2 Exercise Session 2

Exercises on inductive bias, rule induction, genetic algorithms, and instance based learning.

2.1 Inductive Bias

Recall: in the lectures we defined the inductive bias of a learner L as the set of assumptions B for which it holds that, if B is true, L is guaranteed to return a model that equals the correct target hypothesis. This definition is useful because it can help us understand what a learner is "assuming about the world". Sometimes the bias turns out to be a simple and interpretable assumption, and sometimes it does not.

What is the inductive bias:

a. For a version space learner that keeps collecting instances until the version space contains only one hypothesis.

b. For decision trees (built by ID3)?

Solution

The inductive bias is the minimal set of extra assumptions that guarantees the correctness of the inductive leap (or: that guarantees that the learner learns the target theory).

a. $c \in H$.

b. The target theory must be *equivalent* to a decision tree T such that:

   (a) each internal node N of T contains the attribute which has the highest information gain (measured on the training set S_N).

   (b) for each internal node N of T there must exist some training examples in S_N with a different class (if no such examples would exist, then ID3 would generate a leaf and the tree would differ from T).

This shows that the version space approach is "correct" w.r.t. the assumption that the target hypothesis is in the hypothesis space (i.e. if the target hypothesis is in the hypothesis space, then the result is guaranteed to be correct). For the tree learner we cannot say much more than that it is correct if... it is correct. The bias amounts to simply stating that the result is correct if the way in which the algorithm works leads to a correct result.

2.2 Rule Induction

Consider the data set shown in Table 1.

a. Show the candidate literals for the first iteration of:

   - the top-down *LearnOneRule* algorithm.
   - the top-down example-driven *LearnOneRule* algorithm (first pick e_1).

b. Find a set of rules that covers all positive examples of Table 1 using the top-down example-driven *LearnRuleSet* algorithm (always start with the first positive example that is not covered). Use accuracy on the covered examples as heuristic to guide the search.

c. Find a rule that covers at least e_1 using a bottom-up version of the example-driven *LearnOneRule* algorithm. Again use accuracy to guide the search.

ex	temp	hum	wind	play
e_1	mild	norm	strong	pos
e_2	mild	norm	weak	pos
e_3	cool	norm	weak	pos
e_4	mild	high	strong	neg
e_5	cool	norm	strong	neg

Table 1: Example data set for rule induction.

Solution

a. The candidate literals are:

- $temp = mild, temp = cool, hum = norm, hum = high, wind = strong, wind = weak$
- $temp = mild, hum = norm, wind = strong$

b. Notice that the hypothesis space for the example-driven version is much smaller. We only search for rules that cover e_1. The accuracy of the candidates is shown in Table 2.

Literal	Accuracy
temp=mild	66%
hum=norm	75%
wind=strong	33%

Table 2: Candidates for top-down example-driven rule induction.

The second one is the best and the new rule becomes IF $hum = norm$ THEN pos. The rule covers a negative example so we have to make it more specific. We obtain (because temp=mild is the best candidate now):

IF $hum = norm \wedge temp = mild$ THEN pos

This rule covers two positive examples. We remove these and build a rule for the remaining positive example e_3. We obtain (because wind=weak has accuracy 100%):

IF $wind = weak$ THEN pos

c. We start with the most specific rule that covers e_1:

IF $temp = mild \wedge hum = norm \wedge wind = strong$ THEN pos

The accuracy of the candidate rules that are more general than this rule are shown in Table 3. We select the first one as best rule. This rule can not be made more general without covering negative examples.

Rule	Accuracy
$temp = mild \wedge hum = norm$	100%
$temp = mild \wedge wind = strong$	50%
$hum = norm \wedge wind = strong$	50%

Table 3: Candidates for bottom-up example-driven rule induction

2.3 Genetic Algorithms for Rule Induction

Consider a data set with 2 nominal attributes: *outlook* (possible values: *sunny*, *overcast* and *rain*), *wind* (*weak* and *strong*) and the class attribute *play* (*yes* and *no*).

a. Use the representation defined in the slides to represent the following rule sets that predict the *play* attribute.

- IF ($outlook = sunny \vee outlook = overcast$) $\wedge$ $wind = weak$ THEN yes,
  IF $outlook = rain$ THEN no.
- IF $outlook = sunny$ THEN yes,
  IF $outlook = rain \wedge wind = strong$ THEN no.

b. Apply two-point cross-over to the bit-strings you obtained. Place the cross-over points in string one after bit 1 (the actual first bit of the string, **not** counting from zero) and bit 4 and in string two after bit 1 and bit 10. Which new rule sets do you obtain?

c. Assume (i) a data set with 4 attributes A,B,C, and D (ii) a bit stream encoding for rules that employs 2 bits for each of the attributes (01 = needs to have value *false*, 10 = needs to have value *true*, 11 = *don't care*, 00 = always *false*) (iii) each rule encodes the attributes in their alphabetical order. Consider the rules "IF $A \wedge B$ THEN *yes*" and "IF $C \wedge D$ THEN *yes*". Is it possible to obtain "IF $A \wedge D$ THEN *yes*" and "IF $B \wedge C$ THEN *yes*" with two-point cross-over?

d. Are there restrictions on the position of the cross-over points? Under which conditions will the child chromosomes be valid rule sets?

Solution

a. The rules can be represented by these bit-strings:

- `110 10 1 001 11 0`
- `100 11 1 001 01 0`

b. We obtain these rule sets:

- `100 11 1 001 00 1 001 11 0` $\rightarrow$
  IF *outlook* = *sunny* THEN *yes*,
  IF *outlook* = *rain* $\wedge$ *false* THEN *yes*,
  IF *outlook* = *rain* THEN *no*.
- `110 11 0` $\rightarrow$
  IF *outlook* = *sunny* $\vee$ *outlook* = *overcast* THEN *no*.

c. No. This is only possible with 3-point or uniform cross-over.

d. Yes. If one parent is being cut $k \geq 0$ bits after a rule boundary, then the corresponding cross-over point in the second parent must also be k bits after a rule boundary.

2.4 Instance Based Learning

Instance based learning consists of storing examples and comparing new instances with the examples, taking the target values of the examples nearest to the new instance to compute a prediction for the new instance.

Instance based classification requires a distance measure. When data can be represented as n-dimensional vectors with numeric or symbolic components, a generally usable distance function is

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum w_i \cdot d'(x_i, y_i)^2}$$

where for symbolic values d' is defined by $d'(x, y) = 0 \Leftrightarrow x = y$, otherwise 1; and for numerical values $d'(x, y) = |x - y|$. Note that for purely numerical vectors, with $w_i = 1$ we obtain the Euclidean distance, whereas for purely symbolic vectors the same weights give us the Hamming distance.

In this exercise the instance space is $\{true, false\} \times \mathbb{R}$. Unless specified otherwise, use as a distance criterion the function mentioned above.

Consider the following points:

Example	A	B	Class
1	true	1	+
2	false	3	+
3	true	5	+
4	false	2	-

a. Classify (false,4) with Nearest-Neighbour

b. Classify (false,4) with 3NN

c. Compute a prototype for all positive examples and a prototype for all negatives examples.

d. Classify (false,4) using the prototypes

e. We have seen that nearest neighbour methods may severely suffer from irrelevant attributes, and two ways to compensate for this. One way is using cross-validations, another was the following:

   - first normalise the different components to a 0-1 domain; i.e., for each numerical component take the maximal and minimal value and rescale it so that the minimum becomes 0 and the maximum becomes 1. This will allow to better compare numerical and symbolic values.
   - Then compute weights w_i according to $w_i = 1 - \frac{1}{n}\sum_{k=1}^{c}\sum_{j=1}^{n_k} d'(p_{ki}, x_{ji})$ where k denotes a class, $\mathbf{p}_k$ is the prototype of the class, c is the number of classes and n_k is the number of examples in class k. Intuitively, this corresponds to assigning a larger weight to components of the vector that, given one class value, have similar values; and assigning a smaller weight to components that vary greatly within one class.

   Repeat d. with the normalisation and adapted weights.

Solution

a. $d(e1, (false, 4)) = \sqrt{1+9} = 3.16$
   $d(e2, (false, 4)) = \sqrt{1} = 1$
   $d(e3, (false, 4)) = \sqrt{1+1} = 1.41$
   $d(e4, (false, 4)) = \sqrt{4} = 2$
   closest to $e2 \Rightarrow$ Positive

b. closest to $e2$, $e3$ and $e4 \Rightarrow$ 2 pos, 1 neg $\Rightarrow$ Positive

c. $p_+ = (true, 3)$; $p_- = (false, 2)$

d. $d(p_+, (false, 4)) = \sqrt{1+1} = 1.41$
   $d(p_-, (false, 4)) = \sqrt{0+4} = 2$
   $\Rightarrow$ Positive

e. After rescaling: $(false, 4) \rightarrow (false, 0.75)$, $p_+ = (true, 0.5)$, $p_- = (false, 0.25)$.
   $w_A = 1 - 0.25 * [(0 + 1 + 0) + (0)] = 0.75$
   $w_B = 1 - 0.25 * [(0.5 + 0 + 0.5) + (0)] = 0.75$
   $d(p_+, (false, 0.75)) = \sqrt{0.75 \times |false - true|^2 + 0.75 \times |0.5 - 0.75|^2)} = \sqrt{0.75 \times (1 + 0.25^2)} = 0.89$
   $d(p_-, (false, 0.75)) = \sqrt{0.75 \times (0 + |0.25 - 0.75|^2)} = 0.43$
   $\Rightarrow$ Negative

2.5 Voronoi Maps

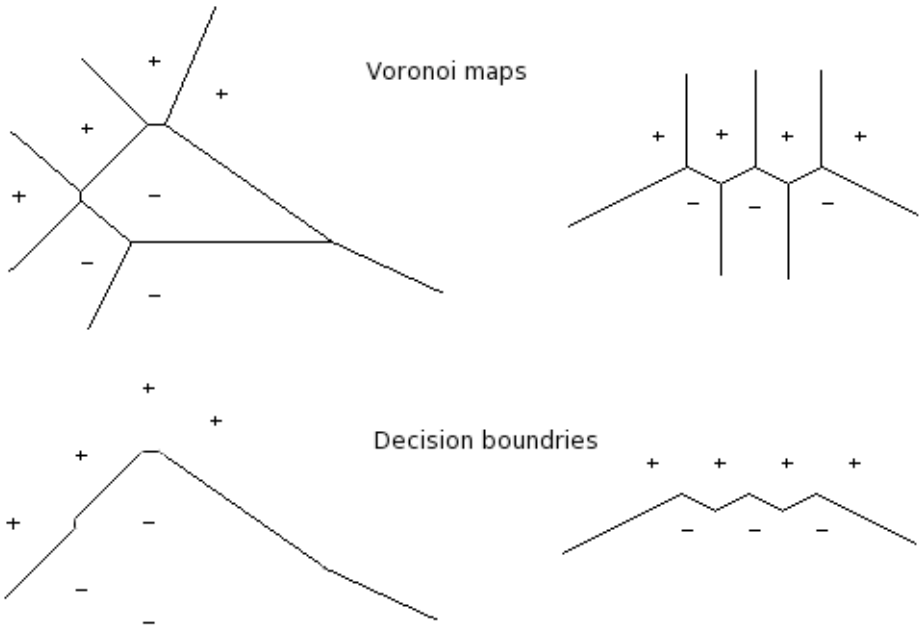
Sketch a Voronoi map for the 1NN method for the following examples, and indicate the decision surface that results.

What would the decision surface look like if you used the prototype based approach (replace the set of positive examples by a single example lying in the middle of them, same for the negatives)?

Think about some other differences between the prototype based approach and storing each individual example.

Solution

With prototypes: faster when making predictions (fewer examples to compare with); stronger and more explicit generalisations are made; flatter decision surfaces are obtained.



2.6 Using Weka: Homework

For more information on Weka, see Exercise 1.9 from the previous session. This exercise assumes you have installed Weka on your computer.

1. As in Exercise 1.9, download and unzip the file `datasets.zip` from Toledo (`Course Documents/Session 1/Weka Data Sets/`). Open the `zoo.arff` file in the Weka Explorer.
2. Go to the classifier tab and select the rule learning algorithm PART. Click on the line behind the choose button. This shows you the parameters you can set and a button called 'More'.
3. Click the start button to run the algorithm. Which percentage of instances is correctly classified by PART? Which families are mistaken for each other? (Hint: Take a look at the confusion matrix.)
4. Compare the performance of PART and interpretability of the produced results to those of j48, which you observed in Exercise 1.9.
5. Also compare the two algorithms on the `vehicle` and the (rather actual) `credit-g` data-sets. Feel free to try other data sets as well.