67577 – Intro. to Machine Learning **Fall semester, 2004/5**

Lecture 1: Introduction: Learning Models I

Lecturer: Amnon Shashua *Scribe: Amnon Shashua* [1]

The course will focus on learning from observations. We will look into the theory and algorithms of constructing inference engines, that when given a large enough sequence of training "examples", compute a function that matches as close as possible the process generating the data. That is, we hope that the classification of subsequent examples is close to the best performance an inference engine exposed to so much training examples can provide.

Noteworthy examples include spam filters, natural speaking language recognition, handwriting recognition systems (Graffiti, printed, cursive), biometric identification systems and visual pattern recognition (find faces, people, animals, familiar objects, and make visual interpretations in general from image input).

The training data for a specific learning problem can be given all in advance. For example, in a face detection task one could be given a database of face images covering people under different illumination conditions, pose, hair-style and facial expressions and the learner's task is to generalize — meaning to be able to capture the specific regularities and features that make up a face image and later find faces in novel images. The training data can consist solely of "positive" instances — such as in the face detection example — or consist of positive and "negative" instances (such as pictures of non-faces in the example above). In the latter case we would say that the training data is "labeled". Generally, the number of labels can be larger than two but in this course will focus on the 2-class (positive/negative) label domain only. When the label consists of a continuous (say, real number) domain, the learning problem is referred to as "regression" instead of classification. We will focus on classification problems only in this course.

The raining data, on the other hand, may come in an incremental fashion where the learner is expected to update its model (the target classification function) "on the fly". Notable examples for such settings are missile tracking, portfolio allocation of financial assets, and age pre-fetching in computers with virtual memory. This type of learning setting is called "on-line learning", and to be more concrete here is an illustrative example[2].

Suppose we would like to teach a robot to examine apples and decide whether the apple is of good quality and thus should be sent to a retailer or it's of industrial grade and should be sent to a juice factory. The robot is able to make various tests: it can check the shade of green of each apple; it can weigh the apple; it can squeeze it a little and check how mushy an apple is; and it can measure the circumstance of each apple. Each apple is therefore characterized by the following vector,

$$(green - level, weight, mushy - level, circumstance) \in R^4 \quad .$$

Based on each measurement the robot should decide whether an apple is market quality or industrial quality. The robot has to start classifying apple from the very first apple it sees. However, whenever it labels an apple wrongly it gets a feedback from a human operator indicating that it made an error. Our goal is therefore to a build a "learning" program for the robot so it will make "good" predictions as fast as possible while making as few mistakes as possible. Put another way, the

[1]

[2]This example was taken from Yoram Singer's course notes, IML'02

learning algorithm should gain experience from its prediction mistakes so as to achieve better predictions for apples that are yet to be seen.

In this course we will consider both labeled and unlabeled classification problems and both batch and on-line settings. We will begin with formal definitions of "learning models" including the consistency and mistake bound models (this lecture) and the more advanced "formal learning model" also known as the Probably Approximate Correct (PAC) model in the next lecture. The principles of the PAC model will resurface several times throughout the course when we discuss formal results on generalization (lectures 6,7).

1.1 Notations

We will describe in this section the definitions and notations we will be using in this and subsequent lectures (more notations would be introduced when appropriate).

- The instance space (also called domain), denoted X, is a space from which the observations (measurements) are drawn. Examples:

$$X = \{0,1\}^n,\ X = R^n, X = \Sigma^\star \quad .$$

- Input *instance*, $\mathbf{x} \in X$, is a single observation/measurement. Examples:

$$\mathbf{x} = (0,1,1,1,0,0),\ \mathbf{x} = (0.5,-2.3,0,1,7.2),\ \mathbf{x} = \text{"go ahead make my day"} \quad .$$

  We denote the individual components of $\mathbf{x}$ by x_i thus $\mathbf{x} = (x_1, x_2, \ldots, x_n)$.

- The set of *labels*, denoted Y, is the set of possible outcomes that can be associated with a measurement. The label of an instance $\mathbf{x}$ is denoted y. Examples:

$$Y = \{-1,+1\}\ (y=1),\ Y = R\ (y=7),\ Y = \{\alpha,\beta,\gamma\}\ (y=\beta) \quad .$$

- An *example* is an instance-label pair $(\mathbf{x}, y)$. If $|Y| = 2$ we typically use $\{0,1\}$ or $\{-1,+1\}$ as the set of labels. We say that an example $(\mathbf{x}, y)$ is a positive example if $y = 1$ and otherwise we call it a negative example.

- A training set Z consists of m instance-label pairs:

$$Z = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)) = (\mathbf{z}_1, ..., \mathbf{z}_m).$$

  In some cases we will refer to the instance training set $S = (\mathbf{x}_1, ..., \mathbf{x}_m)$ which consists only of the input observations without the lables.

So far we just defined a general framework. We now introduce the notion of a *concept class* with which we will facilitate learning. A *concept* (or *hypotheses*) class C is a set (not necessarily finite) of functions of the form,

$$C = \{h \mid h : X \to Y\} \quad .$$

Each $h \in C$ is called a concept or hypothesis and is also often referred to using other terms such as a mapping, a predictor, and a classifier. For example, if $X = \{0,1\}^n$ and $Y = \{0,1\}$ then C might be $C = \{h \mid \exists i : h(\mathbf{x}) = x_i\}$. Other examples:

- Decision Trees: when the instance vectors are binary $\mathbf{x} \in \{0,1\}^n$ and $Y = \{True, False\}$ then any binary function can be described by a binary tree. Thus, the concept class C consists of decision trees ($|C| < \infty$).
- Conjunction learning: A conjunction is a special case of a boolean formula: a *literal* is a variable or its negation and a *term* is a conjunction of literals (i.e., $(x_1 \bar{x}_2 \bar{x}_3)$). A target function is is a term which consists of a subset of the literals. In this case $|X| = 2^n$ and $|C| = 3^n$.
- Threshold formula (separating hyperplanes): $X = R^n$, a concept $h(\mathbf{x})$ is specified by a vecotr $\mathbf{w} \in R^n$ and a scalar b such that $h(\mathbf{x}) = 1$ if $\mathbf{w}^\top \mathbf{x} \geq b$ and $h(\mathbf{x}) = -1$ otherwise.

In most cases we will assume the existence of a target concept function $c_t(\mathbf{x}) \in C$ so that the training set Z consists of pairs $(\mathbf{x}_i, c_t(\mathbf{x}_i))$, $i = 1, ..., m$. We will define later what we mean when the target concept is not included in C.

We need to define next what are the goals of the learner in a formal manner. In the PAC model (which will be introduced in the next lecture) the definition is statistical, but today we will introduce a simpler definition for online learning known as the mistake-bound model.

1.2 The Mistake-bound Model

We say that a learning algorithm L is *consistent* if given a training set Z, the algorithm L generates in polynomial time (over the input length) a hypothesis $h \in C$ which satisfies $h(\mathbf{x}_i) = y_i$, $i = 1, ..., m$ if such a hypothesis exists. If such a hypothesis does not exist then L terminates and notifies that such a hypothesis does not exist.

In the mistake-bound model, the goal of the consistent learner is to bound the number of mistakes made during the learning process, i.e., make the *smallest* number of mistakes over the *worst* training set Z. Formally, let $\hat{y}_i$ be the prediction the learner L makes on the input example $\mathbf{x}_i$. If L makes a mistake then $\hat{y}_i \neq y_i$. Then, the (largest) number of errors of L with respect to a *sample* $Z = (\mathbf{x}_1, y_1), \ldots (\mathbf{x}_m, y_m)$ is,

$$error_Z(L) = \max_{c_t \in C} |\{\hat{y}_i \neq c_t(\mathbf{x}_i) \ : \ (\mathbf{x}_i, c_t(\mathbf{x}_i)) \in Z\}| \ .$$

Note that we replace y_i with $c_t(\mathbf{x}_i)$ since we assumed that there exists a concept $c_t \in C$ that generates the labels for instances. However, since we do not know c_t we use a *worst case* definition by taking the maximum over any $h \in C$. The *mistake bound* of L, denoted $error(L)$, is defined as the largest number of errors L that can be obtained on *any* training sample, that is,

$$error(L) = \max_Z \ error_Z(L) \ .$$

Definition 1 (MB model) *Algorithm L learns a concept class C under the mistake-bound model if for every training sample Z which is consistent with some $c_t \in C$ the total number of mistakes done by L is bounded by a polynomial in the size of the input vector $\mathbf{x}$ and the size (number of bits required for a description) of the simplest and concept in C which is consistent with Z.*

To illustrate the use of the MB model, consider the conjunction learning problem mentioned above. Here is an algorithm:

1. Start with the hypothesis $h_0 = x_1\bar{x}_1 \dots x_n\bar{x}_n$. Note that this hypothesis is consistent with all negative examples because it will always return false on any input. Note that this hypothesis is not an *admissible* concept (i.e., does not belong to C) because we cannot have in a conjunction a variable and its negation. So, we artificially add this special case concept to C so that now $|C| = 3^n + 1$.

2. Wait until you get a positive example $\mathbf{x}_i$. Remove the literals which contradict the positive example. In other words, if the j'th coordinate of $\mathbf{x}_i$ is '1' then remove $\bar{x}_j$ from the current hypothesis, otherwise remove x_j. Continue doing so for all positive examples.

3. If the hypothesis is not consistent with a negative example then there is no consistent solution with the training input. Otherwise terminate when all the examples are processed.

Consider the following example: let $n = 4$ and $c_t = x_1\bar{x}_2x_4$ and consider the two positive examples $\mathbf{x}_1 = (1, 0, 0, 1)$ and $\mathbf{x}_2 = (1, 0, 1, 1)$. Then,

$$h_0 = x_1\bar{x}_1x_2\bar{x}_2x_3\bar{x}_3x_4\bar{x}_4, \quad h_1 = x_1\bar{x}_2\bar{x}_3x_4, \quad h_2 = c_t.$$

First we need to convince ourselves that this algorithm will produce a consistent hypothesis if one exists. For that we prove that the invariant of this algorithm is that the subset of literals of the target concept c_t are always contained in the literals of hypothesis h_k after L observes k examples from Z. We can show this (informally) by induction as follows:

- by construction h_0 contains all the literals so in particular c_t is contained in h_0.

- Let h_k be the hypothesis after the k'th example has been proceesed by L. By construction h_k is consistent with the k examples processed so far (because removing literals would not make previous positive examples inconsistent with the current hypothesis). Assume (the induction assumption) that c_t is contained in h_k and we shall prove that h_{k+1} is consistent with the $k + 1$ examples processed and contains c_t.

- if $\mathbf{x}_{k+1}$ is negative, then $h_{k+1} = h_k$ so then h_{k+1} is consistent with the $k + 1$ examples seen so far and contains c_t (because of the induction hypothesis that h_k contains c_t).

- if $\mathbf{x}_{k+1}$ is positive, then the appropriate literals from h_k are removed so that h_{k+1} is consistent with $\mathbf{x}_{k+1}$. Since removing literals does not affect the consistency over the previous positive examples, then h_{k+1} is consistent with all previously seen positive examples. Since we assume that Z is consistent with some hypothesis c_t, then it is not possible that a negative example will be classified as positive by h_{k+1}. Since we removed only those literals which are not contained in c_t, then h_{k+1} continues to contain c_t.

The number of mistakes made by this algorithm is bounded by $n + 1$. The first mistake will reduce the number of literals from $2n$ to n, then each additional mistake will cause the removal of at least one literal. Therefore it is not possible to make more than $n + 1$ mistakes.

What happens if there is no consistent hypothesis (i.e., the training set Z is is not consistent with any conjunction)? Consider a pair of negative and positive contradictory examples from Z. If the negative example is seen first then the algorithm will not necessarily detect the contradiction (since our hypothesis starts by accepting all negative examples). Therefore, if we wish to handle the case of inconsistent training sets then the algorithm should run through Z a number of times — if the number of mistakes exceeds the upper bound then Z is inconsistent.

What about the lower bound? is it possible to introduce a more efficient online algorithm? one that will have a lower upper bound on the number of mistakes? We will show that n is the lower bound on the number of mistakes over the *worst* training sample Z and over all possible learning algorithms L. In other words, the algorithm we presented above is optimal under the mistake-bound model.

We will construct the following training sample S: the i'th example $\mathbf{x}_i$ contains '0' in the i'th coordinate and '1' everywhere else. Given a learning algorithm L, the labels we will associate with S will depend on the predictions L makes on S (recall we want to create the worst training sample Z for L). The target concept c_t will also be determined based on the predictions of L on S (again, our goal is to create the worst conditions for L). This will be done as follows: if L predicts that $\mathbf{x}_i$ is positive, then the literal x_i will be included in c_t and the label y_i would be '-1'. If L predicts that $\mathbf{x}_i$ is negative, then c_t will not include the literal x_i and the label $y_i = 1$. So we have now a target concept c_t and a training sample Z on which L will makes n mistakes (will make a mistake on each example seen). In other words, in order that L makes a mistake on each example, we need to make the labels y_i to be the *opposite* of the prediction L makes on $\mathbf{x}_i$. What would be the target concept consistent with such a training sample? since each example $\mathbf{x}_i$ has only one coordinate of '0' at the position i, then any labeling will be consistent with some target concept and specifically the one where we include the literal x_i in the target concept if $y_i = -1$. As a result, c_t will be consistent with all the negative examples, and because the remaining coordinates are '1' then it is consistent with the positive examples as well.

For illustration, consider the case of $n = 3$. The $\mathbf{x}_1 = (0, 1, 1), \mathbf{x}_2 = (1, 0, 1)$ and $\mathbf{x}_3 = (1, 1, 0)$. Let L predict '+', '-' and '+' on S. Then the labels would be $y_1 = -1, y_2 = 1$ and $y_3 = -1$. The target concept would be $c_t = x_1 x_3$.

The problem with the mistake-bound model is that in real life one does not see the worst training sample. Typically, the training sample Z is drawn from some distribution (unknown, but very likely to be fixed). The number of mistakes made on Z matters less than the number of mistakes made on examples not seen by the learning algorithm (test examples). So, the MB model bounds the number of mistakes but not *when* those mistakes are made. The When issue introduces the statistical component to the learning model which is lacking in the MB model. Furthermore, in addition to a measure of *accuracy* of the learner (say, number of mistakes on test examples), we would like to obtain a measure of *confidence* on the accuracy measure. These notions will be introduced in the PAC learning model discussed in the next lecture.