1. **Expressivity of Decision Lists.** Read the notes for Lecture 2, which describe the class of decision lists. Here we examine their expressive power.

(a) Explain why any conjunction (like $x_1 \land \bar{x}_2 \land x_3$) can be written as a decision list.

Solution: For each literal $\ell$ in the conjunction, just add the rule “if $\ell$ then negative”, followed by “else positive”.

(b) Explain why any disjunction (like $x_1 \lor \bar{x}_2 \lor x_3$) can be written as a decision list.

Solution: For each literal $\ell$ in the disjunction, just add the rule “if $\ell$ then positive”, followed by “else negative”.

(c) Give an example of a decision list that is not a conjunction or a disjunction.

Solution: One example is “If $x_1$ then positive, else if $x_2$ then negative, else if $x_3$ then positive, else negative”. This rule cannot be a conjunction or disjunction that ignores any of the three variables because for each variable there exists an example such that changing the value of that variable matters. Moreover it cannot be a conjunction or disjunction of all three variables since that would imply there is only one setting of $x_1, x_2, x_3$ that is positive, or that is negative, respectively. But neither of those is the case. (Note that the decision list “If $x_1$ then positive, else if $x_2$ then negative, else positive” can be written as a disjunction $x_1 \lor \bar{x}_2$).

2. **Learning Decision Lists.** Give a PAC algorithm for learning the class of decision lists. This is in the notes, but we want you to describe it and then explain why it is correct in your own words. Give a bound on the sample size and running time needed for finding a rule of error at most $\epsilon$ with probability at least $1 - \delta$.

Solution: See notes.

3. **Learning k-Decision Lists.** A “$k$-decision list” is just like a regular decision list, except the conditions $\ell_i$ are conjunctions of up to $k$ literals, rather than just being single literals. For example, the function “if $x_1 \land \bar{x}_2$ then positive, else if $x_2 \land x_3$ then negative, else positive” is a 2-decision list. So a regular decision list is a 1-decision list.

Give a PAC algorithm for learning $k$-decision lists whose running time and sample size is polynomial in $n^k$ (and so is polynomial in $n$ when $k$ is a constant). Hint: do it by reduction.

Solution: Create one variable $\ell_i$ for each of the $O((2n)^k)$ possible conditions. Then run the standard decision list algorithm over this new variable set.

4. **Expressivity of Decision Lists, contd.** Show that decisions lists are a special case of linear threshold functions. That is, any function that can be expressed as a decision list can also be expressed as a linear threshold function “$f(x) = + \text{iff } w_1 x_1 + \ldots + w_n x_n \geq w_0$.”
Solution: Let’s first suppose there are no negated variables in the decision list. So, the decision list looks like “if \( x_i \) then \( y_1 \), else if \( x_i \) then \( y_2 \), else ...”, where each \( y_j \in \{+, -\} \). Let \( L \) be the length of the list. In that case, we can use the linear threshold function:

\[
-w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + w_{n+1} > 0,
\]

where \( w_{i,j} = 2^{L-j} \) if \( y_j = + \), and \( w_{i,j} = -2^{L-j} \) if \( y_j = - \), and \( w_i = 0 \) if \( x_i \) was not in the list at all. Finally, \( w_{n+1} = 1 \) or \(-1\) depending on whether the last rule is “else +” or “else –” respectively. The point is that since the weights are dropping by factors of 2, the first variable on in the list will override all the ones below it.

If there are negated variables in the decision list, then we just want to view \( \bar{x}_{ij} \) as \( 1 - x_{ij} \), in the LTF above. For example, if \( n = 3 \), the decision list “if \( x_1 \) then +, else if \( \bar{x}_2 \) then –, else +” would become \( 4x_1 - 2(1 - x_2) + 1 > 0 \).

5. Decision Tree Rank. The \textit{rank} of a decision tree is defined as follows. If the tree is a single leaf then the rank is 0. Otherwise, let \( r_L \) and \( r_R \) be the ranks of the left and right subtrees of the root, respectively. If \( r_L = r_R \) then the rank of the tree is \( r_L + 1 \). Otherwise, the rank is the maximum of \( r_L \) and \( r_R \).

Prove that a decision tree with \( s \) leaves has rank at most \( \log_2(s) \).

Solution: Proof by induction on the depth of the tree. Base case: it’s true for a single leaf by definition of “rank”. General case: say the given tree has rank \( r \) and its left and right subtrees have rank \( r_L \) and \( r_R \) respectively. By induction, the number of leaves is at least \( 2^{r_L} + 2^{r_R} \), which is at least \( 2^r \) by definition of “rank”.

6. Expressivity of Decision Lists, contd., contd. Show that the class of rank-\( k \) decision trees is a subclass of \( k \)-decision lists. Hint: There are several different ways of proving this. If you get stuck, try just proving for \( k = 2 \) first.

Solution: Proof by induction on depth. Base case (a single leaf) is true by definition. General case: say the tree has rank \( k \) and its root contains variable \( x_i \). We know one of the subtrees has rank at most \( k-1 \) (without loss of generality, say this is the left subtree) and the other has rank at most \( k \). Inductively, say that \( L \) is a \((k-1)\)-decision list equivalent to the left subtree and \( L' \) is a \( k \)-decision list equivalent to the right subtree. To create a \( k \)-decision list for the entire tree, begin with the list \( L \) but with \( x_i \) appended onto each rule, followed by the list \( L' \). Notice that we do not need to append \( \bar{x}_i \) onto the rules of \( L' \) because every example with \( x_i = 0 \) exits through one of the rules of \( L \).