K-NN}
\end{align*}
\]

- Typical distance = squared Euclidean \( d(m, n) = \sum_{d} (x^{m}_{d} - x^{n}_{d})^{2} \)
- If Euclidean distance is used, decision surfaces are piecewise linear.
- Trick: remember the \( K^{th} \) smallest distance so far, and break out of the summation over dimensions if you exceed it.
- In low-d with lots of training points you can build “KD trees”, “ball trees” or other data structures to speed up the query time.
- In high-d, save time by computing the distance of each training point from the min corner and using the “annulus bound”.
**Error Bounds for NN**

- Amazing fact: asymptotically, $\text{err}(1-\text{NN}) < 2 \text{err}(\text{Bayes})$:
  \[
  e_B \leq e_{1,NN} \leq 2e_B - \frac{M}{M-1}e_B
  \]
  this is a tight upper bound, achieved in the "zero-information" case when the classes have identical densities.

- For K-NN there are also bounds. e.g. for two classes and odd K:
  \[
  e_B \leq e_{K,NN} \leq \sum_{i=0}^{(K-1)/2} \binom{k}{i} \left[ e_{B}^{i+1}(1-e_B)^{k-i} + e_B^{k-i}(1-e_B)^{i+1} \right]
  \]


**Nonparametric (Instance-Based) Models**

- Q: What are the parameters in K-NN? What is the complexity?  
  A: the scalar K and the entire training set.
  Models which need the entire training set at test time but (hopefully) have very few other parameters are known as nonparametric, instance-based or case based.

- What if we want a classifier that uses only a small number of parameters at test time? (e.g. for speed or memory reasons)  
  Idea 1: single linear boundary, of arbitrary orientation  
  Idea 2: many boundaries, but axis-parallel & tree structured

**Example: USPS Digits**

- Take 16x16 grayscale images (8bit) of handwritten digits.
- Use Euclidean distance in raw pixel space (dumb!) and 7-nn.
- Classification error (leave-one-out): 4.85%.

Example 7 Nearest Neighbours

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>4</td>
<td>9</td>
<td>9</td>
<td>4</td>
<td>9</td>
<td>4</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

**Linear Classification for Binary Output**

- Goal: find the line (or hyperplane) which best separates two classes: 
  \[
  c(x) = \text{sign}(\mathbf{x}^\top \mathbf{w} - w_0)
  \]
  - $\mathbf{w}$ is a vector perpendicular to decision boundary
  - This is the opposite of non-parametric: only $d + 1$ parameters!
  - Typically we augment $\mathbf{x}$ with a constant term $\pm 1$ ("bias unit") and then absorb $w_0$ into $\mathbf{w}$, so we don’t have to treat it specially.
Fisher’s Linear Discriminant

- Observation: If each class has a Gaussian distribution (with same covariances) then the Bayes decision boundary is linear:

\[ w^* = \Sigma^{-1}(\mu_0 - \mu_1) \]

\[ w_0^* = \frac{1}{2}w^T(\mu_0 + \mu_1) - w^T(\mu_0 - \mu_1) \left[ \frac{\log p_0 - \log p_1}{(\mu_0 - \mu_1)^T\Sigma^{-1}(\mu_0 - \mu_1)} \right] \]

- Idea (Fisher’36):
  Assume each class is Gaussian even if they aren’t!
  Fit \( \mu_i \) and \( \Sigma \) as sample mean and sample covariance (shared).

- This also maximizes the ratio of cross-class scatter to within class scatter: \( (z_0 - z_1)^2/(\text{var}(z_0) - \text{var}(z_1)) \)

Linear Discriminants are Perceptrons

- The architecture we are using

\[ c(x) = \text{sign}[x^T w - w_0] \]

can be thought of as a circuit/network.

- It was studied extensively in the 1960s and is known as a perceptron.

- There is another way to train the weights, other than Fisher.

Algorithm perceptronTrain (Rosenblatt’56)

\[ w \leftarrow \text{perceptronTrain}(x\text{-train},c\text{-train}) \]

\[ w = \text{‘‘small’’ random values;} \]

\[ \text{do } \}

\[ \text{for n=1:N } \{ \text{if}(c\text{-train(n)} \neq \text{sign}[w^Tx\text{train(n)}]) \text{ then } \}

\[ w = w + c - \text{train(n)}^Tx\text{train(n)}; \text{ errors++; } \}\]

\[ \text{until(overflow=0)} \}

Perceptron Learning Rules

- Now: cycle through examples, when you make an error, add/subtract the example from the weight vector depending on its true class.

- Amazingly, for separable training sets, this always converges.
  (We absorb the threshold as a “bias” variable always equal to -1.)

- For non-separable datasets, you need to remember the sets of weights which you have seen so far, and combine them somehow.

- One way: keep the set that survived unchanged for the longest number of (random) pattern presentations. (Gallant’s pocket algorithm.)

- Better way: Freund & Shapire’s voted perceptron algorithm.
  Remember all sets and the length of time they survived.

- Perceptron, voted-perceptron, weighted-majority, kernel perceptron, Winnow, and other algorithms have a frumpy reputation but they are actually extremely powerful and useful, especially using the kernel trick. Try these before more complex classifiers such as SVMs!
Decision Trees

- What if we want more than two regions?
- We could consider a fixed number of arbitrary linear segments but even cheaper is to use axis-aligned splits (one dimension each).
- If these form a hierarchical partition, then the classifier is called a decision tree or (axis-aligned) classification tree.
- Each internal node tests one attribute; leaves assign a class.
- Equivalent to a disjunction of conjunctions of constraints on attribute values (if-then rules).

Learning (Inducing) Decision Trees

- Need to pick the order of split axes and values of split points.
  Many algorithms: CART, ID3, C4.5, C5.0.
- Almost all have the following structure:
  1. Put all examples into the root node.
  2. At each node: search all dimensions, on each one chose split which most reduces impurity; chose the best split.
  3. Sort the data cases into the daughter nodes based on the split.
  4. Recurse until a leaf condition:
     - number of examples at node is too small
     - all examples at node have same class
     - all examples at node have same inputs
  5. Prune tree down to some maximum number of leaves.
     (Possibly using a different impurity measure than for growing.)

Cost Function for Decision Trees

- Define a measure of “class impurity” in a set of examples.
  Push each example down the tree, how “pure” are leaves?
- Goal: minimize expected sum of impurity at leaves at test time.
- Two problems:
  1) We don’t know true distribution \( p(x, y) \).
  2) Search: even if we knew \( p(x, y) \) finding optimal tree is NP.
- So we will take a suboptimal (greedy) approach.

Impurity Measures

- When considering splitting data \( D \) at a node on \( x_i \), we measure:
  \[
  \text{Gain}(D; x_i) = I(D) - \sum_{v \in \text{split}(x_i)} \frac{|D_{iv}|}{|D|} I(D_{iv})
  \]
- Common impurity measures:
  Entropy: \( I(D) = -\sum_c p_c(D) \log p_c(D) \) (two classes)
  Misclass: \( I(D) = 1 - p_c^* \)
  Gini: \( I(D) = \sum_c \sum_{c' \neq c} p_c(D) p_{c'}(D) \)
\[
= \sum_c p_c(D)(1 - p_c(D))
\]
  (Gini is the average error if we stochastically classify with node prior)
- These often favour multi-way splits.
- One solution: normalize by “split information”:
  \[
  S(D) = -\sum_v \frac{|D_{iv}|}{|D|} \log \frac{|D_{iv}|}{|D|}
  \]
Restrict to Binary Splits

- A better solution is to always constrain ourselves to binary splits.
- For ordered discrete or real valued nodes, split is natural. Also easy to compute.
- For a discrete attribute with $M$ settings, looks like we need to consider $2^M - 1$ splits. But for two classes, there is a trick:
  1. Order the settings according to $p(c|x_i = m)$.
  2. Search exhaustively over $q$, grouping first $q$ and last $M - q$.
  3. Optimal split is one of those.

Real Valued Attributes

- For real valued attributes, what splits should we consider?
- Idea1: discretize the real value into $M$ bins.
- Idea2: Search for a scalar value to split on.
  Sounds hard! Lots of real values. But there is a trick:
  Only need to consider splits at midpoints between observed values.
  In fact, only need to consider splits at midpoints between observed values with different classes.
- Complexity: $N \log N + 2N|C|$

Overfitting in Trees

- Just as with most other models, decision trees can overfit.
  In fact they are quite powerful.
- eg: Expressive power of binary trees
  Q: If all input and outputs are binary, what class of Boolean functions can DTs represent?
  A: All Boolean functions.
- Hence we must regularize to control capacity.
- Typically we do this by limiting the number of leaf nodes.
  Formally, we define: $\Phi(T) = \sum_{\text{leaves}} I(l) + \alpha|\text{leaves}|$.
- Minimizing this for any $\alpha$ is equivalent to finding the tree of a fixed size with smallest impurity. (cf. Lagrange multipliers).
- Practically, we achieve this via pruning.
  Often we use Gini/Entropy to grow tree and Misclass to prune it.
**Pruning Decision Trees**

- Finding the “optimal” pruned tree.
  It can be shown that if you start with a tree $T_0$ and insist on using a rooted subtree of it, the following sequence of trees contains the optimum tree for all numbers of leaves:
  1. Let $U(\text{node}) = I(\text{node}) - I(\text{subtree-rooted-at-node})$
  2. Replace the non-leaf node with the smallest value of:
     $U(\text{node})/\text{leaves-below-node}$
     with a leaf node having majority class.
- Even after pruning, decision trees still have problems:
  - cannot capture additive structure (OR), for this MARS is better
  - cannot deal with linear combinations of variables

**DT Variants**

- **ID3 (Quinlan)**
  - split values are all possible values of $x_i$
  - $I(D)$ is entropy - no pruning
- **C4.5, C5.0 (Quinlan)**
  - binary splits
  - $I(D)$ is entropy - error-pruning
  - “rule simplification”
- **CART (Breiman et. al)**
  - binary splits
  - $I(D)$ is Gini
  - minimum-leaf subtree pruning

**Open Questions…**

- How do we chose $K$ in K-NN? (Cross-validation)
- How do we chose $T_{max}$ for decision trees? (Cross-validation)
- Can Fisher’s Discriminant overfit? (What do you think?)
- What about nearest-neighbour or tree-based models for regression as well as classification? (Good idea!)

Next class: Logistic regression, Neural Nets for Classification, Class-Conditional Models (Gaussian and Naive Bayes)