UNIT- II

Chapter-3

Beyond binary classification

Handling more than two classes

How to evaluate multi-class performance and how to build multi-class models out of binary models.

→Multi-class classification

→Multi-class scores and probabilities

Multi-class classification: **multiclass** or **multinomial classification** is the problem of classifying instances into one of three or more classes. (Classifying instances into one of two classes is called binary classification.)

The existing multi-class classification techniques can be categorized into

(i) Transformation to binary

(ii) Extension from binary(Multi-class scores and probabilities)

(iii) Hierarchical classification.

1. Transformation to binary

This section discusses strategies for reducing the problem of multiclass classification to multiple binary classification problems. It can be categorized into One vs Rest and One vs One. The techniques developed based on reducing the multi-class problem into multiple binary problems can also be called problem transformation techniques.

One-vs.-rest

One-vs.-rest (or *one-vs.-all*, OvA or OvR, *one-against-all*, OAA) strategy involves training a single classifier per class, with the samples of that class as positive samples and all other samples as negatives. This strategy requires the base classifiers to produce a real-valued confidence score for its decision, rather than just a class label; discrete class labels alone can lead to ambiguities, where multiple classes are predicted for a single sample.

In pseudocode, the training algorithm for an OvA learner constructed from a binary classification learner L is as follows:

Inputs:

- L, a learner (training algorithm for binary classifiers)
- samples X
- labels y where $y_i \in \{1, \dots K\}$ is the label for the sample X_i

Output:

- a list of classifiers f_k for $k \in \{1, \dots, K\}$

Procedure:

- For each k in $\{1, \dots, K\}$
  - Construct a new label vector z where $z_i = y_i$ if $y_i = k$ and $z_i = 0$ otherwise

- Apply L to X, z to obtain f_k

Making decisions means applying all classifiers to an unseen sample x and predicting the label k for which the corresponding classifier reports the highest confidence score:

> Although this strategy is popular, it is a heuristic that suffers from several problems. Firstly, the scale of the confidence values may differ between the binary classifiers. Second, even if the class distribution is balanced in the training set, the binary classification learners see unbalanced distributions because typically the set of negatives they see is much larger than the set of positives.

One-vs.-one

In the *one-vs.-one* (OvO) reduction, one trains K (K − 1) / 2 binary classifiers for a K-way multiclass problem; each receives the samples of a pair of classes from the original training set, and must learn to distinguish these two classes. At prediction time, a voting scheme is applied: all K (K − 1) / 2 classifiers are applied to an unseen sample and the class that got the highest number of "+1" predictions gets predicted by the combined classifier.

Like OvR, OvO suffers from ambiguities in that some regions of its input space may receive the same number of votes.

2. Multi-class scores and probabilities

Extension from binary This section discusses strategies of extending the existing binary classifiers to solve multi-class classification problems. Several algorithms have been developed based on neural networks, decision trees, k-nearest neighbors, naive Bayes, support vector machines and Extreme Learning Machines to address multi-class classification problems. These types of techniques can also be called algorithm adaptation techniques.

Neural networks

Multiclass perceptrons provide a natural extension to the multi-class problem. Instead of just having one neuron in the output layer, with binary output, one could have N binary neurons leading to multi-class classification. In practice, the last layer of a neural network is usually a softmax function layer, which is the algebraic simplification of N logistic classifiers, normalized per class by the sum of the N-1 other logistic classifiers.

Extreme learning machines

Extreme Learning Machines (ELM) is a special case of single hidden layer feed-forward neural networks (SLFNs) where in the input weights and the hidden node biases can be chosen at random. Many variants and developments are made to the ELM for multiclass classification.

k-nearest neighbours

k-nearest neighbors kNN is considered among the oldest non-parametric classification algorithms. To classify an unknown example, the distance from that example to every other

training example is measured. The k smallest distances are identified, and the most represented class by these k nearest neighbours is considered the output class label.

Naive Bayes

Naive Bayes is a successful classifier based upon the principle of maximum a posteriori (MAP). This approach is naturally extensible to the case of having more than two classes, and was shown to perform well in spite of the underlying simplifying assumption of conditional independence.

Decision trees

Decision tree learning is a powerful classification technique. The tree tries to infer a split of the training data based on the values of the available features to produce a good generalization. The algorithm can naturally handle binary or multiclass classification problems. The leaf nodes can refer to either of the K classes concerned.

Support vector machines

Support vector machines are based upon the idea of maximizing the margin i.e. maximizing the minimum distance from the separating hyperplane to the nearest example. The basic SVM supports only binary classification, but extensions have been proposed to handle the multiclass classification case as well. In these extensions, additional parameters and constraints are added to the optimization problem to handle the separation of the different classes.

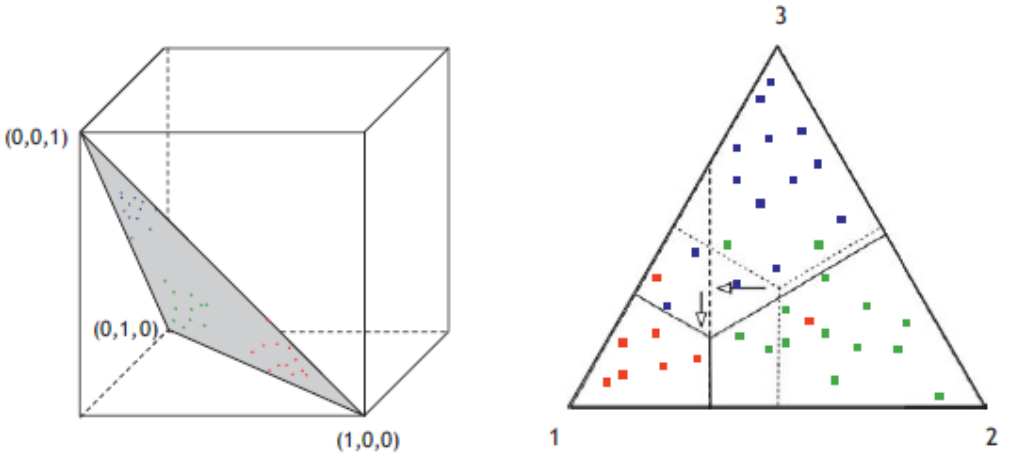


Figure 3.1. **(left)** Triples of probabilistic scores represented as points in an equilateral triangle connecting three corners of the unit cube. **(right)** The arrows show how the weights are adjusted from the initial equal weights (dotted lines), first by optimising the separation of C_2 against C_1 (dashed line), then by optimising the separation of C_3 against the other two classes (solid lines). The end result is that the weight of C_1 is considerably decreased, to the benefit of the other two classes.

$w_1 = 1$. Unfortunately, finding a globally optimal weight vector is computationally intractable. A heuristic approach that works well in practice is to first learn w_2 to optimally separate C_2 from C_1 as in the two-class case; then learn w_3 to separate C_3 from $C_1 \cup C_2$, and so on.

3. Hierarchical classification

Hierarchical classification tackles the multi-class classification problem by dividing the output space i.e. into a tree. Each parent node is divided into multiple child nodes and the process is continued until each child node represents only one class. Several methods have been proposed based on hierarchical classification.

Regression

A *function estimator*, also called a *regressor*, is a mapping $\hat{} f : X \rightarrow$R. The regression learning problem is to learn a function estimator from examples $(xi , f (xi))$

Regression models are used to predict a continuous value. Predicting prices of a house given the features of house like size, price etc is one of the common examples of Regression. It is a supervised technique.

Types of Regression

1. Simple Linear Regression
2. Polynomial Regression
3. Support Vector Regression
4. Decision Tree Regression
5. Random Forest Regression

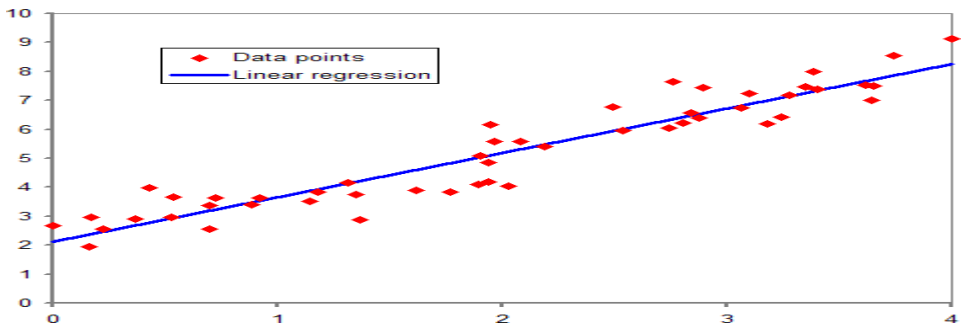
1. Simple Linear Regression

This is one of the most common and interesting type of Regression technique. Here we predict a target variable Y based on the input variable X. A linear relationship should exist between target variable and predictor and so comes the name Linear Regression.

Consider predicting the salary of an employee based on his/her age. We can easily identify that there seems to be a correlation between employee's age and salary (more the age more is the salary). The hypothesis of linear regression is

$$Y = a + bX$$

Y represents salary, X is employee's age and a and b are the coefficients of equation. So in order to predict Y (salary) given X (age), we need to know the values of a and b (the model's coefficients).



While training and building a regression model, it is these coefficients which are learned and fitted to training data. The aim of training is to find a best fit line such that cost function is minimized. The cost function helps in measuring the error. During training process we try to minimize the error between actual and predicted values and thus minimizing cost function.

In the figure, the red points are the data points and the blue line is the predicted line for the training data. To get the predicted value, these data points are projected on to the line.

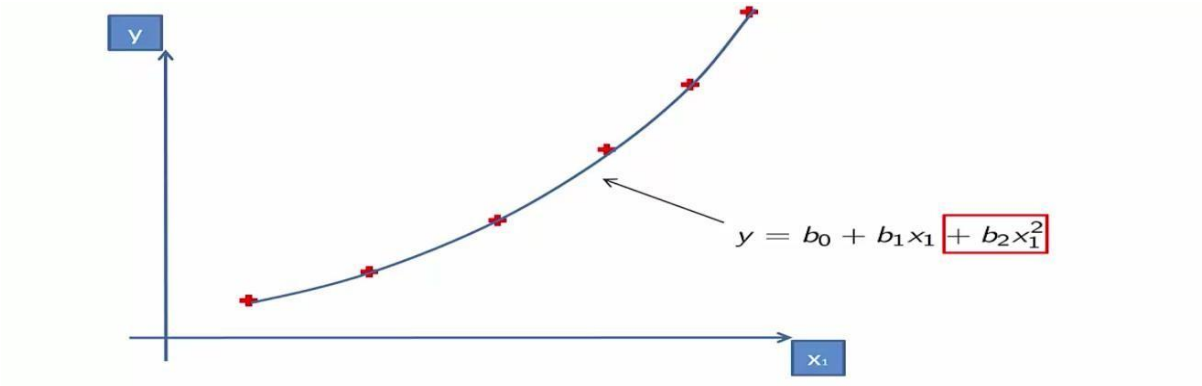
To summarize, our aim is to find such values of coefficients which will minimize the cost function. The most common cost function is **Mean Squared Error** (MSE) which is equal to average squared difference between an observation's actual and predicted values. The coefficient values can be calculated using **Gradient Descent** approach which will be discussed in detail in later articles. To give a brief understanding, in Gradient descent we start with some random values of coefficients, compute gradient of cost function on these values, update the coefficients and calculate the cost function again. This process is repeated until we find a minimum value of cost function.

2. Polynomial Regression

In polynomial regression, we transform the original features into polynomial features of a given degree and then apply Linear Regression on it. Consider the above linear model Y = a+bX is transformed to something like

$$Y = a+bX+cX^2$$

It is still a linear model but the curve is now quadratic rather than a line. Scikit-Learn provides PolynomialFeatures class to transform the features.

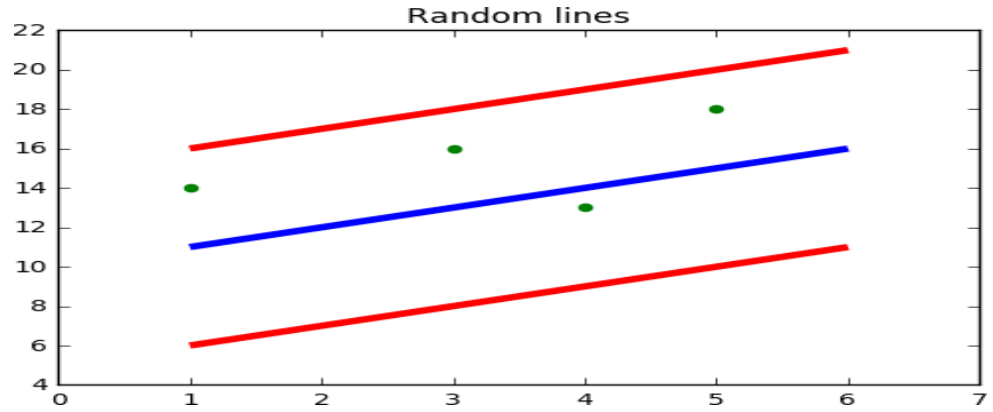


If we increase the degree to a very high value, the curve becomes overfitted as it learns the noise in the data as well.

3. Support Vector Regression

In SVR, we identify a hyperplane with maximum margin such that maximum number of data points are within that margin. SVRs are almost similar to SVM classification algorithm. We will discuss SVM algorithm in detail in my next article.

Instead of minimizing the error rate as in simple linear regression, we try to fit the error within a certain threshold. Our objective in SVR is to basically consider the points that are within the margin. **Our best fit line is the hyperplane that has maximum number of points.**



4. Decision Tree Regression

Decision trees can be used for classification as well as regression. In decision trees, at each level we need to identify the splitting attribute. In case of regression, the ID3 algorithm can be used to identify the splitting node by *reducing standard deviation (in classification information gain is used)*.

A decision tree is built by partitioning the data into subsets containing instances with similar values (homogenous). Standard deviation is used to calculate the homogeneity of a numerical sample. If the numerical sample is completely homogeneous, its standard deviation is zero.

The steps for finding splitting node is briefly described as below:

1. Calculate standard deviation of target variable using below formula.

$$\textbf{Standard Deviation} = S = \sqrt{\frac{\Sigma(x-\bar{x})^2}{n}}$$

2. Split the dataset on different attributes and calculate standard deviation for each branch (standard deviation for target and predictor). This value is subtracted from the standard deviation before the split. The result is the standard deviation reduction.

$$SDR(T,X) = S(T) - S(T,X)$$

3. The attribute with the largest standard deviation reduction is chosen as the splitting node.

4. The dataset is divided based on the values of the selected attribute. This process is run recursively on the non-leaf branches, until all data is processed.

To avoid overfitting, Coefficient of Deviation (**CV**) is used which decides when to stop branching. **Finally the average of each branch is assigned to the related leaf node (in regression mean is taken where as in classification mode of leaf nodes is taken).**

5. Random Forest Regression

Random forest is an ensemble approach where we take into account the predictions of several decision regression trees.

1. Select K random points
2. Identify n where n is the number of decision tree regressors to be created. Repeat step 1 and 2 to create several regression trees.
3. The average of each branch is assigned to leaf node in each decision tree.
4. To predict output for a variable, the average of all the predictions of all decision trees are taken into consideration.

Random Forest prevents overfitting (which is common in decision trees) by creating random subsets of the features and building smaller trees using these subsets.

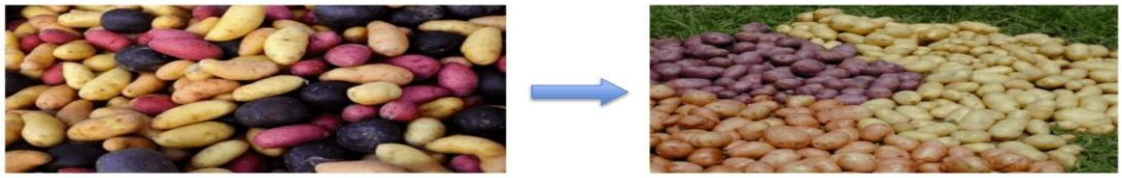
The above explanation is a brief overview of each regression type.

Unsupervised and descriptive learning

- Unsupervised machine learning finds all kind of unknown patterns in data.
- Unsupervised methods help you to find features which can be useful for categorization.
- It is taken place in real time, so all the input data to be analyzed and labeled in the presence of learners.
- It is easier to get unlabeled data from a computer than labeled data, which needs manual intervention.

Types of Unsupervised Learning

Unsupervised learning problems further grouped into clustering and association problems. Clustering



Clustering is an important concept when it comes to unsupervised learning. It mainly deals with finding a structure or pattern in a collection of uncategorized data. Clustering algorithms will process your data and find natural clusters(groups) if they exist in the data. You can also modify how many clusters your algorithms should identify. It allows you to adjust the granularity of these groups.

There are different types of clustering you can utilize:

Exclusive (partitioning)

In this clustering method, Data are grouped in such a way that one data can belong to one cluster only.

Example: K-means

Agglomerative

In this clustering technique, every data is a cluster. The iterative unions between the two nearest clusters reduce the number of clusters.

Example: Hierarchical clustering

Overlapping

In this technique, fuzzy sets is used to cluster data. Each point may belong to two or more clusters with separate degrees of membership.

Descriptive Learning : Using descriptive analysis you came up with the idea that, two products A (Burger) and B (french fries) are brought together with very high frequency. Now you want that if user buys A then machine should automatically give him a suggestion to buy B. So by seeing past data and deducing what could be the possible factors influencing this situation can be achieved using ML.

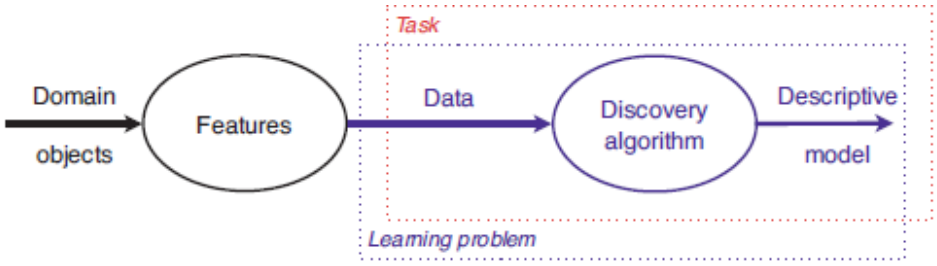


Figure 3.4. In descriptive learning the task and learning problem coincide: we do not have a separate training set, and the task is to produce a descriptive model of the data.

Predictive Learning : We want to increase our sales, using descriptive learning we came to know about what could be the possible factors influencing sales. By tuning the parameters in such a way so that sales should be maximized in the next quarter, and therefore predicting what sales we could generate and hence making investments accordingly. This task can be handled using ML also.

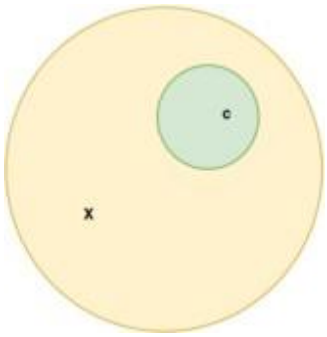
Chapter-4
Concept learning

Concept learning, also known as **category learning.** "The search for and listing of attributes that can be used to distinguish exemplars from non exemplars of various categories". It is Acquiring the definition of a general category from given sample positive and negative training examples of the category.

Much of human learning involves acquiring general concepts from past experiences. For example, humans identify different vehicles among all the vehicles based on specific sets of features defined over a large set of features. This special set of features differentiates the subset of cars in a set of vehicles. This set of features that differentiate cars can be called a concept.

Similarly, machines can learn from concepts to identify whether an object belongs to a specific category by processing past/training data to find a hypothesis that best fits the training examples.

Target concept:



The set of items/objects over which the concept is defined is called the set of instances and denoted by X. The concept or function to be learned is called the target concept and denoted by c. It can be seen as a boolean valued function defined over X and can be represented as **c: X -> {0, 1}**.

If we have a set of training examples with specific features of target concept C, the problem faced by the learner is to estimate C that can be defined on training data.

H is used to denote the set of all possible hypotheses that the learner may consider regarding the identity of the target concept. The goal of a learner is to find a hypothesis H that can identify all the objects in X so that **h(x) = c(x) for all x in X**.

An algorithm that supports concept learning requires:

1. **Training data** (past experiences to train our models)
2. **Target concept** (hypothesis to identify data objects)
3. **Actual data objects** (for testing the models)

The hypothesis space

Each of the data objects represents a concept and hypotheses. Considering a hypothesis **<true, true, false, false>** is more specific because it can cover only one sample. Generally, we can add some notations into this hypothesis. We have the following notations:

1. □ (represents a hypothesis that rejects all)
2. **< ? , ? , ? , ? >** (accepts all)
3. **<true, false, ? , ? >** (accepts some)

The hypothesis □ will reject all the data samples. The hypothesis **<? , ? , ? , ? >** will accept all the data samples. The **?** notation indicates that the values of this specific feature do not affect the result.

The total number of the possible hypothesis is **(3 * 3 * 3 * 3) + 1** — 3 because one feature can have either true, false, or **?** and one hypothesis for rejects all (□).

General to Specific

Many machine learning algorithms rely on the concept of general-to-specific ordering of hypothesis.

1. h1 = < true, true, ?, ? >
2. h2 = < true, ? , ? , ? >

Any instance classified by h1 will also be classified by h2. We can say that h2 is more general than h1. Using this concept, we can find a general hypothesis that can be defined over the entire dataset X.

To find a single hypothesis defined on X, we can use the concept of being more general than partial ordering. One way to do this is start with the most specific hypothesis from H and generalize this hypothesis each time it fails to classify and observe positive training data object as positive.

1. The first step in the Find-S algorithm is to start with the most specific hypothesis, which can be denoted by **h <-** <□, □, □, □>.
2. This step involves picking up next training sample and applying Step 3 on the sample.
3. The next step involves observing the data sample. If the sample is negative, the hypothesis remains unchanged and we pick the next training sample by processing Step 2 again. Otherwise, we process Step 4.
4. If the sample is positive and we find that our initial hypothesis is too specific because it does not cover the current training sample, then we need to update our current hypothesis. This can be done by the pairwise conjunction (logical *and* operation) of the current hypothesis and training sample.

   If the next training sample is **<true, true, false, false>** and the current hypothesis is <□, □, □, □>, then we can directly replace our existing hypothesis with the new one.

If the next positive training sample is **<true, true, false, true>** and current hypothesis is **<true, true, false, false>**, then we can perform a pairwise conjunctive. With the current hypothesis and next training sample, we can find a new hypothesis by putting **?** in the place where the result of conjunction is false:

<true, true, false, true> □ **<true, true, false, false> = <true, true, false, ?>**

Now, we can replace our existing hypothesis with the new one: **h <-<true, true, false, ?>**

5. This step involves repetition of Step 2 until we have more training samples.
6. Once there are no training samples, the current hypothesis is the one we wanted to find. We can use the final hypothesis to classify the real objects.

Paths through the hypothesis space

As we can clearly see in Figure 4.4, in this example we have not one but two most general hypotheses. What we can also notice is that *every concept between the least general one and one of the most general ones is also a possible hypothesis*, i.e., covers all the positives and none of the negatives. Mathematically speaking we say that the set of **Algorithm 4.3:** LGG-Conj-ID(*x*, *y*) – find least general conjunctive generalisation of two conjunctions, employing internal disjunction.

Input : conjunctions *x*, *y*.
Output : conjunction *z*.
1 *z* ←true;
2 for each feature *f* **do**
3 if *f* = *vx* is a conjunct in *x* and *f* = *vy* is a conjunct in *y* **then**
4 add *f* = Combine-ID(*vx* , *vy*) to *z*; // Combine-ID: see text
5 end
6 end
***7* return** *z*

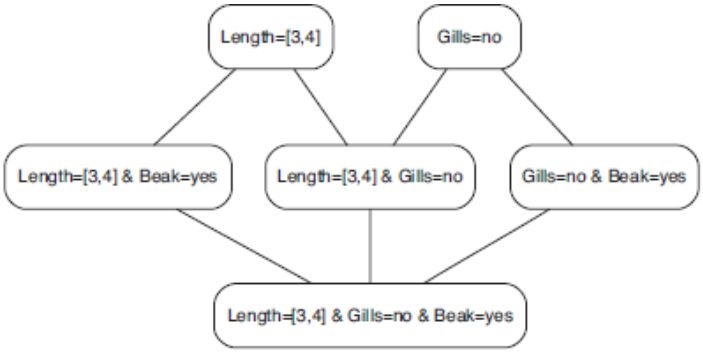


Figure 4.4. **(top)** A snapshot of the expanded hypothesis space that arises when internal disjunction is used for the 'Length' feature. We now need one more generalisation step to travel upwards from a completely specified example to the empty conjunction. **(bottom)** The version space consists of one least general hypothesis, two most general hypotheses, and three in between.

hypotheses that agree with the data is a *convex set*, which basically means that we can interpolate between any two members of the set, and if we find a concept that is less general than one and more general than the other then that concept is also a member of the set. This in turn means that we can describe the set of all possible hypotheses by its least and most general members. This is summed up in the following definition.

Definition 4.1 (Version space). *A concept is* complete *if it covers all positive examples. A concept is* consistent *if it covers none of the negative examples. The* version space *is the set of all complete and consistent concepts. This set is convex and is fully defined by its least and most general elements.*

www.jntufastupdates.com

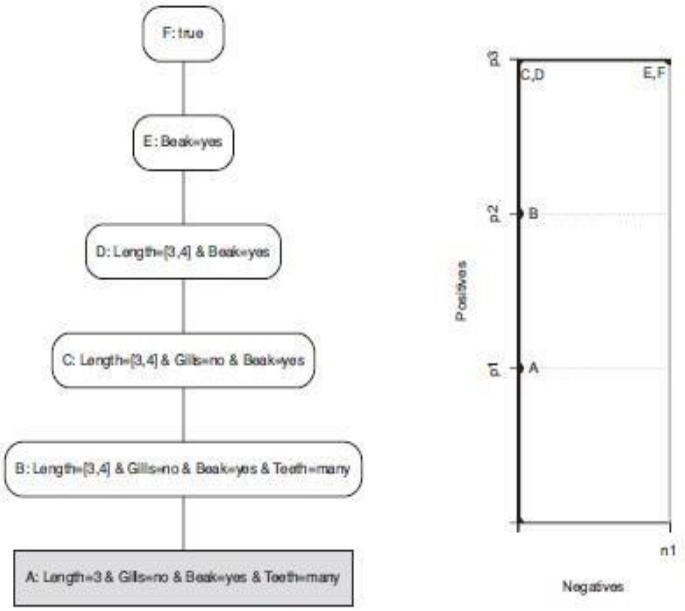


Figure 4.5. **(left)** A path in the hypothesis space of Figure 4.3 from one of the positive examples (p1, see Example 4.2 on p.110) all the way up to the empty concept. Concept A covers a single example; B covers one additional example; C and D are in the version space, and so cover all three positives; E and F also cover the negative. **(right)** The corresponding coverage curve, with ranking p1 – p2 – p3 – n1.

www.jntufastupdates.com

Beyond conjunctive concepts

Recall from Background 4.1 that a conjunctive normal form expression (CNF) is a conjunction of disjunctions of literals, or equivalently, a conjunction of clauses. The conjunctions of literals we have looked at until now are trivially in CNF where each disjunction consists of a single literal. CNF expressions are much more expressive, particularly since literals can occur in several clauses. We will look at an algorithm for learning Horn theories, where each clause $A \rightarrow B$ is a Horn clause, i.e., A is a conjunction of literals and B is a single literal. For ease of notation we will restrict attention to Boolean features, and write F for F = true and ¬F for F = false. In the example below we adapt the dolphins example to Boolean variables ManyTeeth (standing for Teeth = many), Gills, Short (standing for Length = 3) and Beak.

When we looked at learning conjunctive concepts, the main intuition was that uncovered positive examples led us to generalise by dropping literals from the conjunction, while covered negative examples require specialisation by adding literals. This intuition still holds if we are learning Horn theories, but now we need to think 'clauses' rather than 'literals'. Thus, if a Horn theory doesn't cover a positive we need to drop all clauses that violate the positive, where a clause $A \rightarrow B$ violates a positive if all literals in the conjunction A are true in the example, and B is false.

Things get more interesting if we consider covered negatives, since then we need to find one or more clauses to add to the theory in order to exclude the negative. For example, suppose that our current hypothesis covers the negative

$$\text{ManyTeeth} \wedge \text{Gills} \wedge \text{Short} \wedge \neg\text{Beak}$$

To exclude it, we can add the following Horn clause to our theory:

$$\text{ManyTeeth} \wedge \text{Gills} \wedge \text{Short} \rightarrow \text{Beak}$$

While there are other clauses that can exclude the negative (e.g., ManyTeeth → Beak) this is the most specific one, and hence least at risk of also excluding covered positives. However, the most specific clause excluding a negative is only unique if the negative has exactly one literal set to false. For example, if our covered negative is

$$\text{ManyTeeth} \wedge \text{Gills} \wedge \neg\text{Short} \wedge \neg\text{Beak}$$

then we have a choice between the following two Horn clauses:

$$\text{ManyTeeth} \wedge \text{Gills} \rightarrow \text{Short}$$
$$\text{ManyTeeth} \wedge \text{Gills} \rightarrow \text{Beak}$$

Notice that, the fewer literals are set to true in the negative example, the more general the clauses excluding the negative are.

www.jntufastupdates.com

The approach of Algorithm 4.5 is to add *all* of these clauses to the hypothesis. However, the algorithm applies two clever tricks. The first is that it maintains a list S of negative examples, from which it periodically rebuilds the hypothesis. The second is that, rather than simply adding new negative examples to the list, it tries to find negatives with fewer literals set to true, since this will result in more general clauses. This is possible if we assume we have access to a *membership oracle* Mb which can tell us whether a particular example is a member of the concept we're learning or not. So in line 7 of the algorithm we form the *intersection* of a new negative x and an existing one $s \in S$ – i.e., an example with only those literals set to true which are true in both x and s – and pass the result z to the membership oracle to check whether it belongs to the target concept. The algorithm also assumes access to an *equivalence oracle* Eq which either tells us that our current hypothesis h is logically equivalent to the target formula f, or else produces a *counter-example* that can be either a false positive (it is covered by h but not by f) or a false negative (it is covered by f but not by h).

Algorithm 4.5: Horn(Mb, Eq) – learn a conjunction of Horn clauses from membership and equivalence oracles.

Input : equivalence oracle Eq; membership oracle Mb.
Output : Horn theory h equivalent to target formula f.

```
1  h ←true;                                   // conjunction of Horn clauses, initially empty
2  S ←∅ ;                                     // a list of negative examples, initially empty
3  while Eq(h) returns counter-example x do
4      if x violates at least one clause of h then        // x is a false negative
5          specialise h by removing every clause that x violates
6      else                                               // x is a false positive
7          find the first negative example s ∈ S such that (i) z = s ∩ x has fewer true
           literals than s, and (ii) Mb(z) labels it as a negative;
8          if such an example exists then replace s in S with z, else append x to the
           end of S;
9          h ←true;
10         for all s ∈ S do                               // rebuild h from S
11             p ←the conjunction of literals true in s;
12             Q ←the set of literals false in s;
13             for all q ∈ Q do h ←h ∧ (p → q);
14         end
15     end
16 end
17 return h
```

___XXX___

www.jntufastupdates.com